Some Java Basics

Object Oriented Programming
2016375 - 5

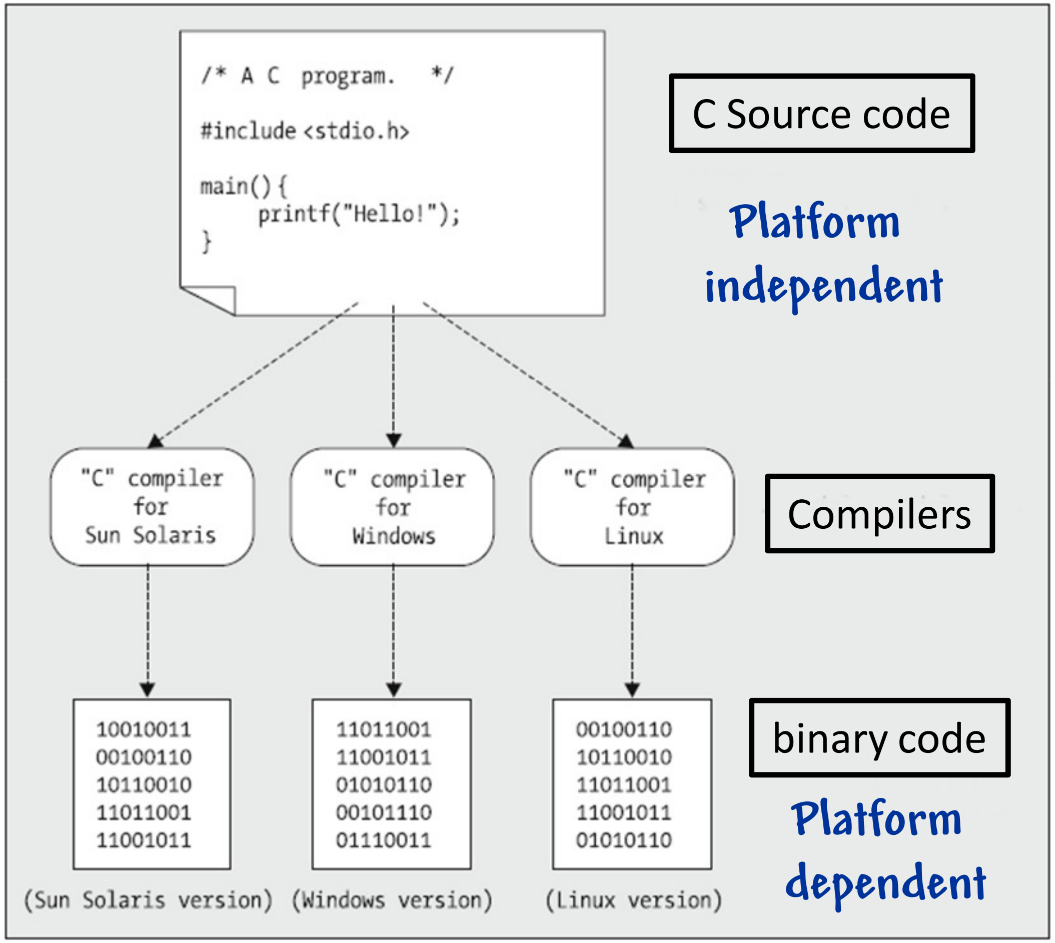
Camilo López

Outline

- Why Java?
- Primitive Data Types
- Variables
- Arrays
- The String Type
- Java Expressions
- Flow Control Structures
- Anatomy of a Simple Java Program
- Intro to Eclipse
- User input
- Some Tips

Why Java?

Java Is Architecture Neutral



```
/* A C  program.   */

#include <stdio.h>

main() {
     printf("Hello!");
}
```

```
10010011
00100110
10110010
11011001
11001011
```

(Sun Solaris version)

```
11011001
11001011
01010110
00101110
01110011
```

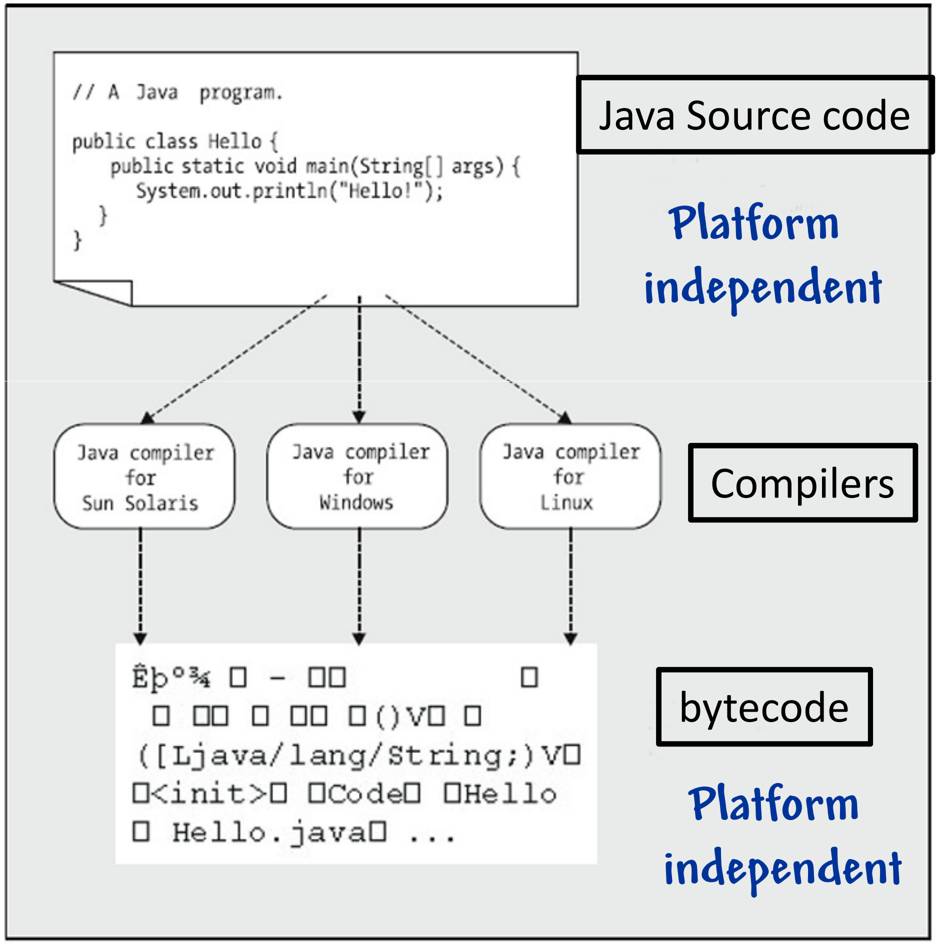
(Windows version)

```
00100110
10110010
11011001
11001011
01010110
```

(Linux version)

Why Java?

Java Is Architecture Neutral



```
// A Java program.

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

Java Source code

Platform independent

Compilers

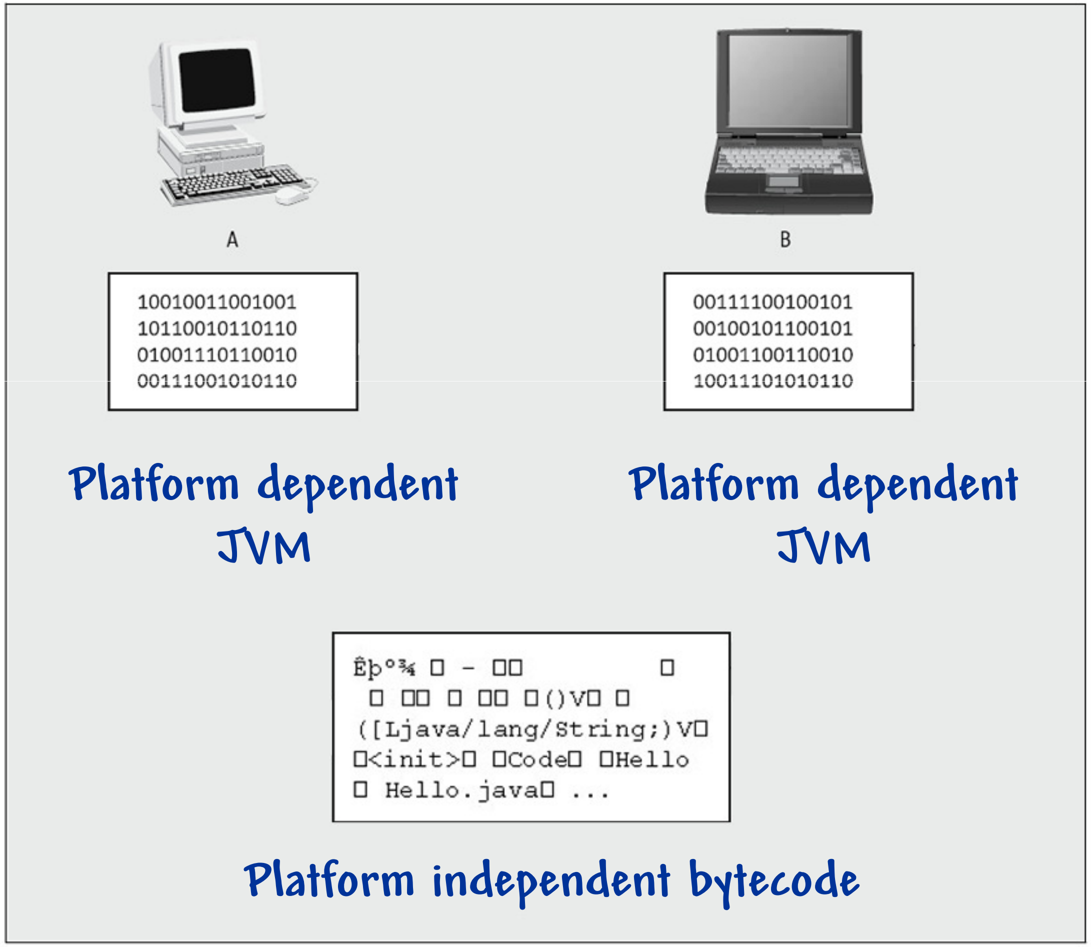
```
Êþº¾ □ – □□          □
 □ □□ □ □□ □()V□ □
([Ljava/lang/String;)V□
□<init>□ □Code□ □Hello
□ Hello.java□ ...
```

bytecode

Platform independent

Why Java?

Java Is Architecture Neutral



```
10010011001001
10110010110110
01001110110010
00111001010110
```

```
00111100100101
00100101100101
01001100110010
10011101010110
```

Platform dependent JVM

Platform dependent JVM

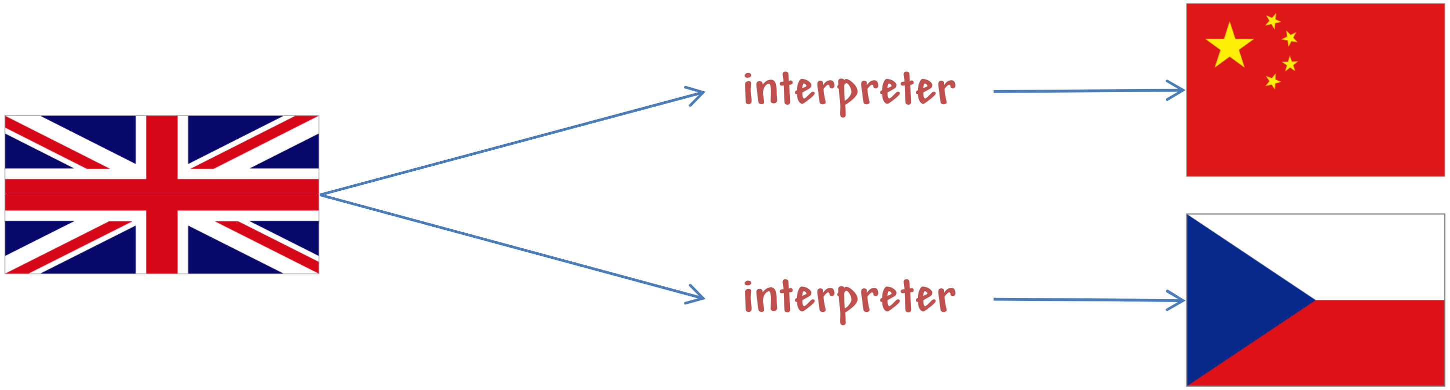
```
Êþº¾ □ - □□          □
 □ □□ □ □□ □()V□ □
([Ljava/lang/String;)V□
□<init>□ □Code□ □Hello
□ Hello.java□ ...
```

Platform independent bytecode

Why Java?

The Java Virtual Machine

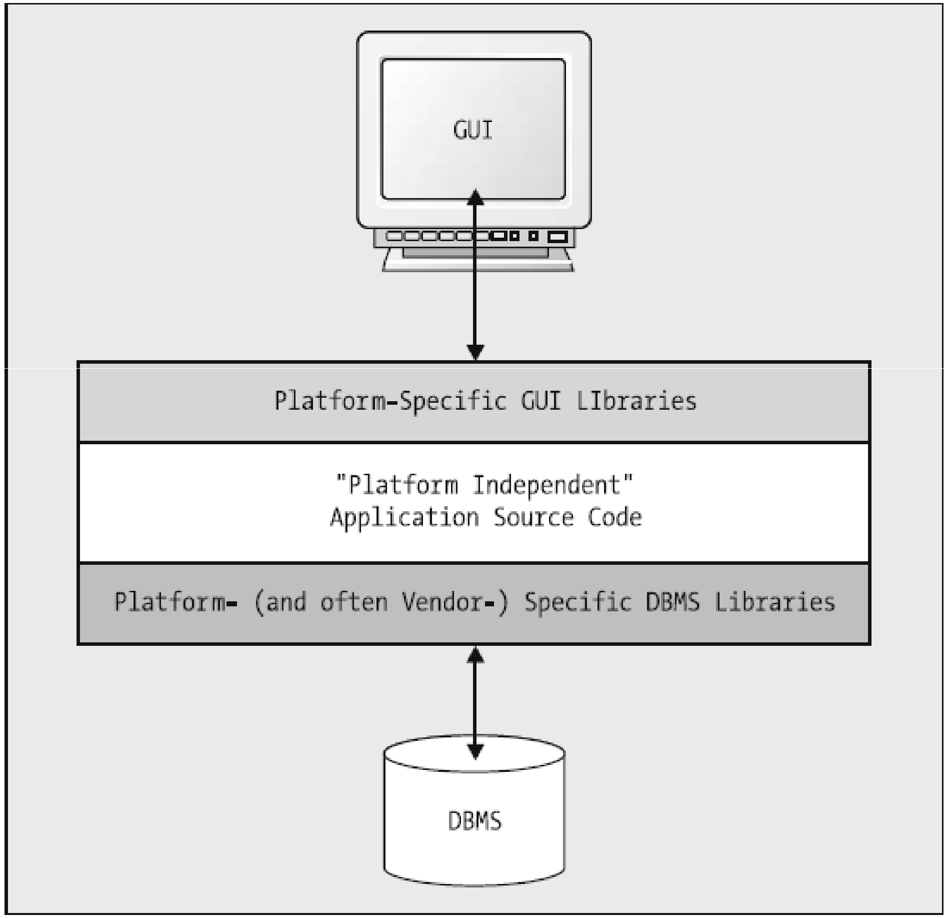
- The JVM is a special piece of software that knows how to interpret and execute Java bytecode.



- **Performance:** a bit slower, in general, than compiled languages
- Java bytecode is, in theory, **forward compatible** with newer versions of the JVM.

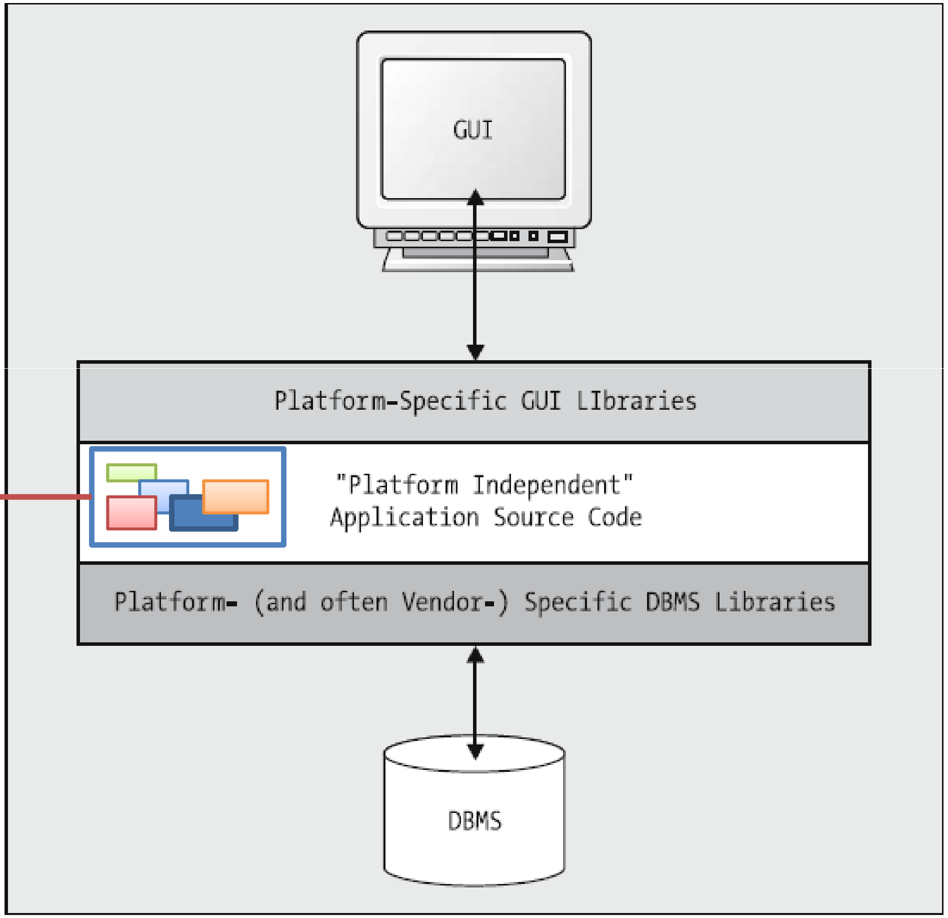
Why Java?

Java Provides "One-Stop Shopping"


GUI
Platform-Specific GUI LIbraries
"Platform Independent"
Application Source Code
Platform- (and often Vendor-) Specific DBMS Libraries
DBMS

Why Java?

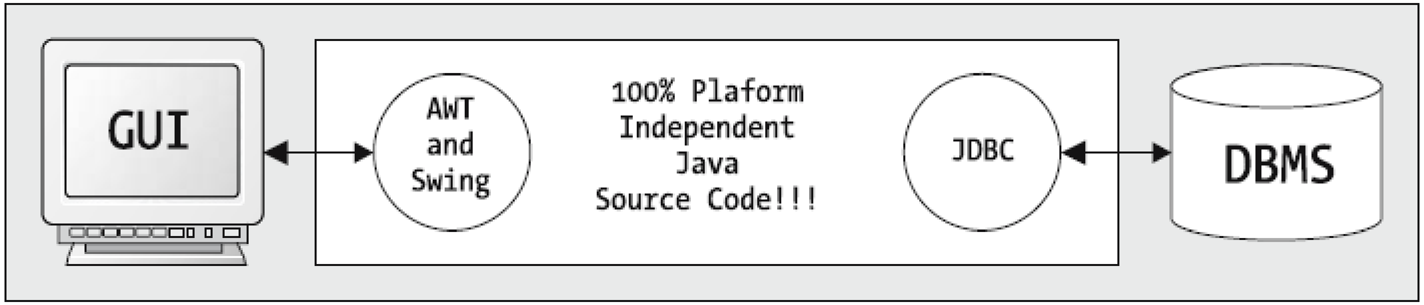
Java Provides "One-Stop Shopping"


GUI
Platform-Specific GUI LIbraries
"Platform Independent"
Application Source Code
Platform- (and often Vendor-) Specific DBMS Libraries
DBMS

Platform specific library calls.

Why Java?

Java Provides "One-Stop Shopping"



- the Java language provides an extensive set of **application programming interfaces (APIs)** → *platform-independent* means of accessing all underlying operating system functions
  - **java.io**: Used for file system access
  - **java.sql**: The JDBC API, used for communicating with relational databases in a vendor independent fashion
  - **java.awt**: The Abstract Windowing Toolkit, used for GUI development
  - **javax.swing**: Swing components, also used for GUI development

Why Java?

Java Provides "One-Stop Shopping"

- Java is OO from the Ground Up
  - C++ can be used as an improved version of C
  - All data, with the exception of a few primitive types, is rendered as objects.
  - All functions are associated with objects and are known as methods—there can be no "free-floating" functions as there are in C/C++.
- Java Is an Open Standard
- Java is Free!

Java lends itself particularly well to writing applications that uphold the OO paradigm.

Primitive Data Types

- The Java programming language is statically-typed

Data Type	Size (bits)	Default Value
byte	8	0
short	16	0
int	32	0
long	64	0L
float	32	0.0f
double	64	0.0d
char	16 (Unicode)	'\u0000'
boolean		false
String	…	null

- Local variables are slightly different; the compiler never assigns a default value to an uninitialized local variable.

Variables

- Before a variable can be used in a Java program, the type and name of the variable must be ***declared*** to the Java compiler.
- One declaration per line is recommended since it encourages commenting.
- Try to initialize local variables where they're declared.
- Naming
  - Meaningful and valid (must start with either an alphabetic character, an underscore, or a dollar sign (*whose use is discouraged, since it is used by the compiler when generating code*), and may contain any of these characters plus numeric digits)
  - camelCasing

Variables

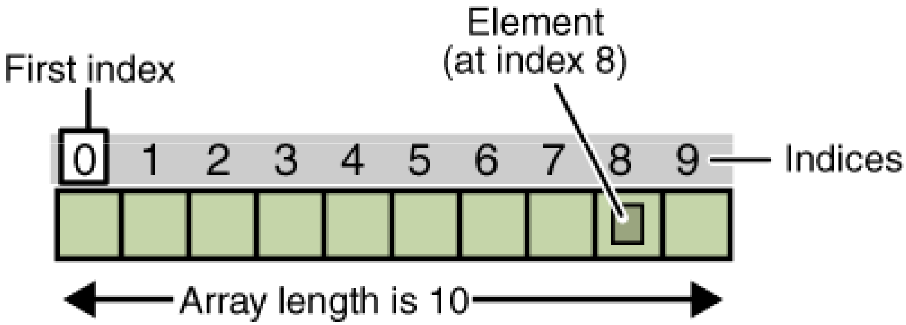
- Some examples

```
int simple;
int _under;
int more$money_is_2much;

int 1bad;
int number#sign;
int foo-bar;
int plus+sign;
int x@y;
int dot.notation;
```

Arrays

An ***array*** is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.

```
datatype[ ] arrayName;
datatype[ ][ ] arrayOfArrays;
```



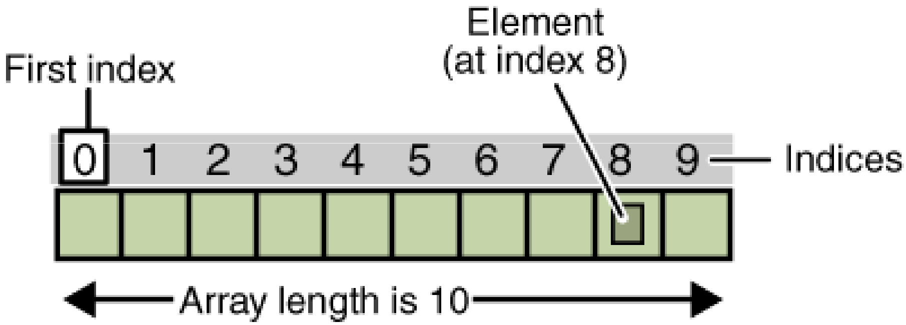
```
int[ ] arrayName;

arrayName = new int[10];
arrayName[0] = 3;
int[ ] anotherArray = {1, 2, 3, 4, 5, 6};
```

Declares an array of integers

Arrays

An ***array*** is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.

```
datatype[ ] arrayName;
datatype[ ][ ] arrayOfArrays;
```



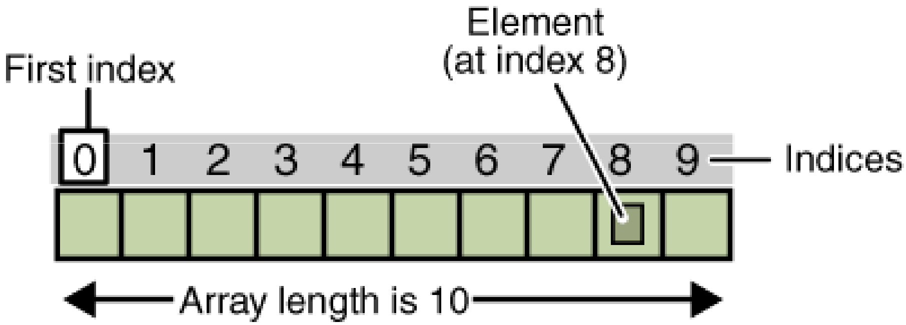
```
int[ ] arrayName;
arrayName = new int[10];
arrayName[0] = 3;
int[ ] anotherArray = {1, 2, 3, 4, 5, 6};
```

allocates memory for 10 integers

Arrays

An ***array*** is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.

```
datatype[ ] arrayName;
datatype[ ][ ] arrayOfArrays;
```



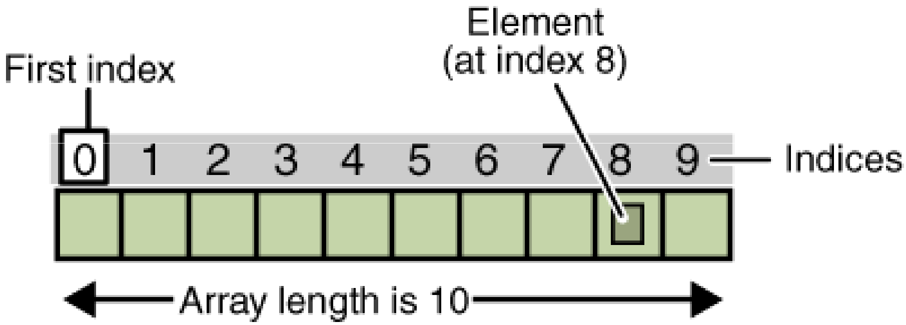
```
int[ ] arrayName;
arrayName = new int[10];
arrayName[0] = 3;
int[ ] anotherArray = {1, 2, 3, 4, 5, 6};
```

initialize first element

Arrays

An ***array*** is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.

```
datatype[ ] arrayName;
datatype[ ][ ] arrayOfArrays;
```



```
int[ ] arrayName;
arrayName = new int[10];
arrayName[0] = 3;
int[ ] anotherArray = {1, 2, 3, 4, 5, 6};
```

Declaration and initialization

The String Type

- A String represents a sequence of zero or more Unicode characters.

```
String stringName = “stringValue”;
String shortString = "A";

String s = "";

String x = "foo";
String y = "bar";
String z = x + y + "!“;
```

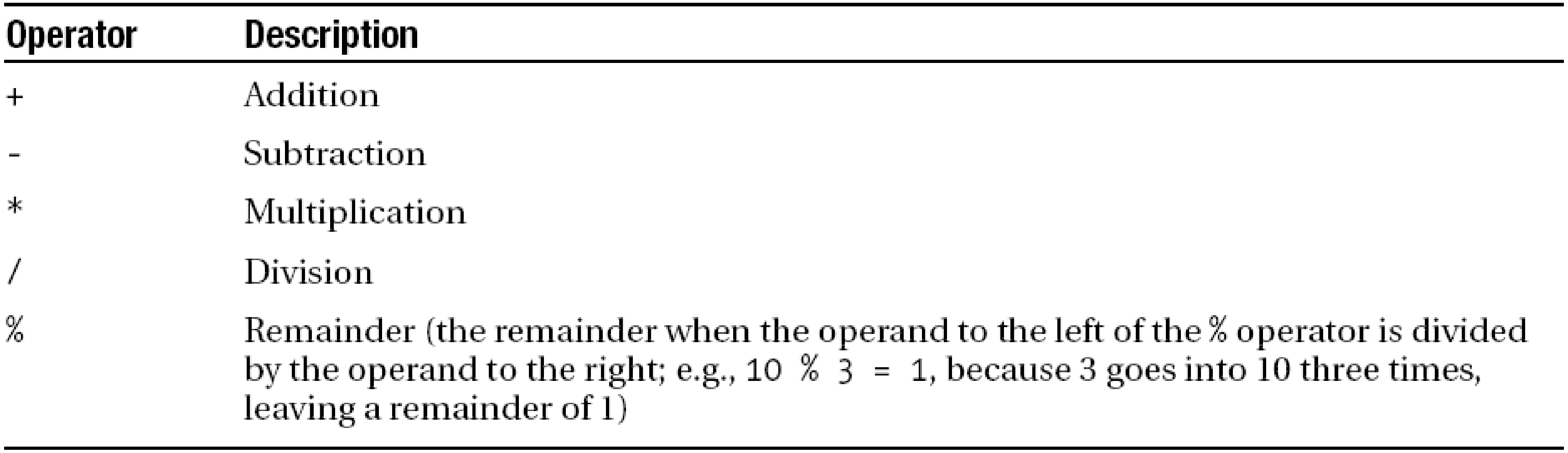
Keep this in mind: it’s a capital ‘S’

Java Expressions

- A constant: 7, false
- A char literal enclosed in single quotes: 'A', '3‘
- A String literal enclosed in double quotes: "foo", "Java“
- The name of any properly declared variables: myString, x
- Any two of the preceding types of expression that are combined with one of the Java binary operators
- Any one of the preceding types of expression that is modified by one of the Java unary operators
- Any of the preceding types of expression enclosed in parentheses: (x + 2)

Java Expressions

Arithmetic Operators



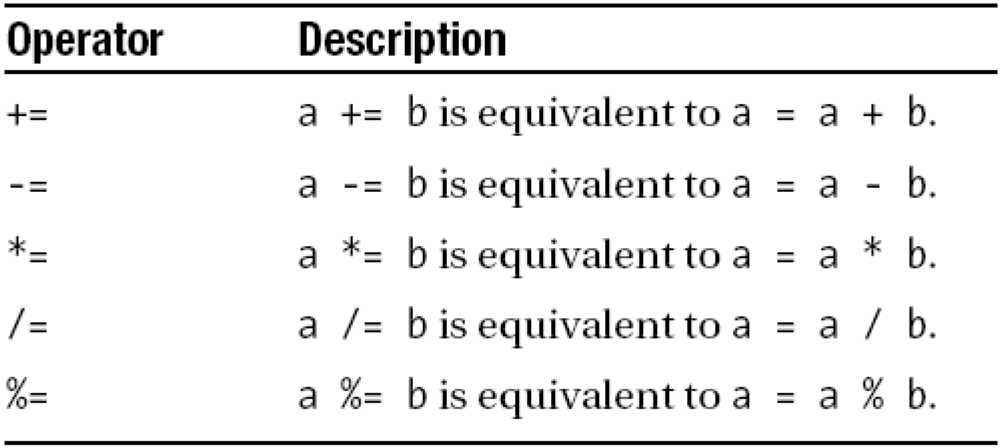
Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Remainder (the remainder when the operand to the left of the % operator is divided by the operand to the right; e.g., 10 % 3 = 1, because 3 goes into 10 three times, leaving a remainder of 1)

unary increment (++) and decrement (--) operators are used to increase or decrease the value of a int variable by 1.

- b = a++ → b = a; a = a + 1;
- b = ++a → a = a + 1; b = a;
- ch = ‘c’; c++;

Java Expressions

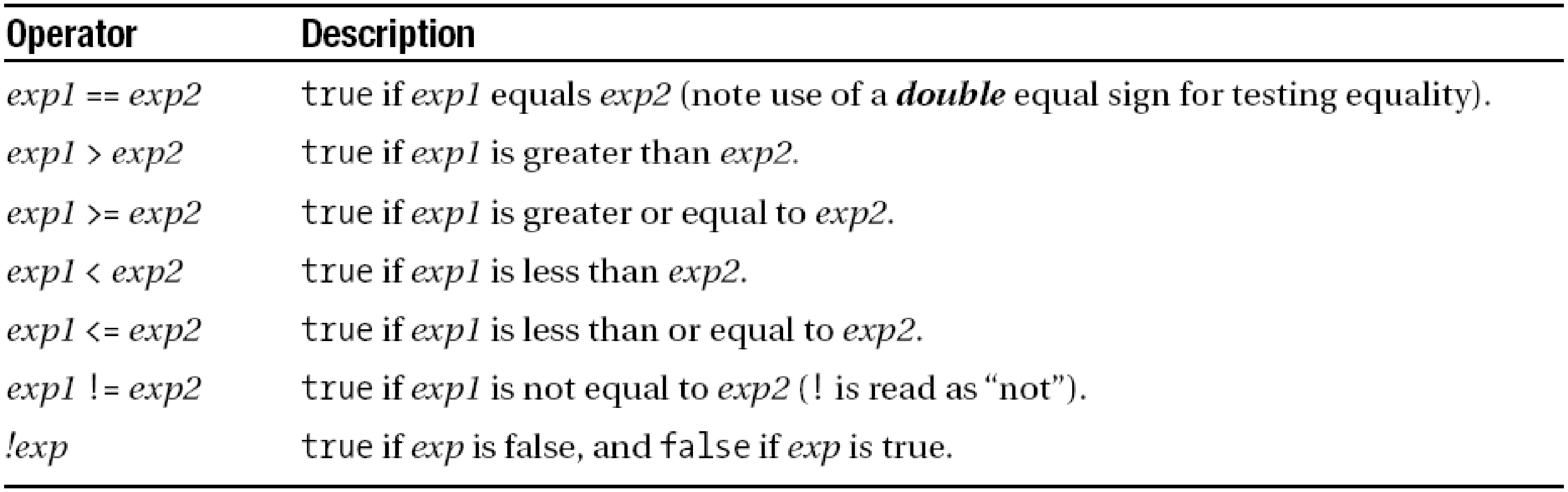
Java Compound Assignment Operators



Operator	Description
+=	a += b is equivalent to a = a + b.
-=	a -= b is equivalent to a = a - b.
*=	a *= b is equivalent to a = a * b.
/=	a /= b is equivalent to a = a / b.
%=	a %= b is equivalent to a = a % b.

Java Expressions

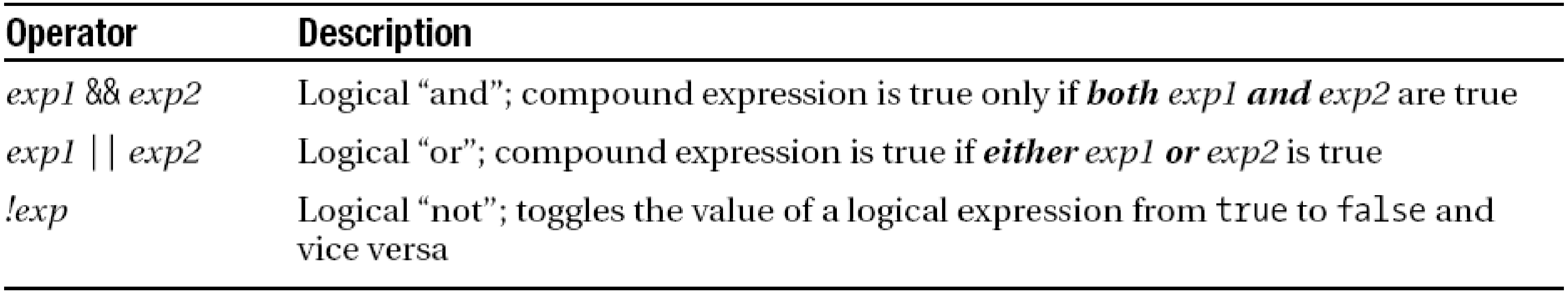
Relational Operators



Operator	Description
exp1 == *exp2*	`true` if *exp1* equals *exp2* (note use of a ***double*** equal sign for testing equality).
exp1 > *exp2*	`true` if *exp1* is greater than *exp2*.
exp1 >= *exp2*	`true` if *exp1* is greater or equal to *exp2*.
exp1 < *exp2*	`true` if *exp1* is less than *exp2*.
exp1 <= *exp2*	`true` if *exp1* is less than or equal to *exp2*.
exp1 != *exp2*	`true` if *exp1* is not equal to *exp2* (`!` is read as "not").
!exp	`true` if *exp* is false, and `false` if *exp* is true.

Java Expressions

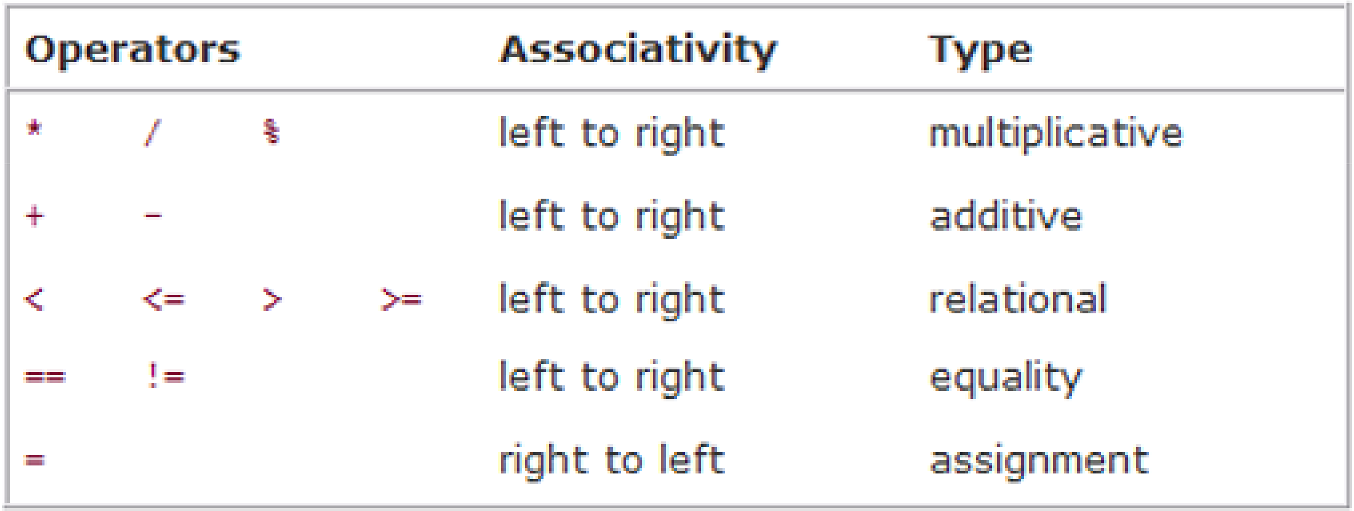
Logical Operators



Operator	Description
exp1 && *exp2*	Logical “and”; compound expression is true only if ***both*** *exp1* ***and*** *exp2* are true
exp1 || *exp2*	Logical “or”; compound expression is true if ***either*** *exp1* ***or*** *exp2* is true
!exp	Logical “not”; toggles the value of a logical expression from `true` to `false` and vice versa

Java Expressions

Precedence and associativity of operations discussed.

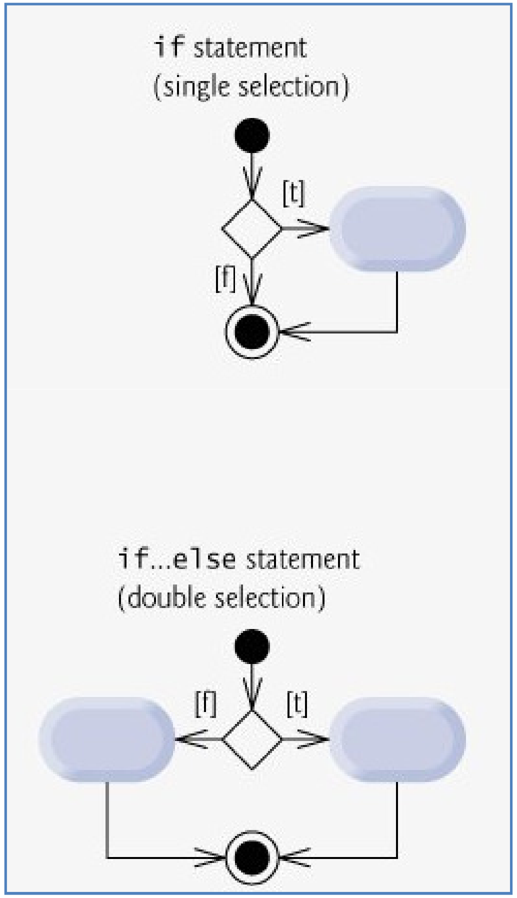


Operators	Associativity	Type
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
=	right to left	assignment

Flow-Control Structures

if Statements

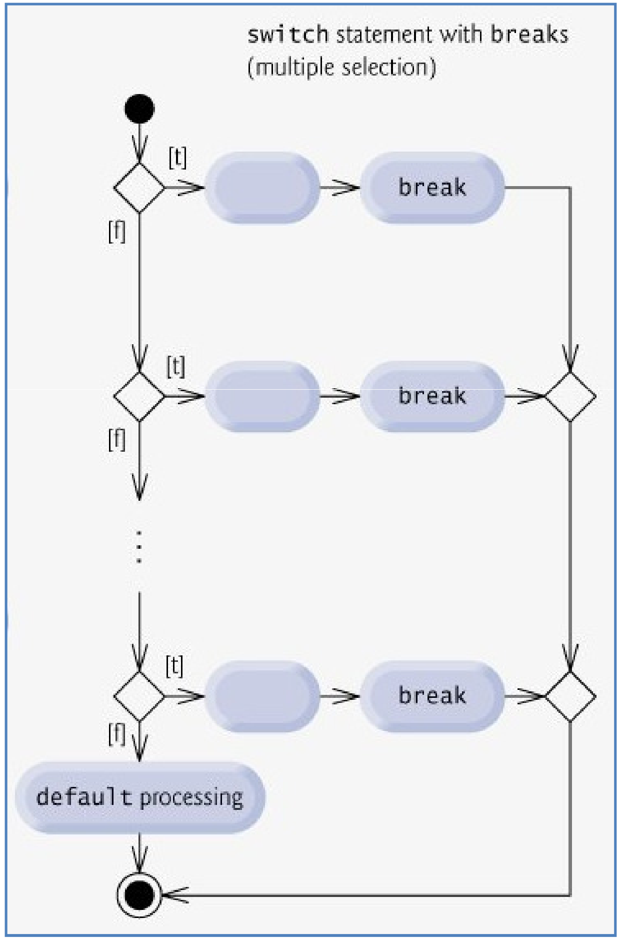
```
if (condition) {
    statements;
}

if (condition) {
    statements;
} else {
    statements;
}

if (condition) {
    statements;
} else if (condition) {
    statements;
} else if (condition) {
    statements;
}
```



Flow-Control Structures

switch Statements

```
switch (condition) {
    case ABC:
        statements;
        /* falls through */

    case DEF:
        statements;
        break;

    case XYZ:
        statements;
        break;

    default:
        statements;
        break;
}
```



Every time a case doesn't include a break statement add a comment where the break statement would normally be

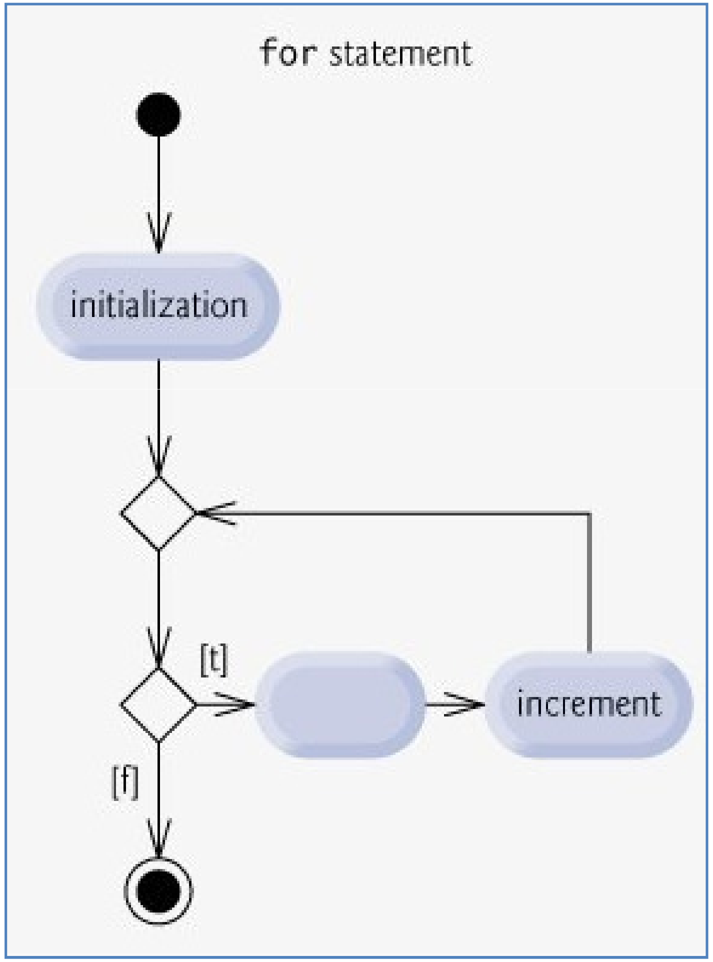
Flow-Control Structures

for Statements

```
for (initialization; condition; update) {
    statements;
}
```

```
for (int i = 0; i < 5; i++) {
    statements;
}
```

code to execute as long as the value of i remains less than 5

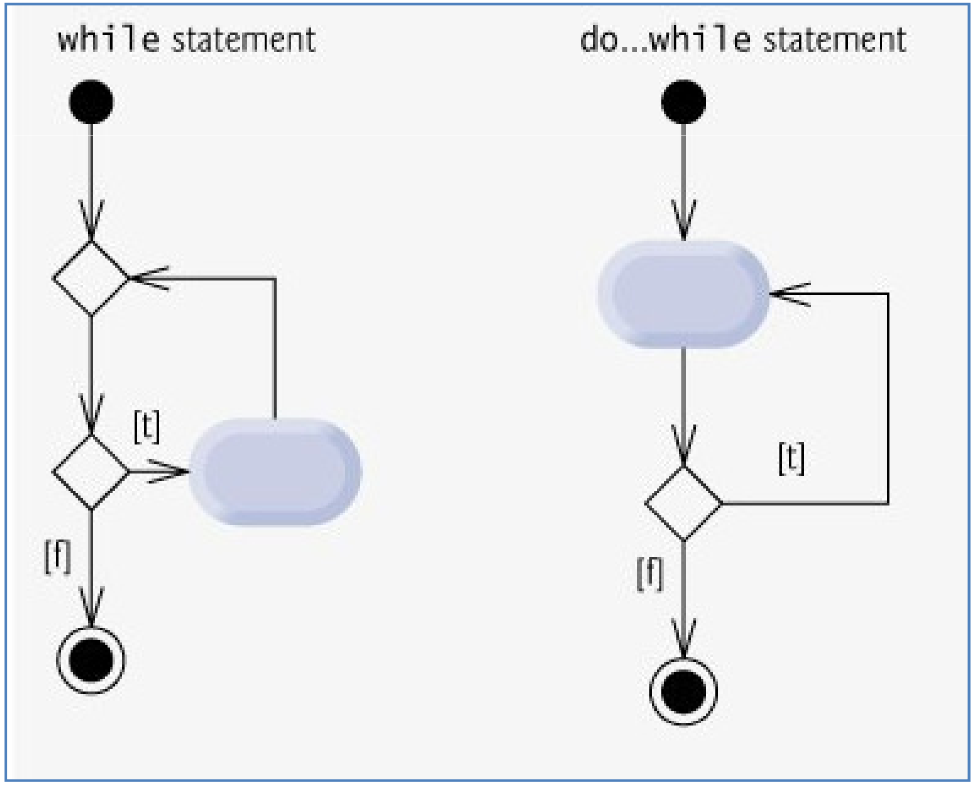


Flow-Control Structures

for Statements

```
while (condition) {
    statements;
}
```

```
do {
    statements;
} while (condition);
```



Anatomy of a Simple Java Program

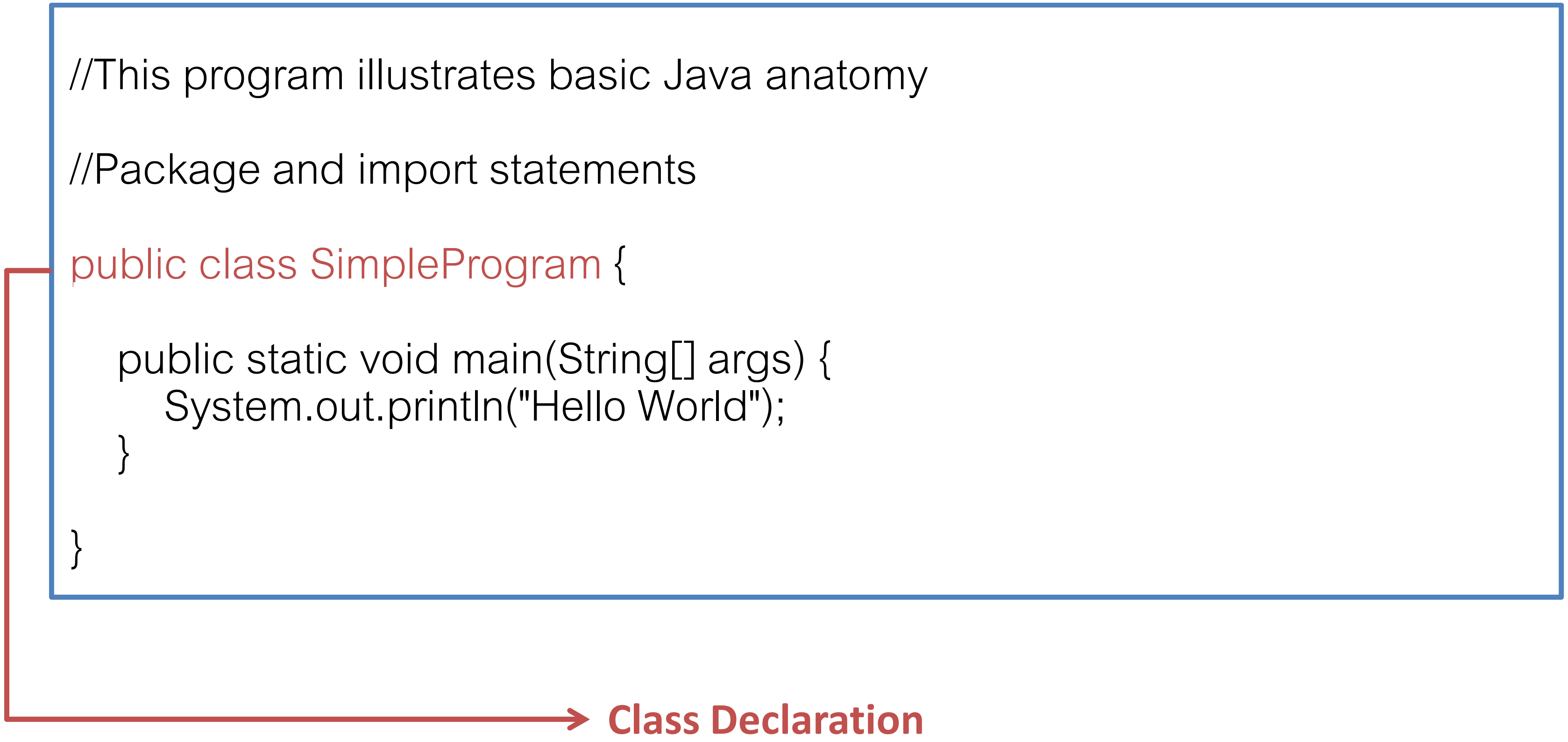
```
//This program illustrates basic Java anatomy


//Package and import statements

public class SimpleProgram {

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

}
```

Beginning comments

Types of comments

- **// end-of-line comments**
- **/* Traditional comments */**

Anatomy of a Simple Java Program



```
//This program illustrates basic Java anatomy

//Package and import statements

public class SimpleProgram {

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

}
```

Class Declaration

Anatomy of a Simple Java Program

```
//This program illustrates basic Java anatomy

//Package and import statements

public class SimpleProgram {

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

}
```

Main Method

Anatomy of a Simple Java Program

```
//This program illustrates basic Java anatomy

//Package and import statements

public class SimpleProgram {

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

}
```

main method begins execution of Java application

Print to the Screen

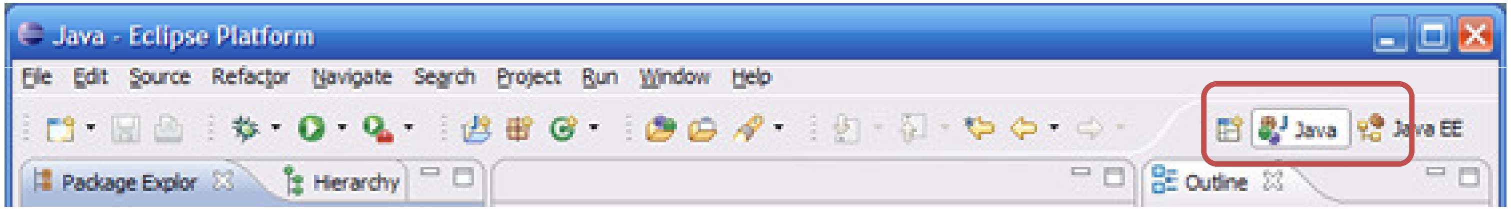
System.out.print(“text to print”);

System.out.println(“text to print”);

Eclipse

Open the Java Perspective

If you're not already in the Java perspective, in the main menu select **Window > Open Perspective > Java** or click on the icon shown below.

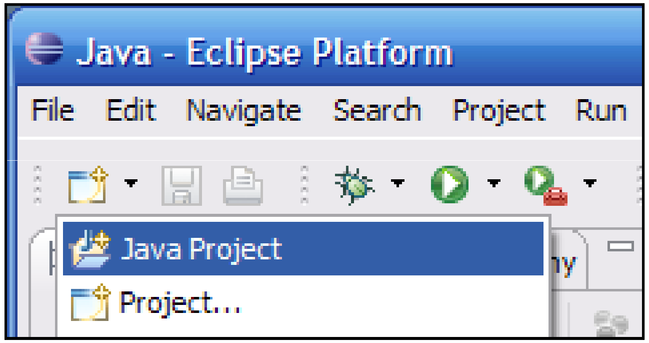

Java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Java EE
Package Explor
Hierarchy
Outline

A perspective is a way to organize and view the files associated with your program.

Eclipse

Create a Java Project

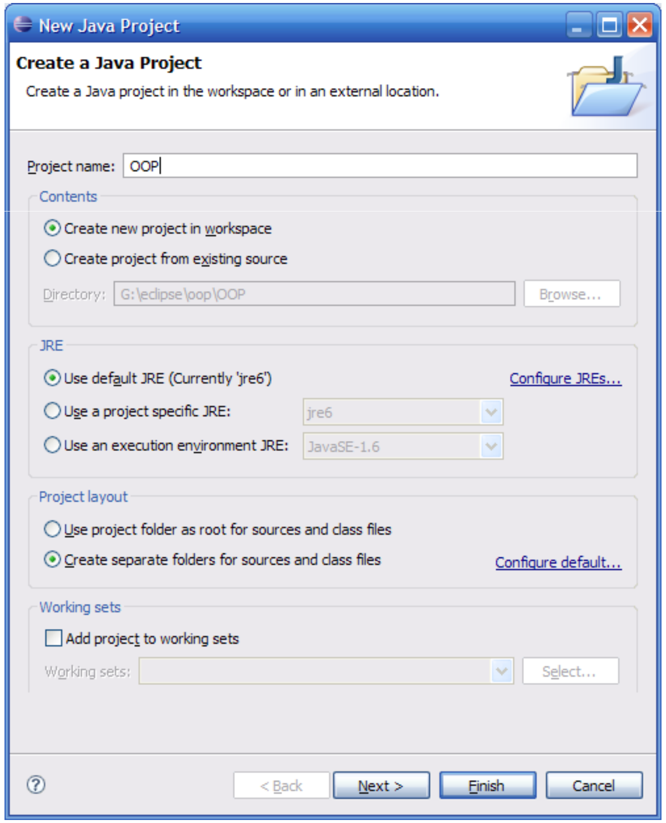
Before creating a class, we need a project to put it in. In the main toolbar, click on the **New Java Project button**



or select **File > New > Java Project**

The project name is any name you choose.

Your project will be created in the workspace associated with Eclipse.



Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: OOP

Contents

● Create new project in workspace

○ Create project from existing source

Directory: G:\eclipse\oop\OOP

JRE

● Use default JRE (Currently 'jre6')

○ Use a project specific JRE: jre6

○ Use an execution environment JRE: JavaSE-1.6

Project layout

○ Use project folder as root for sources and class files

● Create separate folders for sources and class files

Working sets

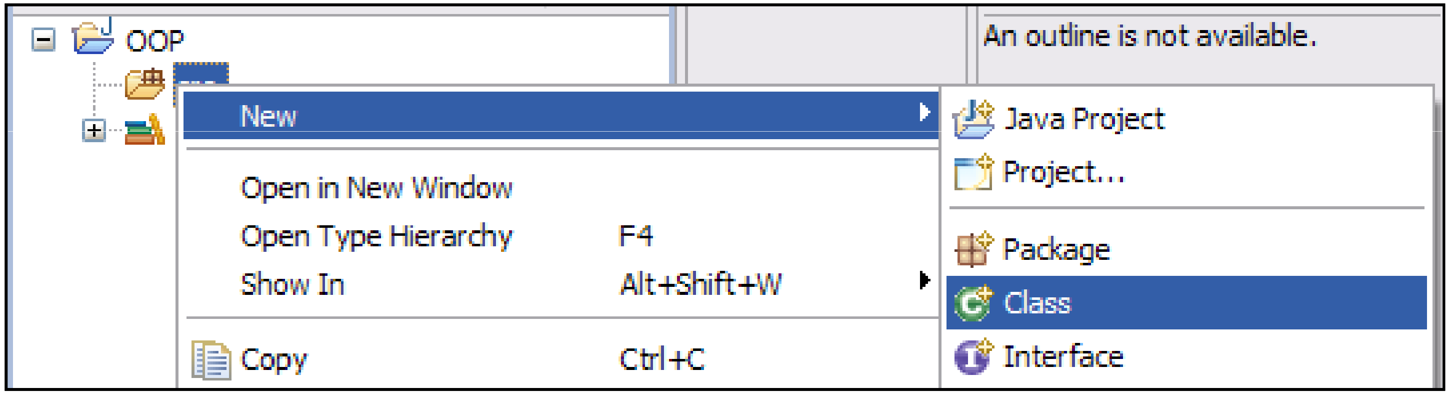
□ Add project to working sets

Working sets:

Eclipse

Create your HelloWorld Class

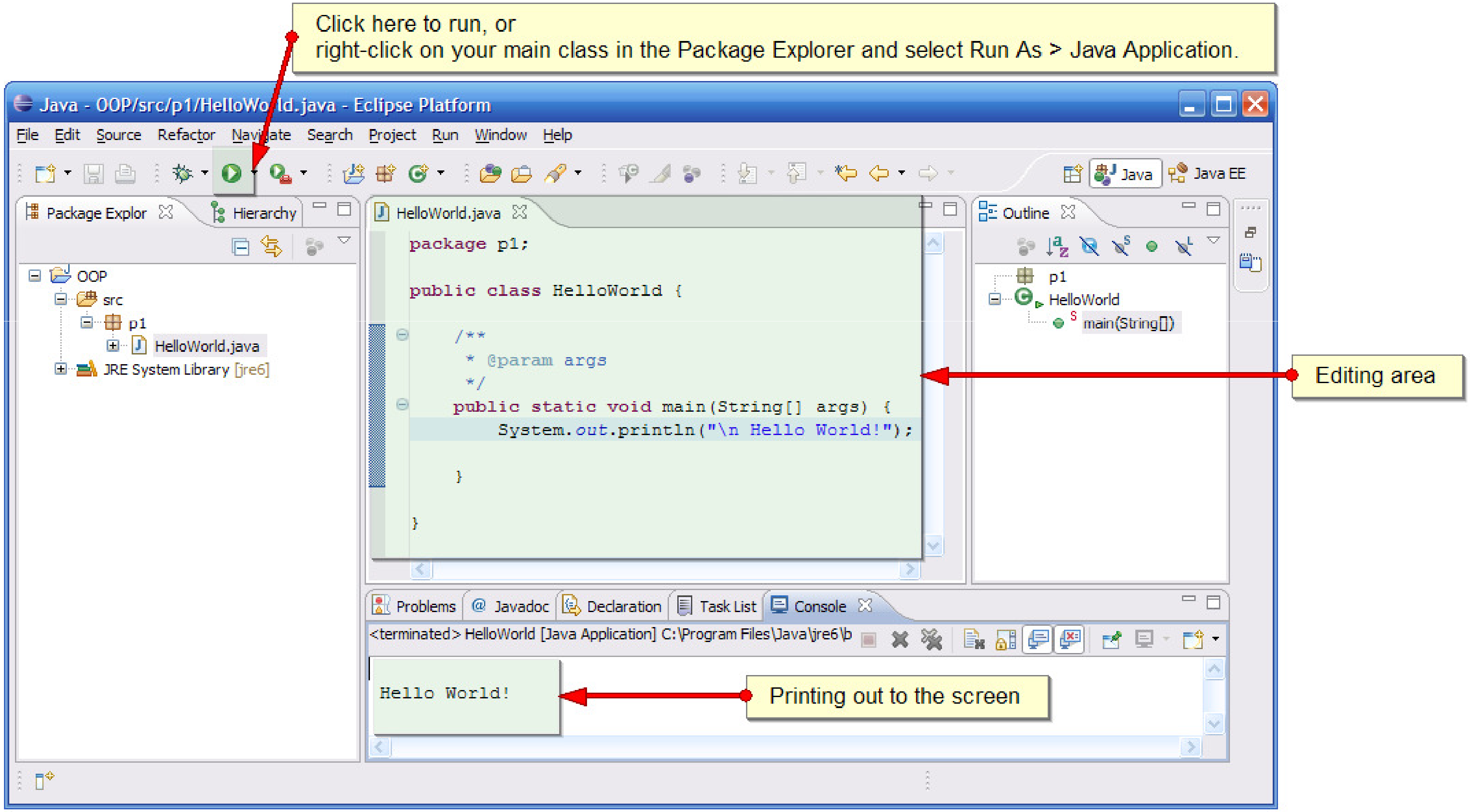
Now, let’s create a new class. In the main toolbar again, click on the **New Java Class button** (or select **File > New > Class**).



Another option is to add the class to the src folder by right-clicking on it in the Package Explorer as shown in the image.

Eclipse

Add a print statement and run your application



Click here to run, or
right-click on your main class in the Package Explorer and select Run As > Java Application.

Java - OOP/src/p1/HelloWorld.java - Eclipse Platform

```
package p1;

public class HelloWorld {

    /**
     * @param args
     */
    public static void main(String[] args) {
        System.out.println("\n Hello World!");

    }

}
```

Editing area

```
<terminated> HelloWorld [Java Application] C:\Program Files\Java\jre6\b

Hello World!
```

Printing out to the screen

User input

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();

System.out.println("The integer was "+i);
```

read user input from console

User input

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();

System.out.println("The integer was "+i);
```

read an integer

```
Scanner sc = new Scanner(System.in);
String line = sc.nextLine();

System.out.println("The string was "+line);
```

read a string of characters

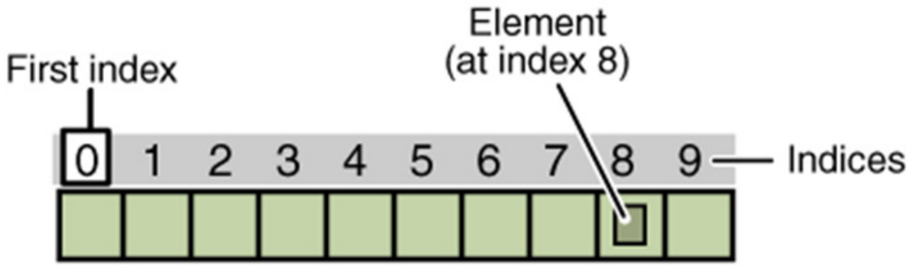
For more info:

http://java.sun.com/javase/6/docs/api/java/util/Scanner.html

Returning char values from a String

```
String word = “abcde”;
char c = word.charAt(3);

System.out.println(c + “is the char at position 4”);
```

This method (function) returns the char value at the specified index.

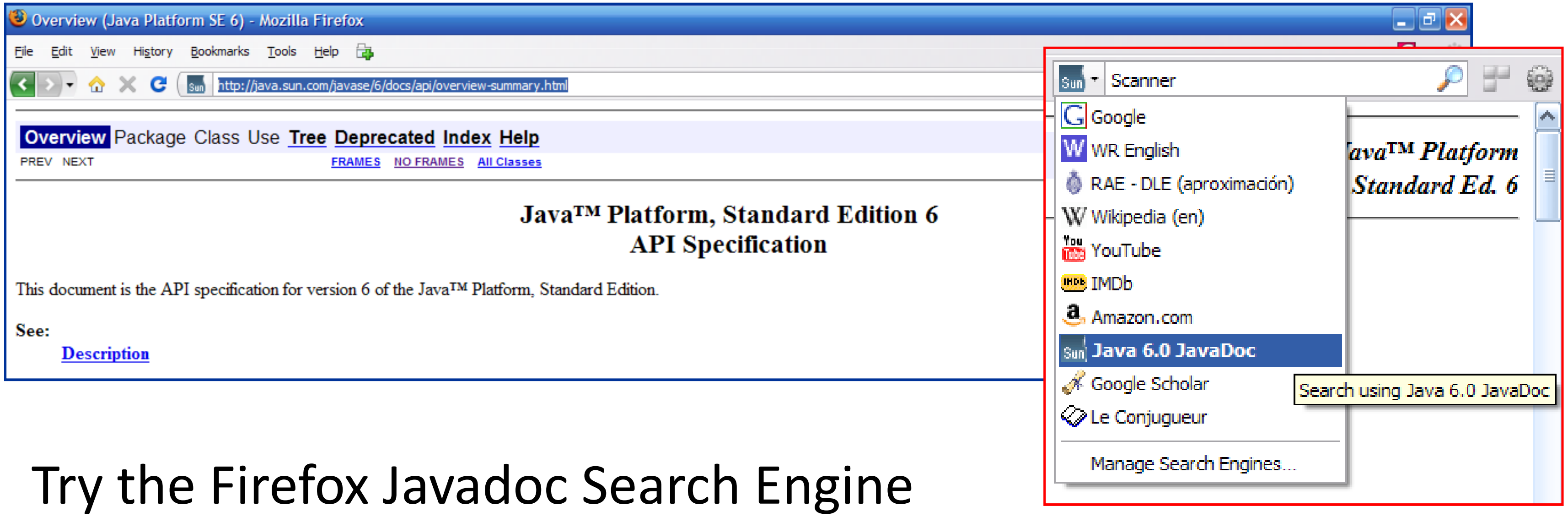


0	**1**	**2**	**3**	**4**
‘a’	‘b’	‘c’	‘d’	‘e’

Considers the index as an array

Some Tips

- Javadoc
  If you want to know more about existing classes check the documentation at:
  http://java.sun.com/javase/6/docs/api/overview-summary.html



Overview (Java Platform SE 6) - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://java.sun.com/javase/6/docs/api/overview-summary.html
Overview Package Class Use Tree Deprecated Index Help
PREV NEXT
FRAMES NO FRAMES All Classes
Scanner
Google
WR English
RAE - DLE (aproximación)
Wikipedia (en)
YouTube
IMDb
Amazon.com
Java 6.0 JavaDoc
Google Scholar
Le Conjugueur
Manage Search Engines...
Search using Java 6.0 JavaDoc
JavaTM Platform
Standard Ed. 6

Java™ Platform, Standard Edition 6
API Specification

This document is the API specification for version 6 of the Java™ Platform, Standard Edition.

See:
Description

- Try the Firefox Javadoc Search Engine
  http://mycroft.mozdev.org/search-engines.html?name=javadoc

Some Tips

- Eclipse – Autocomplete
  CTR + SPACE is an autocompletion shortcut
  - Type **sysout** and press ***CTR + SPACE*** (the autocompletion) and it will be automagically changed to **System.out.println();**
  - Try it Typing **for** and you'll get a skeleton of a for loop.
- Eclipse – Keyboard Shortcuts.
  Of course you know Ctrl+C and Ctrl+V, but how about
  - **Ctrl+F11** to run the application
  - **Ctrl+A** to select all text
  - **Ctrl+i** to correct the Indentation

  and if you don't remember a shortcut
  - **Ctrl+Shift+L** to show the Kew assist

References

- J. Barker, *Beginning Java Objects: From Concepts To Code, Second Edition*, Apress, 2005.
- H.M. Deitel and P.J. Deitel, Java How to Program: Early Objects Version, Prentice Hall, 2009.
- Java SE Tutorials (Last Updated 5/27/2009), which can be found at: http://java.sun.com/docs/books/tutorial
- Code Conventions for the Java Programming Language, available at http://java.sun.com/docs/codeconv/CodeConventions.pdf
- Eclipse Tutorial
  - Eclipse Cheat Sheets
  - http://www.cs.umd.edu/eclipse/EclipseTutorial/project.html