JAVA

ΠΕΡΙΕΧΟΜΕΝΑ

Java Intro σελ3
Java Get Started σελ3
Java Syntax σελ4
Java Output σελ6
Java Comments σελ7
Java Variables σελ8
Java Data Types σελ13
Java Type Casting σελ18
Java Operators σελ20
Java Strings σελ24
Java Math σελ28
Java Booleans σελ30
Java If...Else σελ33
Java Switch σελ38
Java While Loop σελ42
Java For Loop σελ44
Java Break/Continue σελ48
Java Arrays σελ50

Java Methods

Java Methods σελ56
Java Method Parameters σελ58
Java Method Overloading σελ63
Java Scope σελ65
Java Recursion σελ67

Java Classes

Java OOP σελ69
Java Classes/Objects σελ71
Java Class Attributes σελ74
Java Class Methods σελ78
Java Constructors σελ83
Java Modifiers σελ86
Java Encapsulation σελ92
Java Packages / API σελ95
Java Inheritance σελ98
Java Polymorphism σελ100
Java Inner Classes σελ102

Java Abstraction σελ106
Java Interface σελ109
Java Enums σελ112
Java User Input σελ115
Java Date σελ117

Java Data Structures

Java ArrayList σελ121
Java LinkedList σελ127
Java List Sorting σελ130
Java HashMap σελ133
Java HashSet σελ137
Java Iterator σελ141
Java Wrapper Classes σελ143

Java Advanced

Java Exceptions σελ146
Java RegEx σελ150
Java Threads σελ154
Java Lambda σελ158
Java Advanced Sorting σελ161

Java File Handling

Java Files σελ170
Java Create/Write Files σελ171
Java Read Files σελ174
Java Delete Files σελ176

Τι είναι η Java;

Η Java είναι μια δημοφιλής γλώσσα προγραμματισμού, που δημιουργήθηκε το 1995.

Ανήκει στην Oracle και περισσότερες από 3 δισεκατομμύρια συσκευές τρέχουν Java.

Χρησιμοποιείται για:

- Εφαρμογές για κινητά (ειδικά εφαρμογές Android)
- Εφαρμογές επιτραπέζιου υπολογιστή
- διαδικτυακές εφαρμογές
- Διακομιστές Ιστού και διακομιστές εφαρμογών
- Παιχνίδια
- Σύνδεση βάσης δεδομένων
- Και πολλά, πολλά άλλα!

Γιατί να χρησιμοποιήσετε Java;

- Η Java λειτουργεί σε διαφορετικές πλατφόρμες (Windows, Mac, Linux, Raspberry Pi, κ.λπ.)
- Είναι μια από τις πιο δημοφιλείς γλώσσες προγραμματισμού στον κόσμο
- Έχει μεγάλη ζήτηση στην τρέχουσα αγορά εργασίας
- Είναι εύκολο στην εκμάθηση και απλό στη χρήση
- Είναι ανοιχτού κώδικα και δωρεάν
- Είναι ασφαλές, γρήγορο και ισχυρό
- Έχει τεράστια υποστήριξη κοινότητας (δεκάδες εκατομμύρια προγραμματιστές)
- Η Java είναι μια αντικειμενοστραφή γλώσσα που δίνει μια σαφή δομή στα προγράμματα και επιτρέπει την επαναχρησιμοποίηση του κώδικα, μειώνοντας το κόστος ανάπτυξης
- Καθώς η Java είναι κοντά σε C++ και C#, διευκολύνει τους προγραμματιστές να μεταβούν σε Java ή το αντίστροφο

Εγκατάσταση Java

Ορισμένοι υπολογιστές ενδέχεται να έχουν ήδη εγκατεστημένη Java.

Για να ελέγξετε εάν έχετε εγκαταστήσει Java σε υπολογιστή με Windows, αναζητήστε στη γραμμή έναρξης για Java ή πληκτρολογήστε τα ακόλουθα στη Γραμμή εντολών (cmd.exe):

```
C:\Users\Your Name>java -version
```

Εάν είναι εγκατεστημένη η Java, θα δείτε κάτι σαν αυτό (ανάλογα με την έκδοση):

Εάν δεν έχετε εγκατεστημένη Java στον υπολογιστή σας, μπορείτε να την κατεβάσετε δωρεάν στο oracle.com.

Java Quickstart

Στην Java, κάθε εφαρμογή ξεκινά με ένα **όνομα κλάσης** και αυτή η **κλάση** πρέπει να ταιριάζει με το **όνομα αρχείου.**

Ας δημιουργήσουμε το πρώτο μας αρχείο **Java**, που ονομάζεται **Main.java,** το οποίο μπορεί να γίνει σε οποιοδήποτε πρόγραμμα επεξεργασίας κειμένου (όπως το Σημειωματάριο).

Το αρχείο θα πρέπει να περιέχει ένα μήνυμα "Hello World", το οποίο είναι γραμμένο με τον ακόλουθο κώδικα:

Main.java

```
public class Main {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

Αποθηκεύστε τον κώδικα στο Σημειωματάριο ως "**Main.java**". Ανοίξτε τη γραμμή εντολών (cmd.exe), μεταβείτε στον κατάλογο όπου αποθηκεύσατε το αρχείο σας και πληκτρολογήστε **"javac Main.java**":

```
C:\Users\Your Name>javac Main.java
```

Αυτό θα μεταγλωττίσει τον κώδικα σας. Εάν δεν υπάρχουν σφάλματα στον κώδικα, η γραμμή εντολών θα σας μεταφέρει στην επόμενη γραμμή. Τώρα, πληκτρολογήστε "java Main" για να εκτελέσετε το αρχείο:

```
C:\Users\Your Name>java Main
```

Η έξοδος πρέπει να είναι:

```
Hello World
```

Σύνταξη Java

Στο προηγούμενο κεφάλαιο, δημιουργήσαμε ένα αρχείο Java που ονομάζεται Main.java και χρησιμοποιήσαμε τον ακόλουθο κώδικα για να εκτυπώσουμε το "Hello World" στην οθόνη:

Main.java

```
public class Main {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
```

```
}
```

Το παράδειγμα εξηγείται

Κάθε γραμμή κώδικα που εκτελείται σε **Java** πρέπει να βρίσκεται μέσα σε μια **κλάση**. Και το όνομα της κλάσης πρέπει πάντα να ξεκινά με **ένα κεφαλαίο πρώτο γράμμα**. Στο παράδειγμά μας, ονομάσαμε την κλάση **Main.**

Σημείωση: Η Java κάνει διάκριση πεζών-κεφαλαίων: "MyClass" και "myclass" έχουν διαφορετική σημασία.

Το όνομα του αρχείου **java** πρέπει να **ταιριάζει** με το όνομα της **κλάσης**. Κατά την αποθήκευση του αρχείου, αποθηκεύστε το χρησιμοποιώντας το όνομα της κλάσης και προσθέστε ".java" στο τέλος του ονόματος αρχείου. Για να εκτελέσετε το παραπάνω παράδειγμα στον υπολογιστή σας, βεβαιωθείτε ότι η Java έχει εγκατασταθεί σωστά: Μεταβείτε στο κεφάλαιο Ξεκινήστε για τον τρόπο εγκατάστασης της Java. Η έξοδος θα πρέπει να είναι:

```
Hello World
```

The main Method

Απαιτείται η μέθοδος **main()** και θα τη δείτε σε κάθε πρόγραμμα **Java**:

```
public static void main(String[] args)
```

Οποιοσδήποτε κώδικας μέσα στη μέθοδο **main()** θα εκτελεστεί.

Προς το παρόν, απλώς να θυμάστε ότι κάθε πρόγραμμα **Java** έχει ένα όνομα κλάσης που πρέπει να ταιριάζει με το όνομα αρχείου και ότι κάθε πρόγραμμα πρέπει να περιέχει τη μέθοδο main().

System.out.println()

Μέσα στη μέθοδο **main()**, μπορούμε να χρησιμοποιήσουμε τη μέθοδο **println()** για να εκτυπώσουμε μια γραμμή κειμένου στην οθόνη:

```
public static void main(String[] args) {

  System.out.println("Hello World");

}
```

Σημείωση: Οι σγουρές αγκύλες **{}** σηματοδοτούν την αρχή και το τέλος ενός μπλοκ κώδικα.

Το **System** είναι μια ενσωματωμένη **κλάση Java** που περιέχει χρήσιμα μέλη, όπως **out**, που είναι συντομογραφία του "**output**". Η μέθοδος **println()**, συντομογραφία του "**print line**", χρησιμοποιείται για την εκτύπωση μιας τιμής στην οθόνη (ή σε ένα αρχείο).

Μην ανησυχείτε πολύ για το πώς λειτουργεί το System, out και println(). Απλώς ξέρετε ότι τα χρειάζεστε μαζί για να εκτυπώσετε πράγματα στην οθόνη.

Θα πρέπει επίσης να σημειώσετε ότι κάθε δήλωση κώδικα πρέπει να τελειώνει με ένα ερωτηματικό (**;**).

Java Output / Print

Print Text

Μπορείτε να προσθέσετε όσες μεθόδους println() θέλετε. Σημειώστε ότι θα προσθέσει μια νέα γραμμή για κάθε μέθοδο:

```
System.out.println("Hello World!");
System.out.println("I am learning Java.");
System.out.println("It is awesome!");
```

Double Quotes

Το κείμενο πρέπει να είναι τυλιγμένο μέσα σε διπλά εισαγωγικά "".

Εάν ξεχάσετε τα διπλά εισαγωγικά, εμφανίζεται ένα σφάλμα:

The Print() Method

Υπάρχει επίσης μια μέθοδος print(), η οποία είναι παρόμοια με την println().

Η μόνη διαφορά είναι ότι δεν εισάγει μια νέα γραμμή στο τέλος της εξόδου:

```
System.out.print("Hello World! ");
System.out.print("I will print on the same line.");
```

Έξοδος: Hello World! I will print on the same line.

Java Output Numbers

Εκτύπωση αριθμών

Μπορείτε επίσης να χρησιμοποιήσετε τη μέθοδο **println()** για να εκτυπώσετε αριθμούς.

Ωστόσο, σε αντίθεση με το κείμενο, **δεν βάζουμε** αριθμούς μέσα σε διπλά εισαγωγικά:

```
System.out.println(3);
System.out.println(358);
```

```
System.out.println(50000);
```

Μπορείτε επίσης να εκτελέσετε μαθηματικούς υπολογισμούς μέσα στη μέθοδο println():

```
System.out.println(3 + 3);

System.out.println(2 * 5);
```

Java Comments

Τα σχόλια μπορούν να χρησιμοποιηθούν για να εξηγήσουν τον κώδικα Java και να τον κάνουν πιο ευανάγνωστο. Μπορεί επίσης να χρησιμοποιηθεί για να αποτρέψει την εκτέλεση κατά τη δοκιμή εναλλακτικού κώδικα.

Single-line Comments

Τα σχόλια μιας γραμμής ξεκινούν με δύο κάθετες προς τα εμπρός (//).

Οποιοδήποτε κείμενο μεταξύ // και του τέλους της γραμμής αγνοείται από την Java (δεν θα εκτελεστεί).

Αυτό το παράδειγμα χρησιμοποιεί ένα σχόλιο μίας γραμμής πριν από μια γραμμή κώδικα:

```
// This is a comment

System.out.println("Hello World");
```

Αυτό το παράδειγμα χρησιμοποιεί ένα σχόλιο μιας γραμμής στο τέλος μιας γραμμής κώδικα:

```
System.out.println("Hello World"); // This is a comment
```

Java Multi-line Comments

Τα σχόλια πολλών γραμμών ξεκινούν με /* και τελειώνουν με */.

Οποιοδήποτε κείμενο μεταξύ /* και */ θα αγνοηθεί από την Java.

Αυτό το παράδειγμα χρησιμοποιεί ένα σχόλιο πολλών γραμμών (ένα μπλοκ σχολίων) για να εξηγήσει τον κώδικα:

```
/* The code below will print the words Hello World

to the screen, and it is amazing */

System.out.println("Hello World");
```

Java Variables

Οι μεταβλητές είναι κοντέινερ για την αποθήκευση τιμών δεδομένων.

Στην **Java**, υπάρχουν διαφορετικοί τύποι μεταβλητών, για παράδειγμα:

string - αποθηκεύει κείμενο, όπως "Hello". Οι τιμές συμβολοσειράς περιβάλλονται από διπλά εισαγωγικά

int - αποθηκεύει ακέραιους αριθμούς (ακέραιους αριθμούς), χωρίς δεκαδικά, όπως 123 ή -123

float - αποθηκεύει αριθμούς κινητής υποδιαστολής, με δεκαδικούς, όπως 19,99 ή -19,99

char - αποθηκεύει μεμονωμένους χαρακτήρες, όπως 'a' ή 'B'. Οι τιμές χαρακτήρων περιβάλλονται από μεμονωμένα εισαγωγικά

boolean - αποθηκεύει τιμές με δύο καταστάσεις: true ή false

Declaring (Creating) Variables

Για να δημιουργήσετε μια μεταβλητή, πρέπει να καθορίσετε τον τύπο και να της εκχωρήσετε μια τιμή:

```
type variableName = value;
```

Όπου ο τύπος είναι ένας από τους τύπους της Java (όπως int ή String), και το variableName είναι το όνομα της μεταβλητής (όπως x ή name). Το σύμβολο ίσον χρησιμοποιείται για την εκχώρηση τιμών στη μεταβλητή.

Για να δημιουργήσετε μια μεταβλητή που θα πρέπει να αποθηκεύει κείμενο, δείτε το ακόλουθο παράδειγμα:

Δημιουργήστε μια μεταβλητή που ονομάζεται όνομα τύπου String και εκχωρήστε της την τιμή "John".

Στη συνέχεια χρησιμοποιούμε println() για να εκτυπώσουμε τη μεταβλητή ονόματος:

```
String name = "John";

System.out.println(name);
```

Για να δημιουργήσετε μια μεταβλητή που θα πρέπει να αποθηκεύει έναν αριθμό, δείτε το ακόλουθο παράδειγμα:

Δημιουργήστε μια μεταβλητή που ονομάζεται myNum τύπου int και εκχωρήστε της την τιμή 15:

```
int myNum = 15;
```

```
System.out.println(myNum);
```

Μπορείτε επίσης να δηλώσετε μια μεταβλητή χωρίς να εκχωρήσετε την τιμή και να εκχωρήσετε την τιμή αργότερα:

```
int myNum;

myNum = 15;

System.out.println(myNum);
```

Σημειώστε ότι εάν αντιστοιχίσετε μια νέα τιμή σε μια υπάρχουσα μεταβλητή, θα αντικαταστήσει την προηγούμενη τιμή:

Αλλάξτε την τιμή του myNum από 15 σε 20:

```
int myNum = 15;

myNum = 20;  // myNum is now 20

System.out.println(myNum);
```

Final Variables

Εάν δεν θέλετε οι άλλοι (ή εσείς) να αντικαταστήσουν τις υπάρχουσες τιμές, χρησιμοποιήστε την τελική λέξη-κλειδί (αυτό θα δηλώσει τη μεταβλητή ως "τελική" ή "σταθερή", που σημαίνει αμετάβλητη και μόνο για ανάγνωση):

```
final int myNum = 15;

myNum = 20;  // will generate an error: cannot assign a value to a final variable
```

Other Types

```
int myNum = 5;

float myFloatNum = 5.99f;

char myLetter = 'D';

boolean myBool = true;

String myText = "Hello";
```

Java Print Variables

Display Variables

Η μέθοδος println() χρησιμοποιείται συχνά για την εμφάνιση μεταβλητών.

Για να συνδυάσετε κείμενο και μεταβλητή, χρησιμοποιήστε τον χαρακτήρα +:

```
String name = "John";
System.out.println("Hello " + name);
```

You can also use the + character to add a variable to another variable:

```
String firstName = "John ";
String lastName = "Doe";
String fullName = firstName + lastName;
System.out.println(fullName);
```

Για αριθμητικές τιμές, ο χαρακτήρας + λειτουργεί ως μαθηματικός τελεστής (παρατηρήστε ότι χρησιμοποιούμε μεταβλητές int (ακέραιος) εδώ):

```
int x = 5;
int y = 6;
System.out.println(x + y); // Print the value of x + y
```

Java Declare Multiple Variables

Για να δηλώσετε περισσότερες από μία μεταβλητές του ίδιου τύπου, μπορείτε να χρησιμοποιήσετε μια λίστα διαχωρισμένη με κόμματα

```
int x = 5;
int y = 6;
```

```
int z = 50;

System.out.println(x + y + z);
```

```
int x = 5, y = 6, z = 50;

System.out.println(x + y + z);
```

One Value to Multiple Variables

Μπορείτε επίσης να αντιστοιχίσετε την ίδια τιμή σε πολλές μεταβλητές σε μία γραμμή:

```
int x, y, z;

x = y = z = 50;

System.out.println(x + y + z);
```

Java Identifiers

Όλες οι μεταβλητές Java πρέπει να προσδιορίζονται με μοναδικά ονόματα.

Αυτά τα μοναδικά ονόματα ονομάζονται αναγνωριστικά.

Τα αναγνωριστικά μπορεί να είναι σύντομα ονόματα (όπως x και y) ή περισσότερα περιγραφικά ονόματα (ηλικία, άθροισμα, συνολικός όγκος).

Σημείωση: Συνιστάται η χρήση περιγραφικών ονομάτων για τη δημιουργία κατανοητού και διατηρήσιμου κώδικα:

```
// Good

int minutesPerHour = 60;


// OK, but not so easy to understand what m actually is

int m = 60;
```

Οι γενικοί κανόνες για την ονομασία μεταβλητών είναι:

Τα ονόματα μπορεί να περιέχουν γράμματα, ψηφία, κάτω παύλες και σύμβολα δολαρίου

Τα ονόματα πρέπει να ξεκινούν με ένα γράμμα

Τα ονόματα πρέπει να ξεκινούν με πεζό γράμμα και δεν μπορούν να περιέχουν κενά

Τα ονόματα μπορούν επίσης να ξεκινούν με $ και _

Τα ονόματα κάνουν διάκριση πεζών-κεφαλαίων (το "myVar" και το "myvar" είναι διαφορετικές μεταβλητές)

Οι δεσμευμένες λέξεις (όπως λέξεις-κλειδιά Java, όπως int ή boolean) δεν μπορούν να χρησιμοποιηθούν ως ονόματα

Java Variables - Examples

Συχνά στα παραδείγματά μας, απλοποιούμε τα ονόματα μεταβλητών για να ταιριάζουν με τον τύπο δεδομένων τους (myInt ή myNum για τύπους int, myChar για τύπους χαρακτήρων κ.λπ.). Αυτό γίνεται για να αποφευχθεί η σύγχυση.

Ωστόσο, για ένα πρακτικό παράδειγμα χρήσης μεταβλητών, δημιουργήσαμε ένα πρόγραμμα που αποθηκεύει διαφορετικά δεδομένα σχετικά με έναν φοιτητή:

```
// Student data
String studentName = "John Doe";
int studentID = 15;
int studentAge = 23;
float studentFee = 75.25f;
char studentGrade = 'B';


// Print variables
System.out.println("Student name: " + studentName);
System.out.println("Student id: " + studentID);
System.out.println("Student age: " + studentAge);
System.out.println("Student fee: " + studentFee);
System.out.println("Student grade: " + studentGrade);
```

Calculate the Area of a Rectangle

```
// Create integer variables
int length = 4;
int width = 6;
int area;
```

```
// Calculate the area of a rectangle
area = length * width;

// Print variables
System.out.println("Length is: " + length);
System.out.println("Width is: " + width);
System.out.println("Area of the rectangle is: " + area);
```

Java Data Types

Όπως εξηγήθηκε στο προηγούμενο κεφάλαιο, μια μεταβλητή στη Java πρέπει να είναι ένας καθορισμένος τύπος δεδομένων:

```
int myNum = 5;               // Integer (whole number)
float myFloatNum = 5.99f;    // Floating point number
char myLetter = 'D';         // Character
boolean myBool = true;       // Boolean
String myText = "Hello";     // String
```

Οι τύποι δεδομένων χωρίζονται σε δύο ομάδες:

Πρωτόγονοι τύποι δεδομένων - περιλαμβάνει byte, short, int, long, float, double, boolean και char

Μη πρωτόγονοι τύποι δεδομένων - όπως String, Arrays και Classes (θα μάθετε περισσότερα για αυτούς σε επόμενο κεφάλαιο)

Primitive Data Types

Ένας πρωτόγονος τύπος δεδομένων καθορίζει το μέγεθος και τον τύπο των τιμών των μεταβλητών και δεν έχει πρόσθετες μεθόδους.

Υπάρχουν οκτώ πρωτόγονοι τύποι δεδομένων στην Java:

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127

short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Java Numbers

Οι αρχικοί τύποι αριθμών χωρίζονται σε δύο ομάδες:

Οι ακέραιοι τύποι αποθηκεύουν ακέραιους αριθμούς, θετικούς ή αρνητικούς (όπως 123 ή -456), χωρίς δεκαδικούς. Οι έγκυροι τύποι είναι byte, short, int και long. Ποιος τύπος πρέπει να χρησιμοποιήσετε εξαρτάται από την αριθμητική τιμή.

Οι τύποι κινητής υποδιαστολής αντιπροσωπεύουν αριθμούς με κλασματικό μέρος, που περιέχει ένα ή περισσότερα δεκαδικά. Υπάρχουν δύο τύποι: float και διπλό.

Παρόλο που υπάρχουν πολλοί αριθμητικοί τύποι στην Java, οι πιο χρησιμοποιούμενοι για αριθμούς είναι ο int (για ακέραιους αριθμούς) και ο διπλός (για αριθμούς κινητής υποδιαστολής). Ωστόσο, θα τα περιγράψουμε όλα καθώς συνεχίζετε να διαβάζετε.

Integer Types

Byte

Ο τύπος δεδομένων byte μπορεί να αποθηκεύσει ακέραιους αριθμούς από -128 έως 127. Αυτό μπορεί να χρησιμοποιηθεί αντί για int ή άλλους ακέραιους τύπους για εξοικονόμηση μνήμης όταν είστε βέβαιοι ότι η τιμή θα είναι εντός -128 και 127:

```
byte myNum = 100;

System.out.println(myNum);
```

Short

Ο σύντομος τύπος δεδομένων μπορεί να αποθηκεύσει ακέραιους αριθμούς από -32768 έως 32767:

```
short myNum = 5000;

System.out.println(myNum);
```

Int

Ο τύπος δεδομένων int μπορεί να αποθηκεύσει ακέραιους αριθμούς από -2147483648 έως 2147483647. Γενικά, και στο σεμινάριο μας, ο τύπος δεδομένων int είναι ο προτιμώμενος τύπος δεδομένων όταν δημιουργούμε μεταβλητές με αριθμητική τιμή.

```
int myNum = 100000;

System.out.println(myNum);
```

Long

Ο τύπος δεδομένων long μπορεί να αποθηκεύσει ακέραιους αριθμούς από -9223372036854775808 έως 9223372036854775807. Αυτό χρησιμοποιείται όταν το int δεν είναι αρκετά μεγάλο για την αποθήκευση της τιμής. Σημειώστε ότι θα πρέπει να τερματίσετε την τιμή με ένα "L":

```
long myNum = 15000000000L;

System.out.println(myNum);
```

Floating Point Types

Θα πρέπει να χρησιμοποιείτε έναν τύπο κινητής υποδιαστολής όποτε χρειάζεστε έναν αριθμό με δεκαδικό, όπως 9,99 ή 3,14515.

Οι τύποι δεδομένων float και double μπορούν να αποθηκεύσουν κλασματικούς αριθμούς. Σημειώστε ότι θα πρέπει να τερματίσετε την τιμή με ένα "f" για floats και "d" για double:

```
float myNum = 5.75f;

System.out.println(myNum);
```

```
double myNum = 19.99d;
```

```
System.out.println(myNum);
```

Χρησιμοποιήστε float ή double;

Η ακρίβεια μιας τιμής κινητής υποδιαστολής υποδεικνύει πόσα ψηφία μπορεί να έχει η τιμή μετά την υποδιαστολή. Η ακρίβεια του float είναι μόνο έξι ή επτά δεκαδικά ψηφία, ενώ οι διπλές μεταβλητές έχουν ακρίβεια περίπου 15 ψηφίων. Επομένως, είναι ασφαλέστερο να χρησιμοποιείτε το διπλό για τους περισσότερους υπολογισμούς.

Scientific Numbers

Ένας αριθμός κινητής υποδιαστολής μπορεί επίσης να είναι ένας επιστημονικός αριθμός με "e" για να υποδηλώνει την ισχύ του 10:

```
float f1 = 35e3f;

double d1 = 12E4d;

System.out.println(f1);

System.out.println(d1);
```

Java Boolean Data Types

Πολύ συχνά στον προγραμματισμό, θα χρειαστείτε έναν τύπο δεδομένων που μπορεί να έχει μόνο μία από τις δύο τιμές, όπως:

ΝΑΙ / ΟΧΙ

ON / OFF

ΑΛΗΘΕΙΑ / ΛΑΘΟΣ

Για αυτό, η Java έχει έναν τύπο δεδομένων boolean, ο οποίος μπορεί να λάβει μόνο τις τιμές true ή false:

Example

```
boolean isJavaFun = true;

boolean isFishTasty = false;

System.out.println(isJavaFun);     // Outputs true

System.out.println(isFishTasty);   // Outputs false
```

Java Characters

Ο τύπος δεδομένων char χρησιμοποιείται για την αποθήκευση ενός μεμονωμένου χαρακτήρα. Ο χαρακτήρας πρέπει να περιβάλλεται από μεμονωμένα εισαγωγικά, όπως "A" ή "c":

Example Get your own Java Server

```
char myGrade = 'B';

System.out.println(myGrade);
```

Εναλλακτικά, εάν είστε εξοικειωμένοι με τις τιμές ASCII, μπορείτε να τις χρησιμοποιήσετε για να εμφανίσετε ορισμένους χαρακτήρες:

Example

```
char myVar1 = 65, myVar2 = 66, myVar3 = 67;

System.out.println(myVar1);

System.out.println(myVar2);

System.out.println(myVar3);
```

Strings

Ο τύπος δεδομένων συμβολοσειράς χρησιμοποιείται για την αποθήκευση μιας ακολουθίας χαρακτήρων (κείμενο). Οι τιμές συμβολοσειράς πρέπει να περιβάλλονται από διπλά εισαγωγικά:

Example

```
String greeting = "Hello World";

System.out.println(greeting);
```

Παράδειγμα πραγματικής ζωής

Ακολουθεί ένα πραγματικό παράδειγμα χρήσης διαφορετικών τύπων δεδομένων, για τον υπολογισμό και την παραγωγή του συνολικού κόστους ορισμένων στοιχείων:

Example

```
// Create variables of different data types

int items = 50;

float costPerItem = 9.99f;
```

```
float totalCost = items * costPerItem;
char currency = '$';


// Print variables
System.out.println("Number of items: " + items);
System.out.println("Cost per item: " + costPerItem + currency);
System.out.println("Total cost = " + totalCost + currency);
```

Java Non-Primitive Data Types

Οι μη πρωτόγονοι τύποι δεδομένων ονομάζονται τύποι αναφοράς επειδή αναφέρονται σε αντικείμενα.

Η κύρια διαφορά μεταξύ πρωτόγονων και μη τύπων δεδομένων είναι:

Οι πρωτόγονοι τύποι είναι προκαθορισμένοι (ήδη καθορισμένοι) στην Java. Οι μη πρωτόγονοι τύποι δημιουργούνται από τον προγραμματιστή και δεν ορίζονται από την Java (εκτός από το String).

Οι μη πρωτόγονοι τύποι μπορούν να χρησιμοποιηθούν για την κλήση μεθόδων για την εκτέλεση ορισμένων πράξεων, ενώ οι πρωτόγονοι τύποι δεν μπορούν.

Ένας πρωτόγονος τύπος έχει πάντα μια τιμή, ενώ οι μη πρωτόγονοι τύποι μπορεί να είναι μηδενικοί.

Ένας πρωτόγονος τύπος ξεκινά με πεζό γράμμα, ενώ οι μη πρωτόγονοι τύποι ξεκινούν με κεφαλαίο.

Java Type Casting

Η μετάδοση τύπων είναι όταν εκχωρείτε μια τιμή ενός πρωτόγονου τύπου δεδομένων σε έναν άλλο τύπο.

Στην Java, υπάρχουν δύο τύποι χύτευσης:

Διεύρυνση χύτευσης (αυτόματα) - μετατροπή ενός μικρότερου τύπου σε μεγαλύτερο μέγεθος τύπου

byte -> short -> char -> int -> long -> float -> double

Στένωση χύτευσης (χειροκίνητα) - μετατροπή ενός μεγαλύτερου τύπου σε έναν τύπο μικρότερου μεγέθους

διπλό -> float -> long -> int -> char -> short -> byte

Widening Casting

Η χύτευση διεύρυνσης γίνεται αυτόματα όταν περνάτε έναν τύπο μικρότερου μεγέθους σε έναν τύπο μεγαλύτερου μεγέθους:

Example

```
public class Main {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double

    System.out.println(myInt);      // Outputs 9
    System.out.println(myDouble);   // Outputs 9.0
  }
}
```

Narrowing Casting

Η στένωση χύτευσης πρέπει να γίνει χειροκίνητα τοποθετώντας τον τύπο σε παρένθεση () μπροστά από την τιμή:

Example

```
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78d;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

Παράδειγμα πραγματικής ζωής

Ακολουθεί ένα πραγματικό παράδειγμα casting τύπου όπου δημιουργούμε ένα πρόγραμμα για τον υπολογισμό του ποσοστού της βαθμολογίας ενός χρήστη σε σχέση με τη μέγιστη βαθμολογία σε ένα παιχνίδι.

Χρησιμοποιούμε τη χύτευση τύπου για να βεβαιωθούμε ότι το αποτέλεσμα είναι μια τιμή κινητής υποδιαστολής και όχι ένας ακέραιος:

Example

```
// Set the maximum possible score in the game to 500
int maxScore = 500;


// The actual score of the user
int userScore = 423;


/* Calculate the percantage of the user's score in relation to the maximum
available score.
Convert userScore to float to make sure that the division is accurate */
float percentage = (float) userScore / maxScore * 100.0f;


System.out.println("User's percentage is " + percentage);
```

Java Operators

Operators are used to perform operations on variables and values.

In the example below, we use the `+` **operator** to add together two values:

Example

```
int x = 100 + 50;
```

Αν και ο τελεστής + χρησιμοποιείται συχνά για να προσθέσει δύο τιμές, όπως στο παραπάνω παράδειγμα, μπορεί επίσης να χρησιμοποιηθεί για να προσθέσει μια μεταβλητή και μια τιμή ή μια μεταβλητή και μια άλλη μεταβλητή:

Example

```
int sum1 = 100 + 50;         // 150 (100 + 50)
int sum2 = sum1 + 250;       // 400 (150 + 250)
int sum3 = sum2 + sum2;      // 800 (400 + 400)
```

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x – y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Java Assignment Operators

Αν και ο τελεστής + χρησιμοποιείται συχνά για να προσθέσει δύο τιμές, όπως στο παραπάνω παράδειγμα, μπορεί επίσης να χρησιμοποιηθεί για να προσθέσει μια μεταβλητή και μια τιμή ή μια μεταβλητή και μια άλλη μεταβλητή:

Example

```
int x = 10;
```

Ο τελεστής εκχώρησης πρόσθεσης (+=) προσθέτει μια τιμή σε μια μεταβλητή:

Example

```
int x = 10;

x += 5
```

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3

^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Java Comparison Operators

Οι τελεστές σύγκρισης χρησιμοποιούνται για τη σύγκριση δύο τιμών (ή μεταβλητών). Αυτό είναι σημαντικό στον προγραμματισμό, γιατί μας βοηθά να βρίσκουμε απαντήσεις και να παίρνουμε αποφάσεις.

Η επιστρεφόμενη τιμή μιας σύγκρισης είναι είτε true είτε false. Αυτές οι τιμές είναι γνωστές ως τιμές Boolean

Στο παρακάτω παράδειγμα, χρησιμοποιούμε τον τελεστή μεγαλύτερο από (>) για να μάθουμε αν το 5 είναι μεγαλύτερο από το 3:

```
int x = 5;
int y = 3;
System.out.println(x > y); // returns true, because 5 is higher than 3
```

Operator	Name	Example	Try it
==	Equal to	x == y	Try it »
!=	Not equal	x != y	Try it »
>	Greater than	x > y	Try it »

<	Less than	x < y	Try it »
>=	Greater than or equal to	x >= y	Try it »
<=	Less than or equal to	x <= y	

Java Logical Operators

Μπορείτε επίσης να ελέγξετε για αληθείς ή ψευδείς τιμές με λογικούς τελεστές.

Οι λογικοί τελεστές χρησιμοποιούνται για τον προσδιορισμό της λογικής μεταξύ μεταβλητών ή τιμών:

Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Java Strings

Οι συμβολοσειρές χρησιμοποιούνται για την αποθήκευση κειμένου.

Μια μεταβλητή συμβολοσειράς περιέχει μια συλλογή χαρακτήρων που περιβάλλονται από διπλά εισαγωγικά:

```
String greeting = "Hello";
```

String Length

Μια συμβολοσειρά στην Java είναι στην πραγματικότητα ένα αντικείμενο, το οποίο περιέχει μεθόδους που μπορούν να εκτελέσουν ορισμένες λειτουργίες σε συμβολοσειρές. Για παράδειγμα, το μήκος μιας συμβολοσειράς μπορεί να βρεθεί με τη μέθοδο length():

Example

```
String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

System.out.println("The length of the txt string is: " + txt.length());
```

More String Methods

Υπάρχουν πολλές διαθέσιμες μέθοδοι συμβολοσειράς, για παράδειγμα toUpperCase() και toLowerCase():

Example

```
String txt = "Hello World";

System.out.println(txt.toUpperCase());   // Outputs "HELLO WORLD"

System.out.println(txt.toLowerCase());   // Outputs "hello world"
```

Finding a Character in a String

Η μέθοδος indexOf() επιστρέφει το ευρετήριο (τη θέση) της πρώτης εμφάνισης ενός καθορισμένου κειμένου σε μια συμβολοσειρά (συμπεριλαμβανομένου του κενού χώρου):

Example

```
String txt = "Please locate where 'locate' occurs!";

System.out.println(txt.indexOf("locate")); // Outputs 7
```

Java String Concatenation

Ο τελεστής + μπορεί να χρησιμοποιηθεί μεταξύ των συμβολοσειρών για να τις συνδυάσει. Αυτό ονομάζεται συνένωση:

Example Get your own Java Server

```
String firstName = "John";
String lastName = "Doe";
System.out.println(firstName + " " + lastName);
```

Μπορείτε επίσης να χρησιμοποιήσετε τη μέθοδο concat() για να συνδέσετε δύο συμβολοσειρές:

Example

```
String firstName = "John ";
String lastName = "Doe";
System.out.println(firstName.concat(lastName));
```

Java Numbers and Strings

ΠΡΟΕΙΔΟΠΟΙΗΣΗ!

Η Java χρησιμοποιεί τον τελεστή + τόσο για πρόσθεση όσο και για συνένωση.

Προστίθενται αριθμοί. Οι χορδές συνδέονται.

Example Get your own Java Server

```
int x = 10;
int y = 20;
int z = x + y;  // z will be 30 (an integer/number)
```

Εάν προσθέσετε δύο χορδές, το αποτέλεσμα θα είναι μια συνένωση συμβολοσειρών:

Example

```
String x = "10";
```

```
String y = "20";

String z = x + y;  // z will be 1020 (a String)
```

Εάν προσθέσετε έναν αριθμό και μια συμβολοσειρά, το αποτέλεσμα θα είναι μια συνένωση συμβολοσειρών:

Example

```
String x = "10";

int y = 20;

String z = x + y;  // z will be 1020 (a String)
```

Java Special Characters

Επειδή οι συμβολοσειρές πρέπει να γράφονται εντός εισαγωγικών, η Java θα παρεξηγήσει αυτήν τη συμβολοσειρά και θα δημιουργήσει ένα σφάλμα:

```
String txt = "We are the so-called "Vikings" from the north.";
```

Η λύση για να αποφύγετε αυτό το πρόβλημα, είναι να χρησιμοποιήσετε τον χαρακτήρα διαφυγής ανάστροφης κάθετου.

Ο χαρακτήρας διαφυγής ανάστροφης κάθετου (\) μετατρέπει τους ειδικούς χαρακτήρες σε χαρακτήρες συμβολοσειράς:

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

Η ακολουθία \" εισάγει ένα διπλό εισαγωγικό σε μια συμβολοσειρά:

Example

```
String txt = "We are the so-called \"Vikings\" from the north.";
```

Η ακολουθία \' εισάγει ένα μόνο εισαγωγικό σε μια συμβολοσειρά:

Example

```
String txt = "It\'s alright.";
```

Η ακολουθία \\ εισάγει μια μόνο ανάστροφη κάθετο σε μια συμβολοσειρά:

Example

```
String txt = "The character \\ is called backslash.";
```

Άλλες κοινές ακολουθίες διαφυγής που ισχύουν στην Java είναι:

Code	Result
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace
\f	Form Feed

Java Math

Η τάξη Java Math έχει πολλές μεθόδους που σας επιτρέπουν να εκτελείτε μαθηματικές εργασίες σε αριθμούς.

Math.max(*x,y*)

Η μέθοδος μπορεί να χρησιμοποιηθεί για να βρεθεί η υψηλότερη τιμή των *x* και *y* :`Math.max(x,y)`

Παράδειγμα

```
Math.max(5, 10);
```

Math.min(*x,y*)

Η μέθοδος μπορεί να χρησιμοποιηθεί για την εύρεση της χαμηλότερης τιμής των *x* και *y* :`Math.min(x,y)`

Παράδειγμα

```
Math.min(5, 10);
```

Math.sqrt(*x*)

Η μέθοδος επιστρέφει την τετραγωνική ρίζα του *x* :`Math.sqrt(x)`

Παράδειγμα

```
Math.sqrt(64);
```

Math.abs(*x*)

Η μέθοδος επιστρέφει την απόλυτη (θετική) τιμή του *x* :`Math.abs(x)`

Παράδειγμα

```
Math.abs(-4.7);
```

Τυχαίοι Αριθμοί

`Math.random()`επιστρέφει έναν τυχαίο αριθμό μεταξύ 0,0 (συμπεριλαμβανομένου) και 1,0 (αποκλειστικά):

Παράδειγμα

```
Math.random();
```

Για να αποκτήσετε περισσότερο έλεγχο στον τυχαίο αριθμό, για παράδειγμα, εάν θέλετε μόνο έναν τυχαίο αριθμό μεταξύ 0 και 100, μπορείτε να χρησιμοποιήσετε τον ακόλουθο τύπο:

Παράδειγμα

```
int randomNum = (int)(Math.random() * 101);  // 0 to 100
```

Java Booleans

Java Booleans

Πολύ συχνά, στον προγραμματισμό, θα χρειαστείτε έναν τύπο δεδομένων που μπορεί να έχει μόνο μία από τις δύο τιμές, όπως:

- ΝΑΙ / ΟΧΙ
- ΟΝ / OFF
- ΑΛΗΘΕΙΑ / ΛΑΘΟΣ

Για αυτό, η Java έχει έναν `boolean`τύπο δεδομένων, ο οποίος μπορεί να αποθηκεύσει `true`ή `false`τιμές.

Boolean Values

Ένας τύπος boolean δηλώνεται με τη `boolean`λέξη-κλειδί και μπορεί να λάβει μόνο τις τιμές `true`ή `false`:

Παράδειγμα

```
boolean isJavaFun = true;
```

```
boolean isFishTasty = false;

System.out.println(isJavaFun);      // Outputs true

System.out.println(isFishTasty);    // Outputs false
```

Ωστόσο, είναι πιο συνηθισμένο να επιστρέφονται τιμές boolean από εκφράσεις boolean, για έλεγχο υπό όρους (δείτε παρακάτω).

Boolean Έκφραση

Μια Boolean έκφραση επιστρέφει μια Boolean τιμή: `true`ή `false`.

Αυτό είναι χρήσιμο για την οικοδόμηση λογικής και την εύρεση απαντήσεων.

Για παράδειγμα, μπορείτε να χρησιμοποιήσετε έναν τελεστή σύγκρισης , όπως τον τελεστή **μεγαλύτερο από** (`>`), για να μάθετε εάν μια έκφραση (ή μια μεταβλητή) είναι αληθής ή ψευδής:

Παράδειγμα

```
int x = 10;

int y = 9;

System.out.println(x > y); // returns true, because 10 is higher than 9
```

Ή ακόμα πιο εύκολο:

Παράδειγμα

```
System.out.println(10 > 9); // returns true, because 10 is higher than 9
```

Στα παρακάτω παραδείγματα, χρησιμοποιούμε τον τελεστή **ίσο με** (`==`) για να αξιολογήσουμε μια παράσταση:

Παράδειγμα

```
int x = 10;

System.out.println(x == 10); // returns true, because the value of x is
equal to 10
```

Παράδειγμα

```
System.out.println(10 == 15); // returns false, because 10 is not equal to 15
```

Παράδειγμα πραγματικής ζωής

Ας σκεφτούμε ένα «παράδειγμα πραγματικής ζωής» όπου πρέπει να μάθουμε εάν ένα άτομο είναι αρκετά μεγάλο για να ψηφίσει.

Στο παρακάτω παράδειγμα, χρησιμοποιούμε τον `>=`τελεστή σύγκρισης για να μάθουμε εάν η ηλικία (`25`) είναι **μεγαλύτερη από** Ή **ίση με** το όριο ηλικίας ψήφου, το οποίο έχει οριστεί σε `18`:

Παράδειγμα

```
int myAge = 25;

int votingAge = 18;

System.out.println(myAge >= votingAge);
```

Μια ακόμη καλύτερη προσέγγιση (καθώς είμαστε σε ρολό τώρα), θα ήταν να τυλίξουμε τον παραπάνω κώδικα σε μια `if...else`δήλωση, ώστε να μπορούμε να εκτελέσουμε διαφορετικές ενέργειες ανάλογα με το αποτέλεσμα:

Παράδειγμα

Έξοδος "Αρκετά παλιό για να ψηφίσεις!" αν `myAge`είναι **μεγαλύτερο ή ίσο με** `18` . Διαφορετικά, βγείτε "Δεν είναι αρκετά μεγάλο για να ψηφίσω.":

```
int myAge = 25;

int votingAge = 18;


if (myAge >= votingAge) {

  System.out.println("Old enough to vote!");

} else {

  System.out.println("Not old enough to vote.");
```

```
}
```

Java If ... Else

Προϋποθέσεις Java και δηλώσεις Εάν

Γνωρίζετε ήδη ότι η Java υποστηρίζει τις συνήθεις λογικές συνθήκες από τα μαθηματικά:

- Λιγότερο από: `a < b`
- Μικρότερο ή ίσο με: `a <= b`
- Μεγαλύτερο από: `a > b`
- Μεγαλύτερο ή ίσο με: `a >= b`
- Ίσο με `a == b`
- Δεν ισούται με: `a != b`

Μπορείτε να χρησιμοποιήσετε αυτές τις συνθήκες για να εκτελέσετε διαφορετικές ενέργειες για διαφορετικές αποφάσεις.

Η Java έχει τις ακόλουθες δηλώσεις υπό όρους:

- Χρησιμοποιήστε το `if`για να καθορίσετε ένα μπλοκ κώδικα που θα εκτελεστεί, εάν μια καθορισμένη συνθήκη είναι αληθής
- Χρησιμοποιήστε το `else`για να καθορίσετε ένα μπλοκ κώδικα που θα εκτελεστεί, εάν η ίδια συνθήκη είναι ψευδής
- Χρησιμοποιήστε το `else if`για να καθορίσετε μια νέα συνθήκη για δοκιμή, εάν η πρώτη συνθήκη είναι ψευδής
- Χρησιμοποιήστε το `switch`για να καθορίσετε πολλά εναλλακτικά μπλοκ κώδικα που θα εκτελεστούν

Η δήλωση αν

Χρησιμοποιήστε τη `if`δήλωση για να καθορίσετε ένα μπλοκ κώδικα Java που θα εκτελεστεί εάν μια συνθήκη είναι `true`.

Σύνταξη

```
if (condition) {

  // block of code to be executed if the condition is true

}
```

Σημειώστε ότι `if`είναι με πεζά γράμματα. Τα κεφαλαία γράμματα (If ή IF) θα δημιουργήσουν ένα σφάλμα.

Στο παρακάτω παράδειγμα, δοκιμάζουμε δύο τιμές για να διαπιστώσουμε εάν το 20 είναι μεγαλύτερο από 18. Εάν η συνθήκη είναι `true`, εκτυπώστε λίγο κείμενο:

Παράδειγμα

```
if (20 > 18) {

  System.out.println("20 is greater than 18");

}
```

Μπορούμε επίσης να δοκιμάσουμε μεταβλητές:

Παράδειγμα

```
int x = 20;

int y = 18;

if (x > y) {

  System.out.println("x is greater than y");

}
```

Το παράδειγμα εξηγείται

Στο παραπάνω παράδειγμα χρησιμοποιούμε δύο μεταβλητές, **x** και **y** , για να ελέγξουμε αν το x είναι μεγαλύτερο από το y (χρησιμοποιώντας τον `>`τελεστή). Καθώς το x είναι 20 και το y είναι 18 και γνωρίζουμε ότι το 20 είναι μεγαλύτερο από το 18, εκτυπώνουμε στην οθόνη ότι "το x είναι μεγαλύτερο από το y".

Java Else

Χρησιμοποιήστε τη `else`δήλωση για να καθορίσετε ένα μπλοκ κώδικα που θα εκτελεστεί εάν η συνθήκη είναι `false`.

Σύνταξη

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

Παράδειγμα

```
int time = 20;
if (time < 18) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
// Outputs "Good evening."
```

Το παράδειγμα εξηγείται

Στο παραπάνω παράδειγμα, ο χρόνος (20) είναι μεγαλύτερος από 18, επομένως η συνθήκη είναι `false`. Εξαιτίας αυτού, προχωράμε στην `else`κατάσταση και εκτυπώνουμε στην οθόνη "Καλησπέρα". Αν η ώρα ήταν μικρότερη από 18, το πρόγραμμα θα τύπωνε «Καλημέρα».

Java Else If

Χρησιμοποιήστε τη `else if`δήλωση για να καθορίσετε μια νέα συνθήκη εάν η πρώτη συνθήκη είναι `false`.

Σύνταξη

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2
is true
```

```
} else {

  // block of code to be executed if the condition1 is false and condition2
is false

}
```

Παράδειγμα

```
int time = 22;

if (time < 10) {

  System.out.println("Good morning.");

} else if (time < 18) {

  System.out.println("Good day.");

} else {

  System.out.println("Good evening.");

}

// Outputs "Good evening."
```

Το παράδειγμα εξηγείται

Στο παραπάνω παράδειγμα, ο χρόνος (22) είναι μεγαλύτερος από 10, επομένως η **πρώτη συνθήκη** είναι `false`. Η επόμενη συνθήκη, στη `else if`δήλωση, είναι επίσης `false`, οπότε προχωράμε στην `else` συνθήκη αφού **η συνθήκη 1** και **η συνθήκη 2** είναι και τα δύο `false`- και εκτυπώνουμε στην οθόνη "Καλησπέρα".

Ωστόσο, αν η ώρα ήταν 14, το πρόγραμμά μας θα τύπωνε «Καλημέρα».

Java Short Hand If...Else

Υπάρχει επίσης ένας βραχυπρόθεσμος if else , ο οποίος είναι γνωστός ως **τριαδικός τελεστής** επειδή αποτελείται από τρεις τελεστές.

Μπορεί να χρησιμοποιηθεί για την αντικατάσταση πολλαπλών γραμμών κώδικα με μία μόνο γραμμή, και πιο συχνά χρησιμοποιείται για την αντικατάσταση απλών εντολών if else:

Σύνταξη

```
variable = (condition) ? expressionTrue :  expressionFalse;
```

Αντί να γράψεις:

Παράδειγμα

```
int time = 20;
if (time < 18) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
```

Μπορείτε απλά να γράψετε:

Παράδειγμα

```
int time = 20;
String result = (time < 18) ? "Good day." : "Good evening.";
System.out.println(result);
```

Παράδειγμα

Αυτό το παράδειγμα δείχνει πώς μπορείτε να χρησιμοποιήσετε **το if..else** για να "ανοίξει μια πόρτα" εάν ο χρήστης εισάγει τον σωστό κωδικό:

```
int doorCode = 1337;


if (doorCode == 1337) {
  System.out.println("Correct code. The door is now open.");
} else {
  System.out.println("Wrong code. The door remains closed.");
}
```

Αυτό το παράδειγμα δείχνει πώς μπορείτε να χρησιμοποιήσετε **το if..else** για να μάθετε εάν ένας αριθμός είναι θετικός ή αρνητικός:

Παράδειγμα

```
int myNum = 10; // Is this a positive or negative number?


if (myNum > 0) {
  System.out.println("The value is a positive number.");
} else if (myNum < 0) {
  System.out.println("The value is a negative number.");
} else {
  System.out.println("The value is 0.");
}
```

Μάθετε εάν ένα άτομο είναι αρκετά μεγάλο για να ψηφίσει:

Παράδειγμα

```
int myAge = 25;
int votingAge = 18;


if (myAge >= votingAge) {
  System.out.println("Old enough to vote!");
} else {
  System.out.println("Not old enough to vote.");
}
```

Μάθετε αν ένας αριθμός είναι άρτιος ή περιττός:

Παράδειγμα

```
int myNum = 5;


if (myNum % 2 == 0) {
  System.out.println(myNum + " is even");
} else {
  System.out.println(myNum + " is odd");
```

```
}
```

Java Switch

Δηλώσεις Java Switch

Αντί να γράφετε **πολλές** `if..else` δηλώσεις, μπορείτε να χρησιμοποιήσετε τη `switch`δήλωση.

Η `switch`δήλωση επιλέγει ένα από τα πολλά μπλοκ κώδικα που θα εκτελεστούν:

Σύνταξη

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

Έτσι λειτουργεί:

- Η `switch`έκφραση αξιολογείται μία φορά.
- Η τιμή της παράστασης συγκρίνεται με τις τιμές του καθενός `case`.
- Εάν υπάρχει αντιστοιχία, εκτελείται το συσχετισμένο μπλοκ κώδικα.
- Οι λέξεις-κλειδιά `break`και `default`είναι προαιρετικές και θα περιγραφούν αργότερα σε αυτό το κεφάλαιο

Το παρακάτω παράδειγμα χρησιμοποιεί τον αριθμό της ημέρας για τον υπολογισμό του ονόματος της ημέρας:

Παράδειγμα

```
int day = 4;
```

```java
switch (day) {
  case 1:
    System.out.println("Monday");
    break;
  case 2:
    System.out.println("Tuesday");
    break;
  case 3:
    System.out.println("Wednesday");
    break;
  case 4:
    System.out.println("Thursday");
    break;
  case 5:
    System.out.println("Friday");
    break;
  case 6:
    System.out.println("Saturday");
    break;
  case 7:
    System.out.println("Sunday");
    break;
}
// Outputs "Thursday" (day 4)
```

Λέξη-κλειδί διάλειμμα

Όταν η Java φτάσει σε μια break λέξη-κλειδί, ξεφεύγει από το μπλοκ διακόπτη.

Αυτό θα σταματήσει την εκτέλεση περισσότερων δοκιμών κώδικα και περιπτώσεων εντός του μπλοκ.

Όταν βρεθεί ένα ταίρι και τελειώσει η δουλειά, είναι ώρα για διάλειμμα. Δεν χρειάζονται περισσότερες δοκιμές.

Ένα διάλειμμα μπορεί να εξοικονομήσει πολύ χρόνο εκτέλεσης επειδή "αγνοεί" την εκτέλεση όλου του υπόλοιπου κώδικα στο μπλοκ διακόπτη.

Η προεπιλεγμένη λέξη-κλειδί

Η `default`λέξη-κλειδί καθορίζει κάποιον κώδικα που θα εκτελεστεί εάν δεν υπάρχει αντιστοίχιση πεζών-κεφαλαίων:

Παράδειγμα

```
int day = 4;

switch (day) {

  case 6:

    System.out.println("Today is Saturday");

    break;

  case 7:

    System.out.println("Today is Sunday");

    break;

  default:

    System.out.println("Looking forward to the Weekend");

}

// Outputs "Looking forward to the Weekend"
```

Σημειώστε ότι εάν η `default`πρόταση χρησιμοποιείται ως η τελευταία πρόταση σε ένα μπλοκ διακόπτη, δεν χρειάζεται διάλειμμα.

Java while Loop

Βρόχοι

Οι βρόχοι μπορούν να εκτελέσουν ένα μπλοκ κώδικα αρκεί να επιτευχθεί μια καθορισμένη συνθήκη.

Οι βρόχοι είναι εύχρηστοι γιατί εξοικονομούν χρόνο, μειώνουν τα σφάλματα και κάνουν τον κώδικα πιο ευανάγνωστο.

Java while Loop

Ο `while`βρόχος περιστρέφεται μέσω ενός μπλοκ κώδικα, εφόσον μια καθορισμένη συνθήκη είναι `true`:

Σύνταξη

```
while (condition) {
  // code block to be executed
}
```

Στο παρακάτω παράδειγμα, ο κώδικας στον βρόχο θα εκτελείται ξανά και ξανά, εφόσον μια μεταβλητή (i) είναι μικρότερη από 5:

Παράδειγμα

```
int i = 0;
while (i < 5) {
  System.out.println(i);
  i++;
}
```

Σημείωση: Μην ξεχάσετε να αυξήσετε τη μεταβλητή που χρησιμοποιείται στη συνθήκη, διαφορετικά ο βρόχος δεν θα τελειώσει ποτέ!

Java Do/While Loop

Ο βρόχος Do/While

Ο do/whileβρόχος είναι μια παραλλαγή του whileβρόχου. Αυτός ο βρόχος θα εκτελέσει το μπλοκ κώδικα μία φορά, πριν ελέγξει αν η συνθήκη είναι αληθής, και στη συνέχεια θα επαναλάβει τον βρόχο εφόσον η συνθήκη είναι αληθής.

Σύνταξη

```
do {

  // code block to be executed

}

while (condition);
```

Το παρακάτω παράδειγμα χρησιμοποιεί έναν do/whileβρόχο. Ο βρόχος θα εκτελείται πάντα τουλάχιστον μία φορά, ακόμα κι αν η συνθήκη είναι ψευδής, επειδή το μπλοκ κώδικα εκτελείται πριν από τη δοκιμή της συνθήκης:

Παράδειγμα

```
int i = 0;
do {

  System.out.println(i);

  i++;

}

while (i < 5);
```

Μην ξεχάσετε να αυξήσετε τη μεταβλητή που χρησιμοποιείται στη συνθήκη, διαφορετικά ο βρόχος δεν θα τελειώσει ποτέ!

Παραδείγματα Πραγματικής Ζωής

Για να δείξουμε ένα πρακτικό παράδειγμα του **βρόχου while** , δημιουργήσαμε ένα απλό πρόγραμμα "αντίστροφης μέτρησης":

Παράδειγμα

```
int countdown = 3;



while (countdown > 0) {
```

```
  System.out.println(countdown);

  countdown--;

}


System.out.println("Happy New Year!!");
```

Για να δείξουμε ένα πρακτικό παράδειγμα του **βρόχου while** σε συνδυασμό με μια **πρόταση if else** , ας υποθέσουμε ότι παίζουμε ένα παιχνίδι Yatzy:

Παράδειγμα

Εκτύπωση "Yatzy!" Εάν ο αριθμός των ζαριών είναι 6:

```
int dice = 1;


while (dice <= 6) {

  if (dice < 6) {

    System.out.println("No Yatzy.");

  } else {

    System.out.println("Yatzy!");

  }

  dice = dice + 1;

}
```

Εάν ο βρόχος περάσει τις τιμές που κυμαίνονται από 1 έως 5, εκτυπώνει "No Yatzy". Κάθε φορά που περνά την τιμή 6, εκτυπώνει "Yatzy!".

Java For Loop

Όταν γνωρίζετε ακριβώς πόσες φορές θέλετε να κάνετε βρόχο μέσω ενός μπλοκ κώδικα, χρησιμοποιήστε τον `for`βρόχο αντί για έναν `while`βρόχο:

Σύνταξη

```
for (statement 1; statement 2; statement 3) {

  // code block to be executed

}
```

Η δήλωση 1 εκτελείται (μία φορά) πριν από την εκτέλεση του μπλοκ κώδικα.

Η δήλωση 2 ορίζει την συνθήκη για την εκτέλεση του μπλοκ κώδικα.

Η δήλωση 3 εκτελείται (κάθε φορά) μετά την εκτέλεση του μπλοκ κώδικα.

Το παρακάτω παράδειγμα θα εκτυπώσει τους αριθμούς 0 έως 4:

Παράδειγμα

```
for (int i = 0; i < 5; i++) {
  System.out.println(i);
}
```

Το παράδειγμα εξηγείται

Η δήλωση 1 ορίζει μια μεταβλητή πριν από την έναρξη του βρόχου (int i = 0).

Η πρόταση 2 ορίζει την συνθήκη για την εκτέλεση του βρόχου (το i πρέπει να είναι μικρότερο από 5). Εάν η συνθήκη είναι αληθής, ο βρόχος θα ξεκινήσει ξανά από την αρχή, εάν είναι false, ο βρόχος θα τελειώσει.

Η δήλωση 3 αυξάνει μια τιμή (i++) κάθε φορά που εκτελείται το μπλοκ κώδικα στον βρόχο.

Άλλο Παράδειγμα

Αυτό το παράδειγμα θα εκτυπώσει μόνο ζυγές τιμές μεταξύ 0 και 10:

Παράδειγμα

```
for (int i = 0; i <= 10; i = i + 2) {
  System.out.println(i);
}
```

Java Nested Loops

Ένθετοι βρόχοι

Είναι επίσης δυνατό να τοποθετήσετε έναν βρόχο μέσα σε έναν άλλο βρόχο. Αυτό ονομάζεται **ένθετος βρόχος** .

Ο "εσωτερικός βρόχος" θα εκτελείται μία φορά για κάθε επανάληψη του "εξωτερικού βρόχου":

Παράδειγμα

```
// Outer loop
for (int i = 1; i <= 2; i++) {
  System.out.println("Outer: " + i); // Executes 2 times


  // Inner loop
  for (int j = 1; j <= 3; j++) {
    System.out.println(" Inner: " + j); // Executes 6 times (2 * 3)
  }
}
```

Java για κάθε βρόχο

Υπάρχει επίσης ένας βρόχος " **για κάθε** ", ο οποίος χρησιμοποιείται αποκλειστικά για τον βρόχο μέσω στοιχείων σε έναν **πίνακα** (ή άλλα σύνολα δεδομένων):

Σύνταξη

```
for (type variableName : arrayName) {
  // code block to be executed
}
```

Το ακόλουθο παράδειγμα εξάγει όλα τα στοιχεία στον πίνακα **αυτοκινήτων** , χρησιμοποιώντας έναν βρόχο " **για-κάθε ":**

Παράδειγμα

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
```

```
}
```

Παραδείγματα Java For Loop

Για να δείξουμε ένα πρακτικό παράδειγμα του **βρόχου for** , ας δημιουργήσουμε ένα πρόγραμμα που μετράει έως το 100 επί δεκάδες:

Παράδειγμα

```
for (int i = 0; i <= 100; i += 10) {
  System.out.println(i);
}
```

Σε αυτό το παράδειγμα, δημιουργούμε ένα πρόγραμμα που εκτυπώνει μόνο ζυγές τιμές μεταξύ 0 και 10:

Παράδειγμα

```
for (int i = 0; i <= 10; i = i + 2) {
  System.out.println(i);
}
```

Και σε αυτό το παράδειγμα, δημιουργούμε ένα πρόγραμμα που εκτυπώνει τον πίνακα πολλαπλασιασμού για έναν καθορισμένο αριθμό:

Παράδειγμα

```
int number = 2;


// Print the multiplication table for the number 2
for (int i = 1; i <= 10; i++) {
  System.out.println(number + " x " + i + " = " + (number * i));
}
```

Java Break και Continue

Java Break

Έχετε ήδη δει τη `break`δήλωση που χρησιμοποιείται σε προηγούμενο κεφάλαιο αυτού του σεμιναρίου. Χρησιμοποιήθηκε για να «ξεπηδήσει» από μια `switch`δήλωση.

Η `break`δήλωση μπορεί επίσης να χρησιμοποιηθεί για να βγει από έναν **βρόχο** .

Αυτό το παράδειγμα σταματά τον βρόχο όταν το i είναι ίσο με 4:

Παράδειγμα Αποκτήστε τον δικό σας διακομιστή Java

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  System.out.println(i);
}
```

Java Συνέχεια

Η `continue`πρόταση διακόπτει μια επανάληψη (στο βρόχο), εάν προκύψει μια καθορισμένη συνθήκη, και συνεχίζει με την επόμενη επανάληψη στον βρόχο.

Αυτό το παράδειγμα παρακάμπτει την τιμή του 4:

Παράδειγμα

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  System.out.println(i);
}
```

Διακοπή και Continue στο while Loop

Μπορείτε επίσης να χρησιμοποιήσετε βρόχους `break`και `continue`in while:

Παράδειγμα διακοπής

```
int i = 0;
while (i < 10) {
  System.out.println(i);
  i++;
  if (i == 4) {
    break;
  }
}
```

Συνέχεια Παράδειγμα

```
int i = 0;
while (i < 10) {
  if (i == 4) {
    i++;
    continue;
  }
  System.out.println(i);
  i++;
}
```

Οι πίνακες χρησιμοποιούνται για την αποθήκευση πολλαπλών τιμών σε μια μεμονωμένη μεταβλητή, αντί να δηλώνουν ξεχωριστές μεταβλητές για κάθε τιμή.

Για να δηλώσετε έναν πίνακα, ορίστε τον τύπο της μεταβλητής με **αγκύλες** :

```
String[] cars;
```

Τώρα έχουμε δηλώσει μια μεταβλητή που περιέχει έναν πίνακα συμβολοσειρών. Για να εισαγάγετε τιμές σε αυτό, μπορείτε να τοποθετήσετε τις τιμές σε μια λίστα διαχωρισμένη με κόμμα, μέσα σε σγουρά άγκιστρα:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

Για να δημιουργήσετε έναν πίνακα ακεραίων, θα μπορούσατε να γράψετε:

```
int[] myNum = {10, 20, 30, 40};
```

Πρόσβαση στα στοιχεία ενός πίνακα

Μπορείτε να αποκτήσετε πρόσβαση σε ένα στοιχείο πίνακα ανατρέχοντας στον αριθμό ευρετηρίου.

Αυτή η δήλωση έχει πρόσβαση στην τιμή του πρώτου στοιχείου στα αυτοκίνητα:

Παράδειγμα

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

System.out.println(cars[0]);

// Outputs Volvo
```

Σημείωση: Τα ευρετήρια του πίνακα ξεκινούν με 0: Το [0] είναι το πρώτο στοιχείο. Το [1] είναι το δεύτερο στοιχείο κ.λπ.

Αλλάξτε ένα στοιχείο πίνακα

Για να αλλάξετε την τιμή ενός συγκεκριμένου στοιχείου, ανατρέξτε στον αριθμό ευρετηρίου:

Παράδειγμα

```
cars[0] = "Opel";
```

Παράδειγμα

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

cars[0] = "Opel";

System.out.println(cars[0]);

// Now outputs Opel instead of Volvo
```

Μήκος πίνακα

Για να μάθετε πόσα στοιχεία έχει ένας πίνακας, χρησιμοποιήστε την `length`ιδιότητα:

Παράδειγμα

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

System.out.println(cars.length);

// Outputs 4
```

Βρόχος συστοιχιών Java

Μπορείτε να κάνετε βρόχο μέσω των στοιχείων του πίνακα με τον `for`βρόχο και να χρησιμοποιήσετε την `length` ιδιότητα για να καθορίσετε πόσες φορές θα εκτελείται ο βρόχος.

Το ακόλουθο παράδειγμα εξάγει όλα τα στοιχεία του πίνακα **αυτοκινήτων** :

Παράδειγμα

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (int i = 0; i < cars.length; i++) {

  System.out.println(cars[i]);

}
```

Βρόχος μέσα από έναν πίνακα με For-Each

Υπάρχει επίσης ένας βρόχος " **για κάθε** ", ο οποίος χρησιμοποιείται αποκλειστικά για τον βρόχο μέσω στοιχείων σε πίνακες:

Σύνταξη

```
for (type variable : arrayname) {

  ...

}
```

Το ακόλουθο παράδειγμα εξάγει όλα τα στοιχεία στον πίνακα **αυτοκινήτων** , χρησιμοποιώντας έναν βρόχο " **για-κάθε ":**

Παράδειγμα

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

for (String i : cars) {

  System.out.println(i);

}
```

Το παραπάνω παράδειγμα μπορεί να διαβαστεί ως εξής: **για κάθε** `String` στοιχείο (που ονομάζεται **i** - όπως στο **i** ndex) στα **αυτοκίνητα** , εκτυπώστε την τιμή του **i** .

Εάν συγκρίνετε τον `for`βρόχο και τον βρόχο **για κάθε** βρόχο, θα δείτε ότι η μέθοδος **για κάθε** βρόχο είναι πιο εύκολη στην εγγραφή, δεν απαιτεί μετρητή (χρησιμοποιώντας την ιδιότητα μήκος) και είναι πιο ευανάγνωστη.

Πίνακες Java - Παραδείγματα πραγματικής ζωής

Για να δείξουμε ένα πρακτικό παράδειγμα χρήσης πινάκων, ας δημιουργήσουμε ένα πρόγραμμα που υπολογίζει τον μέσο όρο διαφορετικών ηλικιών:

Παράδειγμα

```
// An array storing different ages

int ages[] = {20, 22, 18, 35, 48, 26, 87, 70};
```

```
float avg, sum = 0;


// Get the length of the array
int length = ages.length;


// Loop through the elements of the array
for (int age : ages) {
  sum += age;
}


// Calculate the average by dividing the sum by the length
avg = sum / length;


// Print the average
System.out.println("The average age is: " + avg);
```

Και σε αυτό το παράδειγμα, δημιουργούμε ένα πρόγραμμα που βρίσκει τη χαμηλότερη ηλικία μεταξύ διαφορετικών ηλικιών:

Παράδειγμα

```
// An array storing different ages
int ages[] = {20, 22, 18, 35, 48, 26, 87, 70};


float avg, sum = 0;


// Get the length of the array
int length = ages.length;


// Create a 'lowest age' variable and assign the first array element of ages
to it
```

```java
int lowestAge = ages[0];


// Loop through the elements of the ages array to find the lowest age
for (int age : ages) {
  // Check if the current age is smaller than the current 'lowest age'
  if (lowestAge > age) {
    // If the smaller age is found, update 'lowest age' with that element
    lowestAge = age;
  }
}


// Output the value of the lowest age
System.out.println("The lowest age in the array is: " + lowestAge);
```

Πολυδιάστατοι πίνακες Java

Ένας πολυδιάστατος πίνακας είναι ένας πίνακας πινάκων.

Οι πολυδιάστατοι πίνακες είναι χρήσιμοι όταν θέλετε να αποθηκεύσετε δεδομένα ως μορφή πίνακα, όπως ένας πίνακας με σειρές και στήλες.

Για να δημιουργήσετε έναν δισδιάστατο πίνακα, προσθέστε κάθε πίνακα μέσα στο δικό του σύνολο από **σγουρά στηρίγματα** :

Παράδειγμα

```java
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

Το myNumbers είναι πλέον ένας πίνακας με δύο πίνακες ως στοιχεία του.

Στοιχεία πρόσβασης

Για να αποκτήσετε πρόσβαση στα στοιχεία του πίνακα **myNumbers** , καθορίστε δύο ευρετήρια: ένα για τον πίνακα και ένα για το στοιχείο μέσα σε αυτόν τον

πίνακα. Αυτό το παράδειγμα έχει πρόσβαση στο τρίτο στοιχείο (2) στον δεύτερο πίνακα (1) του myNumbers:

Παράδειγμα

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

System.out.println(myNumbers[1][2]); // Outputs 7
```

Θυμηθείτε ότι: Τα ευρετήρια του πίνακα ξεκινούν με 0: Το [0] είναι το πρώτο στοιχείο. Το [1] είναι το δεύτερο στοιχείο κ.λπ.

Αλλαγή τιμών στοιχείων

Μπορείτε επίσης να αλλάξετε την τιμή ενός στοιχείου:

Παράδειγμα

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

myNumbers[1][2] = 9;

System.out.println(myNumbers[1][2]); // Outputs 9 instead of 7
```

Βρόχος μέσω πολυδιάστατου πίνακα

Μπορείτε επίσης να χρησιμοποιήσετε ένα `for loop`εσωτερικό του άλλου `for loop`για να λάβετε τα στοιχεία ενός δισδιάστατου πίνακα (πρέπει ακόμα να δείξουμε τους δύο δείκτες):

Παράδειγμα

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

for (int i = 0; i < myNumbers.length; ++i) {

  for (int j = 0; j < myNumbers[i].length; ++j) {

    System.out.println(myNumbers[i][j]);

  }
}
```

Ή θα μπορούσατε απλώς να χρησιμοποιήσετε έναν βρόχο **για κάθε** , ο οποίος θεωρείται ευκολότερος στην ανάγνωση και τη γραφή:

Παράδειγμα

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };

for (int[] row : myNumbers) {

  for (int i : row) {

    System.out.println(i);

  }

}
```

Μέθοδοι Java

Μια **μέθοδος** είναι ένα μπλοκ κώδικα που εκτελείται μόνο όταν καλείται.

Μπορείτε να μεταβιβάσετε δεδομένα, γνωστά ως παράμετροι, σε μια μέθοδο.

Οι μέθοδοι χρησιμοποιούνται για την εκτέλεση συγκεκριμένων ενεργειών και είναι επίσης γνωστές ως **συναρτήσεις** .

Γιατί να χρησιμοποιήσετε μεθόδους; Για επαναχρησιμοποίηση κώδικα: ορίστε τον κώδικα μία φορά και χρησιμοποιήστε τον πολλές φορές.

Δημιουργήστε μια μέθοδο

Μια μέθοδος πρέπει να δηλωθεί μέσα σε μια κλάση. Ορίζεται με το όνομα της μεθόδου, ακολουθούμενο από παρενθέσεις **()** . Η Java παρέχει ορισμένες προκαθορισμένες μεθόδους, όπως `System.out.println()`, αλλά μπορείτε επίσης να δημιουργήσετε τις δικές σας μεθόδους για την εκτέλεση συγκεκριμένων ενεργειών:

Παράδειγμα

Δημιουργήστε μια μέθοδο μέσα στο Main:

```
public class Main {

  static void myMethod() {
```

```
    // code to be executed
  }
}
```

Παράδειγμα Επεξήγηση

- `myMethod()`είναι το όνομα της μεθόδου
- `static`σημαίνει ότι η μέθοδος ανήκει στην κλάση Main και όχι αντικείμενο της κλάσης Main. Θα μάθετε περισσότερα σχετικά με τα αντικείμενα και τον τρόπο πρόσβασης σε μεθόδους μέσω αντικειμένων αργότερα σε αυτό το σεμινάριο.
- `void`σημαίνει ότι αυτή η μέθοδος δεν έχει επιστρεφόμενη τιμή. Θα μάθετε περισσότερα σχετικά με τις επιστρεφόμενες τιμές αργότερα σε αυτό το κεφάλαιο

Καλέστε μια μέθοδο

Για να καλέσετε μια μέθοδο σε Java, γράψτε το όνομα της μεθόδου ακολουθούμενο από δύο παρενθέσεις **()** και ένα ερωτηματικό **.**

Στο παρακάτω παράδειγμα, `myMethod()`χρησιμοποιείται για την εκτύπωση ενός κειμένου (η ενέργεια), όταν καλείται:

Παράδειγμα

Στο εσωτερικό `main`, καλέστε τη `myMethod()`μέθοδο:

```
public class Main {
  static void myMethod() {
    System.out.println("I just got executed!");
  }

  public static void main(String[] args) {
    myMethod();
  }
}
```

```
// Outputs "I just got executed!"
```

Μια μέθοδος μπορεί επίσης να κληθεί πολλές φορές:

Παράδειγμα

```
public class Main {
  static void myMethod() {
    System.out.println("I just got executed!");
  }

  public static void main(String[] args) {
    myMethod();
    myMethod();
    myMethod();
  }
}

// I just got executed!
// I just got executed!
// I just got executed!
```

Παράμετροι μεθόδου Java

Οι πληροφορίες μπορούν να μεταβιβαστούν στις μεθόδους ως παράμετρος. Οι παράμετροι λειτουργούν ως μεταβλητές μέσα στη μέθοδο.

Οι παράμετροι καθορίζονται μετά το όνομα της μεθόδου, μέσα στις παρενθέσεις. Μπορείτε να προσθέσετε όσες παραμέτρους θέλετε, απλώς διαχωρίστε τις με κόμμα.

Το παρακάτω παράδειγμα έχει μια μέθοδο που λαμβάνει ως παράμετρο ένα `String`καλούμενο **fname** . Όταν καλείται η μέθοδος, περνάμε ένα μικρό όνομα, το οποίο χρησιμοποιείται μέσα στη μέθοδο για την εκτύπωση του πλήρους ονόματος:

Παράδειγμα

```
public class Main {

  static void myMethod(String fname) {

    System.out.println(fname + " Refsnes");

  }


  public static void main(String[] args) {

    myMethod("Liam");

    myMethod("Jenny");

    myMethod("Anja");

  }

}

// Liam Refsnes

// Jenny Refsnes

// Anja Refsnes
```

Όταν μια **παράμετρος** μεταβιβάζεται στη μέθοδο, ονομάζεται **όρισμα** . Έτσι, από το παραπάνω παράδειγμα: `fname`είναι μια **παράμετρος** , ενώ `Liam`, `Jenny`και `Anja`είναι **ορίσματα** .

Πολλαπλές Παράμετροι

Μπορείτε να έχετε όσες παραμέτρους θέλετε:

Παράδειγμα

```
public class Main {
  static void myMethod(String fname, int age) {
    System.out.println(fname + " is " + age);
  }

  public static void main(String[] args) {
    myMethod("Liam", 5);
    myMethod("Jenny", 8);
    myMethod("Anja", 31);
  }
}

// Liam is 5
// Jenny is 8
// Anja is 31
```

Σημειώστε ότι όταν εργάζεστε με πολλές παραμέτρους, η κλήση της μεθόδου πρέπει να έχει τον ίδιο αριθμό ορισμάτων με τις παραμέτρους και τα ορίσματα πρέπει να μεταβιβάζονται με την ίδια σειρά.

Μια μέθοδος με το If...Else

Είναι σύνηθες να χρησιμοποιούνται `if...else`δηλώσεις μέσα σε μεθόδους:

Παράδειγμα

```
public class Main {

  // Create a checkAge() method with an integer variable called age
```

```
  static void checkAge(int age) {

    // If age is less than 18, print "access denied"
    if (age < 18) {
      System.out.println("Access denied - You are not old enough!");

    // If age is greater than, or equal to, 18, print "access granted"
    } else {
      System.out.println("Access granted - You are old enough!");
    }

  }

  public static void main(String[] args) {
    checkAge(20); // Call the checkAge method and pass along an age of 20
  }
}

// Outputs "Access granted - You are old enough!"
```

Επιστροφή Java

Στην προηγούμενη σελίδα , χρησιμοποιήσαμε τη `void`λέξη-κλειδί σε όλα τα παραδείγματα, γεγονός που υποδεικνύει ότι η μέθοδος δεν πρέπει να επιστρέψει μια τιμή.

Εάν θέλετε η μέθοδος να επιστρέψει μια τιμή, μπορείτε να χρησιμοποιήσετε έναν πρωτόγονο τύπο δεδομένων (όπως `int`, `char`, κ.λπ.) αντί για `void`και να χρησιμοποιήσετε τη `return` λέξη-κλειδί μέσα στη μέθοδο:

Παράδειγμα

```
public class Main {
  static int myMethod(int x) {
    return 5 + x;
  }

  public static void main(String[] args) {
    System.out.println(myMethod(3));
  }
}
// Outputs 8 (5 + 3)
```

Αυτό το παράδειγμα επιστρέφει το άθροισμα των δύο παραμέτρων μιας μεθόδου :

Παράδειγμα

```
public class Main {
  static int myMethod(int x, int y) {
    return x + y;
  }

  public static void main(String[] args) {
    System.out.println(myMethod(5, 3));
  }
}
// Outputs 8 (5 + 3)
```

Μπορείτε επίσης να αποθηκεύσετε το αποτέλεσμα σε μια μεταβλητή (συνιστάται, καθώς είναι πιο εύκολο να διαβαστεί και να διατηρηθεί):

Παράδειγμα

```
public class Main {
```

```
  static int myMethod(int x, int y) {
    return x + y;
  }

  public static void main(String[] args) {
    int z = myMethod(5, 3);
    System.out.println(z);
  }
}
// Outputs 8 (5 + 3)
```

Υπερφόρτωση μεθόδου Java

Με **την υπερφόρτωση μεθόδου** , πολλές μέθοδοι μπορούν να έχουν το ίδιο όνομα με διαφορετικές παραμέτρους:

Παράδειγμα

```
int myMethod(int x)
float myMethod(float x)
double myMethod(double x, double y)
```

Εξετάστε το ακόλουθο παράδειγμα, το οποίο έχει δύο μεθόδους που προσθέτουν αριθμούς διαφορετικού τύπου:

Παράδειγμα

```
static int plusMethodInt(int x, int y) {
  return x + y;
}

static double plusMethodDouble(double x, double y) {
  return x + y;
```

```
}


public static void main(String[] args) {
  int myNum1 = plusMethodInt(8, 5);
  double myNum2 = plusMethodDouble(4.3, 6.26);
  System.out.println("int: " + myNum1);
  System.out.println("double: " + myNum2);
}
```

Αντί να ορίσετε δύο μεθόδους που θα πρέπει να κάνουν το ίδιο πράγμα, είναι καλύτερο να υπερφορτώσετε τη μία.

Στο παρακάτω παράδειγμα, υπερφορτώνουμε τη `plusMethod` μέθοδο για να λειτουργήσει και για τα δύο `int` και `double`για :

Παράδειγμα

```
static int plusMethod(int x, int y) {
  return x + y;
}


static double plusMethod(double x, double y) {
  return x + y;
}


public static void main(String[] args) {
  int myNum1 = plusMethod(8, 5);
  double myNum2 = plusMethod(4.3, 6.26);
  System.out.println("int: " + myNum1);
  System.out.println("double: " + myNum2);
}
```

Σημείωση: Πολλές μέθοδοι μπορεί να έχουν το ίδιο όνομα, εφόσον ο αριθμός ή/και ο τύπος των παραμέτρων είναι διαφορετικοί.

Πεδίο εφαρμογής Java

Στην Java, οι μεταβλητές είναι προσβάσιμες μόνο εντός της περιοχής που έχουν δημιουργηθεί. Αυτό ονομάζεται **πεδίο εφαρμογής** .

Πεδίο μεθόδου

Οι μεταβλητές που δηλώνονται απευθείας μέσα σε μια μέθοδο είναι διαθέσιμες οπουδήποτε στη μέθοδο ακολουθώντας τη γραμμή κώδικα στην οποία δηλώθηκαν:

Παράδειγμα

```
public class Main {

  public static void main(String[] args) {


    // Code here CANNOT use x


    int x = 100;


    // Code here can use x

    System.out.println(x);

  }

}
```

Μπλοκ πεδίου

Ένα μπλοκ κώδικα αναφέρεται σε όλο τον κώδικα ανάμεσα σε σγουρά στηρίγματα `{}`.

Οι μεταβλητές που δηλώνονται μέσα σε μπλοκ κώδικα είναι προσβάσιμες μόνο από τον κώδικα μεταξύ των αγκύλων, ο οποίος ακολουθεί τη γραμμή στην οποία δηλώθηκε η μεταβλητή:

Παράδειγμα

```
public class Main {

  public static void main(String[] args) {


    // Code here CANNOT use x


    { // This is a block


      // Code here CANNOT use x


      int x = 100;


      // Code here CAN use x

      System.out.println(x);


    } // The block ends here


  // Code here CANNOT use x


  }

}
```

Ένα μπλοκ κώδικα μπορεί να υπάρχει από μόνο του ή μπορεί να ανήκει σε μια πρόταση `if`ή . Στην περίπτωση των δηλώσεων, οι μεταβλητές που δηλώνονται στην ίδια τη δήλωση είναι επίσης διαθέσιμες εντός του πεδίου εφαρμογής του μπλοκ.`whileforfor`

Java Recursion

Η αναδρομή είναι η τεχνική της ίδιας της κλήσης μιας συνάρτησης. Αυτή η τεχνική παρέχει έναν τρόπο να χωρίσουμε τα περίπλοκα προβλήματα σε απλά προβλήματα που είναι πιο εύκολο να επιλυθούν.

Η αναδρομή μπορεί να είναι λίγο δύσκολο να κατανοηθεί. Ο καλύτερος τρόπος για να καταλάβετε πώς λειτουργεί είναι να πειραματιστείτε με αυτό.

Παράδειγμα Αναδρομής

Η πρόσθεση δύο αριθμών μαζί είναι εύκολο να γίνει, αλλά η προσθήκη μιας σειράς αριθμών είναι πιο περίπλοκη. Στο παρακάτω παράδειγμα, η αναδρομή χρησιμοποιείται για να προσθέσει ένα εύρος αριθμών μαζί αναλύοντάς το στην απλή εργασία της πρόσθεσης δύο αριθμών:

Παράδειγμα

Χρησιμοποιήστε την αναδρομή για να προσθέσετε όλους τους αριθμούς μέχρι το 10.

```
public class Main {

  public static void main(String[] args) {

    int result = sum(10);

    System.out.println(result);

  } } }

  public static int sum(int k) {

    if (k > 0) {

      return k + sum(k - 1);

    } else {

      return 0;
```

Παράδειγμα Επεξήγηση

Όταν `sum()`καλείται η συνάρτηση, προσθέτει παράμετρο `k`στο άθροισμα όλων των αριθμών μικρότερων από `k`και επιστρέφει το αποτέλεσμα. Όταν το k γίνει 0, η συνάρτηση απλώς επιστρέφει 0. Όταν εκτελείται, το πρόγραμμα ακολουθεί τα εξής βήματα:

```
10 + άθροισμα(9)
10 + ( 9 + άθροισμα(8) )
10 + ( 9 + ( 8 + άθροισμα(7) ) )
...
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + άθροισμα(0)
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0
```

Εφόσον η συνάρτηση δεν καλεί τον εαυτό της όταν `k`είναι 0, το πρόγραμμα σταματά εκεί και επιστρέφει το αποτέλεσμα.

Κατάσταση παύσης

Ακριβώς όπως οι βρόχοι μπορούν να αντιμετωπίσουν το πρόβλημα του άπειρου βρόχου, οι αναδρομικές συναρτήσεις μπορούν να αντιμετωπίσουν το πρόβλημα της άπειρης αναδρομής. Η άπειρη αναδρομή είναι όταν η συνάρτηση δεν σταματά ποτέ να καλεί τον εαυτό της. Κάθε αναδρομική συνάρτηση θα πρέπει να έχει μια συνθήκη παύσης, η οποία είναι η συνθήκη όπου η συνάρτηση σταματά να καλεί τον εαυτό της. Στο προηγούμενο παράδειγμα, η συνθήκη παύσης είναι όταν η παράμετρος `k` γίνεται 0.

Είναι χρήσιμο να δείτε μια ποικιλία διαφορετικών παραδειγμάτων για να κατανοήσετε καλύτερα την έννοια. Σε αυτό το παράδειγμα, η συνάρτηση προσθέτει ένα εύρος αριθμών μεταξύ μιας αρχής και ενός τέλους. Η συνθήκη παύσης για αυτήν την αναδρομική συνάρτηση είναι όταν **το τέλος** δεν είναι μεγαλύτερο από **την αρχή** :

Παράδειγμα

Χρησιμοποιήστε την αναδρομή για να προσθέσετε όλους τους αριθμούς μεταξύ 5 και 10.

```
public class Main {

  public static void main(String[] args) {

    int result = sum(5, 10);

    System.out.println(result);

  }

  public static int sum(int start, int end) {

    if (end > start) {
```

```
      return end + sum(start, end - 1);
    } else {
      return end;
    }
  }
}
```

Ο προγραμματιστής θα πρέπει να είναι πολύ προσεκτικός με την αναδρομή, καθώς μπορεί να είναι πολύ εύκολο να γράψει μια συνάρτηση που δεν τερματίζεται ποτέ ή μια συνάρτηση που χρησιμοποιεί υπερβολική ποσότητα μνήμης ή ισχύος επεξεργαστή. Ωστόσο, όταν γράφεται σωστά η αναδρομή μπορεί να είναι μια πολύ αποτελεσματική και μαθηματικά κομψή προσέγγιση στον προγραμματισμό.

Java OOP

Java - Τι είναι το OOP;

Το OOP σημαίνει **Αντικειμενοστραφής Προγραμματισμός** .

Ο διαδικαστικός προγραμματισμός αφορά τη σύνταξη διαδικασιών ή μεθόδων που εκτελούν λειτουργίες στα δεδομένα, ενώ ο αντικειμενοστραφής προγραμματισμός αφορά τη δημιουργία αντικειμένων που περιέχουν δεδομένα και μεθόδους.

Ο αντικειμενοστραφής προγραμματισμός έχει πολλά πλεονεκτήματα έναντι του διαδικαστικού προγραμματισμού:

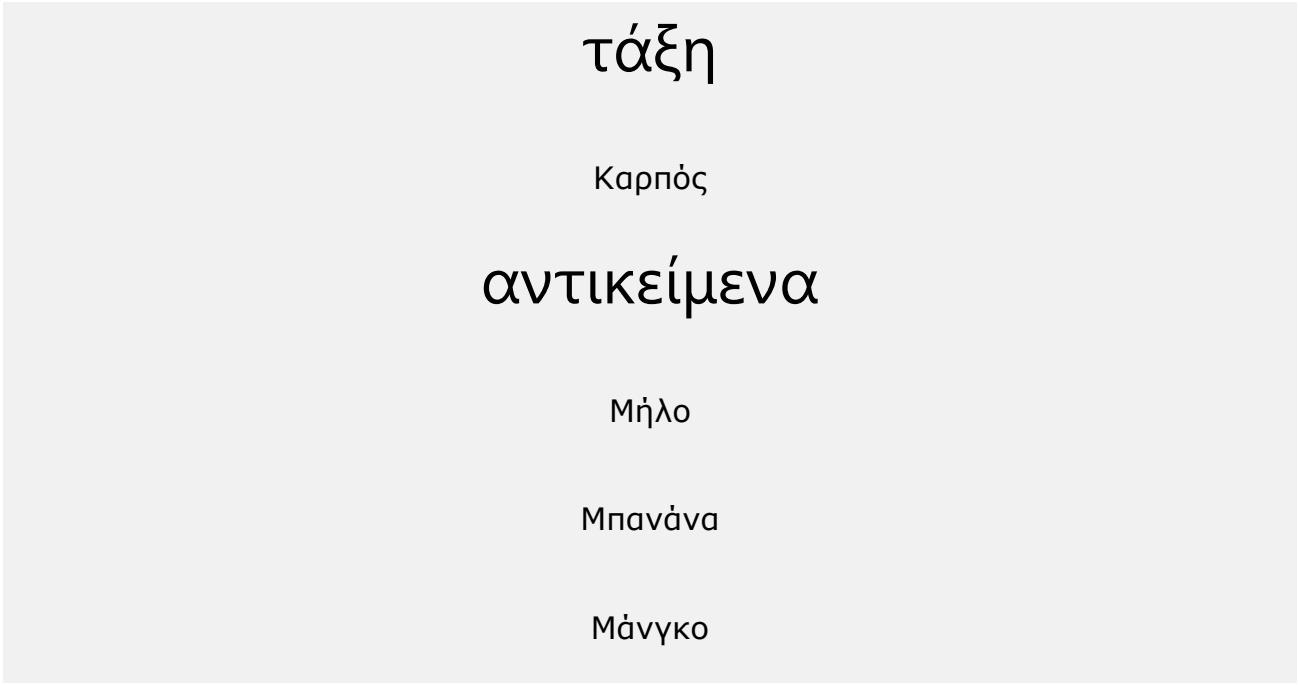
- Το OOP είναι πιο γρήγορο και πιο εύκολο στην εκτέλεση
- Το OOP παρέχει μια σαφή δομή για τα προγράμματα
- Το OOP βοηθά στη διατήρηση του κώδικα Java DRY "Don't Repeat Yourself" και διευκολύνει τη διατήρηση, την τροποποίηση και τον εντοπισμό σφαλμάτων του κώδικα
- Το OOP καθιστά δυνατή τη δημιουργία πλήρους επαναχρησιμοποιήσιμων εφαρμογών με λιγότερο κώδικα και μικρότερο χρόνο ανάπτυξης

Συμβουλή: Η αρχή "Don't Repeat Yourself" (DRY) αφορά τη μείωση της επανάληψης του κώδικα. Θα πρέπει να εξαγάγετε τους κωδικούς που είναι συνηθισμένοι για την εφαρμογή και να τους τοποθετήσετε σε ένα μόνο μέρος και να τους επαναχρησιμοποιήσετε αντί να τους επαναλάβετε.

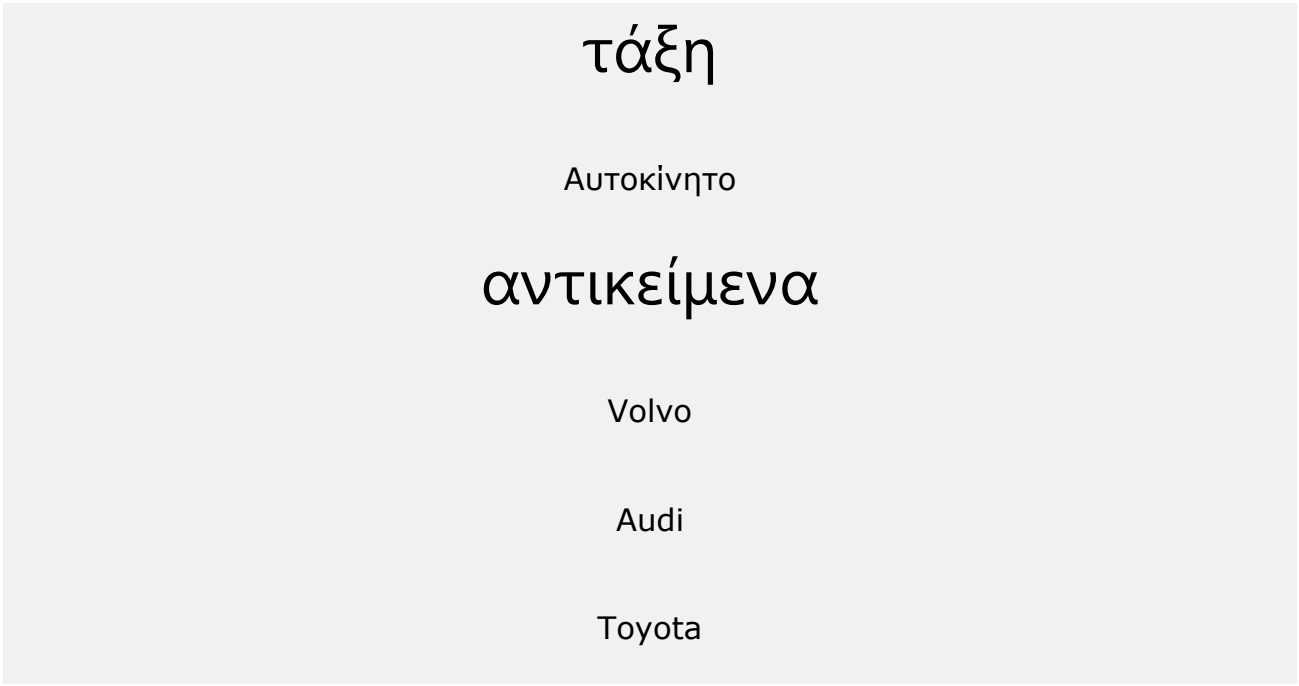
Java - Τι είναι οι κλάσεις και τα αντικείμενα;

Οι κλάσεις και τα αντικείμενα είναι οι δύο κύριες πτυχές του αντικειμενοστρεφούς προγραμματισμού.

Κοιτάξτε την παρακάτω εικόνα για να δείτε τη διαφορά μεταξύ κλάσης και αντικειμένων:



Άλλο παράδειγμα:



Έτσι, μια κλάση είναι ένα πρότυπο για αντικείμενα και ένα αντικείμενο είναι ένα παράδειγμα μιας κλάσης.

Όταν δημιουργούνται τα μεμονωμένα αντικείμενα, κληρονομούν όλες τις μεταβλητές και τις μεθόδους από την κλάση.

Java Classes and Objects

Java Κλάσεις/Αντικείμενα

Η Java είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού.

Τα πάντα στη Java σχετίζονται με κλάσεις και αντικείμενα, μαζί με τα χαρακτηριστικά και τις μεθόδους της. Για παράδειγμα: στην πραγματική ζωή, ένα αυτοκίνητο είναι ένα αντικείμενο. Το αυτοκίνητο έχει **χαρακτηριστικά** , όπως βάρος και χρώμα, και **μεθόδους** , όπως κίνηση και φρένο.

Μια κλάση μοιάζει με έναν κατασκευαστή αντικειμένων ή ένα "σχεδιασμό" για τη δημιουργία αντικειμένων.

Δημιουργήστε μια τάξη

Για να δημιουργήσετε μια τάξη, χρησιμοποιήστε τη λέξη-κλειδί `class`:

Κύρια.java

Δημιουργήστε μια κλάση με το όνομα " `Main`" με μια μεταβλητή x:

```
public class Main {

  int x = 5;

}
```

Θυμηθείτε από το κεφάλαιο Σύνταξη Java ότι μια τάξη πρέπει πάντα να ξεκινά με ένα κεφαλαίο πρώτο γράμμα και ότι το όνομα του αρχείου java πρέπει να ταιριάζει με το όνομα της τάξης.

Δημιουργήστε ένα αντικείμενο

Στην Java, ένα αντικείμενο δημιουργείται από μια κλάση. Έχουμε ήδη δημιουργήσει την κλάση με το όνομα `Main`, οπότε τώρα μπορούμε να τη χρησιμοποιήσουμε για να δημιουργήσουμε αντικείμενα.

Για να δημιουργήσετε ένα αντικείμενο του `Main`, καθορίστε το όνομα της κλάσης, ακολουθούμενο από το όνομα του αντικειμένου και χρησιμοποιήστε τη λέξη-κλειδί `new`:

Παράδειγμα

Δημιουργήστε ένα αντικείμενο που ονομάζεται " `myObj`" και εκτυπώστε την τιμή του x:

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

Πολλαπλά Αντικείμενα

Μπορείτε να δημιουργήσετε πολλά αντικείμενα μιας κλάσης:

Παράδειγμα

Δημιουργήστε δύο αντικείμενα από `Main`:

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj1 = new Main();  // Object 1
    Main myObj2 = new Main();  // Object 2
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
```

```
  }

}
```

Χρήση πολλαπλών τάξεων

Μπορείτε επίσης να δημιουργήσετε ένα αντικείμενο μιας κλάσης και να αποκτήσετε πρόσβαση σε αυτό σε μια άλλη κλάση. Αυτό χρησιμοποιείται συχνά για καλύτερη οργάνωση των κλάσεων (μία κλάση έχει όλα τα χαρακτηριστικά και τις μεθόδους, ενώ η άλλη κλάση κατέχει τη `main()`μέθοδο (κώδικας που θα εκτελεστεί)).

Θυμηθείτε ότι το όνομα του αρχείου java πρέπει να ταιριάζει με το όνομα της κλάσης. Σε αυτό το παράδειγμα, έχουμε δημιουργήσει δύο αρχεία στον ίδιο κατάλογο/φάκελο:

- Κύρια.java
- Δεύτερον.java

Κύρια.java

```
public class Main {

  int x = 5;

}
```

Δεύτερον.java

```
class Second {

  public static void main(String[] args) {

    Main myObj = new Main();

    System.out.println(myObj.x);

  }

}
```

Όταν έχουν μεταγλωττιστεί και τα δύο αρχεία:

```
C:\Users\Your Name>javac Main.java
C:\Users\Your Name>javac Second.java
```

Εκτελέστε το αρχείο Second.java:

```
C:\Users\Your Name>java Second
```

Και η έξοδος θα είναι:

```
5
```

Java Class Attributes

Ιδιότητες κλάσης Java

Στο προηγούμενο κεφάλαιο, χρησιμοποιήσαμε τον όρο "μεταβλητή" `x`στο παράδειγμα (όπως φαίνεται παρακάτω). Είναι στην πραγματικότητα ένα **χαρακτηριστικό** της τάξης. Ή θα μπορούσατε να πείτε ότι τα χαρακτηριστικά κλάσης είναι μεταβλητές μέσα σε μια κλάση:

Παράδειγμα

Δημιουργήστε μια κλάση που ονομάζεται " `Main`" με δύο χαρακτηριστικά: `x`και `y`:

```
public class Main {

  int x = 5;

  int y = 3;

}
```

Ένας άλλος όρος για τα χαρακτηριστικά της κλάσης είναι **τα πεδία** .

Πρόσβαση στις ιδιότητες

Μπορείτε να αποκτήσετε πρόσβαση σε χαρακτηριστικά δημιουργώντας ένα αντικείμενο της κλάσης και χρησιμοποιώντας τη σύνταξη κουκκίδων (`.`):

Το παρακάτω παράδειγμα θα δημιουργήσει ένα αντικείμενο της `Main`κλάσης, με το όνομα `myObj`. Χρησιμοποιούμε το `x` χαρακτηριστικό στο αντικείμενο για να εκτυπώσουμε την τιμή του:

Παράδειγμα

Δημιουργήστε ένα αντικείμενο που ονομάζεται " `myObj`" και εκτυπώστε την τιμή του `x`:

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

Τροποποίηση Ιδιοτήτων

Μπορείτε επίσης να τροποποιήσετε τις τιμές των χαρακτηριστικών:

Παράδειγμα

Ορίστε την τιμή του xσε 40:

```
public class Main {
  int x;

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 40;
    System.out.println(myObj.x);
  }
}
```

Ή να παρακάμψετε τις υπάρχουσες τιμές:

Παράδειγμα

Αλλάξτε την τιμή του xσε 25:

```
public class Main {

  int x = 10;


  public static void main(String[] args) {

    Main myObj = new Main();

    myObj.x = 25; // x is now 25

    System.out.println(myObj.x);

  }

}
```

Εάν δεν θέλετε τη δυνατότητα παράκαμψης υπαρχουσών τιμών, δηλώστε το χαρακτηριστικό ως final:

Παράδειγμα

```
public class Main {

  final int x = 10;


  public static void main(String[] args) {

    Main myObj = new Main();

    myObj.x = 25; // will generate an error: cannot assign a value to a
final variable

    System.out.println(myObj.x);

  }

}
```

Η finalλέξη-κλειδί είναι χρήσιμη όταν θέλετε μια μεταβλητή να αποθηκεύει πάντα την ίδια τιμή, όπως το PI (3.14159...).

Η finalλέξη-κλειδί ονομάζεται "τροποποιητής". Θα μάθετε περισσότερα για αυτά στο κεφάλαιο Java Modifiers .

Πολλαπλά Αντικείμενα

Εάν δημιουργήσετε πολλά αντικείμενα μιας κλάσης, μπορείτε να αλλάξετε τις τιμές των χαρακτηριστικών σε ένα αντικείμενο, χωρίς να επηρεάσετε τις τιμές των χαρακτηριστικών στο άλλο:

Παράδειγμα

Αλλάξτε την τιμή του xσε 25 in myObj2και αφήστε xτο myObj1αμετάβλητο:

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj1 = new Main();  // Object 1
    Main myObj2 = new Main();  // Object 2
    myObj2.x = 25;
    System.out.println(myObj1.x);  // Outputs 5
    System.out.println(myObj2.x);  // Outputs 25
  }
}
```

Πολλαπλές Ιδιότητες

Μπορείτε να καθορίσετε όσα χαρακτηριστικά θέλετε:

Παράδειγμα

```
public class Main {
  String fname = "John";
```

```
  String lname = "Doe";
  int age = 24;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println("Name: " + myObj.fname + " " + myObj.lname);
    System.out.println("Age: " + myObj.age);
  }
}
```

Java Class Methods

Μέθοδοι κλάσης Java

Μάθατε από το κεφάλαιο Μέθοδοι Java ότι οι μέθοδοι δηλώνονται σε μια κλάση και ότι χρησιμοποιούνται για την εκτέλεση συγκεκριμένων ενεργειών:

Παράδειγμα

Δημιουργήστε μια μέθοδο `myMethod()`με το όνομα Main:

```
public class Main {
  static void myMethod() {
    System.out.println("Hello World!");
  }
}
```

`myMethod()`εκτυπώνει ένα κείμενο (τη δράση), όταν **ονομάζεται** . Για να καλέσετε μια μέθοδο, γράψτε το όνομα της μεθόδου ακολουθούμενο από δύο παρενθέσεις **()** και ένα ερωτηματικό **.**

Παράδειγμα

Μέσα `main`, καλέστε `myMethod()`:

```
public class Main {
  static void myMethod() {
    System.out.println("Hello World!");
  }

  public static void main(String[] args) {
    myMethod();
  }
}

// Outputs "Hello World!"
```

Στατική εναντίον Δημοσίου

Θα δείτε συχνά προγράμματα Java που έχουν ιδιότητες και μεθόδους `static`είτε είτε.`public`

Στο παραπάνω παράδειγμα, δημιουργήσαμε μια `static` μέθοδο, που σημαίνει ότι μπορεί να προσπελαστεί χωρίς να δημιουργηθεί ένα αντικείμενο της κλάσης, σε αντίθεση με το `public`, το οποίο είναι προσβάσιμο μόνο από αντικείμενα:

Παράδειγμα

Ένα παράδειγμα για να δείξετε τις διαφορές μεταξύ `static`και `public` **μεθόδων** :

```
public class Main {
  // Static method
  static void myStaticMethod() {
    System.out.println("Static methods can be called without creating
objects");
  }

  // Public method
```

```
  public void myPublicMethod() {
    System.out.println("Public methods must be called by creating objects");
  }

  // Main method
  public static void main(String[] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would compile an error

    Main myObj = new Main(); // Create an object of Main
    myObj.myPublicMethod(); // Call the public method on the object
  }
}
```

Σημείωση: Θα μάθετε περισσότερα για αυτές τις λέξεις-κλειδιά (που ονομάζονται τροποποιητές) στο κεφάλαιο Java Modifiers .

Μέθοδοι πρόσβασης με αντικείμενο

Παράδειγμα

Δημιουργήστε ένα αντικείμενο αυτοκινήτου με το όνομα `myCar`. Καλέστε τις μεθόδους `fullThrottle()`και `speed()` στο `myCar`αντικείμενο και εκτελέστε το πρόγραμμα:

```
// Create a Main class
public class Main {

  // Create a fullThrottle() method
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
```

```
  }

  // Create a speed() method and add a parameter
  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }

  // Inside main, call the methods on the myCar object
  public static void main(String[] args) {
    Main myCar = new Main();   // Create a myCar object
    myCar.fullThrottle();      // Call the fullThrottle() method
    myCar.speed(200);          // Call the speed() method
  }
}

// The car is going as fast as it can!
// Max speed is: 200
```

Το παράδειγμα εξηγείται

1) Δημιουργήσαμε μια προσαρμοσμένη `Main`τάξη με τη `class`λέξη-κλειδί.

2) Δημιουργήσαμε τις μεθόδους `fullThrottle()`και `speed()` στην `Main`τάξη.

3) Η `fullThrottle()`μέθοδος και η `speed()` μέθοδος θα εκτυπώσουν κάποιο κείμενο, όταν καλούνται.

4) Η `speed()` μέθοδος δέχεται μια `int`παράμετρο που ονομάζεται `maxSpeed`- θα τη χρησιμοποιήσουμε στο **8)** .

5) Για να χρησιμοποιήσουμε την `Main`κλάση και τις μεθόδους της, πρέπει να δημιουργήσουμε ένα **αντικείμενο** της `Main`Κλάσης.

6) Στη συνέχεια, μεταβείτε στη `main()`μέθοδο, η οποία μέχρι τώρα γνωρίζετε ότι είναι μια ενσωματωμένη μέθοδος Java που εκτελεί το πρόγραμμά σας (εκτελείται οποιοσδήποτε κώδικας μέσα στο main).

7) Χρησιμοποιώντας τη `new`λέξη-κλειδί δημιουργήσαμε ένα αντικείμενο με το όνομα `myCar`.

8) Στη συνέχεια, καλούμε τις μεθόδους `fullThrottle()`και `speed()` στο `myCar`αντικείμενο και εκτελούμε το πρόγραμμα χρησιμοποιώντας το όνομα του αντικειμένου (`myCar`), ακολουθούμενο από μια τελεία (`.`), ακολουθούμενη από το όνομα της μεθόδου (`fullThrottle();`και `speed(200);`). Παρατηρήστε ότι προσθέτουμε μια `int`παράμετρο **200** μέσα στη `speed()`μέθοδο.

Να θυμάστε ότι..

Η τελεία (`.`) χρησιμοποιείται για την πρόσβαση στα χαρακτηριστικά και τις μεθόδους του αντικειμένου.

Για να καλέσετε μια μέθοδο σε Java, γράψτε το όνομα της μεθόδου ακολουθούμενο από ένα σύνολο παρενθέσεων **()** , ακολουθούμενο από ένα ερωτηματικό (`;`).

Μια κλάση πρέπει να έχει ένα αντίστοιχο όνομα αρχείου (`Main`και **Main.java**).

Χρήση πολλαπλών τάξεων

Όπως προσδιορίσαμε στο κεφάλαιο Classes , είναι καλή πρακτική να δημιουργείτε ένα αντικείμενο μιας κλάσης και να έχετε πρόσβαση σε αυτό σε μια άλλη κλάση.

Θυμηθείτε ότι το όνομα του αρχείου java πρέπει να ταιριάζει με το όνομα της κλάσης. Σε αυτό το παράδειγμα, έχουμε δημιουργήσει δύο αρχεία στον ίδιο κατάλογο:

- Κύρια.java
- Δεύτερον.java

Κύρια.java

```
public class Main {

  public void fullThrottle() {

    System.out.println("The car is going as fast as it can!");

  }



  public void speed(int maxSpeed) {

    System.out.println("Max speed is: " + maxSpeed);

  }
```

```
}
```

Δεύτερον.java

```
class Second {

  public static void main(String[] args) {

    Main myCar = new Main();      // Create a myCar object

    myCar.fullThrottle();       // Call the fullThrottle() method

    myCar.speed(200);           // Call the speed() method

  }

}
```

Όταν έχουν μεταγλωττιστεί και τα δύο αρχεία:

```
C:\Users\Your Name>javac Main.java
C:\Users\Your Name>javac Second.java
```

Εκτελέστε το αρχείο Second.java:

```
C:\Users\Your Name>java Second
```

Και η έξοδος θα είναι:

```
The car is going as fast as it can!
Max speed is: 200
```

Java Constructors

Ένας κατασκευαστής στην Java είναι μια **ειδική μέθοδος** που χρησιμοποιείται για την προετοιμασία αντικειμένων. Ο κατασκευαστής καλείται όταν δημιουργείται ένα αντικείμενο μιας κλάσης. Μπορεί να χρησιμοποιηθεί για τον ορισμό αρχικών τιμών για τα χαρακτηριστικά αντικειμένων:

Παράδειγμα

Δημιουργήστε έναν κατασκευαστή:

```
// Create a Main class

public class Main {
```

```
  int x;  // Create a class attribute


  // Create a class constructor for the Main class
  public Main() {
    x = 5;  // Set the initial value for the class attribute x
  }


  public static void main(String[] args) {
    Main myObj = new Main(); // Create an object of class Main (This will
call the constructor)
    System.out.println(myObj.x); // Print the value of x
  }
}


// Outputs 5
```

Σημειώστε ότι το όνομα του κατασκευαστή πρέπει **να ταιριάζει με το όνομα της κλάσης** και δεν μπορεί να έχει **τύπο επιστροφής** (όπως `void`).

Σημειώστε επίσης ότι ο κατασκευαστής καλείται όταν δημιουργείται το αντικείμενο.

Όλες οι κλάσεις έχουν κατασκευαστές από προεπιλογή: εάν δεν δημιουργήσετε μόνοι σας έναν κατασκευαστή κλάσεων, η Java δημιουργεί έναν για εσάς. Ωστόσο, τότε δεν μπορείτε να ορίσετε αρχικές τιμές για τα χαρακτηριστικά των αντικειμένων.

Παράμετροι κατασκευαστή

Οι κατασκευαστές μπορούν επίσης να λάβουν παραμέτρους, οι οποίες χρησιμοποιούνται για την προετοιμασία των χαρακτηριστικών.

Το ακόλουθο παράδειγμα προσθέτει μια `int y`παράμετρο στον κατασκευαστή. Μέσα στον κατασκευαστή θέτουμε το x σε y (x=y). Όταν καλούμε τον κατασκευαστή,

περνάμε μια παράμετρο στον κατασκευαστή (5), η οποία θα ορίσει την τιμή του x σε 5:

Παράδειγμα

```
public class Main {
  int x;

  public Main(int y) {
    x = y;
  }

  public static void main(String[] args) {
    Main myObj = new Main(5);
    System.out.println(myObj.x);
  }
}

// Outputs 5
```

Μπορείτε να έχετε όσες παραμέτρους θέλετε:

Παράδειγμα

```
public class Main {
  int modelYear;
  String modelName;

  public Main(int year, String name) {
    modelYear = year;
    modelName = name;
  }
```

```
  public static void main(String[] args) {
    Main myCar = new Main(1969, "Mustang");
    System.out.println(myCar.modelYear + " " + myCar.modelName);
  }
}

// Outputs 1969 Mustang
```

Java Modifiers

Τροποποιητές

Μέχρι τώρα, είστε αρκετά εξοικειωμένοι με τη `public`λέξη-κλειδί που εμφανίζεται σχεδόν σε όλα τα παραδείγματά μας:

```
public class Main
```

Η `public`λέξη-κλειδί είναι ένας **τροποποιητής πρόσβασης** , που σημαίνει ότι χρησιμοποιείται για τον ορισμό του επιπέδου πρόσβασης για κλάσεις, χαρακτηριστικά, μεθόδους και κατασκευαστές.

Χωρίζουμε τους τροποποιητές σε δύο ομάδες:

- **Τροποποιητές πρόσβασης** - ελέγχει το επίπεδο πρόσβασης
- **Τροποποιητές χωρίς πρόσβαση** - δεν ελέγχουν το επίπεδο πρόσβασης, αλλά παρέχουν άλλες λειτουργίες

Τροποποιητές πρόσβασης

Για **τις τάξεις** , μπορείτε να χρησιμοποιήσετε ένα `public`ή *προεπιλεγμένο* :

Modifier	Description	Try it
`public`	The class is accessible by any other class	Try it »
default	The class is only accessible by classes in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter	Try it »

Για **χαρακτηριστικά, μεθόδους και κατασκευαστές** , μπορείτε να χρησιμοποιήσετε ένα από τα ακόλουθα:

Modifier	Description	Try it
`public`	The code is accessible for all classes	Try it »
`private`	The code is only accessible within the declared class	Try it »
default	The code is only accessible in the same package. This is used when you don't specify a modifier. You will learn more about packages in the Packages chapter	Try it »
`protected`	The code is accessible in the same package and **subclasses**. You will learn more about subclasses and superclasses in the Inheritance chapter	Try it »

Τροποποιητές χωρίς πρόσβαση

Για **τις τάξεις** , μπορείτε να χρησιμοποιήσετε ένα `final`ή `abstract`:

Modifier	Description	Try it
`final`	The class cannot be inherited by other classes (You will learn more about inheritance in the Inheritance chapter)	Try it »
`abstract`	The class cannot be used to create objects (To access an abstract class, it must be inherited from another class. You will learn more about inheritance and abstraction in the Inheritance and Abstraction chapters)	Try it »

Για **χαρακτηριστικά και μεθόδους** , μπορείτε να χρησιμοποιήσετε ένα από τα παρακάτω:

Modifier	Description
`final`	Attributes and methods cannot be overridden/modified
`static`	Attributes and methods belongs to the class, rather than an object
`abstract`	Can only be used in an abstract class, and can only be used on methods. The method does not have a body, for example **abstract void run();**. The body is provided by the subclass (inherited from). You will learn more about inheritance and abstraction in the Inheritance and Abstraction chapters
`transient`	Attributes and methods are skipped when serializing the object containing them

`synchronized`	Methods can only be accessed by one thread at a time
`volatile`	The value of an attribute is not cached thread-locally, and is always read from the "main memory"

Τελικός

Εάν δεν θέλετε τη δυνατότητα παράκαμψης των υπαρχουσών τιμών χαρακτηριστικών, δηλώστε τα χαρακτηριστικά ως `final`:

Παράδειγμα Αποκτήστε τον δικό σας διακομιστή Java

```
public class Main {

  final int x = 10;

  final double PI = 3.14;


  public static void main(String[] args) {

    Main myObj = new Main();

    myObj.x = 50; // will generate an error: cannot assign a value to a
final variable

    myObj.PI = 25; // will generate an error: cannot assign a value to a
final variable

    System.out.println(myObj.x);

  }

}
```

Στατικός

Μια `static`μέθοδος σημαίνει ότι μπορεί να προσπελαστεί χωρίς να δημιουργηθεί ένα αντικείμενο της κλάσης, σε αντίθεση με `public`:

Παράδειγμα

Ένα παράδειγμα για να καταδείξετε τις διαφορές μεταξύ `static`και `public`μεθόδων:

```
public class Main {

  // Static method

  static void myStaticMethod() {

    System.out.println("Static methods can be called without creating
objects");

  }


  // Public method

  public void myPublicMethod() {

    System.out.println("Public methods must be called by creating objects");

  }


  // Main method

  public static void main(String[ ] args) {

    myStaticMethod(); // Call the static method

    // myPublicMethod(); This would output an error


    Main myObj = new Main(); // Create an object of Main

    myObj.myPublicMethod(); // Call the public method

  }

}
```

Περίληψη

Μια abstractμέθοδος ανήκει σε μια abstractκλάση και δεν έχει σώμα. Το σώμα παρέχεται από την υποκατηγορία:

Παράδειγμα

```
// Code from filename: Main.java

// abstract class
abstract class Main {

  public String fname = "John";

  public int age = 24;

  public abstract void study(); // abstract method

}


// Subclass (inherit from Main)

class Student extends Main {

  public int graduationYear = 2018;

  public void study() { // the body of the abstract method is provided here

    System.out.println("Studying all day long");

  }

}

// End code from filename: Main.java


// Code from filename: Second.java

class Second {

  public static void main(String[] args) {

    // create an object of the Student class (which inherits attributes and
methods from Main)

    Student myObj = new Student();
```

```
    System.out.println("Name: " + myObj.fname);
    System.out.println("Age: " + myObj.age);
    System.out.println("Graduation Year: " + myObj.graduationYear);
    myObj.study(); // call abstract method
  }
}
```

Java Encapsulation

Ενθυλάκωση

Η έννοια του **Encapsulation** , είναι να βεβαιωθείτε ότι τα "ευαίσθητα" δεδομένα είναι κρυμμένα από τους χρήστες. Για να το πετύχετε αυτό θα πρέπει:

- δηλώνουν μεταβλητές/χαρακτηριστικά κλάσης ως`private`
- Παρέχετε μεθόδους δημόσιας **λήψης** και **ρύθμισης** για πρόσβαση και ενημέρωση της τιμής μιας `private` μεταβλητής

Λήψη και Ρύθμιση

Μάθατε από το προηγούμενο κεφάλαιο ότι `private`οι μεταβλητές είναι προσβάσιμες μόνο εντός της ίδιας τάξης (μια εξωτερική τάξη δεν έχει πρόσβαση σε αυτήν). Ωστόσο, είναι δυνατή η πρόσβαση σε αυτά εάν παρέχουμε μεθόδους δημόσιας **λήψης** και **ρύθμισης .**

Η `get`μέθοδος επιστρέφει την τιμή της μεταβλητής και η `set`μέθοδος ορίζει την τιμή.

Η σύνταξη και για τα δύο είναι ότι ξεκινούν με ένα `get`ή `set`, ακολουθούμενο από το όνομα της μεταβλητής, με το πρώτο γράμμα με κεφαλαία:

Παράδειγμα Αποκτήστε τον δικό σας διακομιστή Java

```
public class Person {
  private String name; // private = restricted access
```

```
  // Getter
  public String getName() {
    return name;
  }

  // Setter
  public void setName(String newName) {
    this.name = newName;
  }
}
```

Το παράδειγμα εξηγείται

Η `get`μέθοδος επιστρέφει την τιμή της μεταβλητής `name`.

Η `set`μέθοδος παίρνει μια παράμετρο (`newName`) και την εκχωρεί στη `name`μεταβλητή.
Η `this`λέξη-κλειδί χρησιμοποιείται για να αναφέρεται στο τρέχον αντικείμενο.

Ωστόσο, καθώς η `name`μεταβλητή δηλώνεται ως `private`, **δεν μπορούμε** να έχουμε πρόσβαση σε αυτήν εκτός αυτής της κλάσης:

Παράδειγμα

```
public class Main {
  public static void main(String[] args) {
    Person myObj = new Person();
    myObj.name = "John";  // error
    System.out.println(myObj.name); // error
  }
}
```

Παράδειγμα εκτέλεσης »

Εάν η μεταβλητή είχε δηλωθεί ως `public`, θα περιμέναμε την ακόλουθη έξοδο:

```
John
```

Ωστόσο, καθώς προσπαθούμε να αποκτήσουμε πρόσβαση σε μια `private`μεταβλητή, λαμβάνουμε ένα σφάλμα:

```
MyClass.java:4: error: name has private access in Person
    myObj.name = "John";
          ^
MyClass.java:5: error: name has private access in Person
    System.out.println(myObj.name);
                       ^
2 errors
```

Αντίθετα, χρησιμοποιούμε τις μεθόδους `getName()`και `setName()`για να αποκτήσουμε πρόσβαση και να ενημερώσουμε τη μεταβλητή:

Παράδειγμα

```
public class Main {

  public static void main(String[] args) {

    Person myObj = new Person();

    myObj.setName("John"); // Set the value of the name variable to "John"

    System.out.println(myObj.getName());

  }

}


// Outputs "John"
```

Γιατί ενθυλάκωση;

- Καλύτερος έλεγχος χαρακτηριστικών και μεθόδων κλάσης
- Τα χαρακτηριστικά κλάσης μπορούν να γίνουν **μόνο για ανάγνωση** (αν χρησιμοποιείτε μόνο τη `get`μέθοδο) ή **μόνο για εγγραφή** (αν χρησιμοποιείτε μόνο τη `set`μέθοδο)
- Ευέλικτο: ο προγραμματιστής μπορεί να αλλάξει ένα μέρος του κώδικα χωρίς να επηρεάσει άλλα μέρη
- Αυξημένη ασφάλεια δεδομένων

Java Packages

Πακέτα Java & API

Ένα πακέτο σε Java χρησιμοποιείται για την ομαδοποίηση σχετικών κλάσεων. Σκεφτείτε το ως **φάκελο σε έναν κατάλογο αρχείων** . Χρησιμοποιούμε πακέτα για να αποφύγουμε διενέξεις ονομάτων και για να γράψουμε έναν καλύτερο συντηρήσιμο κώδικα. Τα πακέτα χωρίζονται σε δύο κατηγορίες:

- Ενσωματωμένα πακέτα (πακέτα από το Java API)
- Πακέτα καθορισμένα από τον χρήστη (δημιουργήστε τα δικά σας πακέτα)

Ενσωματωμένα πακέτα

Το Java API είναι μια βιβλιοθήκη προγραμμένων κλάσεων, που είναι δωρεάν στη χρήση, που περιλαμβάνονται στο περιβάλλον ανάπτυξης Java.

Η βιβλιοθήκη περιέχει στοιχεία για τη διαχείριση εισόδου, τον προγραμματισμό της βάσης δεδομένων και πολλά πολλά άλλα. Η πλήρης λίστα βρίσκεται στον ιστότοπο της Oracles: https://docs.oracle.com/javase/8/docs/api/ .

Η βιβλιοθήκη χωρίζεται σε **πακέτα** και **τάξεις** . Αυτό σημαίνει ότι μπορείτε είτε να εισαγάγετε μια κλάση (μαζί με τις μεθόδους και τα χαρακτηριστικά της), είτε ένα ολόκληρο πακέτο που περιέχει όλες τις κλάσεις που ανήκουν στο καθορισμένο πακέτο.

Για να χρησιμοποιήσετε μια τάξη ή ένα πακέτο από τη βιβλιοθήκη, πρέπει να χρησιμοποιήσετε τη `import` λέξη-κλειδί:

Σύνταξη

```
import package.name.Class;   // Import a single class

import package.name.*;   // Import the whole package
```

Εισαγωγή κλάσης

Εάν βρείτε μια κλάση που θέλετε να χρησιμοποιήσετε, για παράδειγμα, την `Scanner`κλάση, **η οποία χρησιμοποιείται για τη λήψη εισόδου χρήστη** , γράψτε τον ακόλουθο κώδικα:

Παράδειγμα

```
import java.util.Scanner;
```

Στο παραπάνω παράδειγμα, `java.util`είναι ένα πακέτο, ενώ `Scanner`είναι μια κλάση του `java.util`πακέτου.

Για να χρησιμοποιήσετε την `Scanner`κλάση, δημιουργήστε ένα αντικείμενο της κλάσης και χρησιμοποιήστε οποιαδήποτε από τις διαθέσιμες μεθόδους που βρίσκονται στην `Scanner`τεκμηρίωση της κλάσης. Στο παράδειγμά μας, θα χρησιμοποιήσουμε τη `nextLine()`μέθοδο, η οποία χρησιμοποιείται για την ανάγνωση μιας πλήρους γραμμής:

Παράδειγμα

Χρησιμοποιώντας την `Scanner`τάξη για να λάβετε στοιχεία από τον χρήστη:

```
import java.util.Scanner;


class MyClass {

  public static void main(String[] args) {

    Scanner myObj = new Scanner(System.in);

    System.out.println("Enter username");


    String userName = myObj.nextLine();

    System.out.println("Username is: " + userName);

  }

}
```

Εισαγωγή πακέτου

Υπάρχουν πολλά πακέτα για να διαλέξετε. Στο προηγούμενο παράδειγμα, χρησιμοποιήσαμε την `Scanner`κλάση από το `java.util`πακέτο. Αυτό το πακέτο περιέχει επίσης εγκαταστάσεις ημερομηνίας και ώρας, γεννήτρια τυχαίων αριθμών και άλλες κατηγορίες χρησιμότητας.

Για να εισαγάγετε ένα ολόκληρο πακέτο, τελειώστε την πρόταση με αστερίσκο (`*`). Το παρακάτω παράδειγμα θα εισάγει ΟΛΕΣ τις κλάσεις στο `java.util`πακέτο:

Παράδειγμα

```
import java.util.*;
```

Πακέτα καθορισμένα από τον χρήστη

Για να δημιουργήσετε το δικό σας πακέτο, πρέπει να καταλάβετε ότι η Java χρησιμοποιεί έναν κατάλογο συστήματος αρχείων για να τα αποθηκεύσει. Ακριβώς όπως οι φάκελοι στον υπολογιστή σας:

Παράδειγμα

```
└── root
    └── mypack
        └── MyPackageClass.java
```

Για να δημιουργήσετε ένα πακέτο, χρησιμοποιήστε τη `package`λέξη-κλειδί:

MyPackageClass.java

```
package mypack;

class MyPackageClass {

  public static void main(String[] args) {

    System.out.println("This is my package!");

  }

}
```

Αποθηκεύστε το αρχείο ως **MyPackageClass.java** και μεταγλωττίστε το:

```
C:\Users\Your Name>javac MyPackageClass.java
```

Στη συνέχεια, μεταγλωττίστε το πακέτο:

```
C:\Users\Your Name>javac -d . MyPackageClass.java
```

Αυτό αναγκάζει τον μεταγλωττιστή να δημιουργήσει το πακέτο "mypack".

Η `-d`λέξη-κλειδί καθορίζει τον προορισμό για την αποθήκευση του αρχείου κλάσης. Μπορείτε να χρησιμοποιήσετε οποιοδήποτε όνομα καταλόγου, όπως c:/user (windows) ή, εάν θέλετε να διατηρήσετε το πακέτο στον ίδιο κατάλογο, μπορείτε να χρησιμοποιήσετε το σύμβολο κουκκίδας " .", όπως στο παραπάνω παράδειγμα.

Σημείωση: Το όνομα του πακέτου πρέπει να γράφεται με πεζά γράμματα για να αποφευχθεί η σύγκρουση με τα ονόματα κλάσεων.

Όταν μεταγλωττίσαμε το πακέτο στο παραπάνω παράδειγμα, δημιουργήθηκε ένας νέος φάκελος που ονομάζεται "mypack".

Για να εκτελέσετε το αρχείο **MyPackageClass.java** , γράψτε τα εξής:

```
C:\Users\Your Name>java mypack.MyPackageClass
```

Η έξοδος θα είναι:

```
This is my package!
```

Java Inheritance

Κληρονομικότητα Java (Υποκατηγορία και Υπερκλάση)

Στην Java, είναι δυνατό να κληρονομηθούν χαρακτηριστικά και μέθοδοι από τη μια κλάση στην άλλη. Ομαδοποιούμε την «έννοια της κληρονομικότητας» σε δύο κατηγορίες:

- **υποκλάση** (παιδί) - η κλάση που κληρονομεί από μια άλλη κλάση
- **superclass** (γονικός) - η κλάση από την οποία κληρονομείται

Για να κληρονομήσετε από μια κλάση, χρησιμοποιήστε τη `extends` λέξη-κλειδί.

Στο παρακάτω παράδειγμα, η `Car`κλάση (υποκλάση) κληρονομεί τα χαρακτηριστικά και τις μεθόδους από την `Vehicle`κλάση (υπερκλάση):

Παράδειγμα

```
class Vehicle {
  protected String brand = "Ford";        // Vehicle attribute
  public void honk() {                    // Vehicle method
    System.out.println("Tuut, tuut!");
  }
```

```
}


class Car extends Vehicle {
  private String modelName = "Mustang";    // Car attribute
  public static void main(String[] args) {


    // Create a myCar object
    Car myCar = new Car();


    // Call the honk() method (from the Vehicle class) on the myCar object
    myCar.honk();


    // Display the value of the brand attribute (from the Vehicle class) and
the value of the modelName from the Car class
    System.out.println(myCar.brand + " " + myCar.modelName);
  }
}
```

Προσέξατε τον `protected`τροποποιητή στο Vehicle;

Ορίσαμε το χαρακτηριστικό **brand στο Vehicle** σε έναν `protected` τροποποιητή πρόσβασης . Εάν είχε οριστεί σε `private`, η κατηγορία Car δεν θα μπορούσε να έχει πρόσβαση σε αυτό.

Γιατί και πότε να χρησιμοποιήσετε την "Κληρονομικότητα";

- Είναι χρήσιμο για επαναχρησιμοποίηση κώδικα: επαναχρησιμοποιήστε χαρακτηριστικά και μεθόδους μιας υπάρχουσας κλάσης όταν δημιουργείτε μια νέα κλάση.

Συμβουλή: Ρίξτε επίσης μια ματιά στο επόμενο κεφάλαιο, Πολυμορφισμός , το οποίο χρησιμοποιεί κληρονομικές μεθόδους για την εκτέλεση διαφορετικών εργασιών.

Η τελική Λέξη-κλειδί

Εάν δεν θέλετε άλλες κλάσεις να κληρονομήσουν από μια τάξη, χρησιμοποιήστε τη `final`λέξη-κλειδί:

Εάν προσπαθήσετε να αποκτήσετε πρόσβαση σε μια `final`τάξη, η Java θα δημιουργήσει ένα σφάλμα:

```
final class Vehicle {

  ...

}


class Car extends Vehicle {

  ...

}
```

Η έξοδος θα είναι κάπως έτσι:

```
Main.java:9: error: cannot inherit from final Vehicle
class Main extends Vehicle {
                   ^
1 error)
```

Java Polymorphism

Πολυμορφισμός Java

Πολυμορφισμός σημαίνει «πολλές μορφές» και εμφανίζεται όταν έχουμε πολλές τάξεις που σχετίζονται μεταξύ τους κληρονομικά.

Όπως προσδιορίσαμε στο προηγούμενο κεφάλαιο. **Η κληρονομικότητα** μας επιτρέπει να κληρονομήσουμε χαρακτηριστικά και μεθόδους από άλλη κλάση. **Ο πολυμορφισμός** χρησιμοποιεί αυτές τις μεθόδους για να εκτελέσει διαφορετικές εργασίες. Αυτό μας επιτρέπει να εκτελέσουμε μια ενιαία ενέργεια με διαφορετικούς τρόπους.

Για παράδειγμα, σκεφτείτε μια υπερκλάση που ονομάζεται `Animal`και έχει μια μέθοδο που ονομάζεται `animalSound()`. Οι υποκατηγορίες των ζώων θα μπορούσαν να είναι Χοίροι, Γάτες, Σκύλοι, Πουλιά - Και έχουν επίσης τη δική τους εφαρμογή ενός ήχου ζώων (το γουρούνι μελάνη, και η γάτα νιαουρίζει κ.λπ.):

Παράδειγμα

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}

class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}

class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}
```

Θυμηθείτε από το κεφάλαιο Κληρονομικότητα ότι χρησιμοποιούμε τη `extends`λέξη-κλειδί για να κληρονομήσουμε από μια κλάση.

Τώρα μπορούμε να δημιουργήσουμε `Pig`και `Dog`αντικείμενα και να καλέσουμε τη `animalSound()`μέθοδο και στα δύο:

Παράδειγμα

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}

class Pig extends Animal {
```

```
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}

class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}

class Main {
  public static void main(String[] args) {
    Animal myAnimal = new Animal();  // Create a Animal object
    Animal myPig = new Pig();  // Create a Pig object
    Animal myDog = new Dog();  // Create a Dog object
    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
  }
}
```

Γιατί και πότε να χρησιμοποιήσετε το "Κληρονομικότητα" και τον "Πολυμορφισμό";

- Είναι χρήσιμο για επαναχρησιμοποίηση κώδικα: επαναχρησιμοποιήστε χαρακτηριστικά και μεθόδους μιας υπάρχουσας κλάσης όταν δημιουργείτε μια νέα κλάση.

Java Inner Classes

Java Inner Classes

Στην Java, είναι επίσης δυνατή η ένθεση κλάσεων (μια κλάση μέσα σε μια κλάση). Ο σκοπός των ένθετων κλάσεων είναι να ομαδοποιήσουν τις κλάσεις που ανήκουν μεταξύ τους, κάτι που κάνει τον κώδικά σας πιο ευανάγνωστο και διατηρήσιμο.

Για να αποκτήσετε πρόσβαση στην εσωτερική κλάση, δημιουργήστε ένα αντικείμενο της εξωτερικής κλάσης και, στη συνέχεια, δημιουργήστε ένα αντικείμενο της εσωτερικής κλάσης:

Παράδειγμα

```
class OuterClass {
  int x = 10;

  class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.y + myOuter.x);
  }
}

// Outputs 15 (5 + 10)
```

Ιδιωτική εσωτερική τάξη

Σε αντίθεση με μια "κανονική" τάξη, μια εσωτερική τάξη μπορεί να είναι `private`ή `protected`. Εάν δεν θέλετε εξωτερικά αντικείμενα να έχουν πρόσβαση στην εσωτερική κλάση, δηλώστε την κλάση ως `private`:

Παράδειγμα

```
class OuterClass {

  int x = 10;


  private class InnerClass {

    int y = 5;

  }

}


public class Main {

  public static void main(String[] args) {

    OuterClass myOuter = new OuterClass();

    OuterClass.InnerClass myInner = myOuter.new InnerClass();

    System.out.println(myInner.y + myOuter.x);

  }

}
```

Εάν προσπαθήσετε να αποκτήσετε πρόσβαση σε μια ιδιωτική εσωτερική κλάση από μια εξωτερική τάξη, εμφανίζεται ένα σφάλμα:

```
Main.java:13: error: OuterClass.InnerClass has private access in
OuterClass
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
               ^
```

Στατική εσωτερική τάξη

Μια εσωτερική κλάση μπορεί επίσης να είναι `static`, πράγμα που σημαίνει ότι μπορείτε να έχετε πρόσβαση σε αυτήν χωρίς να δημιουργήσετε ένα αντικείμενο της εξωτερικής κλάσης:

Παράδειγμα

```
class OuterClass {
  int x = 10;

  static class InnerClass {
    int y = 5;
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass.InnerClass myInner = new OuterClass.InnerClass();
    System.out.println(myInner.y);
  }
}

// Outputs 5
```

Σημείωση: όπως `static`τα χαρακτηριστικά και οι μέθοδοι, μια `static`εσωτερική κλάση δεν έχει πρόσβαση σε μέλη της εξωτερικής κλάσης.

Πρόσβαση στην εξωτερική τάξη από την εσωτερική τάξη

Ένα πλεονέκτημα των εσωτερικών κλάσεων είναι ότι μπορούν να έχουν πρόσβαση σε χαρακτηριστικά και μεθόδους της εξωτερικής κλάσης:

Παράδειγμα

```
class OuterClass {
  int x = 10;

  class InnerClass {
    public int myInnerMethod() {
      return x;
    }
  }
}

public class Main {
  public static void main(String[] args) {
    OuterClass myOuter = new OuterClass();
    OuterClass.InnerClass myInner = myOuter.new InnerClass();
    System.out.println(myInner.myInnerMethod());
  }
}

// Outputs 10
```

Java Abstraction

Η αφαίρεση δεδομένων είναι η διαδικασία απόκρυψης ορισμένων λεπτομερειών και εμφάνισης μόνο βασικών πληροφοριών στον χρήστη.
Η αφαίρεση μπορεί να επιτευχθεί είτε με **αφηρημένες κλάσεις** είτε **με διεπαφές** (για τις οποίες θα μάθετε περισσότερα στο επόμενο κεφάλαιο).

Η abstractλέξη-κλειδί είναι ένας τροποποιητής χωρίς πρόσβαση, που χρησιμοποιείται για κλάσεις και μεθόδους:

- **Abstract class:** είναι μια περιορισμένη κλάση που δεν μπορεί να χρησιμοποιηθεί για τη δημιουργία αντικειμένων (για πρόσβαση σε αυτήν, πρέπει να κληρονομηθεί από άλλη κλάση).

- **Αφηρημένη μέθοδος:** μπορεί να χρησιμοποιηθεί μόνο σε μια αφηρημένη κλάση και δεν έχει σώμα. Το σώμα παρέχεται από την υποκατηγορία (κληρονομείται από).

Μια αφηρημένη κλάση μπορεί να έχει αφηρημένες και κανονικές μεθόδους:

```
abstract class Animal {

  public abstract void animalSound();

  public void sleep() {

    System.out.println("Zzz");

  }

}
```

Από το παραπάνω παράδειγμα, δεν είναι δυνατό να δημιουργηθεί ένα αντικείμενο της κλάσης Animal:

```
Animal myObj = new Animal(); // will generate an error
```

Για να αποκτήσετε πρόσβαση στην αφηρημένη κλάση, πρέπει να κληρονομηθεί από άλλη κλάση. Ας μετατρέψουμε την κλάση Animal που χρησιμοποιήσαμε στο κεφάλαιο Polymorphism σε μια αφηρημένη τάξη:

Θυμηθείτε από το κεφάλαιο Κληρονομικότητα ότι χρησιμοποιούμε τη extendsλέξη-κλειδί για να κληρονομήσουμε από μια κλάση.

Παράδειγμα

```
// Abstract class

abstract class Animal {

  // Abstract method (does not have a body)

  public abstract void animalSound();
```

```
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Γιατί και πότε να χρησιμοποιήσετε αφηρημένες τάξεις και μεθόδους;

Για να επιτύχετε ασφάλεια - αποκρύψτε ορισμένες λεπτομέρειες και εμφανίστε μόνο τις σημαντικές λεπτομέρειες ενός αντικειμένου.

Σημείωση: Η αφαίρεση μπορεί επίσης να επιτευχθεί με [τις διεπαφές](), για τις οποίες θα μάθετε περισσότερα στο επόμενο κεφάλαιο.

Java Interface

Διεπαφές

Ένας άλλος τρόπος για να επιτευχθεί αφαίρεση στην Java, είναι με τις διεπαφές.

Η An `interface`είναι μια εντελώς " **αφηρημένη κλάση** " που χρησιμοποιείται για την ομαδοποίηση σχετικών μεθόδων με κενά σώματα:

Παράδειγμα

```
// interface

interface Animal {

  public void animalSound(); // interface method (does not have a body)

  public void run(); // interface method (does not have a body)

}
```

Για να αποκτήσετε πρόσβαση στις μεθόδους διεπαφής, η διεπαφή πρέπει να "υλοποιηθεί" (κάπως σαν να κληρονομείται) από μια άλλη κλάση με τη `implements` λέξη-κλειδί (αντί για `extends`). Το σώμα της μεθόδου διεπαφής παρέχεται από την κλάση "implement":

Παράδειγμα

```
// Interface

interface Animal {

  public void animalSound(); // interface method (does not have a body)

  public void sleep(); // interface method (does not have a body)

}


// Pig "implements" the Animal interface

class Pig implements Animal {

  public void animalSound() {

    // The body of animalSound() is provided here
```

```
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}


class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Σημειώσεις για τις διεπαφές:

- Όπως και **οι αφηρημένες κλάσεις** , οι διεπαφές **δεν μπορούν** να χρησιμοποιηθούν για τη δημιουργία αντικειμένων (στο παραπάνω παράδειγμα, δεν είναι δυνατή η δημιουργία αντικειμένου "Animal" στην MyMainClass)
- Οι μέθοδοι διεπαφής δεν έχουν σώμα - το σώμα παρέχεται από την κλάση "υλοποίηση".
- Κατά την υλοποίηση μιας διεπαφής, πρέπει να παρακάμψετε όλες τις μεθόδους της
- Οι μέθοδοι διεπαφής είναι από προεπιλογή `abstract`και `public`
- Τα χαρακτηριστικά διεπαφής είναι από προεπιλογή `public`και `staticfinal`
- Μια διεπαφή δεν μπορεί να περιέχει έναν κατασκευαστή (καθώς δεν μπορεί να χρησιμοποιηθεί για τη δημιουργία αντικειμένων)

Γιατί και πότε να χρησιμοποιήσετε τις διεπαφές;

1) Για να επιτύχετε ασφάλεια - αποκρύψτε ορισμένες λεπτομέρειες και εμφανίστε μόνο τις σημαντικές λεπτομέρειες ενός αντικειμένου (διεπαφής).

2) Η Java δεν υποστηρίζει "πολλαπλή κληρονομικότητα" (μια κλάση μπορεί να κληρονομήσει μόνο από μία υπερκλάση). Ωστόσο, μπορεί να επιτευχθεί με διεπαφές, επειδή η κλάση μπορεί **να υλοποιήσει** πολλαπλές διεπαφές. **Σημείωση:** Για να εφαρμόσετε πολλαπλές διεπαφές, διαχωρίστε τις με κόμμα (βλ. παράδειγμα παρακάτω).

Πολλαπλές διεπαφές

Για να εφαρμόσετε πολλές διεπαφές, διαχωρίστε τις με κόμμα:

Παράδειγμα

```
interface FirstInterface {
  public void myMethod(); // interface method
}

interface SecondInterface {
  public void myOtherMethod(); // interface method
}

class DemoClass implements FirstInterface, SecondInterface {
  public void myMethod() {
    System.out.println("Some text..");
  }
  public void myOtherMethod() {
    System.out.println("Some other text...");
  }
}

class Main {
  public static void main(String[] args) {
    DemoClass myObj = new DemoClass();
    myObj.myMethod();
    myObj.myOtherMethod();
```

```
  }
}
```

Java Enums

Το An `enum`είναι μια ειδική "κλάση" που αντιπροσωπεύει μια ομάδα **σταθερών** (αμετάβλητες μεταβλητές, όπως `final`μεταβλητές).

Για να δημιουργήσετε ένα `enum`, χρησιμοποιήστε τη `enum`λέξη-κλειδί (αντί για κλάση ή διεπαφή) και διαχωρίστε τις σταθερές με κόμμα. Σημειώστε ότι θα πρέπει να είναι με κεφαλαία γράμματα:

Παράδειγμα

```
enum Level {
  LOW,
  MEDIUM,
  HIGH
}
```

Μπορείτε να αποκτήσετε πρόσβαση `enum`σε σταθερές με τη σύνταξη **κουκκίδων** :

```
Level myVar = Level.MEDIUM;
```

Το Enum είναι συντομογραφία του "enumerations", που σημαίνει "συγκεκριμένη λίστα".

Enum μέσα σε μια Τάξη

Μπορείτε επίσης να έχετε μια `enum`εσωτερική τάξη:

Παράδειγμα

```
public class Main {
  enum Level {
```

```
    LOW,
    MEDIUM,
    HIGH
  }

  public static void main(String[] args) {
    Level myVar = Level.MEDIUM;
    System.out.println(myVar);
  }
}
```

Η έξοδος θα είναι:

```
MEDIUM
```

Enum σε μια δήλωση διακόπτη

Τα enums χρησιμοποιούνται συχνά σε `switch`δηλώσεις για τον έλεγχο των αντίστοιχων τιμών:

Παράδειγμα

```
enum Level {
  LOW,
  MEDIUM,
  HIGH
}

public class Main {
  public static void main(String[] args) {
    Level myVar = Level.MEDIUM;
```

```
    switch(myVar) {
      case LOW:
        System.out.println("Low level");
        break;
      case MEDIUM:
         System.out.println("Medium level");
        break;
      case HIGH:
        System.out.println("High level");
        break;
    }
  }
}
```

Η έξοδος θα είναι:

```
Medium level
```

Βρόχος μέσω Enum

Ο τύπος enum έχει μια `values()`μέθοδο, η οποία επιστρέφει έναν πίνακα με όλες τις σταθερές enum. Αυτή η μέθοδος είναι χρήσιμη όταν θέλετε να κάνετε βρόχο μέσω των σταθερών ενός enum:

Παράδειγμα

```
for (Level myVar : Level.values()) {
  System.out.println(myVar);
}
```

Η έξοδος θα είναι:

```
LOW
MEDIUM
HIGH
```

Διαφορά μεταξύ Enums και Classes

Ένα `enum`κουτί, ακριβώς όπως ένα `class`, έχει ιδιότητες και μεθόδους. Η μόνη διαφορά είναι ότι οι σταθερές enum είναι `public`, `static`και `final` (αμετάβλητες - δεν μπορούν να παρακαμφθούν).

Το an `enum`δεν μπορεί να χρησιμοποιηθεί για τη δημιουργία αντικειμένων και δεν μπορεί να επεκτείνει άλλες κλάσεις (αλλά μπορεί να υλοποιήσει διεπαφές).

Γιατί και πότε να χρησιμοποιήσετε το Enums;

Χρησιμοποιήστε enums όταν έχετε τιμές που γνωρίζετε ότι δεν πρόκειται να αλλάξουν, όπως ημέρες μηνών, ημέρες, χρώματα, τράπουλα κ.λπ.

Java User Input (Scanner)

Είσοδος χρήστη Java

Η `Scanner`κλάση χρησιμοποιείται για τη λήψη εισόδου χρήστη και βρίσκεται στο `java.util`πακέτο.

Για να χρησιμοποιήσετε την `Scanner`κλάση, δημιουργήστε ένα αντικείμενο της κλάσης και χρησιμοποιήστε οποιαδήποτε από τις διαθέσιμες μεθόδους που βρίσκονται στην `Scanner`τεκμηρίωση της κλάσης. Στο παράδειγμά μας, θα χρησιμοποιήσουμε τη `nextLine()`μέθοδο, η οποία χρησιμοποιείται για την ανάγνωση συμβολοσειρών:

Παράδειγμα Αποκτήστε τον δικό σας διακομιστή Java

```
import java.util.Scanner;  // Import the Scanner class


class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);  // Create a Scanner object
    System.out.println("Enter username");


    String userName = myObj.nextLine();  // Read user input
    System.out.println("Username is: " + userName);  // Output user input
  }
}
```

Τύποι εισόδου

Στο παραπάνω παράδειγμα, χρησιμοποιήσαμε τη `nextLine()`μέθοδο, η οποία χρησιμοποιείται για την ανάγνωση συμβολοσειρών. Για να διαβάσετε άλλους τύπους, δείτε τον παρακάτω πίνακα:

Method	Description
`nextBoolean()`	Reads a `boolean` value from the user
`nextByte()`	Reads a `byte` value from the user
`nextDouble()`	Reads a `double` value from the user
`nextFloat()`	Reads a `float` value from the user
`nextInt()`	Reads a `int` value from the user
`nextLine()`	Reads a `String` value from the user
`nextLong()`	Reads a `long` value from the user
`nextShort()`	Reads a `short` value from the user

Στο παρακάτω παράδειγμα, χρησιμοποιούμε διαφορετικές μεθόδους για την ανάγνωση δεδομένων διαφόρων τύπων:

Παράδειγμα

```java
import java.util.Scanner;


class Main {
  public static void main(String[] args) {
    Scanner myObj = new Scanner(System.in);


    System.out.println("Enter name, age and salary:");


    // String input
    String name = myObj.nextLine();


    // Numerical input
    int age = myObj.nextInt();
    double salary = myObj.nextDouble();


    // Output input by user
    System.out.println("Name: " + name);
    System.out.println("Age: " + age);
    System.out.println("Salary: " + salary);
  }
}
```

Σημείωση: Εάν εισάγετε λάθος είσοδο (π.χ. κείμενο σε μια αριθμητική είσοδο), θα λάβετε ένα μήνυμα εξαίρεσης/λάθους (όπως "InputMismatchException").

Java Date and Time

Ημερομηνίες Java

Η Java δεν έχει ενσωματωμένη κλάση Date, αλλά μπορούμε να εισαγάγουμε το `java.time` πακέτο για να λειτουργήσει με το API ημερομηνίας και ώρας. Το πακέτο περιλαμβάνει πολλά μαθήματα ημερομηνίας και ώρας. Για παράδειγμα:

Class	Description
`LocalDate`	Represents a date (year, month, day (yyyy-MM-dd))
`LocalTime`	Represents a time (hour, minute, second and nanoseconds (HH-mm-ss-ns))
`LocalDateTime`	Represents both a date and a time (yyyy-MM-dd-HH-mm-ss-ns)
`DateTimeFormatter`	Formatter for displaying and parsing date-time objects

Εμφάνιση της τρέχουσας ημερομηνίας

Για να εμφανίσετε την τρέχουσα ημερομηνία, εισαγάγετε την `java.time.LocalDate`κλάση και χρησιμοποιήστε τη `now()`μέθοδό της:

Παράδειγμα Αποκτήστε τον δικό σας διακομιστή Java

```
import java.time.LocalDate; // import the LocalDate class


public class Main {
  public static void main(String[] args) {
    LocalDate myObj = LocalDate.now(); // Create a date object
    System.out.println(myObj); // Display the current date
  }
}
```

Η έξοδος θα είναι:

```
2025-03-23
```

Εμφάνιση τρέχουσας ώρας

Για να εμφανίσετε την τρέχουσα ώρα (ώρα, λεπτά, δευτερόλεπτα και νανοδευτερόλεπτα), εισαγάγετε την `java.time.LocalTime`κλάση και χρησιμοποιήστε τη `now()`μέθοδό της:

Παράδειγμα

```
import java.time.LocalTime; // import the LocalTime class


public class Main {

  public static void main(String[] args) {

    LocalTime myObj = LocalTime.now();

    System.out.println(myObj);

  }

}
```

Αυτό το παράδειγμα εμφανίζει την τοπική ώρα του διακομιστή, η οποία μπορεί να διαφέρει από την τοπική σας ώρα:

```
18:32:00.691040
```

Εμφάνιση της τρέχουσας ημερομηνίας και ώρας

Για να εμφανίσετε την τρέχουσα ημερομηνία και ώρα, εισαγάγετε την `java.time.LocalDateTime`κλάση και χρησιμοποιήστε τη `now()`μέθοδό της:

Παράδειγμα

```
import java.time.LocalDateTime; // import the LocalDateTime class


public class Main {

  public static void main(String[] args) {

    LocalDateTime myObj = LocalDateTime.now();

    System.out.println(myObj);
```

```
  }
}
```

Η έξοδος θα είναι κάπως έτσι:

```
2025-03-23T18:32:00.691736
```

Μορφοποίηση ημερομηνίας και ώρας

Το "T" στο παραπάνω παράδειγμα χρησιμοποιείται για να διαχωρίσει την ημερομηνία από την ώρα. Μπορείτε να χρησιμοποιήσετε την `DateTimeFormatter`κλάση με τη `ofPattern()`μέθοδο στο ίδιο πακέτο για να μορφοποιήσετε ή να αναλύσετε αντικείμενα ημερομηνίας-ώρας. Το ακόλουθο παράδειγμα θα αφαιρέσει τόσο το "T" και τα νανοδευτερόλεπτα από την ημερομηνία-ώρα:

Παράδειγμα

```
import java.time.LocalDateTime; // Import the LocalDateTime class

import java.time.format.DateTimeFormatter; // Import the DateTimeFormatter
class


public class Main {

  public static void main(String[] args) {

    LocalDateTime myDateObj = LocalDateTime.now();

    System.out.println("Before formatting: " + myDateObj);

    DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("dd-MM-yyyy
HH:mm:ss");


    String formattedDate = myDateObj.format(myFormatObj);

    System.out.println("After formatting: " + formattedDate);

  }

}
```

Η έξοδος θα είναι:

```
Before Formatting: 2025-03-23T18:32:00.692140
After Formatting: 23-03-2025 18:32:00
```

Η `ofPattern()`μέθοδος δέχεται κάθε είδους τιμές, εάν θέλετε να εμφανίσετε την ημερομηνία και την ώρα σε διαφορετική μορφή. Για παράδειγμα:

Value	Example	Tryit
yyyy-MM-dd	"1988-09-29"	Try it »
dd/MM/yyyy	"29/09/1988"	Try it »
dd-MMM-yyyy	"29-Sep-1988"	Try it »
E, MMM dd yyyy	"Thu, Sep 29 1988"	

Java ArrayList

Η `ArrayList`κλάση είναι ένας πίνακας με δυνατότητα αλλαγής μεγέθους , ο οποίος μπορεί να βρεθεί στο `java.util`πακέτο.

Η διαφορά μεταξύ ενός ενσωματωμένου πίνακα και ενός πίνακα `ArrayList`σε Java, είναι ότι το μέγεθος ενός πίνακα δεν μπορεί να τροποποιηθεί (αν θέλετε να προσθέσετε ή να αφαιρέσετε στοιχεία σε/από έναν πίνακα, πρέπει να δημιουργήσετε έναν νέο). Ενώ τα στοιχεία μπορούν να προστεθούν και να αφαιρεθούν από ένα `ArrayList`όποτε θέλετε. Η σύνταξη είναι επίσης ελαφρώς διαφορετική:

Παράδειγμα

Δημιουργήστε ένα `ArrayList`αντικείμενο που ονομάζεται **αυτοκίνητα** που θα αποθηκεύει συμβολοσειρές:

```
import java.util.ArrayList; // import the ArrayList class



ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList
object
```

Εάν δεν ξέρετε τι είναι ένα πακέτο, διαβάστε τον οδηγό μας για τα πακέτα Java .

Προσθήκη αντικειμένων

Η `ArrayList`τάξη έχει πολλές χρήσιμες μεθόδους. Για παράδειγμα, για να προσθέσετε στοιχεία στη λίστα, χρησιμοποιήστε τη `add()`μέθοδο:

Παράδειγμα

```
import java.util.ArrayList;


public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
  }
}
```

Μπορείτε επίσης να προσθέσετε ένα στοιχείο σε μια καθορισμένη θέση ανατρέχοντας στον αριθμό ευρετηρίου:

Παράδειγμα

```
import java.util.ArrayList;


public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
```

```
    cars.add("Ford");


    cars.add(0, "Mazda"); // Insert element at the beginning of the list (0)


    System.out.println(cars);
  }
}
```

Θυμηθείτε: Τα ευρετήρια του πίνακα ξεκινούν με 0: Το [0] είναι το πρώτο στοιχείο. Το [1] είναι το δεύτερο στοιχείο κ.λπ.

Πρόσβαση σε ένα στοιχείο

Για να αποκτήσετε πρόσβαση σε ένα στοιχείο στο `ArrayList`, χρησιμοποιήστε τη `get()`μέθοδο και ανατρέξτε στον αριθμό ευρετηρίου:

Παράδειγμα

```
cars.get(0);
```

Αλλάξτε ένα Στοιχείο

Για να τροποποιήσετε ένα στοιχείο, χρησιμοποιήστε τη `set()`μέθοδο και ανατρέξτε στον αριθμό ευρετηρίου:

Παράδειγμα

```
cars.set(0, "Opel");
```

Αφαίρεση αντικειμένου

Για να αφαιρέσετε ένα στοιχείο, χρησιμοποιήστε τη `remove()`μέθοδο και ανατρέξτε στον αριθμό ευρετηρίου:

Παράδειγμα

```
cars.remove(0);
```

Για να αφαιρέσετε όλα τα στοιχεία στο `ArrayList`, χρησιμοποιήστε τη `clear()`μέθοδο:

Παράδειγμα

```
cars.clear();
```

Μέγεθος ArrayList

Για να μάθετε πόσα στοιχεία έχει μια ArrayList, χρησιμοποιήστε τη `size`μέθοδο:

Παράδειγμα

```
cars.size();
```

Βρόχος μέσω μιας ArrayList

Κάντε βρόχο στα στοιχεία του an `ArrayList`με έναν `for`βρόχο και χρησιμοποιήστε τη `size()`μέθοδο για να καθορίσετε πόσες φορές θα εκτελείται ο βρόχος:

Παράδειγμα

```
public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    for (int i = 0; i < cars.size(); i++) {
      System.out.println(cars.get(i));
    }
  }
}
```

Μπορείτε επίσης να κάνετε βρόχο μέσω ενός `ArrayList`με τον βρόχο **για κάθε** :

Παράδειγμα

```
public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    for (String i : cars) {
      System.out.println(i);
    }
  }
}
```

Άλλα είδη

Τα στοιχεία σε μια ArrayList είναι στην πραγματικότητα αντικείμενα. Στα παραπάνω παραδείγματα δημιουργήσαμε στοιχεία (αντικείμενα) τύπου "String". Θυμηθείτε ότι μια συμβολοσειρά στην Java είναι ένα αντικείμενο (όχι ένας πρωτόγονος τύπος). Για να χρησιμοποιήσετε άλλους τύπους, όπως int, πρέπει να καθορίσετε μια ισοδύναμη κλάση περιτυλίγματος : `Integer`. Για άλλους πρωτόγονους τύπους, χρησιμοποιήστε: `Boolean`για boolean, `Character`για char, `Double`για double, κ.λπ.

Παράδειγμα

Δημιουργήστε ένα `ArrayList`για αποθήκευση αριθμών (προσθέστε στοιχεία τύπου `Integer`):

```
import java.util.ArrayList;


public class Main {
  public static void main(String[] args) {
```

```
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(10);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(25);
    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
}
```

Ταξινόμηση μιας ArrayList

Μια άλλη χρήσιμη κλάση στο `java.util`πακέτο είναι η `Collections`κλάση, η οποία περιλαμβάνει τη `sort()`μέθοδο ταξινόμησης λιστών αλφαβητικά ή αριθμητικά:

Παράδειγμα

Ταξινόμηση ArrayList of strings:

```
import java.util.ArrayList;
import java.util.Collections;  // Import the Collections class

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    Collections.sort(cars);  // Sort cars
    for (String i : cars) {
      System.out.println(i);
```

```
    }
  }
}
```

Παράδειγμα

Ταξινόμηση ArrayList ακέραιων αριθμών:

```
import java.util.ArrayList;
import java.util.Collections;  // Import the Collections class

public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(33);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(34);
    myNumbers.add(8);
    myNumbers.add(12);

    Collections.sort(myNumbers);  // Sort myNumbers

    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
}
```

Java LinkedList

Στο προηγούμενο κεφάλαιο, μάθατε για την ArrayListτάξη. Η LinkedListτάξη είναι σχεδόν ίδια με την ArrayList:

Παράδειγμα

```
// Import the LinkedList class

import java.util.LinkedList;


public class Main {

  public static void main(String[] args) {

    LinkedList<String> cars = new LinkedList<String>();

    cars.add("Volvo");

    cars.add("BMW");

    cars.add("Ford");

    cars.add("Mazda");

    System.out.println(cars);

  }

}
```

ArrayList εναντίον LinkedList

Η LinkedListκλάση είναι μια συλλογή που μπορεί να περιέχει πολλά αντικείμενα του ίδιου τύπου, όπως ακριβώς το ArrayList.

Η LinkedListκλάση έχει όλες τις ίδιες μεθόδους με την ArrayListκλάση επειδή και οι δύο υλοποιούν τη Listδιεπαφή. Αυτό σημαίνει ότι μπορείτε να προσθέσετε στοιχεία, να αλλάξετε στοιχεία, να αφαιρέσετε στοιχεία και να διαγράψετε τη λίστα με τον ίδιο τρόπο.

Ωστόσο, ενώ η ArrayListκλάση και η LinkedListκλάση μπορούν να χρησιμοποιηθούν με τον ίδιο τρόπο, έχουν κατασκευαστεί πολύ διαφορετικά.

Πώς λειτουργεί το ArrayList

Η ArrayListτάξη έχει έναν κανονικό πίνακα μέσα της. Όταν προστίθεται ένα στοιχείο, τοποθετείται στον πίνακα. Εάν ο πίνακας δεν είναι αρκετά μεγάλος, δημιουργείται ένας νέος, μεγαλύτερος πίνακας για να αντικαταστήσει τον παλιό και ο παλιός αφαιρείται.

Πώς λειτουργεί το LinkedList

Αποθηκεύει `LinkedList`τα προϊόντα του σε «κοντέινερ». Η λίστα έχει έναν σύνδεσμο προς το πρώτο κοντέινερ και κάθε κοντέινερ έχει έναν σύνδεσμο προς το επόμενο κοντέινερ στη λίστα. Για να προσθέσετε ένα στοιχείο στη λίστα, το στοιχείο τοποθετείται σε ένα νέο κοντέινερ και αυτό το κοντέινερ συνδέεται με ένα από τα άλλα κοντέινερ της λίστας.

Πότε να χρησιμοποιείται

Χρησιμοποιήστε ένα `ArrayList`για αποθήκευση και πρόσβαση σε δεδομένα και `LinkedList` για χειρισμό δεδομένων.

Μέθοδοι LinkedList

Για πολλές περιπτώσεις, `ArrayList`είναι πιο αποτελεσματικό, καθώς είναι συνηθισμένο να απαιτείται πρόσβαση σε τυχαία στοιχεία στη λίστα, αλλά `LinkedList`παρέχει πολλές μεθόδους για να κάνετε ορισμένες λειτουργίες πιο αποτελεσματικά:

Method	Description	Try it
addFirst()	Adds an item to the beginning of the list	Try it »
addLast()	Add an item to the end of the list	Try it »
removeFirst()	Remove an item from the beginning of the list	Try it »
removeLast()	Remove an item from the end of the list	Try it »
getFirst()	Get the item at the beginning of the list	Try it »

getLast()	Get the item at the end of the list

Java List Sorting

Στα προηγούμενα κεφάλαια, μάθατε πώς να χρησιμοποιείτε δύο δημοφιλείς λίστες στην Java: ArrayListκαι LinkedList, που βρίσκονται στο java.utilπακέτο.

Μια άλλη χρήσιμη κλάση στο java.utilπακέτο είναι η Collectionsκλάση, η οποία περιλαμβάνει τη sort()μέθοδο ταξινόμησης λιστών αλφαβητικά ή αριθμητικά.

Ταξινόμηση μιας ArrayList

Ταξινομήστε μια ArrayList από συμβολοσειρές αλφαβητικά σε αύξουσα σειρά:

Παράδειγμα

```
import java.util.ArrayList;
import java.util.Collections;  // Import the Collections class

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");

    Collections.sort(cars);  // Sort cars

    for (String i : cars) {
```

```
      System.out.println(i);
    }
  }
}
```

Ταξινομήστε μια ArrayList ακεραίων αριθμητικά σε αύξουσα σειρά:

Παράδειγμα

```
import java.util.ArrayList;
import java.util.Collections;  // Import the Collections class

public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(33);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(34);
    myNumbers.add(8);
    myNumbers.add(12);

    Collections.sort(myNumbers);  // Sort myNumbers

    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
}
```

Αντιστρέψτε τη σειρά

Μπορείτε επίσης να ταξινομήσετε μια λίστα με αντίστροφη σειρά, χρησιμοποιώντας τη `reverseOrder()` μέθοδο.

Στο παρακάτω παράδειγμα, ταξινομούμε μια ArrayList από Strings αλφαβητικά με αντίστροφη/φθίνουσα σειρά:

Παράδειγμα

```
import java.util.ArrayList;
import java.util.Collections;  // Import the Collections class


public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");

    Collections.sort(cars, Collections.reverseOrder()); // Sort cars

    for (String i : cars) {
      System.out.println(i);
    }
  }
}
```

Ταξινομήστε μια ArrayList ακεραίων αριθμητικά με αντίστροφη/φθίνουσα σειρά:

Παράδειγμα

```
import java.util.ArrayList;
import java.util.Collections;  // Import the Collections class


public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(33);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(34);
    myNumbers.add(8);
    myNumbers.add(12);

    Collections.sort(myNumbers, Collections.reverseOrder()); // Sort
myNumbers

    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
```

Java HashMap

Στο ArrayListκεφάλαιο, μάθατε ότι οι Πίνακες αποθηκεύουν στοιχεία ως μια ταξινομημένη συλλογή και πρέπει να έχετε πρόσβαση σε αυτά με έναν αριθμό ευρετηρίου (inttύπος). Ωστόσο HashMap, αποθηκεύστε τα στοιχεία σε ζεύγη " **κλειδί** / **τιμή** " και μπορείτε να τα έχετε πρόσβαση από ένα ευρετήριο άλλου τύπου (π.χ. a String).

Ένα αντικείμενο χρησιμοποιείται ως κλειδί (ευρετήριο) σε ένα άλλο αντικείμενο (τιμή). Μπορεί να αποθηκεύσει διαφορετικούς τύπους: `String`κλειδιά και `Integer`τιμές, ή τον ίδιο τύπο, όπως: `String`κλειδιά και `String`τιμές:

Παράδειγμα

Δημιουργήστε ένα `HashMap`αντικείμενο που ονομάζεται **capitalCities** που θα αποθηκεύει `String` **κλειδιά** και `String` **τιμές** :

```
import java.util.HashMap; // import the HashMap class


HashMap<String, String> capitalCities = new HashMap<String, String>();
```

Προσθήκη αντικειμένων

Η `HashMap`τάξη έχει πολλές χρήσιμες μεθόδους. Για παράδειγμα, για να προσθέσετε στοιχεία σε αυτό, χρησιμοποιήστε τη `put()`μέθοδο:

Παράδειγμα

```
// Import the HashMap class
import java.util.HashMap;


public class Main {
  public static void main(String[] args) {
    // Create a HashMap object called capitalCities
    HashMap<String, String> capitalCities = new HashMap<String, String>();


    // Add keys and values (Country, City)
    capitalCities.put("England", "London");
    capitalCities.put("Germany", "Berlin");
    capitalCities.put("Norway", "Oslo");
    capitalCities.put("USA", "Washington DC");
    System.out.println(capitalCities);
```

```
  }
}
```

Πρόσβαση σε ένα στοιχείο

Για να αποκτήσετε πρόσβαση σε μια τιμή στο `HashMap`, χρησιμοποιήστε τη `get()`μέθοδο και ανατρέξτε στο κλειδί της:

Παράδειγμα

```
capitalCities.get("England");
```

Αφαίρεση αντικειμένου

Για να αφαιρέσετε ένα στοιχείο, χρησιμοποιήστε τη `remove()`μέθοδο και ανατρέξτε στο κλειδί:

Παράδειγμα

```
capitalCities.remove("England");
```

Για να αφαιρέσετε όλα τα στοιχεία, χρησιμοποιήστε τη `clear()`μέθοδο:

Παράδειγμα

```
capitalCities.clear();
```

Μέγεθος HashMap

Για να μάθετε πόσα αντικείμενα υπάρχουν, χρησιμοποιήστε τη `size()`μέθοδο:

Παράδειγμα

```
capitalCities.size();
```

Βρόχος μέσω ενός HashMap

Κάντε βρόχο στα στοιχεία του a `HashMap`με έναν βρόχο **για κάθε .**

Σημείωση: Χρησιμοποιήστε τη `keySet()`μέθοδο εάν θέλετε μόνο τα κλειδιά και χρησιμοποιήστε τη `values()`μέθοδο εάν θέλετε μόνο τις τιμές:

Παράδειγμα

```
// Print keys
for (String i : capitalCities.keySet()) {
  System.out.println(i);
}
```

Παράδειγμα

```
// Print values
for (String i : capitalCities.values()) {
  System.out.println(i);
}
```

Παράδειγμα

```
// Print keys and values
for (String i : capitalCities.keySet()) {
  System.out.println("key: " + i + " value: " + capitalCities.get(i));
}
```

Άλλα είδη

Τα κλειδιά και οι τιμές σε ένα HashMap είναι στην πραγματικότητα αντικείμενα. Στα παραπάνω παραδείγματα χρησιμοποιήσαμε αντικείμενα τύπου "String". Θυμηθείτε ότι μια συμβολοσειρά στην Java είναι ένα αντικείμενο (όχι ένας πρωτόγονος τύπος). Για να χρησιμοποιήσετε άλλους τύπους, όπως int, πρέπει να καθορίσετε μια ισοδύναμη κλάση περιτυλίγματος : `Integer`. Για άλλους πρωτόγονους τύπους, χρησιμοποιήστε: `Boolean`για boolean, `Character`για char, `Double`για double, κ.λπ.

Παράδειγμα

Δημιουργήστε ένα `HashMap`αντικείμενο που ονομάζεται **άτομα** που θα αποθηκεύει `String` **κλειδιά** και `Integer` **τιμές** :

```
// Import the HashMap class
import java.util.HashMap;
```

```
public class Main {
  public static void main(String[] args) {

    // Create a HashMap object called people
    HashMap<String, Integer> people = new HashMap<String, Integer>();

    // Add keys and values (Name, Age)
    people.put("John", 32);
    people.put("Steve", 30);
    people.put("Angie", 33);

    for (String i : people.keySet()) {
      System.out.println("key: " + i + " value: " + people.get(i));
    }
  }
}
```

Java HashSet

Ένα HashSet είναι μια συλλογή αντικειμένων όπου κάθε αντικείμενο είναι μοναδικό και βρίσκεται στη `java.util` συσκευασία:

Παράδειγμα

Δημιουργήστε ένα `HashSet`αντικείμενο που ονομάζεται **αυτοκίνητα** που θα αποθηκεύει συμβολοσειρές:

```
import java.util.HashSet; // Import the HashSet class

HashSet<String> cars = new HashSet<String>();
```

Προσθήκη αντικειμένων

Η `HashSet`τάξη έχει πολλές χρήσιμες μεθόδους. Για παράδειγμα, για να προσθέσετε στοιχεία σε αυτό, χρησιμοποιήστε τη `add()`μέθοδο:

Παράδειγμα

```
// Import the HashSet class

import java.util.HashSet;


public class Main {

  public static void main(String[] args) {

    HashSet<String> cars = new HashSet<String>();

    cars.add("Volvo");

    cars.add("BMW");

    cars.add("Ford");

    cars.add("BMW");

    cars.add("Mazda");

    System.out.println(cars);

  }

}
```

Σημείωση: Στο παραπάνω παράδειγμα, παρόλο που η BMW έχει προστεθεί δύο φορές, εμφανίζεται μόνο μία φορά στο σετ, επειδή κάθε στοιχείο σε ένα σετ πρέπει να είναι μοναδικό.

Ελέγξτε εάν υπάρχει ένα στοιχείο

Για να ελέγξετε εάν ένα στοιχείο υπάρχει σε ένα HashSet, χρησιμοποιήστε τη `contains()`μέθοδο:

Παράδειγμα

```
cars.contains("Mazda");
```

Αφαίρεση αντικειμένου

Για να αφαιρέσετε ένα στοιχείο, χρησιμοποιήστε τη `remove()`μέθοδο:

Παράδειγμα

```
cars.remove("Volvo");
```

Για να αφαιρέσετε όλα τα στοιχεία, χρησιμοποιήστε τη `clear()`μέθοδο:

Παράδειγμα

```
cars.clear();
```

Μέγεθος HashSet

Για να μάθετε πόσα αντικείμενα υπάρχουν, χρησιμοποιήστε τη `size`μέθοδο:

Παράδειγμα

```
cars.size();
```

Βρόχος μέσω ενός HashSet

Κάντε βρόχο στα στοιχεία του an `HashSet`με έναν βρόχο **για κάθε :**

Παράδειγμα

```
for (String i : cars) {
  System.out.println(i);
}
```

Άλλα είδη

Τα στοιχεία σε ένα HashSet είναι στην πραγματικότητα αντικείμενα. Στα παραπάνω παραδείγματα δημιουργήσαμε αντικείμενα (αντικείμενα) τύπου "String". Θυμηθείτε ότι μια συμβολοσειρά στην Java είναι ένα αντικείμενο (όχι ένας πρωτόγονος τύπος). Για να χρησιμοποιήσετε άλλους τύπους, όπως int, πρέπει να καθορίσετε μια ισοδύναμη κλάση περιτυλίγματος : `Integer`. Για άλλους πρωτόγονους τύπους, χρησιμοποιήστε: `Boolean`για boolean, `Character`για char, `Double`για double, κ.λπ.

Παράδειγμα

Χρησιμοποιήστε ένα `HashSet`που αποθηκεύει `Integer`αντικείμενα:

```
import java.util.HashSet;


public class Main {
  public static void main(String[] args) {


    // Create a HashSet object called numbers
    HashSet<Integer> numbers = new HashSet<Integer>();


    // Add values to the set
    numbers.add(4);
    numbers.add(7);
    numbers.add(8);


    // Show which numbers between 1 and 10 are in the set
    for(int i = 1; i <= 10; i++) {
      if(numbers.contains(i)) {
        System.out.println(i + " was found in the set.");
      } else {
        System.out.println(i + " was not found in the set.");
      }
    }
  }
```

```
}
```

Java Iterator

Το An `Iterator`είναι ένα αντικείμενο που μπορεί να χρησιμοποιηθεί για επαναφορά συλλογών, όπως ArrayList και HashSet . Ονομάζεται "επαναλήπτης" επειδή "επαναλαμβάνω" είναι ο τεχνικός όρος για το βρόχο.

Για να χρησιμοποιήσετε ένα Iterator, πρέπει να το εισαγάγετε από το `java.util`πακέτο.

Λήψη ενός Iterator

Η `iterator()`μέθοδος μπορεί να χρησιμοποιηθεί για τη λήψη μιας συλλογής `Iterator`για οποιαδήποτε συλλογή:

Παράδειγμα

```
// Import the ArrayList class and the Iterator class
import java.util.ArrayList;
import java.util.Iterator;

public class Main {
  public static void main(String[] args) {

    // Make a collection
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");

    // Get the iterator
```

```
    Iterator<String> it = cars.iterator();


    // Print the first item
    System.out.println(it.next());
  }
}
```

Looping Through a Collection

Για να κάνετε βρόχο μέσω μιας συλλογής, χρησιμοποιήστε τις μεθόδους `hasNext()`και `next()`του `Iterator`:

Παράδειγμα

```
while(it.hasNext()) {
  System.out.println(it.next());
}
```

Αφαίρεση αντικειμένων από μια συλλογή

Οι επαναλήψεις έχουν σχεδιαστεί για να αλλάζουν εύκολα τις συλλογές στις οποίες πραγματοποιούν βρόχο. Η `remove()`μέθοδος μπορεί να αφαιρέσει στοιχεία από μια συλλογή κατά την επαναφορά.

Παράδειγμα

Χρησιμοποιήστε έναν επαναλήπτη για να αφαιρέσετε αριθμούς μικρότερους από 10 από μια συλλογή:

```
import java.util.ArrayList;
import java.util.Iterator;


public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> numbers = new ArrayList<Integer>();
    numbers.add(12);
    numbers.add(8);
```

```
    numbers.add(2);
    numbers.add(23);
    Iterator<Integer> it = numbers.iterator();
    while(it.hasNext()) {
      Integer i = it.next();
      if(i < 10) {
        it.remove();
      }
    }
    System.out.println(numbers);
  }
}
```

Σημείωση: Η προσπάθεια κατάργησης στοιχείων χρησιμοποιώντας έναν **βρόχο for** ή έναν **βρόχο για κάθε** δεν θα λειτουργούσε σωστά επειδή η συλλογή αλλάζει μέγεθος την ίδια στιγμή που ο κώδικας προσπαθεί να επαναφέρει.

Java Wrapper Classes

Οι κλάσεις Wrapper παρέχουν έναν τρόπο χρήσης πρωτόγονων τύπων δεδομένων (`int`, `boolean`, κ.λπ..) ως αντικείμενα.

Ο παρακάτω πίνακας δείχνει τον αρχέγονο τύπο και την ισοδύναμη κατηγορία περιτυλίγματος:

Primitive Data Type	Wrapper Class
byte	Byte

short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

Μερικές φορές πρέπει να χρησιμοποιείτε κλάσεις περιτυλίγματος, για παράδειγμα όταν εργάζεστε με αντικείμενα συλλογής, όπως `ArrayList`, όπου δεν μπορούν να χρησιμοποιηθούν πρωτόγονοι τύποι (η λίστα μπορεί να αποθηκεύσει μόνο αντικείμενα):

Παράδειγμα

```
ArrayList<int> myNumbers = new ArrayList<int>(); // Invalid

ArrayList<Integer> myNumbers = new ArrayList<Integer>(); // Valid
```

Δημιουργία αντικειμένων Wrapper

Για να δημιουργήσετε ένα αντικείμενο wrapper, χρησιμοποιήστε την κλάση wrapper αντί για τον αρχέγονο τύπο. Για να λάβετε την τιμή, μπορείτε απλώς να εκτυπώσετε το αντικείμενο:

Παράδειγμα

```
public class Main {
  public static void main(String[] args) {
    Integer myInt = 5;
    Double myDouble = 5.99;
    Character myChar = 'A';
    System.out.println(myInt);
    System.out.println(myDouble);
    System.out.println(myChar);
  }
}
```

Δεδομένου ότι τώρα εργάζεστε με αντικείμενα, μπορείτε να χρησιμοποιήσετε ορισμένες μεθόδους για να λάβετε πληροφορίες σχετικά με το συγκεκριμένο αντικείμενο.

Για παράδειγμα, χρησιμοποιούνται οι ακόλουθες μέθοδοι για να ληφθεί η τιμή που σχετίζεται με το αντίστοιχο αντικείμενο περιτυλίγματος: `intValue()`, `byteValue()`, `shortValue()`, `longValue()`, `floatValue()`, `doubleValue()`, `charValue()`, `booleanValue()`.

Αυτό το παράδειγμα θα δώσει το ίδιο αποτέλεσμα με το παραπάνω παράδειγμα:

Παράδειγμα

```
public class Main {
  public static void main(String[] args) {
    Integer myInt = 5;
    Double myDouble = 5.99;
    Character myChar = 'A';
    System.out.println(myInt.intValue());
    System.out.println(myDouble.doubleValue());
    System.out.println(myChar.charValue());
  }
}
```

Μια άλλη χρήσιμη μέθοδος είναι η `toString()`μέθοδος, η οποία χρησιμοποιείται για τη μετατροπή αντικειμένων περιτυλίγματος σε συμβολοσειρές.

Στο παρακάτω παράδειγμα, μετατρέπουμε το an `Integer`σε ένα `String`και χρησιμοποιούμε τη `length()`μέθοδο της `String`κλάσης για να εξάγουμε το μήκος της "string":

Παράδειγμα

```
public class Main {

  public static void main(String[] args) {

    Integer myInt = 100;

    String myString = myInt.toString();

    System.out.println(myString.length());

  }

}
```

Java Exceptions - Try...Catch

Εξαιρέσεις Java

Κατά την εκτέλεση του κώδικα Java, μπορεί να προκύψουν διαφορετικά σφάλματα: σφάλματα κωδικοποίησης που έγιναν από τον προγραμματιστή, σφάλματα λόγω λανθασμένης εισαγωγής ή άλλα απρόβλεπτα πράγματα.

Όταν παρουσιαστεί ένα σφάλμα, η Java κανονικά θα σταματήσει και θα δημιουργήσει ένα μήνυμα σφάλματος. Ο τεχνικός όρος για αυτό είναι: Η Java θα ρίξει μια **εξαίρεση** (ρίξει ένα σφάλμα).

Java δοκιμάστε και πιάστε

Η tryδήλωση σάς επιτρέπει να ορίσετε ένα μπλοκ κώδικα που θα ελεγχθεί για σφάλματα κατά την εκτέλεσή του.

Η catchδήλωση σάς επιτρέπει να ορίσετε ένα μπλοκ κώδικα που θα εκτελεστεί, εάν παρουσιαστεί σφάλμα στο μπλοκ δοκιμής.

Οι λέξεις-κλειδιά tryκαι catchέρχονται σε ζεύγη:

Σύνταξη

```
try {
  //  Block of code to try
}
catch(Exception e) {
  //  Block of code to handle errors
}
```

Εξετάστε το ακόλουθο παράδειγμα:

Αυτό θα δημιουργήσει ένα σφάλμα, επειδή **το myNumbers[10]** δεν υπάρχει.

```
public class Main {
  public static void main(String[ ] args) {
    int[] myNumbers = {1, 2, 3};
    System.out.println(myNumbers[10]); // error!
  }
}
```

Η έξοδος θα είναι κάπως έτσι:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
10
        at Main.main(Main.java:4)
```

Σημείωση: ArrayIndexOutOfBoundsExceptionεμφανίζεται όταν προσπαθείτε να αποκτήσετε πρόσβαση σε έναν αριθμό ευρετηρίου που δεν υπάρχει.

Εάν παρουσιαστεί κάποιο σφάλμα, μπορούμε να χρησιμοποιήσουμε try...catchγια να συλλάβουμε το σφάλμα και να εκτελέσουμε κάποιο κώδικα για να το χειριστούμε:

Παράδειγμα

```
public class Main {
  public static void main(String[ ] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    }
  }
}
```

Η έξοδος θα είναι:

```
Something went wrong.
```

Τελικά

Η `finally`δήλωση σάς επιτρέπει να εκτελέσετε κώδικα, μετά από `try...catch`, ανεξάρτητα από το αποτέλεσμα:

Παράδειγμα

```
public class Main {
  public static void main(String[] args) {
    try {
      int[] myNumbers = {1, 2, 3};
      System.out.println(myNumbers[10]);
    } catch (Exception e) {
      System.out.println("Something went wrong.");
    } finally {
      System.out.println("The 'try catch' is finished.");
    }
```

```
  }

}
```

Η έξοδος θα είναι:

```
Something went wrong.
The 'try catch' is finished.
```

Η λέξη-κλειδί ρίχνω

Η throwδήλωση σάς επιτρέπει να δημιουργήσετε ένα προσαρμοσμένο σφάλμα.

Η throwδήλωση χρησιμοποιείται μαζί με έναν **τύπο εξαίρεσης** . Υπάρχουν πολλοί τύποι εξαιρέσεων διαθέσιμοι στην
Java: ArithmeticException, FileNotFoundException, ArrayIndexOutOfBoundsException, SecurityException, κ.λπ.:

Παράδειγμα

Βάλτε μια εξαίρεση εάν **η ηλικία** είναι κάτω των 18 ετών (εκτύπωση "Δεν επιτρέπεται η πρόσβαση"). Εάν η ηλικία είναι 18 ετών και άνω, εκτυπώστε "Παρέχεται πρόσβαση"

```
public class Main {

  static void checkAge(int age) {

    if (age < 18) {

      throw new ArithmeticException("Access denied - You must be at least 18 years old.");

    }

    else {

      System.out.println("Access granted - You are old enough!");

    }

  }


  public static void main(String[] args) {
```

```
    checkAge(15); // Set age to 15 (which is below 18...)
  }
}
```

Η έξοδος θα είναι:

```
Exception in thread "main" java.lang.ArithmeticException: Access
denied - You must be at least 18 years old.
        at Main.checkAge(Main.java:4)
        at Main.main(Main.java:12)
```

Εάν **η ηλικία** ήταν 20, **δεν** θα είχατε εξαίρεση:

Παράδειγμα

```
checkAge(20);
```

Η έξοδος θα είναι:

```
Access granted - You are old enough!
```

Java Regular Expressions

Μια κανονική έκφραση είναι μια ακολουθία χαρακτήρων που σχηματίζει ένα μοτίβο αναζήτησης. Όταν αναζητάτε δεδομένα σε ένα κείμενο, μπορείτε να χρησιμοποιήσετε αυτό το μοτίβο αναζήτησης για να περιγράψετε αυτό που αναζητάτε.

Μια τυπική έκφραση μπορεί να είναι ένας μεμονωμένος χαρακτήρας ή ένα πιο περίπλοκο μοτίβο.

Οι τυπικές εκφράσεις μπορούν να χρησιμοποιηθούν για την εκτέλεση όλων των τύπων εργασιών **αναζήτησης κειμένου** και **αντικατάστασης κειμένου** .

Η Java δεν έχει ενσωματωμένη κλάση Regular Expression, αλλά μπορούμε να εισαγάγουμε το `java.util.regex` πακέτο για να λειτουργήσει με κανονικές εκφράσεις. Το πακέτο περιλαμβάνει τις ακόλουθες κατηγορίες:

- `Pattern`Τάξη - Καθορίζει ένα μοτίβο (που θα χρησιμοποιηθεί σε μια αναζήτηση)
- `Matcher`Τάξη - Χρησιμοποιείται για την αναζήτηση του μοτίβου
- `PatternSyntaxException`Κλάση - Υποδεικνύει συντακτικό σφάλμα σε ένα τυπικό μοτίβο έκφρασης

Παράδειγμα

Μάθετε εάν υπάρχουν εμφανίσεις της λέξης "w3schools" σε μια πρόταση:

```
import java.util.regex.Matcher;

import java.util.regex.Pattern;


public class Main {

  public static void main(String[] args) {

    Pattern pattern = Pattern.compile("w3schools",
Pattern.CASE_INSENSITIVE);

    Matcher matcher = pattern.matcher("Visit W3Schools!");

    boolean matchFound = matcher.find();

    if(matchFound) {

      System.out.println("Match found");

    } else {

      System.out.println("Match not found");

    }

  }

}

// Outputs Match found
```

Παράδειγμα Επεξήγηση

Σε αυτό το παράδειγμα, η λέξη "w3schools" αναζητείται σε μια πρόταση.

Αρχικά, το μοτίβο δημιουργείται χρησιμοποιώντας τη `Pattern.compile()`μέθοδο. Η πρώτη παράμετρος υποδεικνύει ποιο μοτίβο αναζητείται και η δεύτερη παράμετρος έχει μια σημαία που υποδεικνύει ότι η αναζήτηση δεν πρέπει να γίνεται με διάκριση πεζών-κεφαλαίων. Η δεύτερη παράμετρος είναι προαιρετική.

Η `matcher()`μέθοδος χρησιμοποιείται για την αναζήτηση του μοτίβου σε μια συμβολοσειρά. Επιστρέφει ένα αντικείμενο Match που περιέχει πληροφορίες σχετικά με την αναζήτηση που εκτελέστηκε.

Η `find()`μέθοδος επιστρέφει true αν το μοτίβο βρέθηκε στη συμβολοσειρά και false αν δεν βρέθηκε.

Σημαίες

Οι σημαίες στη `compile()`μέθοδο αλλάζουν τον τρόπο με τον οποίο εκτελείται η αναζήτηση. Εδώ είναι μερικά από αυτά:

- `Pattern.CASE_INSENSITIVE`- Η περίπτωση των γραμμάτων θα αγνοηθεί κατά την εκτέλεση αναζήτησης.
- `Pattern.LITERAL`- Οι ειδικοί χαρακτήρες στο μοτίβο δεν θα έχουν ιδιαίτερο νόημα και θα αντιμετωπίζονται ως συνηθισμένοι χαρακτήρες κατά την εκτέλεση μιας αναζήτησης.
- `Pattern.UNICODE_CASE`- Χρησιμοποιήστε το μαζί με τη `CASE_INSENSITIVE`σημαία για να αγνοήσετε επίσης την περίπτωση των γραμμάτων εκτός του αγγλικού αλφαβήτου

Μοτίβα κανονικής έκφρασης

Η πρώτη παράμετρος της `Pattern.compile()`μεθόδου είναι το μοτίβο. Περιγράφει αυτό που αναζητείται.

Οι αγκύλες χρησιμοποιούνται για την εύρεση μιας σειράς χαρακτήρων:

Expression	Description
[abc]	Find one character from the options between the brackets
[^abc]	Find one character NOT between the brackets
[0-9]	Find one character from the range 0 to 9

Μεταχαρακτήρες

Οι μεταχαρακτήρες είναι χαρακτήρες με ιδιαίτερη σημασία:

Metacharacter	Description
|	Find a match for any one of the patterns separated by | as in: cat|dog|fish
.	Find just one instance of any character
^	Finds a match as the beginning of a string as in: ^Hello
$	Finds a match at the end of the string as in: World$
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Ποσοτικοί δείκτες

Οι ποσοτικοί δείκτες ορίζουν τις ποσότητες:

Quantifier	Description

n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{x}	Matches any string that contains a sequence of *X* *n*'s
n{x,y}	Matches any string that contains a sequence of X to Y *n*'s
n{x,}	Matches any string that contains a sequence of at least X *n*'s

Java Threads

Το Threads επιτρέπει σε ένα πρόγραμμα να λειτουργεί πιο αποτελεσματικά κάνοντας πολλά πράγματα ταυτόχρονα.

Τα νήματα μπορούν να χρησιμοποιηθούν για την εκτέλεση περίπλοκων εργασιών στο παρασκήνιο χωρίς διακοπή του κύριου προγράμματος.

Δημιουργία νήματος

Υπάρχουν δύο τρόποι για να δημιουργήσετε ένα νήμα.

Μπορεί να δημιουργηθεί επεκτείνοντας την `Thread`κλάση και παρακάμπτοντας τη `run()` μέθοδο της:

Επέκταση ΣύνταξηςΑποκτήστε τον δικό σας διακομιστή Java

```
public class Main extends Thread {

  public void run() {

    System.out.println("This code is running in a thread");

  }

}
```

Ένας άλλος τρόπος για να δημιουργήσετε ένα νήμα είναι να εφαρμόσετε τη `Runnable`διεπαφή:

Εφαρμογή Σύνταξης

```
public class Main implements Runnable {

  public void run() {

    System.out.println("This code is running in a thread");

  }

}
```

Τρέχοντα νήματα

Εάν η κλάση επεκτείνει την `Thread`κλάση, το νήμα μπορεί να εκτελεστεί δημιουργώντας μια παρουσία της κλάσης και καλώντας τη `start()`μέθοδο της:

Επέκταση του παραδείγματος

```
public class Main extends Thread {

  public static void main(String[] args) {

    Main thread = new Main();

    thread.start();

    System.out.println("This code is outside of the thread");

  }

  public void run() {

    System.out.println("This code is running in a thread");
```

```
  }
}
```

Εάν η κλάση υλοποιεί τη `Runnable`διεπαφή, το νήμα μπορεί να εκτελεστεί περνώντας μια παρουσία της κλάσης στον `Thread`κατασκευαστή ενός αντικειμένου και στη συνέχεια καλώντας τη `start()`μέθοδο του νήματος:

Παράδειγμα εφαρμογής

```
public class Main implements Runnable {

  public static void main(String[] args) {

    Main obj = new Main();

    Thread thread = new Thread(obj);

    thread.start();

    System.out.println("This code is outside of the thread");

  }

  public void run() {

    System.out.println("This code is running in a thread");

  }

}
```

Διαφορές μεταξύ «επέκτασης» και «υλοποίησης» νημάτων

Η κύρια διαφορά είναι ότι όταν μια κλάση επεκτείνει την κλάση Thread, δεν μπορείτε να επεκτείνετε καμία άλλη κλάση, αλλά εφαρμόζοντας τη διεπαφή Runnable, είναι δυνατή η επέκταση και από μια άλλη κλάση, όπως: class `MyClass extends OtherClass implements Runnable`.

Προβλήματα συγχρονισμού

Επειδή τα νήματα εκτελούνται ταυτόχρονα με άλλα μέρη του προγράμματος, δεν υπάρχει τρόπος να γνωρίζουμε με ποια σειρά θα εκτελεστεί ο κώδικας. Όταν τα νήματα και το κύριο πρόγραμμα διαβάζουν και γράφουν τις ίδιες μεταβλητές, οι τιμές είναι απρόβλεπτες. Τα προβλήματα που προκύπτουν από αυτό ονομάζονται προβλήματα συγχρονισμού.

Παράδειγμα

Ένα παράδειγμα κώδικα όπου η τιμή της μεταβλητής **ποσότητας** είναι απρόβλεπτη:

```
public class Main extends Thread {
  public static int amount = 0;

  public static void main(String[] args) {
    Main thread = new Main();
    thread.start();
    System.out.println(amount);
    amount++;
    System.out.println(amount);
  } }

  public void run() {
    amount++;
```

Για να αποφύγετε προβλήματα ταυτόχρονης χρήσης, είναι καλύτερο να μοιράζεστε όσο το δυνατόν λιγότερα χαρακτηριστικά μεταξύ των νημάτων. Εάν τα χαρακτηριστικά πρέπει να κοινοποιηθούν, μια πιθανή λύση είναι να χρησιμοποιήσετε τη `isAlive()` μέθοδο του νήματος για να ελέγξετε εάν το νήμα έχει τελειώσει πριν χρησιμοποιήσετε οποιαδήποτε χαρακτηριστικά που μπορεί να αλλάξει το νήμα.

Παράδειγμα

Χρησιμοποιήστε `isAlive()`για την αποφυγή προβλημάτων ταυτόχρονης χρήσης:

```
public class Main extends Thread {
  public static int amount = 0;

  public static void main(String[] args) {
    Main thread = new Main();
    thread.start();
    // Wait for the thread to finish
```

```
    while(thread.isAlive()) {
      System.out.println("Waiting...");
    } } }
    // Update amount and print its value
    System.out.println("Main: " + amount);
    amount++;
    System.out.println("Main: " + amount);

  public void run() {
    amount++;
```

Java Lambda Expressions

Οι εκφράσεις λάμδα προστέθηκαν στην Java 8.

Μια έκφραση λάμδα είναι ένα σύντομο μπλοκ κώδικα που λαμβάνει παραμέτρους και επιστρέφει μια τιμή. Οι εκφράσεις λάμδα είναι παρόμοιες με τις μεθόδους, αλλά δεν χρειάζονται όνομα και μπορούν να εφαρμοστούν απευθείας στο σώμα μιας μεθόδου.

Σύνταξη

Η απλούστερη έκφραση λάμδα περιέχει μια παράμετρο και μια έκφραση:

```
parameter -> expression
```

Για να χρησιμοποιήσετε περισσότερες από μία παραμέτρους, τυλίξτε τις σε παρένθεση:

```
(parameter1, parameter2) -> expression
```

Οι εκφράσεις είναι περιορισμένες. Πρέπει να επιστρέψουν αμέσως μια τιμή και δεν μπορούν να περιέχουν μεταβλητές, εκχωρήσεις ή δηλώσεις όπως `if`ή `for`. Για να γίνουν πιο περίπλοκες λειτουργίες, μπορεί να χρησιμοποιηθεί ένα μπλοκ κώδικα με σγουρά στηρίγματα. Εάν η έκφραση λάμδα πρέπει να επιστρέψει μια τιμή, τότε το μπλοκ κώδικα θα πρέπει να έχει μια `return`δήλωση.

```
(parameter1, parameter2) -> { code block }
```

Χρήση εκφράσεων λάμδα

Οι εκφράσεις λάμδα μεταβιβάζονται συνήθως ως παράμετροι σε μια συνάρτηση:

Παράδειγμα

Χρησιμοποιήστε μια έκφραση λάμδα στη μέθοδο `ArrayList`'s `forEach()`για να εκτυπώσετε κάθε στοιχείο στη λίστα:

```
import java.util.ArrayList;


public class Main {

  public static void main(String[] args) {

    ArrayList<Integer> numbers = new ArrayList<Integer>();

    numbers.add(5);

    numbers.add(9);

    numbers.add(8);

    numbers.add(1);

    numbers.forEach( (n) -> { System.out.println(n); } );

  }

}
```

Οι εκφράσεις λάμδα μπορούν να αποθηκευτούν σε μεταβλητές εάν ο τύπος της μεταβλητής είναι μια διεπαφή που έχει μόνο μία μέθοδο. Η έκφραση λάμδα θα πρέπει να έχει τον ίδιο αριθμό παραμέτρων και τον ίδιο τύπο επιστροφής με αυτήν τη μέθοδο. Η Java έχει ενσωματωμένα πολλά από αυτά τα είδη διεπαφών, όπως η `Consumer`διεπαφή (που βρίσκεται στο `java.util`πακέτο) που χρησιμοποιείται από λίστες.

Παράδειγμα

Χρησιμοποιήστε τη διεπαφή της Java `Consumer`για να αποθηκεύσετε μια έκφραση λάμδα σε μια μεταβλητή:

```
import java.util.ArrayList;
import java.util.function.Consumer;


public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> numbers = new ArrayList<Integer>();
    numbers.add(5);
    numbers.add(9);
    numbers.add(8);
    numbers.add(1);
    Consumer<Integer> method = (n) -> { System.out.println(n); };
    numbers.forEach( method );
  }
}
```

Για να χρησιμοποιήσετε μια έκφραση λάμδα σε μια μέθοδο, η μέθοδος θα πρέπει να έχει μια παράμετρο με μια διεπαφή μίας μεθόδου ως τύπο. Η κλήση της μεθόδου της διεπαφής θα εκτελέσει την έκφραση λάμδα:

Παράδειγμα

Δημιουργήστε μια μέθοδο που λαμβάνει μια έκφραση λάμδα ως παράμετρο:

```
interface StringFunction {
  String run(String str);
} } }


public class Main {
  public static void main(String[] args) {
    StringFunction exclaim = (s) -> s + "!";
    StringFunction ask = (s) -> s + "?";
    printFormatted("Hello", exclaim);
    printFormatted("Hello", ask);
```

```
public static void printFormatted(String str, StringFunction format) {
  String result = format.run(str);
  System.out.println(result);
```

Java Advanced Sorting

Java Advanced Sorting

Στο Κεφάλαιο ταξινόμησης λίστας , μάθατε πώς να ταξινομείτε τις λίστες αλφαβητικά και αριθμητικά, αλλά τι γίνεται αν η λίστα έχει αντικείμενα;

Για να ταξινομήσετε αντικείμενα πρέπει να καθορίσετε έναν κανόνα που αποφασίζει πώς πρέπει να ταξινομηθούν τα αντικείμενα. Για παράδειγμα, εάν έχετε μια λίστα με αυτοκίνητα που μπορεί να θέλετε να τα ταξινομήσετε ανά έτος, ο κανόνας θα μπορούσε να είναι ότι τα αυτοκίνητα με προηγούμενο έτος πηγαίνουν πρώτα.

Οι διεπαφές `Comparator`και `Comparable`σάς επιτρέπουν να καθορίσετε ποιος κανόνας χρησιμοποιείται για την ταξινόμηση αντικειμένων.

Η δυνατότητα καθορισμού ενός κανόνα ταξινόμησης σάς επιτρέπει επίσης να αλλάξετε τον τρόπο ταξινόμησης των συμβολοσειρών και των αριθμών.

Συγκριτές

Ένα αντικείμενο που υλοποιεί τη `Comparator`διεπαφή ονομάζεται συγκριτής.

Η `Comparator`διεπαφή σάς επιτρέπει να δημιουργήσετε μια κλάση με μια `compare()`μέθοδο που συγκρίνει δύο αντικείμενα για να αποφασίσετε ποιο θα πρέπει να πάει πρώτο σε μια λίστα.

Η `compare()`μέθοδος πρέπει να επιστρέψει έναν αριθμό που είναι:

- Αρνητικό εάν το πρώτο αντικείμενο πρέπει να μπει πρώτο σε μια λίστα.
- Θετικό εάν το δεύτερο αντικείμενο πρέπει να μπει πρώτο σε μια λίστα.
- Μηδέν αν η σειρά δεν έχει σημασία.

Μια κλάση που υλοποιεί τη `Comparator`διεπαφή μπορεί να μοιάζει κάπως έτσι:

```
// Sort Car objects by year
class SortByYear implements Comparator {
  public int compare(Object obj1, Object obj2) {
    // Make sure that the objects are Car objects
    Car a = (Car) obj1;
    Car b = (Car) obj2;

    // Compare the objects
    if (a.year < b.year) return -1; // The first car has a smaller year
    if (a.year > b.year) return 1;  // The first car has a larger year
    return 0; // Both cars have the same year
  }
}
```

Για να χρησιμοποιήσετε τον συγκριτικό, περάστε τον ως όρισμα σε μια μέθοδο ταξινόμησης:

```
// Use a comparator to sort the cars
Comparator myComparator = new SortByYear();
Collections.sort(myCars, myComparator);
```

Ακολουθεί ένα πλήρες παράδειγμα με χρήση συγκριτή για την ταξινόμηση μιας λίστας αυτοκινήτων ανά έτος:

Παράδειγμα

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

// Define a Car class
class Car {
  public String brand;
  public String model;
```

```
  public int year;

  public Car(String b, String m, int y) {
    brand = b;
    model = m;
    year = y;
  }
}

// Create a comparator
class SortByYear implements Comparator {
  public int compare(Object obj1, Object obj2) {
    // Make sure that the objects are Car objects
    Car a = (Car) obj1;
    Car b = (Car) obj2;

    // Compare the year of both objects
    if (a.year < b.year) return -1; // The first car has a smaller year
    if (a.year > b.year) return 1;  // The first car has a larger year
    return 0; // Both cars have the same year
  }
}

public class Main {
  public static void main(String[] args) {
    // Create a list of cars
    ArrayList<Car> myCars = new ArrayList<Car>();
    myCars.add(new Car("BMW", "X5", 1999));
    myCars.add(new Car("Honda", "Accord", 2006));
```

```
    myCars.add(new Car("Ford", "Mustang", 1970));


    // Use a comparator to sort the cars
    Comparator myComparator = new SortByYear();
    Collections.sort(myCars, myComparator);


    // Display the cars
    for (Car c : myCars) {
      System.out.println(c.brand + " " + c.model + " " + c.year);
    }
  }
}
```

Χρήση έκφρασης λάμδα

Για να γίνει ο κώδικας συντομότερος, ο συγκριτής μπορεί να αντικατασταθεί με μια έκφραση λάμδα που έχει τα ίδια ορίσματα και την ίδια τιμή με τη `compare()`μέθοδο:

Παράδειγμα

Χρησιμοποιήστε μια έκφραση λάμδα ως σύγκριση:

```
Collections.sort(myCars, (obj1, obj2) -> {
  Car a = (Car) obj1;
  Car b = (Car) obj2;
  if (a.year < b.year) return -1;
  if (a.year > b.year) return 1;
  return 0;
});
```

Ειδικοί κανόνες ταξινόμησης

Οι συγκριτές μπορούν επίσης να χρησιμοποιηθούν για τη δημιουργία ειδικών κανόνων ταξινόμησης για συμβολοσειρές και αριθμούς. Σε αυτό το παράδειγμα χρησιμοποιούμε έναν συγκριτικό για να απαριθμήσουμε όλους τους ζυγούς αριθμούς πριν από τους περιττούς:

Παράδειγμα

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

class SortEvenFirst implements Comparator {
  public int compare(Object obj1, Object obj2) {
    // Make sure the objects are integers
    Integer a = (Integer)obj1;
    Integer b = (Integer)obj2;

    // Check each number to see if it is even
    // A number is even if the remainder when dividing by 2 is 0
    boolean aIsEven = (a % 2) == 0;
    boolean bIsEven = (b % 2) == 0;

    if (aIsEven == bIsEven) {

      // If both numbers are even or both are odd then use normal sorting
rules
      if (a < b) return -1;
      if (a > b) return 1;
      return 0;

    } else {

      // If a is even then it goes first, otherwise b goes first
      if (aIsEven) {
       return -1;
```

```
      } else {
        return 1;
      }
    }
  }
}

public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(33);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(34);
    myNumbers.add(8);
    myNumbers.add(12);

    Comparator myComparator = new SortEvenFirst();
    Collections.sort(myNumbers, myComparator);

    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
}
```

Η συγκρίσιμη διεπαφή

Η `Comparable`διεπαφή επιτρέπει σε ένα αντικείμενο να καθορίσει τον δικό του κανόνα ταξινόμησης με μια `compareTo()`μέθοδο.

Η `compareTo()`μέθοδος παίρνει ένα αντικείμενο ως όρισμα και συγκρίνει το συγκρίσιμο με το όρισμα για να αποφασίσει ποιο πρέπει να πάει πρώτο σε μια λίστα.

Όπως ο συγκριτής, η `compareTo()`μέθοδος επιστρέφει έναν αριθμό που είναι:

- Αρνητικό εάν το συγκρίσιμο πρέπει να είναι πρώτο σε μια λίστα.
- Θετικό εάν το άλλο αντικείμενο πρέπει να μπει πρώτο σε μια λίστα.
- Μηδέν αν η σειρά δεν έχει σημασία.

Πολλές εγγενείς κλάσεις Java υλοποιούν τη `Comparable`διεπαφή, όπως `String`και `Integer`.

Αυτός είναι ο λόγος για τον οποίο οι συμβολοσειρές και οι αριθμοί δεν χρειάζονται συγκριτικό για να ταξινομηθούν.

Ένα αντικείμενο που υλοποιεί τη `Comparable`διεπαφή μπορεί να μοιάζει κάπως έτσι:

```
class Car implements Comparable {

  public String brand;

  public String model;

  public int year;


  // Decide how this object compares to other objects

  public int compareTo(Object obj) {

        Car other = (Car)obj;

    if(year < other.year) return -1; // This object is smaller than the
other one

    if(year > other.year) return 1;  // This object is larger than the other
one

    return 0; // Both objects are the same

  }

}
```

Εδώ είναι το ίδιο παράδειγμα όπως πριν, αλλά χρησιμοποιώντας τη `Comparable`διεπαφή αντί για έναν συγκριτικό:

Παράδειγμα

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

// Define a Car class which is comparable
class Car implements Comparable {
  public String brand;
  public String model;
  public int year;

  public Car(String b, String m, int y) {
    brand = b;
    model = m;
    year = y;
  }

  // Decide how this object compares to other objects
  public int compareTo(Object obj) {
       Car other = (Car)obj;
    if(year < other.year) return -1; // This object is smaller than the
other one
    if(year > other.year) return 1;  // This object is larger than the other
one
    return 0; // Both objects are the same
  }
}

public class Main {
  public static void main(String[] args) {
    // Create a list of cars
```

```
    ArrayList<Car> myCars = new ArrayList<Car>();
    myCars.add(new Car("BMW", "X5", 1999));
    myCars.add(new Car("Honda", "Accord", 2006));
    myCars.add(new Car("Ford", "Mustang", 1970));

    // Sort the cars
    Collections.sort(myCars);

    // Display the cars
    for (Car c : myCars) {
      System.out.println(c.brand + " " + c.model + " " + c.year);
    }
  }
}
```

Ένα κοινό κόλπο ταξινόμησης

Ο πιο προφανής τρόπος για να ταξινομήσετε δύο αριθμούς φυσικά είναι να γράψετε κάτι σαν αυτό:

```
if(a.year < b.year) return -1; // a is less than b
if(a.year > b.year) return 1; // a is greater than b
return 0; // a is equal to b
```

Αλλά στην πραγματικότητα μπορεί να γίνει με μία μόνο γραμμή:

```
return a.year - b.year;
```

Αυτό το τέχνασμα μπορεί επίσης να χρησιμοποιηθεί για να ταξινομήσετε εύκολα τα πράγματα αντίστροφα:

```
return b.year - a.year;
```

Συγκριτής εναντίον Συγκρίσιμου

Συγκριτής είναι ένα αντικείμενο με μία μέθοδο που χρησιμοποιείται για τη σύγκριση δύο διαφορετικών αντικειμένων.

Συγκρίσιμο είναι ένα αντικείμενο που μπορεί να συγκριθεί με άλλα αντικείμενα.

Είναι ευκολότερο να χρησιμοποιήσετε τη `Comparable`διεπαφή όταν είναι δυνατόν, αλλά η `Comparator`διεπαφή είναι πιο ισχυρή επειδή σας επιτρέπει να ταξινομήσετε οποιοδήποτε είδος αντικειμένου, ακόμα κι αν δεν μπορείτε να αλλάξετε τον κώδικά του.

Java Files

Ο χειρισμός αρχείων είναι ένα σημαντικό μέρος οποιασδήποτε εφαρμογής.

Η Java έχει πολλές μεθόδους για τη δημιουργία, την ανάγνωση, την ενημέρωση και τη διαγραφή αρχείων.

Χειρισμός αρχείων Java

Η `File`κλάση από το `java.io`πακέτο, μας επιτρέπει να δουλεύουμε με αρχεία.

Για να χρησιμοποιήσετε την `File`κλάση, δημιουργήστε ένα αντικείμενο της κλάσης και καθορίστε το όνομα αρχείου ή το όνομα καταλόγου:

Παράδειγμα

```
import java.io.File;  // Import the File class


File myObj = new File("filename.txt"); // Specify the filename
```

Εάν δεν ξέρετε τι είναι ένα πακέτο, διαβάστε τον οδηγό μας για τα πακέτα Java .

Η `File`τάξη έχει πολλές χρήσιμες μεθόδους για τη δημιουργία και τη λήψη πληροφοριών σχετικά με αρχεία. Για παράδειγμα:

Method	Type	Description

`canRead()`	Boolean	Tests whether the file is readable or not
`canWrite()`	Boolean	Tests whether the file is writable or not
`createNewFile()`	Boolean	Creates an empty file
`delete()`	Boolean	Deletes a file
`exists()`	Boolean	Tests whether the file exists
`getName()`	String	Returns the name of the file
`getAbsolutePath()`	String	Returns the absolute pathname of the file
`length()`	Long	Returns the size of the file in bytes
`list()`	String[]	Returns an array of the files in the directory
`mkdir()`	Boolean	Creates a directory

Java Create and Write To Files

Δημιουργήστε ένα Αρχείο

Για να δημιουργήσετε ένα αρχείο σε Java, μπορείτε να χρησιμοποιήσετε τη `createNewFile()`μέθοδο. Αυτή η μέθοδος επιστρέφει μια τιμή boolean: `true`εάν το αρχείο δημιουργήθηκε με επιτυχία και `false`εάν το αρχείο υπάρχει ήδη. Σημειώστε ότι η μέθοδος περικλείεται σε ένα `try...catch` μπλοκ. Αυτό είναι απαραίτητο γιατί εκπέμπει ένα `IOException`εάν παρουσιαστεί σφάλμα (αν το αρχείο δεν μπορεί να δημιουργηθεί για κάποιο λόγο):

Παράδειγμα

```
import java.io.File;  // Import the File class

import java.io.IOException;  // Import the IOException class to handle
errors


public class CreateFile {

  public static void main(String[] args) {

    try {

      File myObj = new File("filename.txt");

      if (myObj.createNewFile()) {

        System.out.println("File created: " + myObj.getName());

      } else {

        System.out.println("File already exists.");

      }

    } catch (IOException e) {

      System.out.println("An error occurred.");

      e.printStackTrace();

    }

  }

}
```

Η έξοδος θα είναι:

```
File created: filename.txt
```

Για να δημιουργήσετε ένα αρχείο σε έναν συγκεκριμένο κατάλογο (απαιτείται άδεια), καθορίστε τη διαδρομή του αρχείου και χρησιμοποιήστε διπλές ανάστροφες κάθετες για να διαφύγετε από τον `\`χαρακτήρα " " (για Windows). Σε Mac και Linux μπορείτε απλώς να γράψετε τη διαδρομή, όπως: /Users/name/filename.txt

Παράδειγμα

```
File myObj = new File("C:\\Users\\MyName\\filename.txt");
```

Εγγραφή σε αρχείο

Στο παρακάτω παράδειγμα, χρησιμοποιούμε την `FileWriter`κλάση μαζί με τη `write()`μέθοδο της για να γράψουμε κάποιο κείμενο στο αρχείο που δημιουργήσαμε στο παραπάνω παράδειγμα. Σημειώστε ότι όταν ολοκληρώσετε την εγγραφή στο αρχείο, θα πρέπει να το κλείσετε με τη `close()`μέθοδο:

Παράδειγμα

```
import java.io.FileWriter;   // Import the FileWriter class

import java.io.IOException;  // Import the IOException class to handle
errors


public class WriteToFile {

  public static void main(String[] args) {

    try {

      FileWriter myWriter = new FileWriter("filename.txt");

      myWriter.write("Files in Java might be tricky, but it is fun
enough!");

      myWriter.close();

      System.out.println("Successfully wrote to the file.");

    } catch (IOException e) {

      System.out.println("An error occurred.");

      e.printStackTrace();

    }

  }

}
```

Η έξοδος θα είναι:

```
Successfully wrote to the file.
```

Java Read Files

Διαβάστε ένα αρχείο

Στο προηγούμενο κεφάλαιο, μάθατε πώς να δημιουργείτε και να γράφετε σε ένα αρχείο.

Στο παρακάτω παράδειγμα, χρησιμοποιούμε την Scannerκλάση για να διαβάσουμε τα περιεχόμενα του αρχείου κειμένου που δημιουργήσαμε στο προηγούμενο κεφάλαιο:

Παράδειγμα

```
import java.io.File;  // Import the File class

import java.io.FileNotFoundException;  // Import this class to handle errors

import java.util.Scanner; // Import the Scanner class to read text files


public class ReadFile {

  public static void main(String[] args) {

    try {

      File myObj = new File("filename.txt");

      Scanner myReader = new Scanner(myObj);

      while (myReader.hasNextLine()) {

        String data = myReader.nextLine();

        System.out.println(data);

      }

      myReader.close();

    } catch (FileNotFoundException e) {

      System.out.println("An error occurred.");

      e.printStackTrace();

    }
```

```
  }
}
```

Η έξοδος θα είναι:

```
Files in Java might be tricky, but it is fun enough!
```

Λήψη πληροφοριών αρχείου

Για να λάβετε περισσότερες πληροφορίες σχετικά με ένα αρχείο, χρησιμοποιήστε οποιαδήποτε από τις `File`μεθόδους:

Παράδειγμα

```
import java.io.File;  // Import the File class


public class GetFileInfo {
  public static void main(String[] args) {

    File myObj = new File("filename.txt");

    if (myObj.exists()) {

      System.out.println("File name: " + myObj.getName());

      System.out.println("Absolute path: " + myObj.getAbsolutePath());

      System.out.println("Writeable: " + myObj.canWrite());

      System.out.println("Readable " + myObj.canRead());

      System.out.println("File size in bytes " + myObj.length());

    } else {

      System.out.println("The file does not exist.");

    }

  }

}
```

Η έξοδος θα είναι:

```
File name: filename.txt
Absolute path: C:\Users\MyName\filename.txt
Writeable: true
```

```
Readable: true
File size in bytes: 0
```

Σημείωση: Υπάρχουν πολλές διαθέσιμες κλάσεις στο Java API που μπορούν να χρησιμοποιηθούν για την ανάγνωση και την εγγραφή αρχείων σε Java: `FileReader`, `BufferedReader`, `Files`, `Scanner`, `FileInputStream`, `FileWriter`, `BufferedWriter`, `FileOutputStream`, κ.λπ. Ποια θα χρησιμοποιηθεί εξαρτάται από την έκδοση Java με την οποία εργάζεστε και αν χρειάζεται να διαβάσετε byte ή χαρακτήρες και το μέγεθος του αρχείου/γραμμών κ.λπ.

Java Delete Files

Διαγραφή αρχείου

Για να διαγράψετε ένα αρχείο σε Java, χρησιμοποιήστε τη `delete()`μέθοδο:

Παράδειγμα

```
import java.io.File;  // Import the File class


public class DeleteFile {

  public static void main(String[] args) {

    File myObj = new File("filename.txt");

    if (myObj.delete()) {

      System.out.println("Deleted the file: " + myObj.getName());

    } else {

      System.out.println("Failed to delete the file.");

    }

  }

}
```

Η έξοδος θα είναι:

```
Deleted the file: filename.txt
```

Διαγραφή φακέλου

Μπορείτε επίσης να διαγράψετε έναν φάκελο. Ωστόσο, πρέπει να είναι κενό:

Παράδειγμα

```
import java.io.File;


public class DeleteFolder {
  public static void main(String[] args) {
    File myObj = new File("C:\\Users\\MyName\\Test");
    if (myObj.delete()) {
      System.out.println("Deleted the folder: " + myObj.getName());
    } else {
      System.out.println("Failed to delete the folder.");
    }
  }
}
```

Η έξοδος θα είναι:

```
Deleted the folder: Test
```