An Introduction to Java

v3.0 - March 2014

Francois de Coligny

INRA - UMR AMAP
Botany and computational plant architecture

Java training - Contents

Introduction

- history
- specificities
- programming environment
- installation

Bases

Object Oriented Programming

Resources

History

James Gosling and Sun Microsystems

- java: May 20, 1995
- java 1 -> Java 8 (i.e. 1.8)
- Oracle since 2010

Specificities

Java is an Object Oriented language

- clean, simple and powerful
- interpreted (needs a virtual machine)
- portable (Linux, Mac, Windows...): "write once, run everywhere"
- static typing (checks during compilation)
- simpler than C++ (memory management, pointers, headers...)

Programming environment

Java environment

- JRE (Java Runtime Environment)

- JDK (Java Development Kit) ●

contains the compiler

Several versions

- Jave SE (Standard Edition)

- Java EE (Enterprise Edition → Web)

- Java ME (Micro Edition)

Editors

- simple editors: Notepad++, TextPad, Scite (syntax coloring...)

- IDEs (Integrated Development Environment):

Eclipse, NetBeans (completion, refactoring...)

Installation

Windows

- download and install the JDK (Java SE 6)
- environment variable

  add the java/bin/ directory at the beginning of the **PATH** variable

  e.g. 'C :/Program Files/Java/jdk1.6.0_21/bin'
- install editor: TextPad or Notepad++

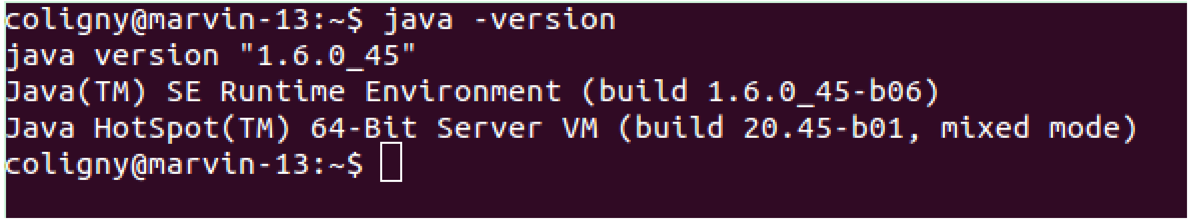
Linux

- sudo apt-get install sun-java6-jdk
- sudo apt-get remove openjdk-6-jdk
- editor: use gedit (default editor under Ubuntu)

  or SciTE: sudo apt-get install scite

Check the installation

- in a terminal: java -version and javac -version

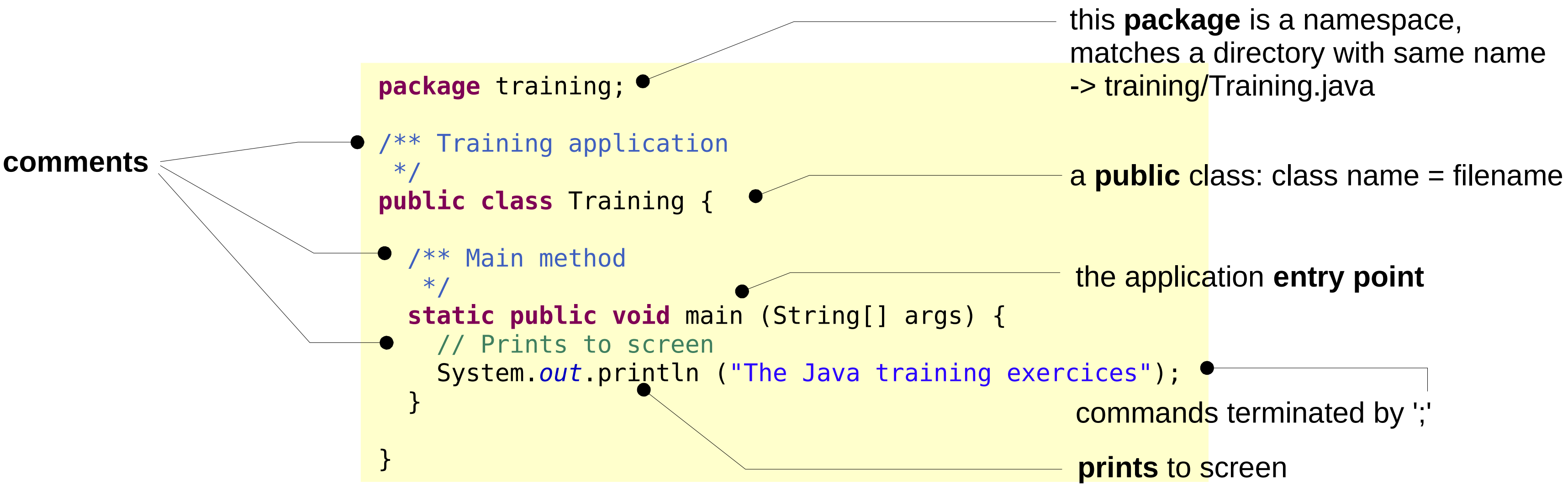


```
coligny@marvin-13:~$ java -version
java version "1.6.0_45"
Java(TM) SE Runtime Environment (build 1.6.0_45-b06)
Java HotSpot(TM) 64-Bit Server VM (build 20.45-b01, mixed mode)
coligny@marvin-13:~$ 
```

Bases

- a Java application
- the development process
- variables, simple types
- operators
- boolean arithmetics
- math
- arrays
- conditions: if else
- loops: while, do... while
- loops: for
- loops: continue or break
- runtime exceptions
- exceptions management

A Java application



```
package training;

/** Training application
 */
public class Training {

  /** Main method
   */
  static public void main (String[] args) {
    // Prints to screen
    System.out.println ("The Java training exercices");
  }

}
```

comments

this **package** is a namespace,
matches a directory with same name
-> training/Training.java

a **public** class: class name = filename

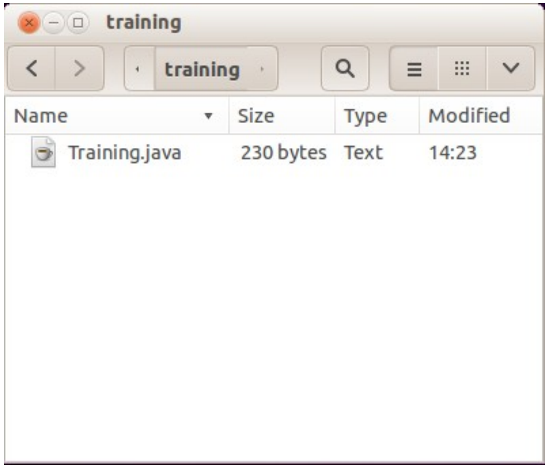
the application **entry point**

commands terminated by ';'

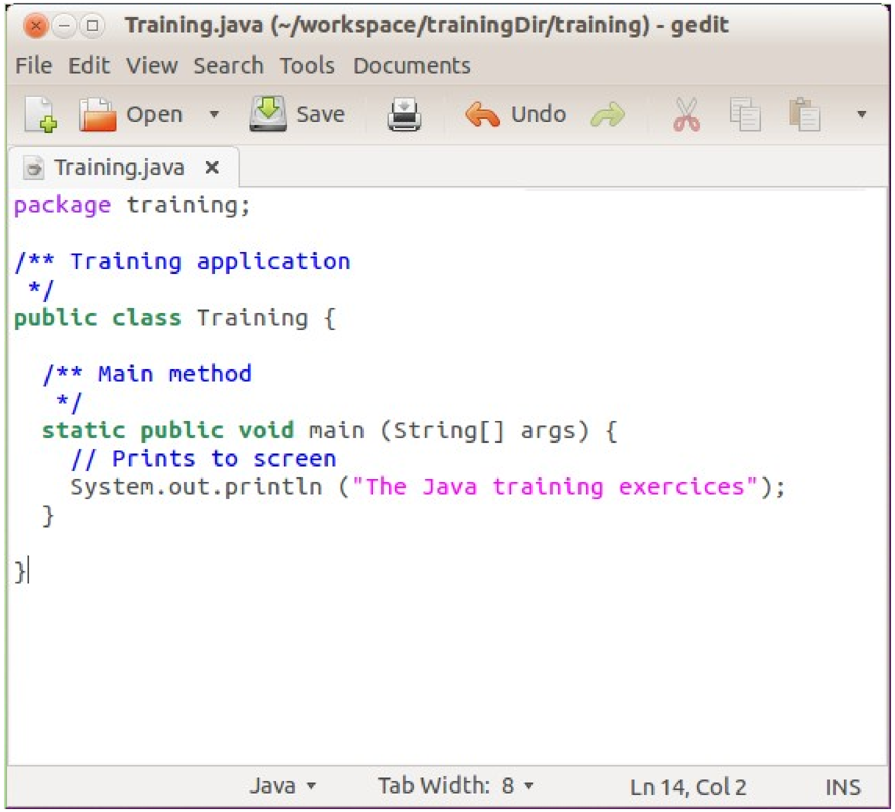
prints to screen

A Java application

- Java programs are written with an editor in files with a **'.java'** extension
- applications are .java files with a **public static void main(...) {...}** method



Name	Size	Type	Modified
Training.java	230 bytes	Text	14:23

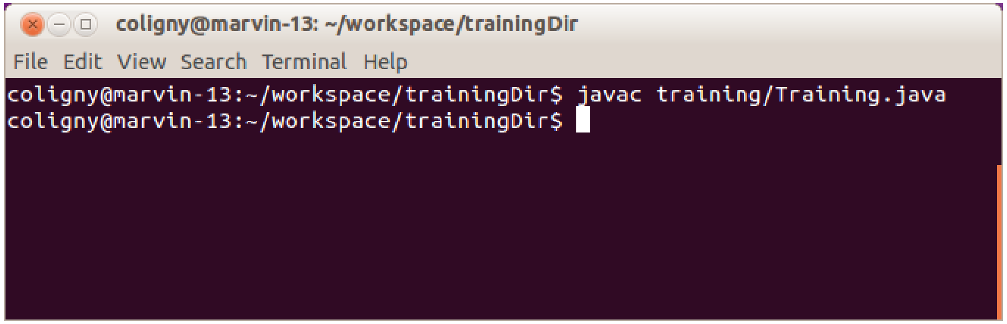

Training.java (~/workspace/trainingDir/training) - gedit
File Edit View Search Tools Documents
Open
Save
Undo
Training.java ×

```
package training;

/** Training application
 */
public class Training {

  /** Main method
   */
  static public void main (String[] args) {
    // Prints to screen
    System.out.println ("The Java training exercices");
  }

}
```

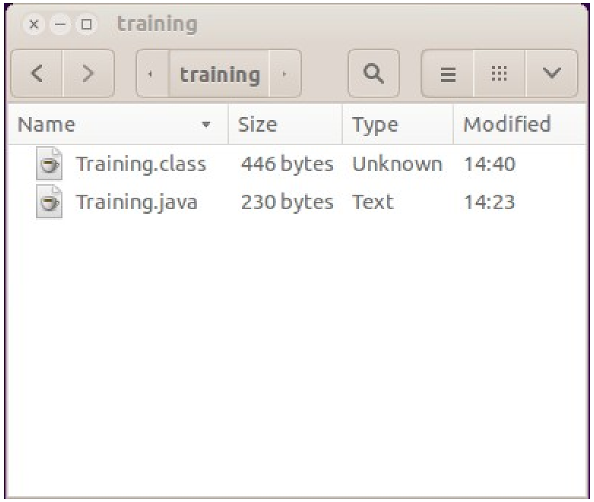
Java
Tab Width: 8
Ln 14, Col 2
INS

A Java application

- to compile a Java application, use the javac compiler (part of the jdk) in a terminal
- returns a Java byte code file: Training.class


coligny@marvin-13: ~/workspace/trainingDir
File Edit View Search Terminal Help

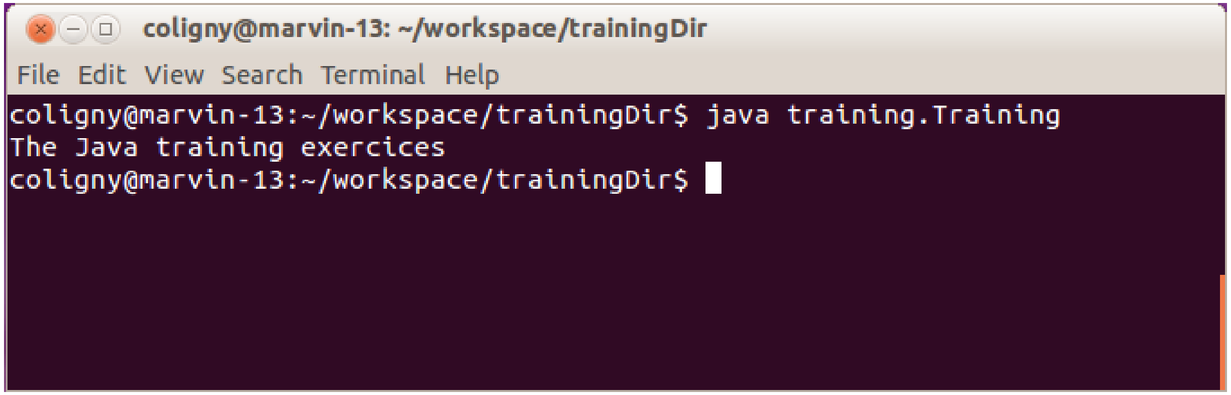
```
coligny@marvin-13:~/workspace/trainingDir$ javac training/Training.java
coligny@marvin-13:~/workspace/trainingDir$ 
```



Name	Size	Type	Modified
Training.class	446 bytes	Unknown	14:40
Training.java	230 bytes	Text	14:23

A Java application

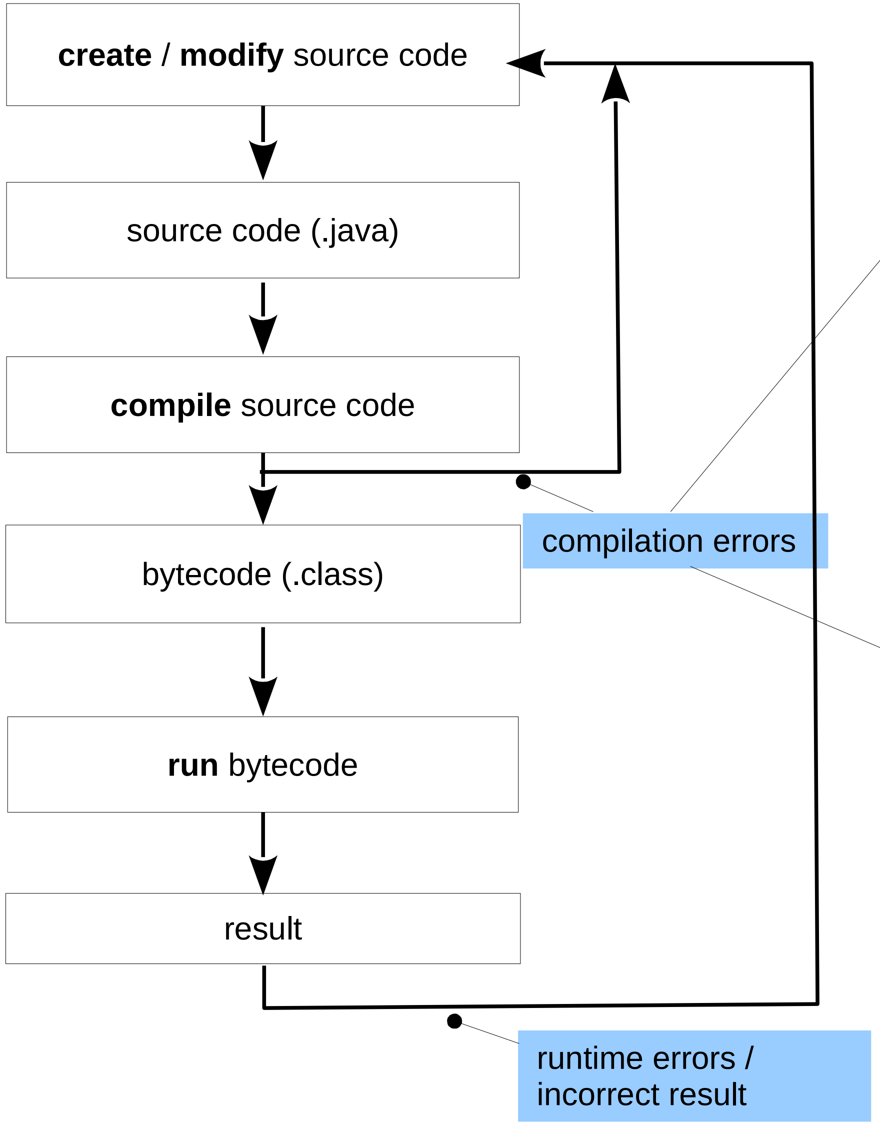
- to run a Java application, use the java interpreter (or Java Virtual Machine, JVM) in a terminal


coligny@marvin-13: ~/workspace/trainingDir
File Edit View Search Terminal Help

```
coligny@marvin-13:~/workspace/trainingDir$ java training.Training
The Java training exercices
coligny@marvin-13:~/workspace/trainingDir$
```

the result

The development process



create / **modify** source code

source code (.java)

compile source code

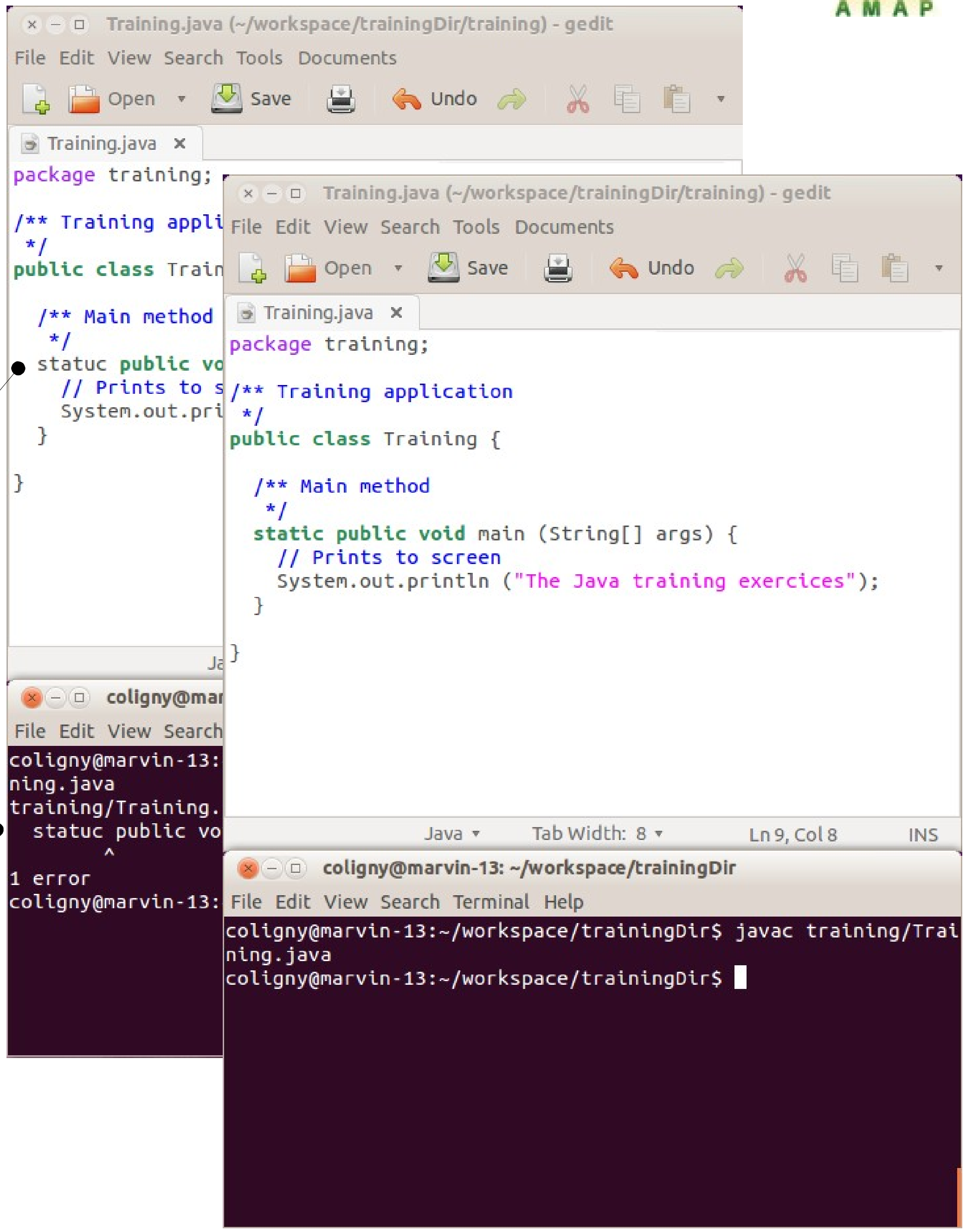
bytecode (.class)

run bytecode

result

compilation errors

runtime errors / incorrect result



Training.java (~/workspace/trainingDir/training) - gedit
File Edit View Search Tools Documents
Open Save Undo
Training.java

```
package training;

/** Training appli
 */
public class Train

  /** Main method
   */
  statuc public vo
    // Prints to s
    System.out.pri
  }

}
```

coligny@mar
File Edit View Search

```
coligny@marvin-13:
ning.java
training/Training.
  statuc public vo
        ^
1 error
coligny@marvin-13:
```

Training.java (~/workspace/trainingDir/training) - gedit
File Edit View Search Tools Documents
Open Save Undo
Training.java

```
package training;

/** Training application
 */
public class Training {

  /** Main method
   */
  static public void main (String[] args) {
    // Prints to screen
    System.out.println ("The Java training exercices");
  }

}
```

Java
Tab Width: 8
Ln 9, Col 8
INS
coligny@marvin-13: ~/workspace/trainingDir
File Edit View Search Terminal Help

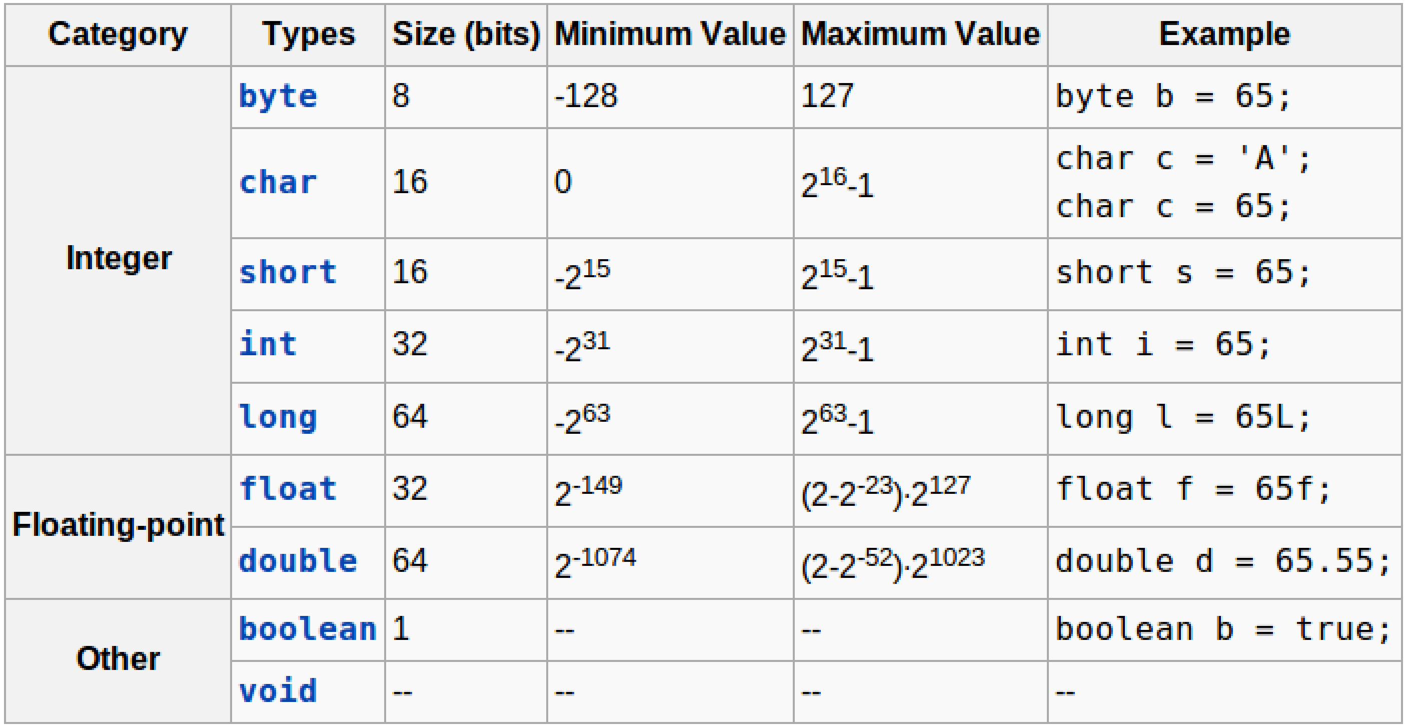
```
coligny@marvin-13:~/workspace/trainingDir$ javac training/Trai
ning.java
coligny@marvin-13:~/workspace/trainingDir$
```

errors fixed, result is correct

Variables, simple types

Variable

- a variable has a **type** and holds a **value**
- a **variable name** starts with a lowercase letter, e.g. myVariable

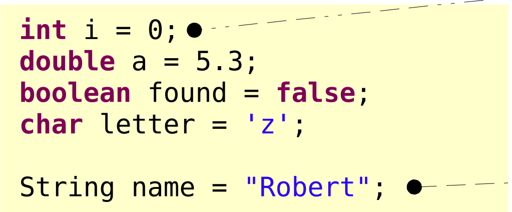


Category	Types	Size (bits)	Minimum Value	Maximum Value	Example
Integer	byte	8	-128	127	`byte b = 65;`

	char	16	0	2^{16}-1	`char c = 'A';`
					`char c = 65;`

	short	16	-2^{15}	2^{15}-1	`short s = 65;`
	int	32	-2^{31}	2^{31}-1	`int i = 65;`
	long	64	-2^{63}	2^{63}-1	`long l = 65L;`
Floating-point	float	32	2^{-149}	$(2-2^{-23})\cdot 2^{127}$	`float f = 65f;`
	double	64	2^{-1074}	$(2-2^{-52})\cdot 2^{1023}$	`double d = 65.55;`
Other	boolean	1	--	--	`boolean b = true;`
	void	--	--	--	--

Declaration



```
int i = 0;
double a = 5.3;
boolean found = false;
char letter = 'z';

String name = "Robert";
```

value assignment

not a simple type (seen later)

Operators

Arithmetic

- simple: **+**, **-**, *****, **/**, **%**
- increment / decrement: ++, --
- combined: **+=**, **-=**, ***=**, **/=**
- precedence with **parentheses**
- comparison: **<**, **<=**, **>**, **>=**, **==**, **!=**
- boolean: **&&**, **||**, **!** (see next slide)

index = index + 2;

i++;

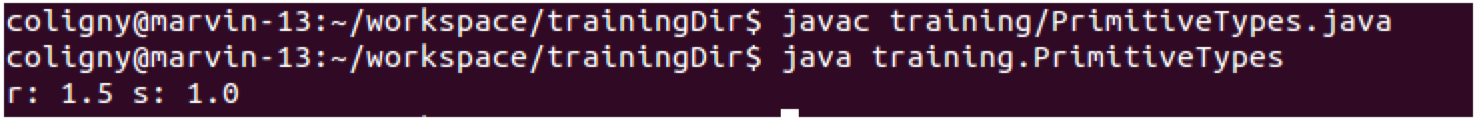
index += 2;

(a + b) * c;

Beware of the int division

String concatenation:
“a string” **+** *something* turns *something* into a String and appends it

```java
double r = 3d / 2d;
double s = 3 / 2;

System.out.println ("r: "+r+" s: "+s);
```



```
coligny@marvin-13:~/workspace/trainingDir$ javac training/PrimitiveTypes.java
coligny@marvin-13:~/workspace/trainingDir$ java training.PrimitiveTypes
r: 1.5 s: 1.0
```

Caution

Boolean arithmetics

Boolean variables are true or false

- boolean v = **true**;
- NOT: **!**
- AND: **&&**
- OR: **||**
- test equality: **==**
- test non equality: **!=**
- use **()** for precedence

```
// Did we find ?
boolean trouble = !found;
```

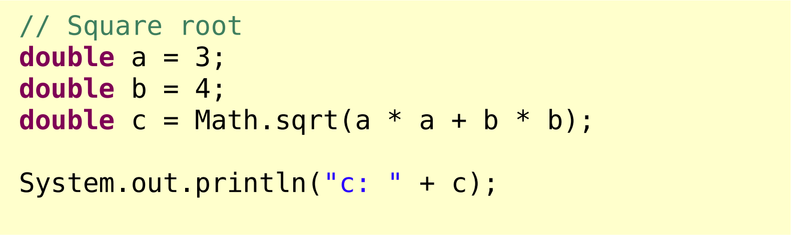
Math

Constants

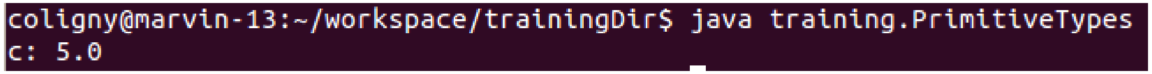
- Math.PI, Math.E

Trigonometry and other operations

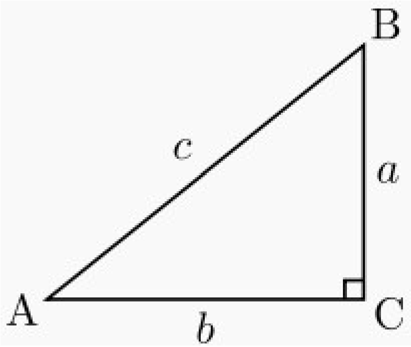
- Math.cos (), Math.sin (), Math.tan ()...
- Math.pow (), Math.sqrt (), Math.abs (), Math.exp (), Math.log ()...
- Math.min (), Math.max (), Math.round (), Math.floor (), Math.ceil ()...
- Math.toDegrees (), Math.toRadians ()...



```java
// Square root
double a = 3;
double b = 4;
double c = Math.sqrt(a * a + b * b);

System.out.println("c: " + c);
```



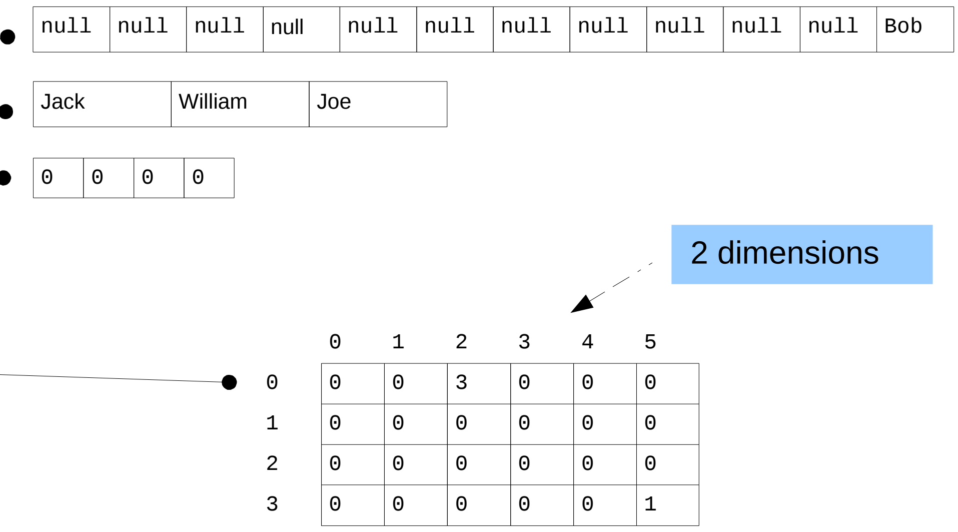
```
coligny@marvin-13:~/workspace/trainingDir$ java training.PrimitiveTypes
c: 5.0
```



Arrays

- 1, 2 or more dimensions arrays
- dynamic allocation: with the **new** keyword
- null if not initialised
- can not be resized
- access elements with the [ ] operator
- indices begin at 0
- size: myArray.length

```
String[] a = new String[12];
a[11] = "Bob";

String[] b = {"Jack", "William", "Joe"};

int size = 4;
double[] c = new double[size];

double[][] d = new double[4][6];
d[0][2] = 3d ;
d[3][5] = 1d ;

// Index error: max is d[3][5]
System.out.println (d[4][6]);
```



null	null	null	null	null	null	null	null	null	null	null	Bob

Jack	William	Joe

0	0	0	0

2 dimensions

	0	1	2	3	4	5
0	0	0	3	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	1

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
        at training.Training.main(Training.java:31)
```

a runtime exception

Conditions: if else

Tests a simple condition

- can be combined

```
// Simple if
if (i == 10) {
  // do something
}

// Complex if
if (count < 50) {
  // do something
} else if (count > 50) {
  // do something else
} else {
  // count == 50
}

// Boolean expression
if (index >= 5 && !found) {
  System.out.println ("Could not find in 5 times");
}
```

Loops: while, do... while

Loop with condition

- while (condition) {...}
- do {...} while (condition);

while:
condition is tested first

```
int count = 0;
while (count < 0) {
  count++;
}

System.out.println ("count: " + count);
```

```
count: 0
```

do... while:
condition is tested at the end
-> always at least one iteration

```
int count = 0;
do {
  count++;
} while (count < 0);

System.out.println ("count: " + count);
```

test is at the end

```
count: 1
```

Loops: for

Loop a number of times

- for (initialisation; stop condition; advance code) {...}

```
// With an array
int[] array = new int[12];
int sum = 0 ;
for (int i = 0; i < array.length; i++) {
  array[i] = i;
  sum += array[i];
}
```

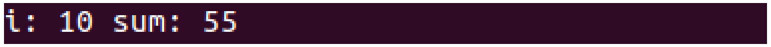
from 0 to 11

sum: 66

Loops: continue or break

```
// Search an array
int sum = 0;
int i = 0;

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

  if (array[i] == 0) continue;

  sum += array[i];

  if (sum > 50) break;

}
System.out.println ("i: " + i+" sum: " + sum);
```

from 0 to 11



```
i: 10 sum: 55
```

- an internal **continue** jumps to the next iteration
- an internal **break** gets out of the loop

- for all kinds of loops (for, while, do while)

Runtime exceptions

Something wrong during the execution

- could not be checked at compilation time

- **e.g.** try to access to an element outside the bounds of an array

  -> java.lang.ArrayIndexOutOfBoundsException

- **e.g.** try to use an array that was not initialised

  -> java.lang.NullPointerException

- **e.g.** try to read a file that could not be found

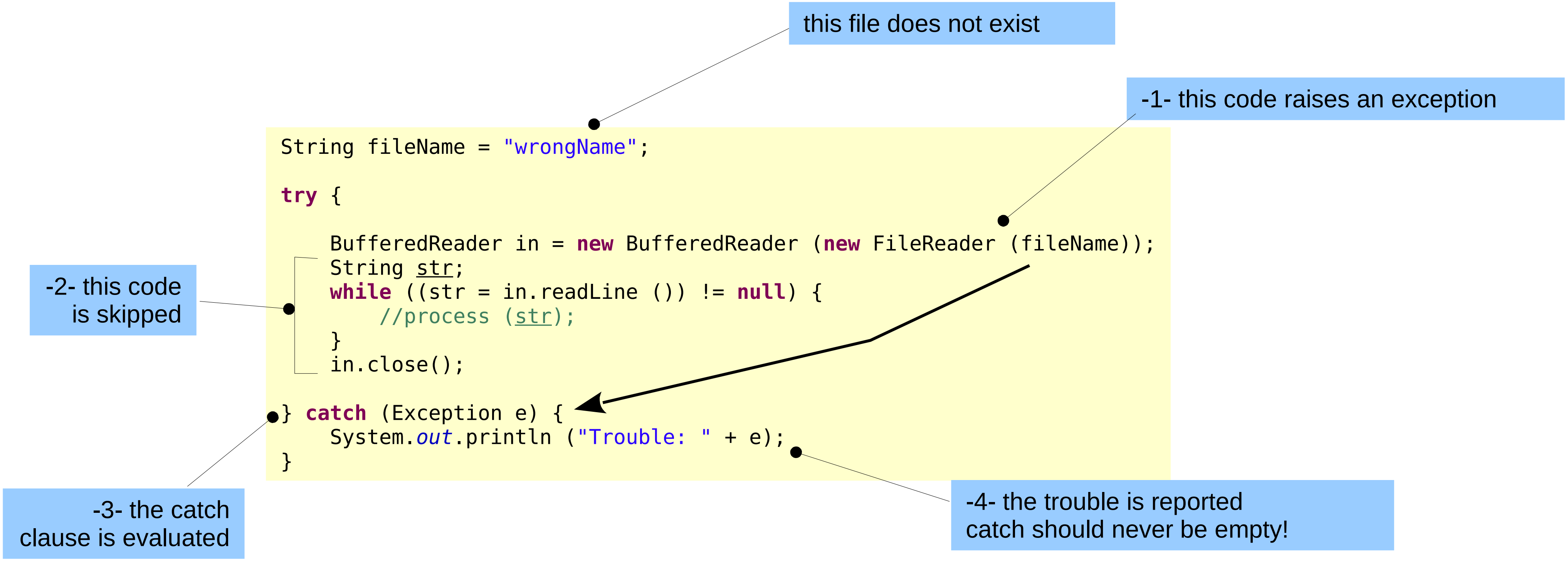
  -> java.io.FileNotFoundException

- exceptions stop the program <u>if not managed...</u>

Exceptions management

Exceptions can be managed everywhere

-> use a try / catch statement



```
String fileName = "wrongName";

try {

    BufferedReader in = new BufferedReader (new FileReader (fileName));
    String str;
    while ((str = in.readLine ()) != null) {
        //process (str);
    }
    in.close();

} catch (Exception e) {
    System.out.println ("Trouble: " + e);
}
```

this file does not exist

-1- this code raises an exception

-2- this code is skipped

-3- the catch clause is evaluated

-4- the trouble is reported
catch should never be empty!

```
Trouble: java.io.FileNotFoundException: wrongName (No such file or directory)
```

Object Oriented Programming

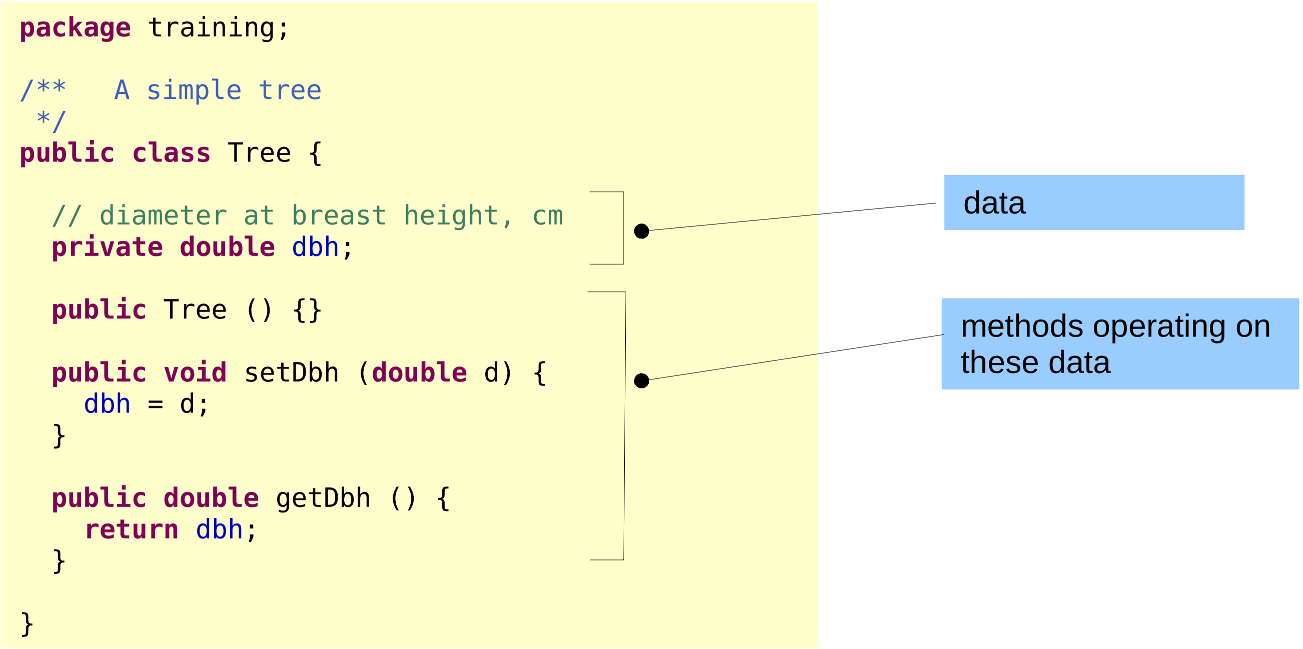
Java is an object oriented language...

- encapsulation
- vocabulary
- class
- properties
- constructor
- instance(s)
- memory management
- inheritance
- specific references
- constructors chaining
- method
- method overloading / overriding
- calling methods
- static method and variable
- static initializer
- interface
- abstract class
- The 'Object' superclass
- enums
- nested class
- polymorphism
- the instanceof operator
- cast
- packages and import
- java reserved keywords
- java modifiers

Encapsulation

Bundle data and methods operating on these data in a unique container:
-> the object

Hide the implementation details to the users of the object, they only know its 'interface'



```
package training;

/**   A simple tree
 */
public class Tree {

  // diameter at breast height, cm
  private double dbh;

  public Tree () {}

  public void setDbh (double d) {
    dbh = d;
  }

  public double getDbh () {
    return dbh;
  }

}
```

data

methods operating on these data

Vocabulary

Class
- a class = a new data type
- source files describe classes, i.e. object 'templates'

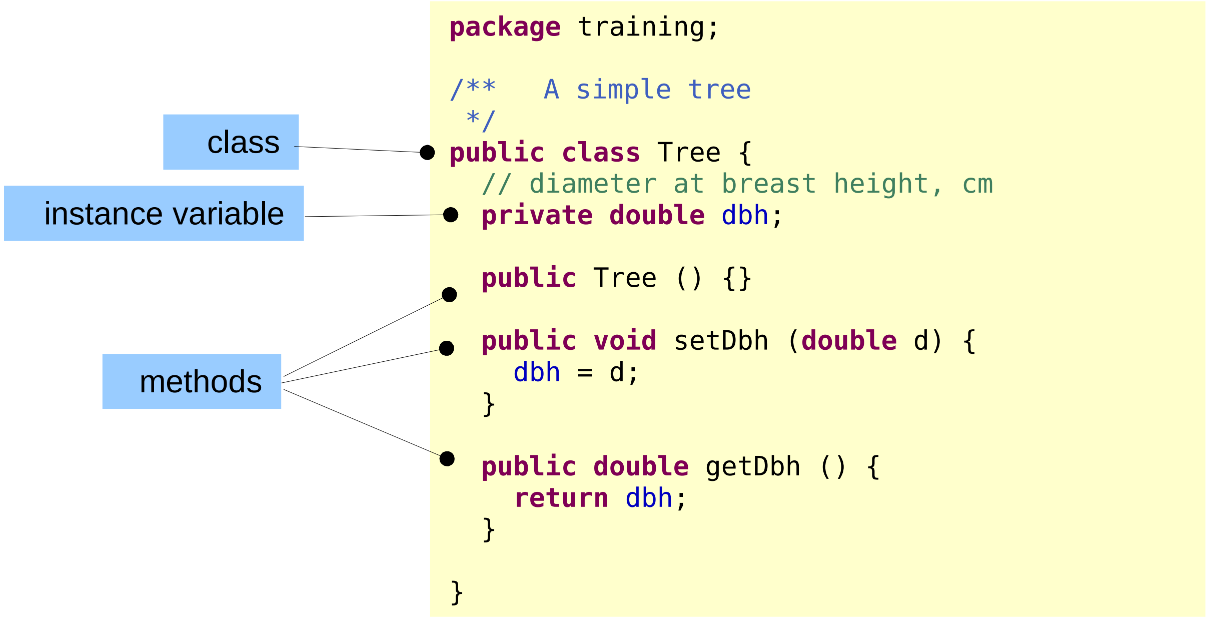
Object
- instance of a class at runtime
- memory allocation
- several objects may be build with the same class

Instance variable (iv)
- main variables of an object
- (field, attribute, member data)

Method
- function of an object
- (procedure, member function)

Property
- instance variable or method

Class



```
package training;

/**   A simple tree
 */
public class Tree {
  // diameter at breast height, cm
  private double dbh;

  public Tree () {}

  public void setDbh (double d) {
    dbh = d;
  }

  public double getDbh () {
    return dbh;
  }

}
```

A class is a new data type

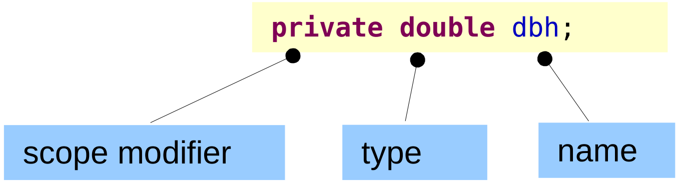
e.g. int, double, float, boolean, String, Tree...

Scope modifiers for the properties

- **public** : visible by all (interface)
- **protected** : visible in the package (and in later seen subclasses...)
- **private** : scope is limited to the class (hidden to the others)

Properties

Instance variable



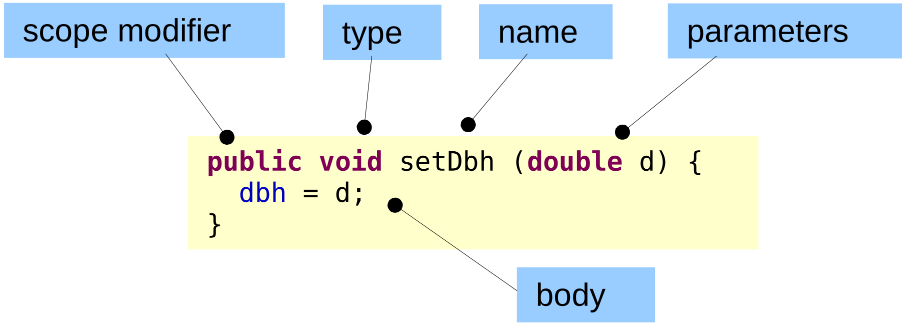
```
private double dbh;
```

scope modifier

type

name

Method



scope modifier

type

name

parameters

```
public void setDbh (double d) {
  dbh = d;
}
```

body

A rule:

parentheses after the name => it is a method

Constructor

- **particular method** called at object creation time
- **same name** than the class (starts with an uppercase letter)
- **no return type**
- deals with instance variables **initialisation**
- **several** constructors may coexist if they have different parameter types

```java
package training;

/**   A simple tree
 */
public class Tree {
  // diameter at breast height, cm
  private double dbh;

  public Tree () {}

  public Tree (double d) {
    dbh = d;
  }

  public void setDbh (double d) {
    dbh = d;
  }

  public double getDbh () {
    return dbh;
  }

}
```

a default constructor (no parameter)

another constructor (takes a parameter)

regular method with a parameter

Notes:
this default constructor does nothing particular
-> dbh is a numeric instance variable
-> set to 0 automatically
the other constructor initializes dbh

Instance

Vocabulary:
object = instance

Vocabulary:
the properties of the object
the properties of the class
-> instance variables + methods

Instanciation

- creates an instance of a given class
- i.e. an <u>object</u>

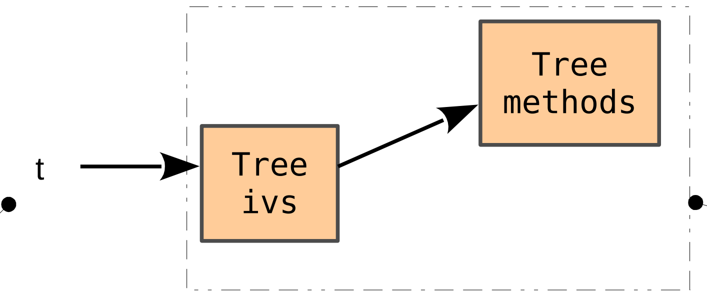
-1- declaration of a reference
type + name
no object created yet

```
// make an instance of Tree
Tree t;
t = new Tree ();

// same than
Tree t = new Tree ();
```

-2- creation of the object
new => instanciation
class name = constructor name

What happens in memory

- <u>new</u> -> instanciation = <u>memory reservation for the instance variables + the methods</u>
- the <u>constructor</u> is called (initialisations)
- returns a <u>reference</u> to the created object
- we assign it to the reference named 't'



a reference
to use the object

a Tree object in memory:
instance variables
+ methods

Instances

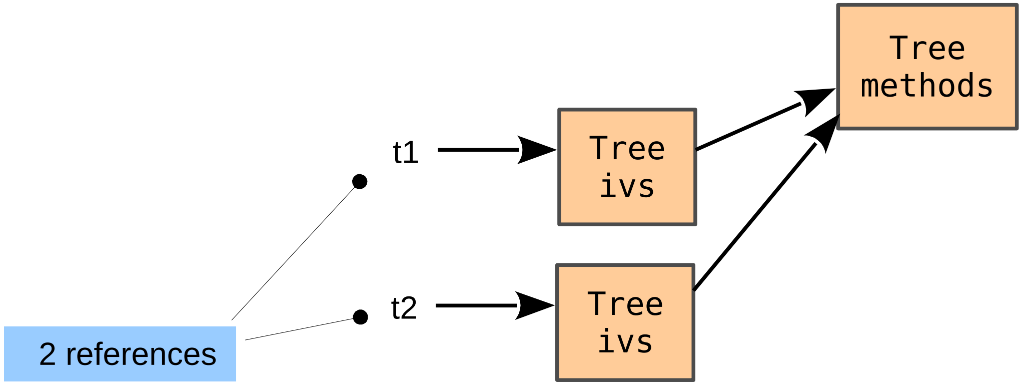
Creation of several objects

```
// create 2 trees
Tree t1 = new Tree ();

Tree t2 = new Tree ();
```

2 times new -> 2 objects

What happens in memory

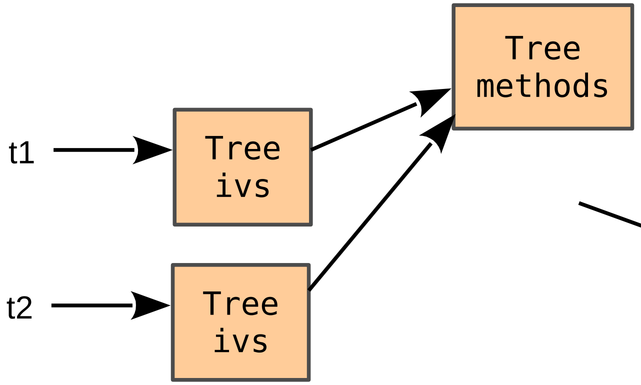
- 2 times new: 2 memory reservations for the instance variables of the 2 objects (their dbh may be different)
- the constructor is called for each object
- the methods of the 2 objects are shared in memory
- each new returns a reference to the corresponding object
- we assign them to 2 different references named 't1' and 't2'



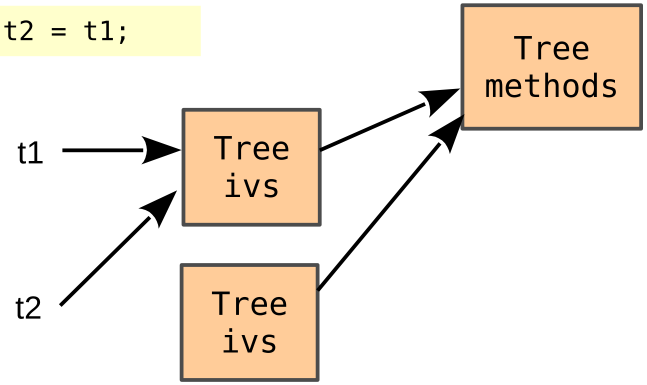
Instances

Using the references

```java
// Create 2 trees
Tree t1 = new Tree ();

Tree t2 = new Tree ();
```

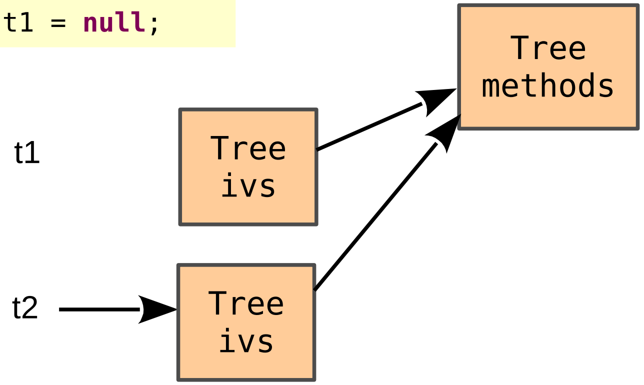


```java
t2 = t1;
```



- both t1 and t2 point to the first tree
- the second tree is 'lost'

```java
t1 = null;
```



- t1 points to nothing
- t2 points to the second tree
- the first Tree is 'lost'

Memory management

- objects are instantiated with the keyword **new** -> memory allocation

- objects are **destroyed** when there is no more reference on them -> garbage collecting

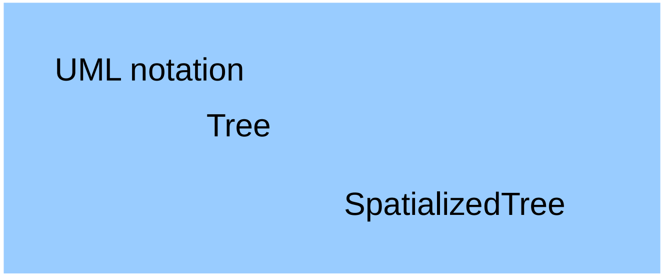
  -> this process is automatic

  -> to help remove a big object from memory, set all refs to null

```
// declare two references
Tree t1 = null;
Tree t2 = null;

// create an object (instanciation)
t1 = new Tree ();

// the object can be used
double v = t1.getDbh ();

// assignment
t2 = t1;
...

// set both references to null
t1 = null;
t2 = null;  // the object will be destroyed by the garbage collector
```

no object created yet

Inheritance


UML notation

Tree

SpatializedTree

How to create a spatialized tree ?

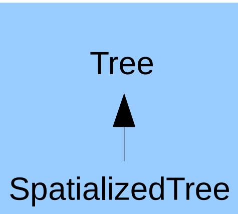
Simple manner results in **duplicates**...

```java
package training;

/**   A simple tree
 */
public class Tree {
  // diameter at breast height, cm
  private double dbh;

  public Tree () {}

  public void setDbh (double d) {
    dbh = d;
  }

  public double getDbh () {
    return dbh;
  }

}
```

```java
package training;

/**   A tree with coordinates
 */
public class SpatializedTree {
  // diameter at breast height, cm
  private double dbh;
  // x, y of the base of the trunk (m)
  private double x;
  private double y;

  /** Default constructor
   */
  public SpatializedTree () {
    setXY (0, 0);
  }

  public void setDbh (double d) {
    dbh = d;
  }

  public double getDbh () {
    return dbh;
  }

  public void setXY (double x, double y) {
    this.x = x;
    this.y = y;
  }

  public double getX () {return x;}
  public double getY () {return y;}

}
```

Inheritance

UML notation



Reuse a class to make more specific classes
- e.g. a tree with coordinates
- inheritance corresponds to a **'is a' relation**
- a **subclass** has all the data and methods of its parent: the **superclass**
- all classes inherit from the **Object** class
- multiple inheritance is not allowed in Java

a spatialized tree **is a** tree (with coordinates)

superclass

```
package training;

/**   A simple tree
 */
public class Tree {
  // diameter at breast height, cm
  private double dbh;

  public Tree () {}

  public void setDbh (double d) {
    dbh = d;
  }

  public double getDbh () {
    return dbh;
  }

}
```

subclass

```
package training;

/**   A tree with coordinates
 */
public class SpatializedTree extends Tree {
  // x, y of the base of the trunk (m)
  private double x;
  private double y;

  /** Default constructor
   */
  public SpatializedTree () {
    super ();
    setXY (0, 0);
  }

  public void setXY (double x, double y) {
    this.x = x;
    this.y = y;
  }

  public double getX () {return x;}
  public double getY () {return y;}

}
```

inheritance keyword

calls constructor of the superclass

new methods

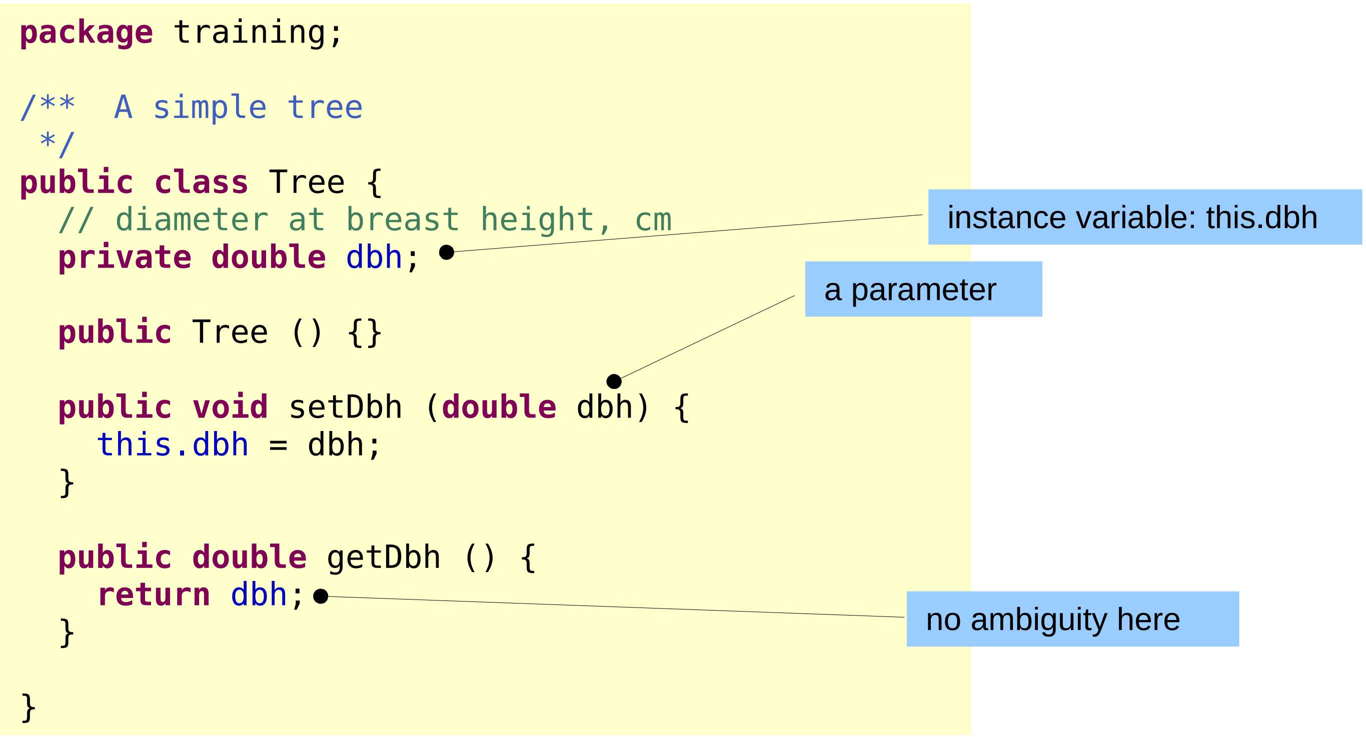
```
// SpatializedTree
SpatializedTree t3 = new SpatializedTree ();

t3.setDbh (15.5);
t3.setXY (1, 5);

double d = t3.getDbh ();  // 15.5
double x = t3.getX ();    // 1
```

inherited methods

Specific references

A keyword for the reference to the current class: this

- to remove ambiguities



```java
package training;

/**  A simple tree
 */
public class Tree {
  // diameter at breast height, cm
  private double dbh;

  public Tree () {}

  public void setDbh (double dbh) {
    this.dbh = dbh;
  }

  public double getDbh () {
    return dbh;
  }

}
```

instance variable: this.dbh

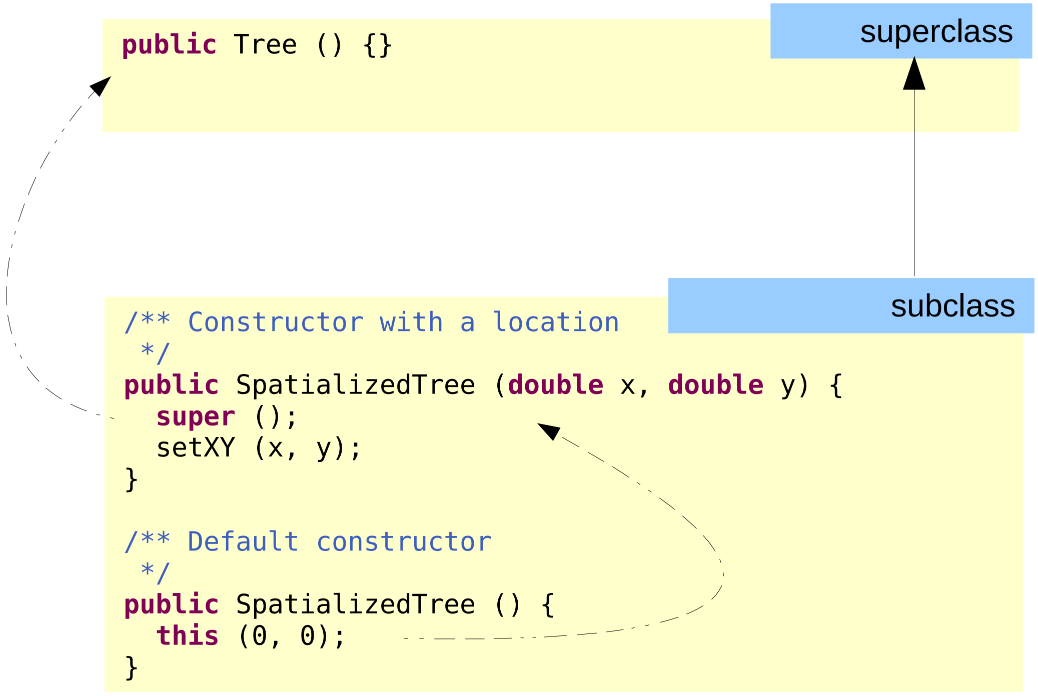
a parameter

no ambiguity here

A keyword for the reference to the superclass: super

Constructors chaining

- **chain the constructors** to avoid duplication of code



```
public Tree () {}
```

```
/** Constructor with a location
 */
public SpatializedTree (double x, double y) {
  super ();
  setXY (x, y);
}

/** Default constructor
 */
public SpatializedTree () {
  this (0, 0);
}
```

```
new Tree ();
// calls Tree ()

new SpatializedTree (1, 5);
// calls SpatializedTree (x, y)
// calls Tree ()

new SpatializedTree ();
// calls SpatializedTree ()
// calls SpatializedTree (x, y)
// calls Tree ()
```

Method

Classes contain instance variables and methods

- a class can contain several methods
- if no parameters, use **()**
- if no return type, use **void**

```
package training;

/**   A tree with coordinates
 */
public class SpatializedTree extends Tree {
  // x, y of the base of the trunk (m)
  private double x;
  private double y;

  /** Default constructor
   */
  public SpatializedTree () {
    super ();
    setXY (0, 0);
  }

  public void setXY (double x, double y) {
    this.x = x;
    this.y = y;
  }

  public double getX () {return x;}
  public double getY () {return y;}

}
```

constructors are particular methods without a return type

setXY () method: 2 parameters

getSomething () is an **accessor**
returns something

Method overloading / overriding

Overload ("surcharge")
- in the same class
- several methods with same name
- and <u>different parameter types</u>

BiomassCalculator

```
public double calculateBiomass (Tree t) {
  return t.getTrunkBiomass ();
}

public double calculateBiomass (TreeWithCrown t) {
  return t.getTrunkBiomass () + t.getCrownBiomass ();
}
```

Override ("redéfinition")
- in a class and a subclass
- several methods with <u>same signature</u>
  i.e. same name and parameter types

superclass

```
public double getVolume () {
  return trunkVolume;
}
```

subclass

```
@Override
public double getVolume () {
  return trunkVolume + crownVolume;
}
```

optional:
tell the compiler
-> it will check

e.g. if TreeWithCrown **extends** Tree

Calling methods

Method returning void

reference.method (parameters);

Method returning something

returnType = reference.method (parameters);

```
package training;

/**   A simple tree
 */
public class Tree {
  // diameter at breast height, cm
  private double dbh;

  public Tree () {}

  public void setDbh (double d) {
    dbh = d;
  }

  public double getDbh () {
    return dbh;
  }

}
```

```
// Create a tree
Tree t1 = new Tree ();

// Set its diameter
t1.setDbh (12.5);

// Print the diameter
double d1 = t1.getDbh ();

System.out.println ("t1 dbh: " + d1);
```

System is a class
out is a static public instance variable, type PrintStream
println () is a method of PrintStream

writing in out writes on the 'standard output'

Static method and variable

A method at the class level: no access to the instance variables

- like the Math methods: Math.cos ()...
- to reuse a block of code
- uses only its parameters
- returns a result

example: in class **Tree**

```
/**
 * Quadratic diameter
 */
public static double calculate_dg (double basalArea, int numberOfTrees) {
    return Math.sqrt (basalArea / numberOfTrees * 40000d / Math.PI);
}
```

- basalArea and numberOfTrees are the parameters
- their names have a local scope: they are only available in the method

```
double dg = Tree.calculate_dg (23.7, 1250);
```

ClassName.method (parameters)

A common variable shared by all the instances of a class

- can be a constants: Math.PI

```
public static final double PI = 3.14...;
```

- can be a variable

```
public static int counter;
```

Static initializer

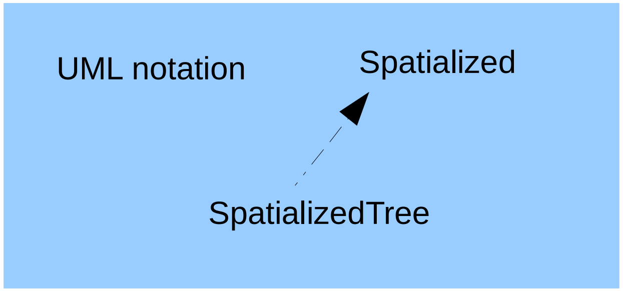
A code executed just once: **first new Species ()**

```
public class Species {

    public static FileLoader speciesLoader;

    // static initializer: load a file
    static {
        speciesLoader = Species.loadSpeciesFile ();
    }

    public static void loadSpeciesFile () {
        ...
    }


    // Constructor
    public Species (...) {
        ...
    }
}
```

executed once before the first constructor call

```
Species sp1 = new Species (...); // the species file is loaded
Species sp2 = new Species (...);
Species sp3 = new Species (...);
```

Interface



UML notation

Spatialized

SpatializedTree

A particular kind of class

- a list of methods without a body
- a way to **make sure** a class implements a set of methods
- a kind of **contract**
- classes extend other classes
- classes **implement** interfaces
- implementing several interfaces is possible

```java
public interface Spatialized {

  public void setXYZ (double x, double y, double z);
  public double getX ();
  public double getY ();
  public double getZ ();

}
```

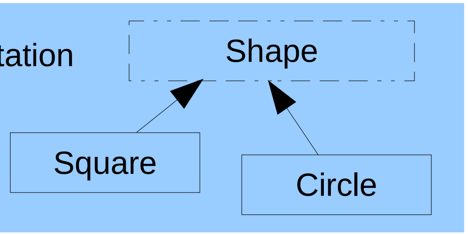
no method body in the interface

```java
/**   A tree with coordinates
 */
public class SpatializedTree extends Tree implements Spatialized {
  ...

  public void setXYZ (double x, double y, double z) {
    this.x = x;
    this.y = y;
    this.z = z;
  }

  public double getX () {return x;}
  public double getY () {return y;}
  public double getZ () {return z;}

}
```

an **implementation is required** for the methods in the subclasses

Abstract class

UML notation



An <u>incomplete</u> superclass with common methods

- class 'template' containing **abstract methods** to be implemented in all subclasses
- can also have regular methods (unlike an interface)
- each subclass implements the abstract methods
- <u>can not be instanciated</u> directly

```
public abstract class Shape {
    private String name;
    ...
    public String getName () {return name;}
    public abstract double area (); // m2

}
```

an **abstract class** (at least one abstract method):
can not be instanciated

a regular method

an **abstract method**: no body

```
public class Square extends Shape {
    private double width;  // m
    ...
    @Override
    public double area () {
        return width * width;
    }
}
```

two subclasses:
they **implement** the abstract method

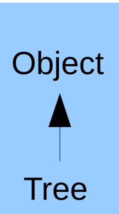
```
public class Circle extends Shape {
    private double radius;  // m
    ...
    @Override
    public double area () {
        return Math.PI * radius * radius;
    }
}
```

```
// Example
Shape sh = new Shape (); // ** Compilation error

Square s = new Square (”square 1”, 10);

Circle c = new Circle (”circle 1”, 3);

String name1 = s.getName (); // square 1

double a1 = s.area (); // 100
double a2 = c.area (); // 28.27
```

The 'Object' superclass

If no 'extends' keyword...
...then the class extends Object
-> All classes extend Object

UML notation



note: native methods have a body in native language (e.g. C)
-> they are not abstract

extends Object

```
package training;

/**   A simple tree
 */
public class Tree {
  // diameter at breast height, cm
  private double dbh;

  public Tree () {}

  public void setDbh (double d) {
    dbh = d;
  }

  public double getDbh () {
    return dbh;
  }

  public String toString () {
    return “Tree dbh: “ + dbh;
  }

}
```

a superclass for all classes

```
package java.lang;

public class Object {

  public final native Class<?> getClass();

  public native int hashCode();

  public boolean equals(Object obj) {
    return (this == obj);
  }

  protected native Object clone() throws
      CloneNotSupportedException;

  public String toString() {
    return getClass().getName() + "@" +
        Integer.toHexString(hashCode());
  }

  (...)

}
```

all these methods can be called on all objects

```
// Tree
Tree t = new Tree ();
t.setDbh (14.5);
System.out.println (“” + t);
```

appended to a String: i.e. t.toString ()

```
training.tree.Tree@37dd7056
```

toString () can be overriden for a better result

```
Tree dbh: 14.5
```

Enum

Another particular kind of class: a type for enumerations

- an enum is a type with a limited number of value

Declaration

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
```

An example of use

```
private Day day;
...

day = Day.SUNDAY;
...
```

Nested class

A class within another class

- may be static (no access to the instance variables)
- can be a member class (like a method)
- can be a local class (in a method)
- can be an anonymous class (on the fly)

```java
public class ScriptRaphael2013FileLoader extends FileLoader {
    public String parameterFileName;
    public int plantAge;
    public int plantSeed;
    public String archimed1ConfigFileName;
    public String pattern;
    public String patternBounds;

    public List<SimRecord> simRecords;

    // A line in the command file
    static public class SimRecord extends Record {
        public String simName;
        public String simulationParameters;

        public SimRecord(String line) throws Exception {
            super(line);
        }
    }

    public static void main(String[] args) {

        FileLoader l = new ScriptRaphael2013FileLoader();
        String report = l.load("someFileName.txt");

        System.out.println(l.toString());

    }

}
```

someFileName.txt

```
# Simeo-Principes command file

parameterFileName = PhoenixRomana.txt

plantAge = 200
plantSeed = 1

archimed1ConfigFileName = archimed1.config

# Pattern coordinates in m
pattern = ((0,10);(10,0))
patternBounds = ((-5,-5);(15,15))

# simNamesimulationParameters
sim1        PinnaeDistance_1(2D;1;(0:2.3);(50:10);(100:2))
sim2        PinnaeDistance_1(2D;1;(0:6);(50:15);(100:4)) ! FrondNervureLength_1(2D;1;(1:100);(175:400))
```

Note:

Nested classes are not often used
-> No more details here

Polymorphism

Write generic code to be executed with several types

- more abstract and general implementations

```
public abstract class Shape {

    public abstract double area ();  // m2

}
```

```
public class Square extends Shape {
    private double width;  // m
    ...
    @Override
    public double area () {
        return width * width;
    }
}
```

```
public class Circle extends Shape {
    private double radius;  // m
    ...
    @Override
    public double area () {
        return Math.PI * radius * radius;
    }
}
```

```
private float totalArea (Shape[] a) {
  double sum = 0;
  for (int i = 0; i < a.length; i++) {

     // the program knows what method to call
     sum += a[i].area ();

  }
  return sum;
}
```

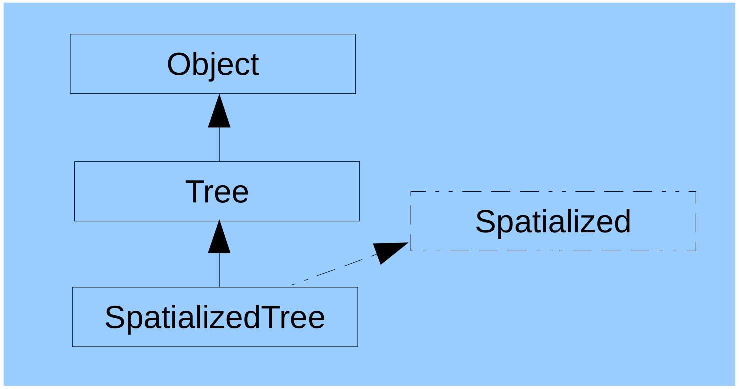
this code is generic
works with all shapes

several classes, all Shapes

Example of use

```
Shape[] a = {new Square (5), new Circle (3), new Square (10)};

float total = totalArea (a);
```

The instanceof operator



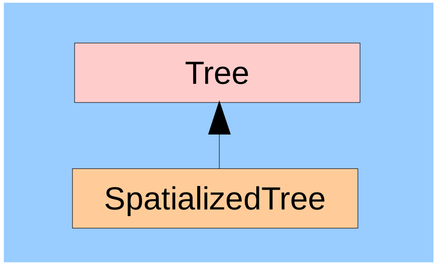
All classes inherit the Object class

- **instanceof** checks the type of an object

```
SpatializedTree t1 = new SpatializedTree ();

if (t1 instanceof SpatializedTree) ...  // true
if (t1 instanceof Tree) ...             // true
if (t1 instanceof Object) ...           // true

if (t1 instanceof Spatialized) ...  // true


Tree t2 = new Tree ();

if (t2 instanceof Tree) ...             // true
if (t2 instanceof SpatializedTree) ...  // false
```

also with an interface

Cast



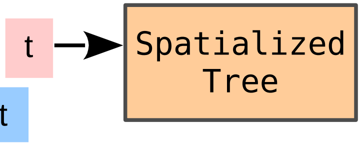
In an inheritance graph

- a reference can have any supertype of the real type

```
Tree t = new SpatializedTree ();
```

type of the reference

real type of the object

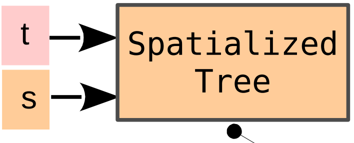


- we can only use the methods the reference knows

```
t.setDbh (10);    // ok
t.setXY (2, 10);  // ** compilation error: Tree does not define setXY ()
```

- to access the methods of the real type, we can create another reference

```
SpatializedTree s = (SpatializedTree) t;  // cast: creates another reference

s.setXY (2, 1);  // ok: SpatializedTree does define setXY ()
```



same object

- a check can be done with instanceof before the cast

```
if (t instanceof SpatializedTree) {
  SpatializedTree s = (SpatializedTree) t;
  ...
}
```

- cast can also be used for numbers

```
double d = 12.3;
int i = (int) d; // 12
```

Packages and import

Packages

- namespaces to organize the developments: groups of related classes
- first statement in the class (all lowercase)
- match directories with the same names

e.g.

- **java.lang**: String, Math and other basic Java classes
- **java.util**: List, Set... (see below)
- **training**: Tree and SpatializedTree

The package is part of the class name: java.lang.String, training.Tree

Import

- to simplify notation, import classes and packages

instead of:

```
training.Tree t = new training.Tree ();
```

write:

```
import training.Tree;
...
Tree t = new Tree ();
```

Java reserved keywords

abstract
boolean
break
byte
case
cast
catch
char
class
const
continue
default
do
double
else
enum
extends
false
final
finally
float
for
goto (unused)
if
implements
import
instanceof
int
interface
long
native
new
null
package
private
protected
public
return
short
static
super
switch
synchronized
this
throw
throws
transient
true
try
void
volatile
while

Java modifiers

a final field cannot be changed e.g. Math.PI

a final class can not be subclassed

a final method can not be overriden

	class	interface	field	method	initializer	variable
abstract	X	X		X		
final	X		X	X		X
native				X		
none (package)	X	X	X	X		
private			X	X		
protected			X	X		
public	X	X	X	X		
static	X		X	X	X	
synchronized				X		
transient			X			
volatile			X			

Resources

- a focus on the collection framework
- the Collection interface
- arrayList
- hashSet
- maps
- the tools in the Collections class
- how to iterate on objects in collections
- how to iterate on objects in maps
- generics
- online documentation
- online documentation: javadoc
- online documentation: tutorials
- links to go further

A focus on the collection framework

A collection is like an array, but without a size limitation

- contains references

- may have distinctive features
    - a **list** keeps insertion order
    - a **set** contains no duplicates and has no order

- the 8 simple types (int, double, boolean...) are not objects -> need a **wrapper object**

    Byte, Short, Integer, Long, Float, Double, Boolean, Character

    java helps: **Integer i = 12;** (autoboxing / unboxing)

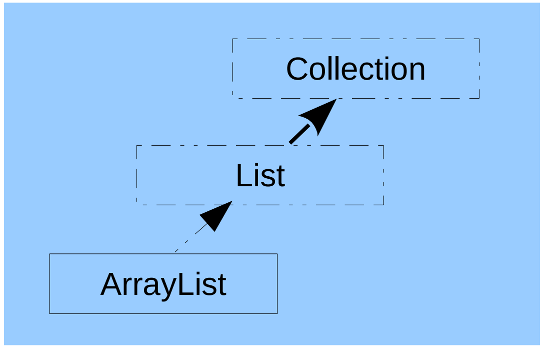
- all collections implement **the Collection interface**

The Collection interface

Implemented by all collections

```
public boolean add (Object o);         // adds o
public boolean remove (Object o);      // removes o

public void clear ();                  // removes all objects
public boolean isEmpty ();             // true if the collection is empty

public int size ();                    // number of objects in the collection
public boolean contains (Object o);    // true if o is in the collection
...
```

ArrayList

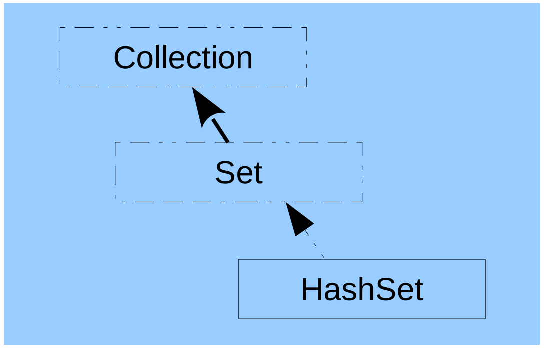


ArrayList
- implements the **List** interface
- keeps insertion order
- accepts duplicates
- specific methods added

```
public void add (int index, Object o);    // adds o at the given index (shifts subsequent elts)
public Object get (int index);            // returns the object at the given index
public int indexOf (Object o);            // returns the index of o
public Object remove (int index);         // removes the object at the given index
...
```

```
List l = new ArrayList ();
l.add ("Robert");  // add () comes from Collection
l.add ("Brad");
l.add ("Robert");

int n = l.size (); // 3
String s = (String) l.get (0);  // "Robert"
```

HashSet

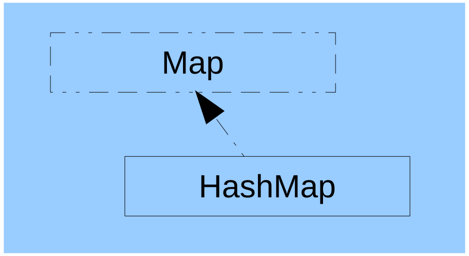


HashSet

- implements the **Set** interface
- does **not** keep insertion order
- does **not** accept duplicates

```java
Set s = new HashSet ();

s.add ("one");
s.add ("two");
s.add ("one"); // duplicate, ignored

int n = s.size ();  // 2

if (s.contains ("one"))...    // true
if (s.contains ("three"))...  // false
```

Maps



A Map associates a key with a value

- the common Map implementation is **HashMap**
- keys must be unique (like in a Set)
- keys and values are references

```java
Map m = new HashMap ();

m.put ("Red", new Color (1, 0, 0));
m.put ("Green", new Color (0, 1, 0));
m.put ("Blue", new Color (0, 0, 1));

Color c = (Color) m.get ("Red");  // returns a color object

if (m.containsKey ("Blue"))...  // true

Set s = m.keySet ();  // set of keys: Red, Green, Blue
```

The tools in the Collections class

Tools for the collections are proposed in a class: Collections

```
public static final List EMPTY_LIST
public static final Set EMPTY_SET
public static final Map EMPTY_MAP

public static void sort(List list)
public static void sort(List list, Comparator c)

public static void shuffle(List list)
public static void reverse(List list)

public static Object min(Collection coll)
public static Object max(Collection coll)
```

empty collections & maps

sorting

changing elements order

```
// Random order
Collections.shuffle (list);
```

How to iterate on objects in collections

Two syntaxes to loop on a list

```
// List of Tree
List l = new ArrayList ();
l.add (new Tree (5.5));
l.add (new Tree (2.3));
l.add (new Tree (4.1));
...

// Loop with an Iterator
for (Iterator i = l.iterator (); i.hasNext ();) {
  Tree t = (Tree) i.next ();

  if (t.getDbh () < 3) {i.remove ();}

}
```

constructor takes a dbh

an Iterator + a cast

the iterator can remove the current element from the list

a cast is needed at iteration time

```
// Loop with a foreach
for (Object o : l) {
  Tree t = (Tree) o;

  t.setDbh (t.getDbh () * 1.1);

}
```

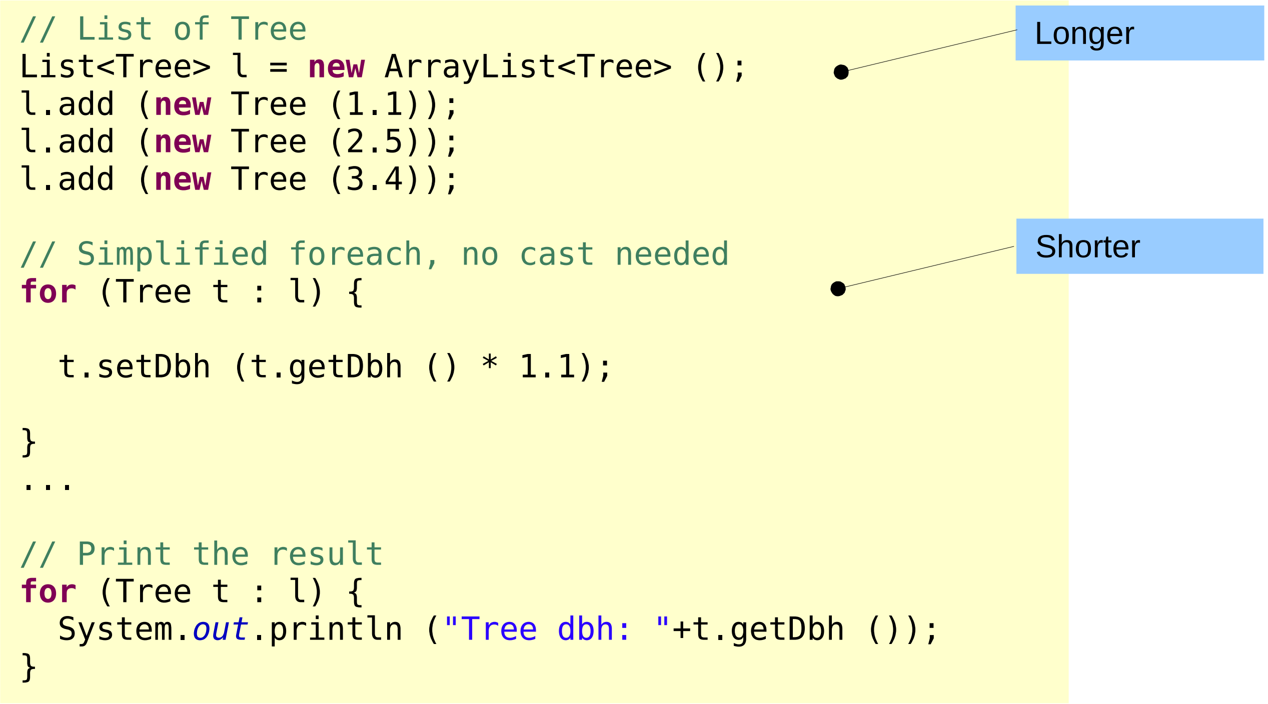
How to iterate on objects in maps

```
Map m = new HashMap ();
m.put ("Red", new Color (1, 0, 0));
m.put ("Green", new Color (0, 1, 0));
m.put ("Blue", new Color (0, 0, 1));

for (Object o : m.keySet ()) {
  String key = (String) o;
  //...
}

for (Object o : m.values ()) {
  Color value = (Color) o;
  //...
}
```

iterate on keys

iterate on values

Generics



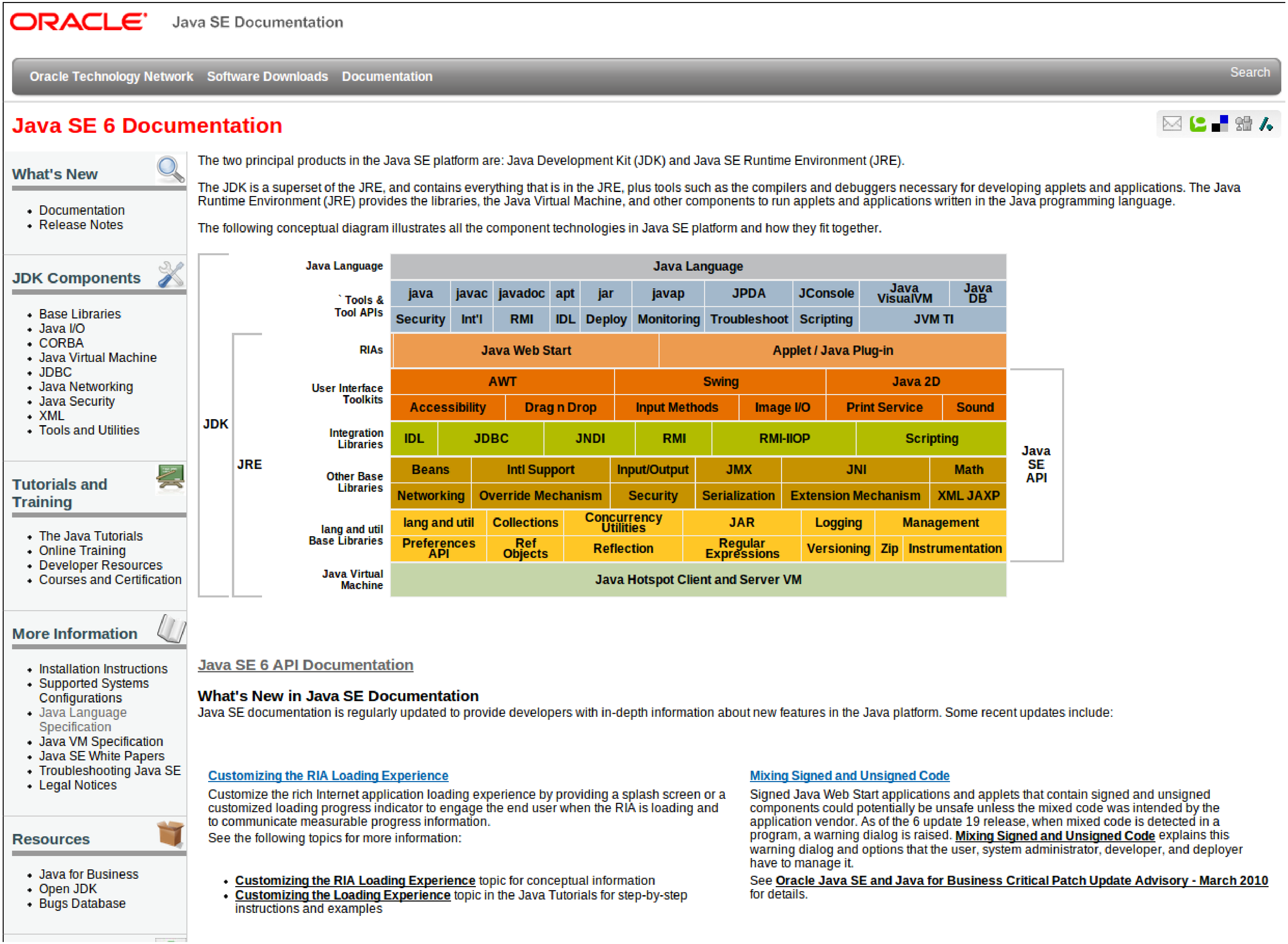
```
// List of Tree
List<Tree> l = new ArrayList<Tree> ();
l.add (new Tree (1.1));
l.add (new Tree (2.5));
l.add (new Tree (3.4));

// Simplified foreach, no cast needed
for (Tree t : l) {

  t.setDbh (t.getDbh () * 1.1);

}
...

// Print the result
for (Tree t : l) {
  System.out.println ("Tree dbh: "+t.getDbh ());
}
```

Longer

Shorter

Online documentation

http://download.oracle.com/javase/6/docs/



ORACLE Java SE Documentation

Java SE 6 Documentation

What's New

- Documentation
- Release Notes

JDK Components

- Base Libraries
- Java I/O
- CORBA
- Java Virtual Machine
- JDBC
- Java Networking
- Java Security
- XML
- Tools and Utilities

Tutorials and Training

- The Java Tutorials
- Online Training
- Developer Resources
- Courses and Certification

More Information

- Installation Instructions
- Supported Systems Configurations
- Java Language Specification
- Java VM Specification
- Java SE White Papers
- Troubleshooting Java SE
- Legal Notices

Resources

- Java for Business
- Open JDK
- Bugs Database

The two principal products in the Java SE platform are: Java Development Kit (JDK) and Java SE Runtime Environment (JRE).

The JDK is a superset of the JRE, and contains everything that is in the JRE, plus tools such as the compilers and debuggers necessary for developing applets and applications. The Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine, and other components to run applets and applications written in the Java programming language.

The following conceptual diagram illustrates all the component technologies in Java SE platform and how they fit together.

JDK | JRE | Java SE API

Java Language	Java Language
` Tools & Tool APIs	java, javac, javadoc, apt, jar, javap, JPDA, JConsole, Java VisualVM, Java DB Security, Int'l, RMI, IDL, Deploy, Monitoring, Troubleshoot, Scripting, JVM TI
RIAs	Java Web Start, Applet / Java Plug-in
User Interface Toolkits	AWT, Swing, Java 2D Accessibility, Drag n Drop, Input Methods, Image I/O, Print Service, Sound
Integration Libraries	IDL, JDBC, JNDI, RMI, RMI-IIOP, Scripting
Other Base Libraries	Beans, Intl Support, Input/Output, JMX, JNI, Math Networking, Override Mechanism, Security, Serialization, Extension Mechanism, XML JAXP
lang and util Base Libraries	lang and util, Collections, Concurrency Utilities, JAR, Logging, Management Preferences API, Ref Objects, Reflection, Regular Expressions, Versioning, Zip, Instrumentation
Java Virtual Machine	Java Hotspot Client and Server VM

Java SE 6 API Documentation

What's New in Java SE Documentation

Java SE documentation is regularly updated to provide developers with in-depth information about new features in the Java platform. Some recent updates include:

Customizing the RIA Loading Experience

Customize the rich Internet application loading experience by providing a splash screen or a customized loading progress indicator to engage the end user when the RIA is loading and to communicate measurable progress information.

See the following topics for more information:

- **Customizing the RIA Loading Experience** topic for conceptual information
- **Customizing the Loading Experience** topic in the Java Tutorials for step-by-step instructions and examples

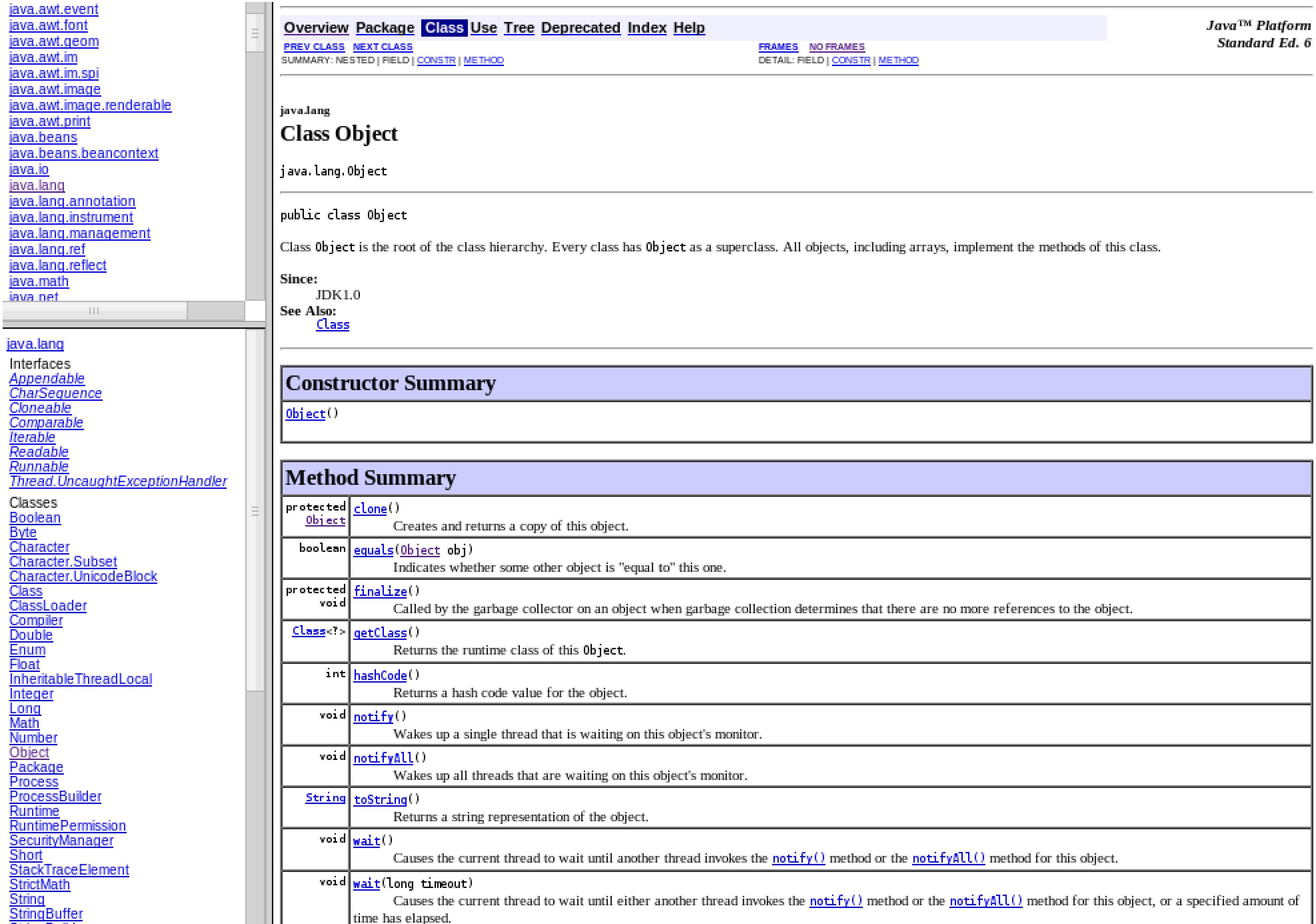
Mixing Signed and Unsigned Code

Signed Java Web Start applications and applets that contain signed and unsigned components could potentially be unsafe unless the mixed code was intended by the application vendor. As of the 6 update 19 release, when mixed code is detected in a program, a warning dialog is raised. **Mixing Signed and Unsigned Code** explains this warning dialog and options that the user, system administrator, developer, and deployer have to manage it.

See **Oracle Java SE and Java for Business Critical Patch Update Advisory - March 2010** for details.

Online documentation: javadoc

http://download.oracle.com/javase/6/docs/api/



java.awt.event
java.awt.font
java.awt.geom
java.awt.im
java.awt.im.spi
java.awt.image
java.awt.image.renderable
java.awt.print
java.beans
java.beans.beancontext
java.io
java.lang
java.lang.annotation
java.lang.instrument
java.lang.management
java.lang.ref
java.lang.reflect
java.math
java.net

java.lang

Interfaces
Appendable
CharSequence
Cloneable
Comparable
Iterable
Readable
Runnable
Thread.UncaughtExceptionHandler

Classes
Boolean
Byte
Character
Character.Subset
Character.UnicodeBlock
Class
ClassLoader
Compiler
Double
Enum
Float
InheritableThreadLocal
Integer
Long
Math
Number
Object
Package
Process
ProcessBuilder
Runtime
RuntimePermission
SecurityManager
Short
StackTraceElement
StrictMath
String
StringBuffer

Overview Package Class Use Tree Deprecated Index Help

Java™ Platform
Standard Ed. 6

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

java.lang

Class Object

`java.lang.Object`

`public class Object`

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

Since:
JDK1.0

See Also:
`Class`

Constructor Summary

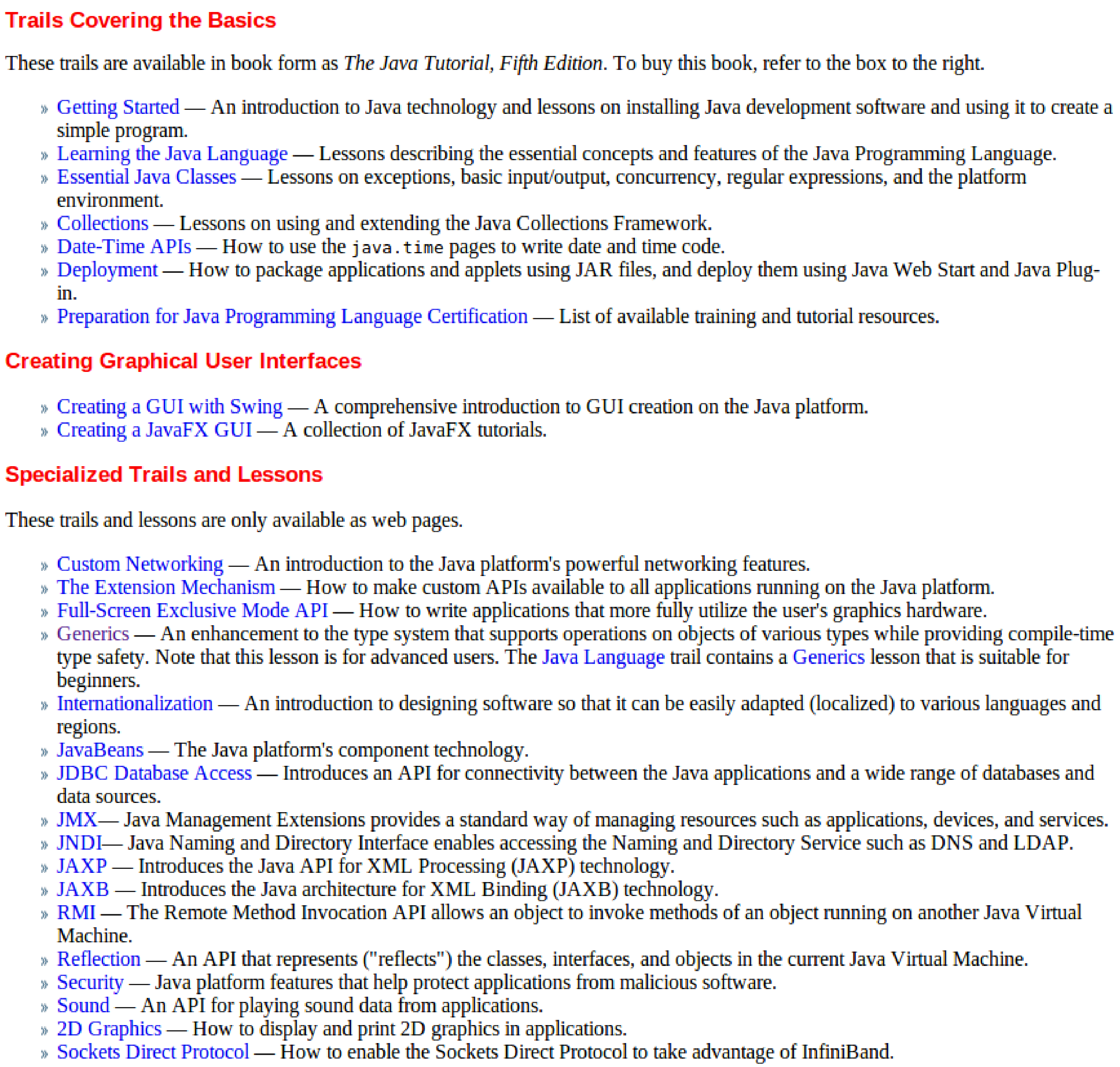
`Object()`

Method Summary

`protected Object`	`clone()` Creates and returns a copy of this object.
`boolean`	`equals(Object obj)` Indicates whether some other object is "equal to" this one.
`protected void`	`finalize()` Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
`Class<?>`	`getClass()` Returns the runtime class of this `Object`.
`int`	`hashCode()` Returns a hash code value for the object.
`void`	`notify()` Wakes up a single thread that is waiting on this object's monitor.
`void`	`notifyAll()` Wakes up all threads that are waiting on this object's monitor.
`String`	`toString()` Returns a string representation of the object.
`void`	`wait()` Causes the current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object.
`void`	`wait(long timeout)` Causes the current thread to wait until either another thread invokes the `notify()` method or the `notifyAll()` method for this object, or a specified amount of time has elapsed.

Online documentation: tutorials

http://docs.oracle.com/javase/tutorial/



Trails Covering the Basics

These trails are available in book form as *The Java Tutorial, Fifth Edition*. To buy this book, refer to the box to the right.

- Getting Started — An introduction to Java technology and lessons on installing Java development software and using it to create a simple program.
- Learning the Java Language — Lessons describing the essential concepts and features of the Java Programming Language.
- Essential Java Classes — Lessons on exceptions, basic input/output, concurrency, regular expressions, and the platform environment.
- Collections — Lessons on using and extending the Java Collections Framework.
- Date-Time APIs — How to use the `java.time` pages to write date and time code.
- Deployment — How to package applications and applets using JAR files, and deploy them using Java Web Start and Java Plug-in.
- Preparation for Java Programming Language Certification — List of available training and tutorial resources.

Creating Graphical User Interfaces

- Creating a GUI with Swing — A comprehensive introduction to GUI creation on the Java platform.
- Creating a JavaFX GUI — A collection of JavaFX tutorials.

Specialized Trails and Lessons

These trails and lessons are only available as web pages.

- Custom Networking — An introduction to the Java platform's powerful networking features.
- The Extension Mechanism — How to make custom APIs available to all applications running on the Java platform.
- Full-Screen Exclusive Mode API — How to write applications that more fully utilize the user's graphics hardware.
- Generics — An enhancement to the type system that supports operations on objects of various types while providing compile-time type safety. Note that this lesson is for advanced users. The Java Language trail contains a Generics lesson that is suitable for beginners.
- Internationalization — An introduction to designing software so that it can be easily adapted (localized) to various languages and regions.
- JavaBeans — The Java platform's component technology.
- JDBC Database Access — Introduces an API for connectivity between the Java applications and a wide range of databases and data sources.
- JMX— Java Management Extensions provides a standard way of managing resources such as applications, devices, and services.
- JNDI— Java Naming and Directory Interface enables accessing the Naming and Directory Service such as DNS and LDAP.
- JAXP — Introduces the Java API for XML Processing (JAXP) technology.
- JAXB — Introduces the Java architecture for XML Binding (JAXB) technology.
- RMI — The Remote Method Invocation API allows an object to invoke methods of an object running on another Java Virtual Machine.
- Reflection — An API that represents ("reflects") the classes, interfaces, and objects in the current Java Virtual Machine.
- Security — Java platform features that help protect applications from malicious software.
- Sound — An API for playing sound data from applications.
- 2D Graphics — How to display and print 2D graphics in applications.
- Sockets Direct Protocol — How to enable the Sockets Direct Protocol to take advantage of InfiniBand.

Links to go further

Oracle and Sun’s tutorials
http://docs.oracle.com/javase/tutorial/
see the 'Getting Started' section

Learning the Java language
http://docs.oracle.com/javase/tutorial/java/index.html

Coding conventions
http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

Resources on the Capsis web site
http://capsis.cirad.fr

Millions of books...

Including this reference
Java In A Nutshell
David Flanagan - O'Reilly (several editions)