An Introduction to Java

dec 2010

Francois de Coligny
Samuel Dufour

INRA - UMR AMAP
Botany and computational plant architecture

Java training - Contents

Introduction

Bases
- a Java application
- variables, simple types
- operators
- Math
- arrays
- conditions
- loops
- exceptions

Object Oriented Programming
- encapsulation
- class
- instance
- methods
- inheritance
- abstract class
- interface
- polymorphism
- collections
- maps

Java resources to go further

History

James Gosling and Sun Microsystems

- java, May 20, 1995
- java 1 -> Java 6 (i.e. 1.6)
- license: GPL with classpath exception since 2006
- 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"
- dynamic (introspection)
- 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 last JDK (Java SE 6)
- environment variable

  add the java/bin/ directory <u>at the beginning</u> 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 <u>remove</u> openjdk-6-jdk
- editor: use gedit (default editor under Ubuntu)

  or SciTE: sudo apt-get install scite

Test

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

Bases

- a Java application
- variables, simple types
- operators
- Math
- arrays
- conditions
- loops
- exceptions

A Java application

- Java programs are written in files with a **'.java'** extension

- applications are .java files with a **public static void main(...) {...}** method

- how to compile and run a Java application:

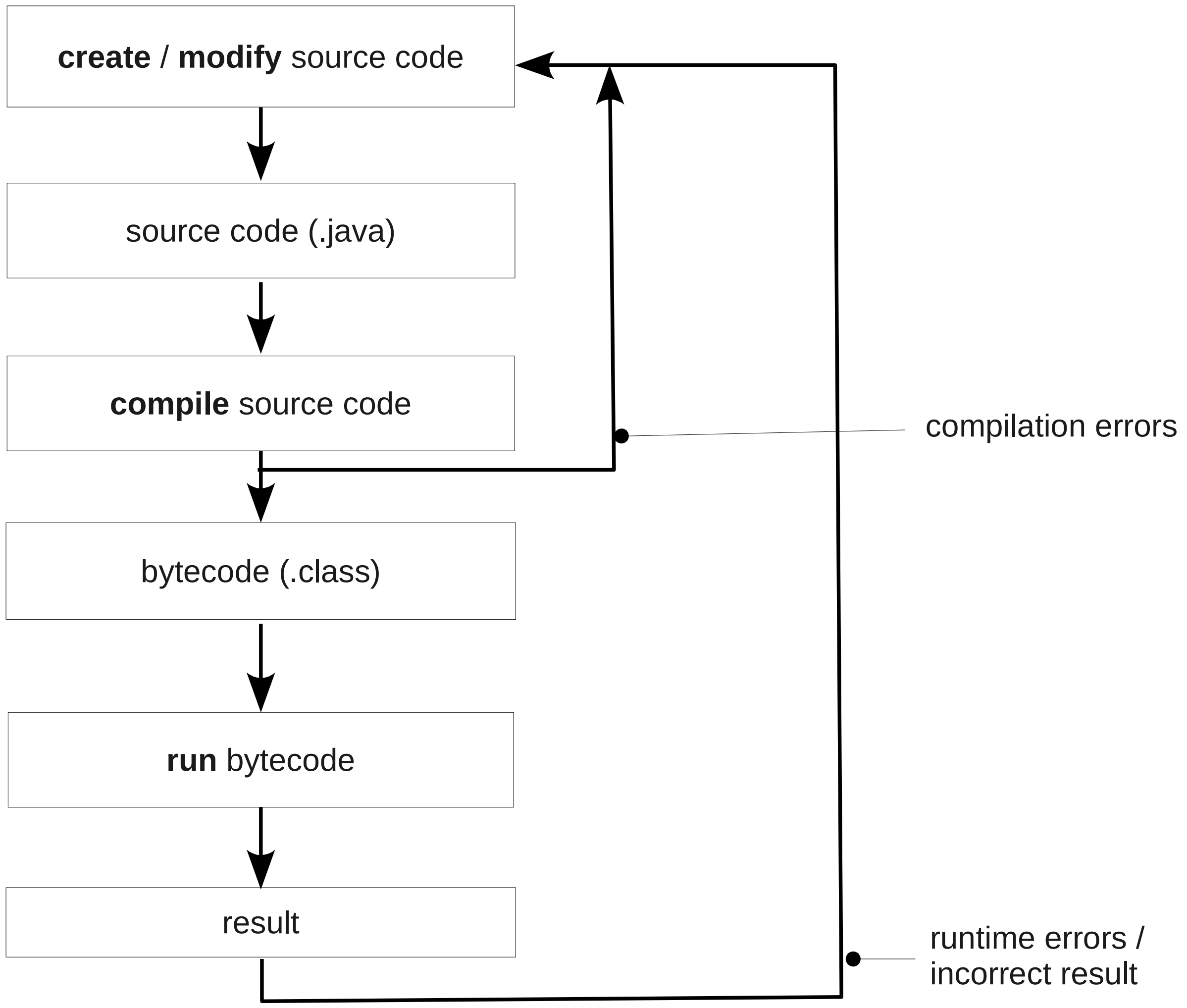
    run the compiler on a .java file : javac package/MyProgram.java

    returns a Java byte code file : MyProgram.class

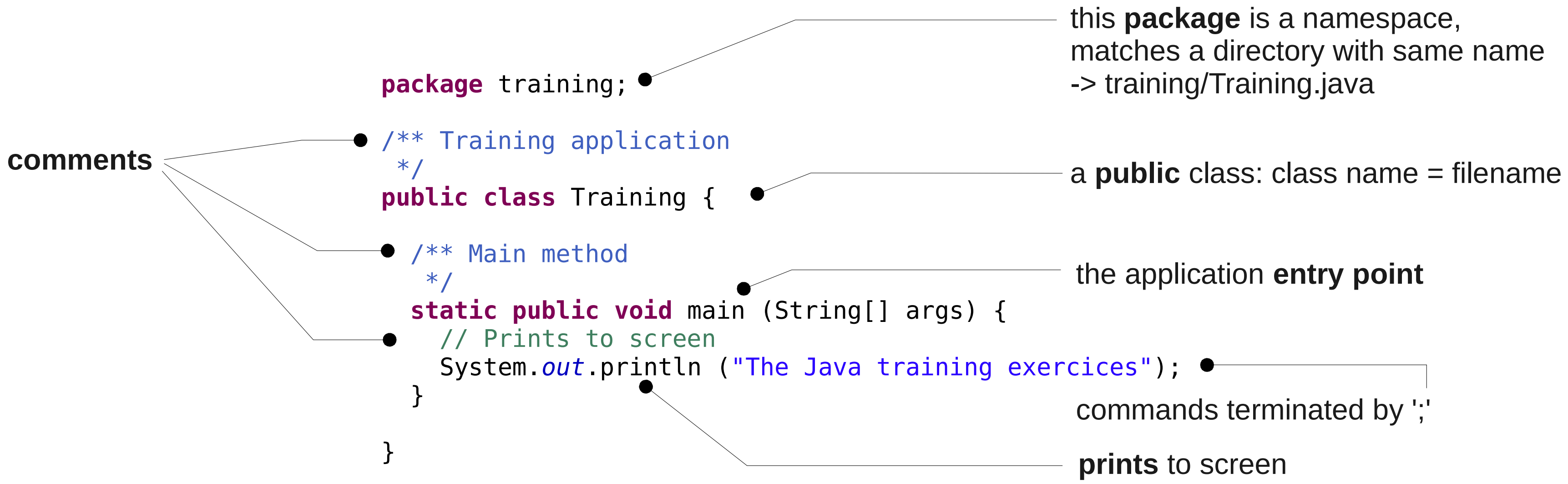
    run the interperter on a .class file : java package.MyProgram

- the tools **javac** and **java** are part of the JDK

The development process



A first 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

```
javac training/Training.java
java training.Training

The Java training exercices
```

Exercice: write, compile and run the Training application

Variables, simple types

Variable

- a variable has a **type** and holds a **value**

- value type can be **primitive** or a reference to an Object (seen later)

- a **variable name** starts with a lowercase letter, e.g. myVariable

8 primitive types

- signed integer : byte (8 bits), short (16 bits), **int** (32 bits), long (64 bits), e.g. 25

- floating point : float (32 bits) e.g. 2.3f, **double** (64 bits), e.g. 0.1d

- character : **char**, e.g. 'a', newline: '\n', tab: '\t'...

- boolean : **boolean**, e.g. true or false

Initialisation

- default for numbers: 0

- double crownDiameter = 2.5;

- constants: **final** double EPSILON = 0.01;

constants: uppercase
e.g. **MAX_CROWN_DIAMETER**

A special case: String

- String s = "Humprey";

Operators

Arithmetic

- simple: **+**, **-**, *****, **/**, **%**

- increment / decrement: **anInt++**; **anotherInt−−** ; e.g. index++;

- combined: **+=**, **-=**, ***=**, **/=**, e.g. index += 2; is the same than index = index + 2;

- precedence with **parentheses**, e.g. (a + b) * c;

- comparison: **<**, **<=**, **>**, **>=**, **==**, **!=**

- boolean: **&&**, **||**, **!**

Division

- real: 3. / 2. = 1.5

- int: 3 / 2 = 1

beware of the int division

- Division by zero

  - 3. / 0. -> Infinity

  - 3 / 0 -> java.lang.ArithmeticException

an exception (later)

Boolean arithmetics

Boolean variables are true or false

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

isReadyToApply () performs tests and
returns true or false

```
// Check if apply is possible
if (!isReadyToApply ()) {
  throw new Exception ("TraThinner.apply () - Wrong input parameters, see Log");
}
```

Math

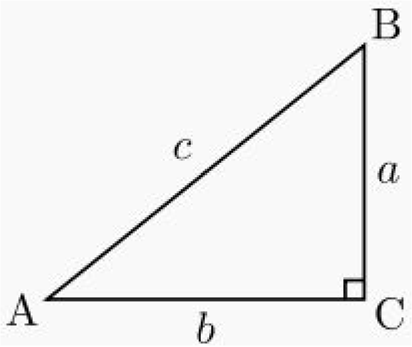
Constants

- 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 ()...

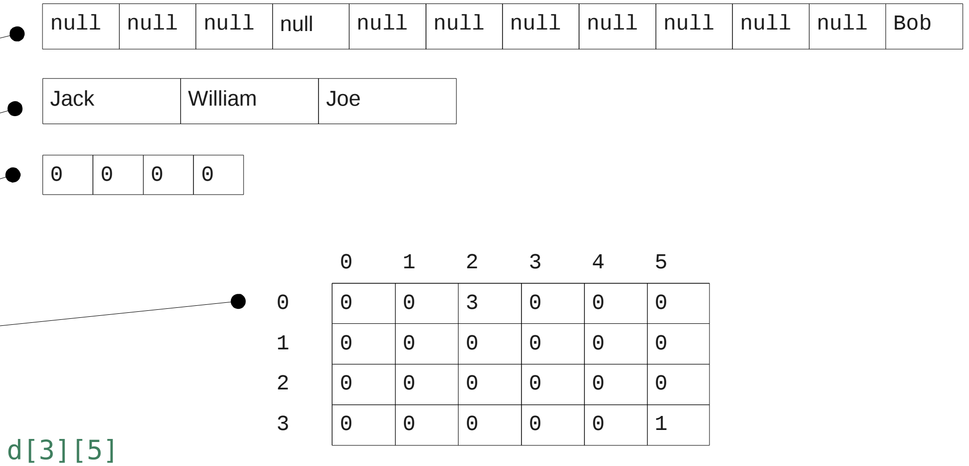
Exercice: Calculate the hypothenuse of a right-angled triangle with the Pythagorean theorem



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

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



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

Jack	William	Joe

0	0	0	0

	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 (condition) {
  block;
}

// complex if
if (condition) {
  block1
} else if (otherCondition) {
  block2
} else {
  block3
}

// boolean expression
if (v1 && !v2) {
  System.out.println ("v1 and not v2");
}
```

Loops: while, do... while

Loop with condition

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

while:
condition is tested first

```
int count = 0;
while (count < 100) {
  System.out.println ("count: " + count);
  count++;
}
```

```
int count2 = 0;
int sum = 0 ;
while (count2 < 100) {
  sum += Math.random ();
  if (sum > 20) {break;}
  count2 += 1;
}
```

may stop before the while test is true

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

```
do {
  double value = getValue ();
  System.out.println ("value: " + value);
} while (testCondition ());
```

test is at the end

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++) {
  sum += array[i];
}
```

from 0 to 11

- an internal **break** gets out of the loop
- an internal **continue** jumps to the next iteration
- for **while**, **do... while** and **for** loops

Exercice: loop on a double array and print the sum of its elements

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

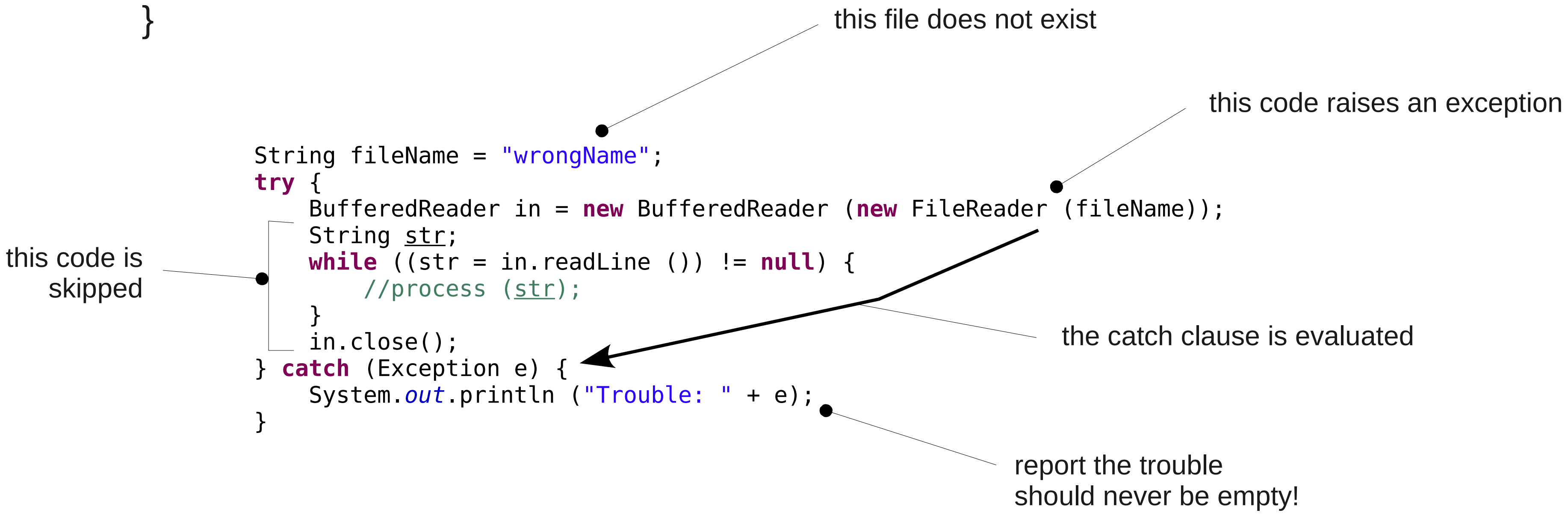
- exceptions stop the program

Exceptions management

Exceptions can be managed everywhere

-> use a try / catch statement

```
try {
        // code that possibly can raise an exception
} catch (Exception e) {
        // report the problem
}
```



```
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

this code raises an exception

this code is
skipped

the catch clause is evaluated

report the trouble
should never be empty!

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

Object Oriented Programming

Java is an object oriented language...

- encapsulation
- class
- instance
- methods
- inheritance
- abstract class
- interface
- polymorphism
- collections
- maps

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'

```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;
  }

}
```

data

methods operating on these data

Vocabulary

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

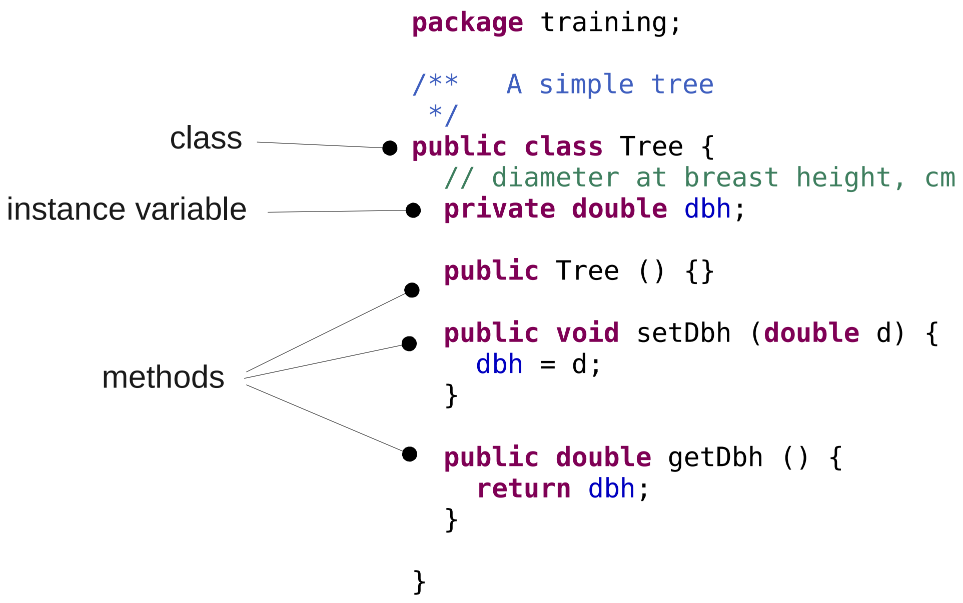
Object
- instance of a class at runtime
- memory allocation
- several objects may be build with the same class

Instance variable (IV)
- field of an object, i.e. its main variables
- (attribute, member data)

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

Property
- IV or method

A class



class

instance variable

methods

```
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;
  }

}
```

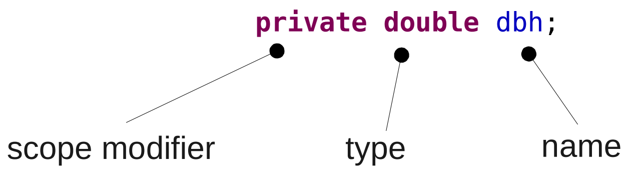
Scope modifiers for the properties

- **public** : visible by all (interface)
- **protected** : visible by subclasses (see hereafter) and in the package
- **private** : scope is limited to the class (hiden to the others)

Properties

the class properties...

Instance variable



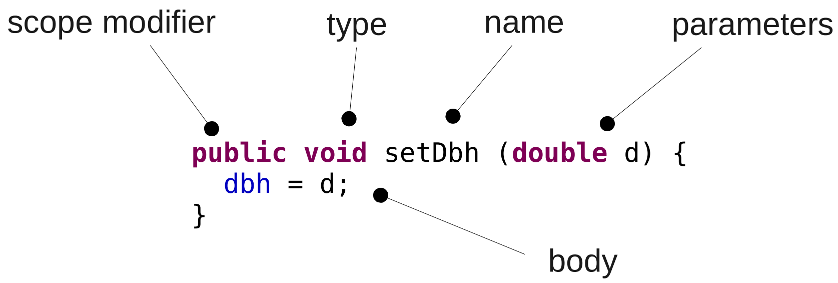
```
private double dbh;
```

scope modifier

type

name

Method



scope modifier

type

name

parameters

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

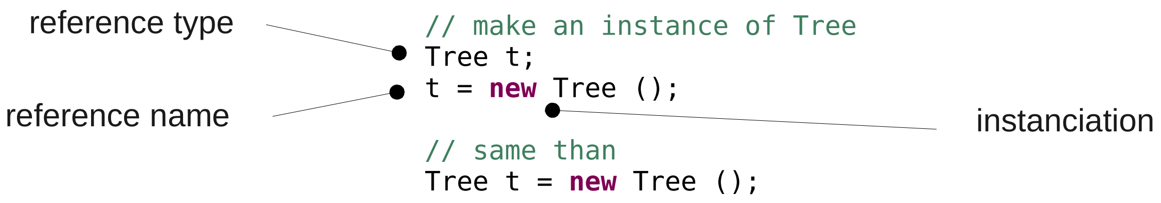
body

Instance

...by extension: the object / instance properties

Instanciation

- creates an object of a given class
- an object = an instance of the class



reference type

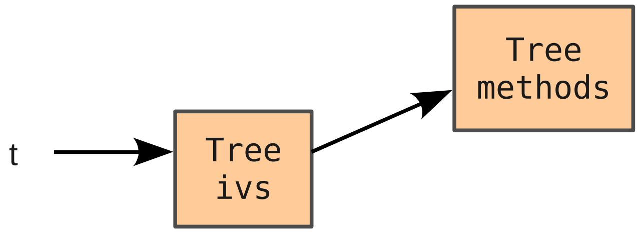
reference name

instanciation

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

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

What happens in memory

- new -> instanciation = memory reservation for the instance variables + the methods
- returns a reference to the created object
- we assign it to the 't' reference



Instances

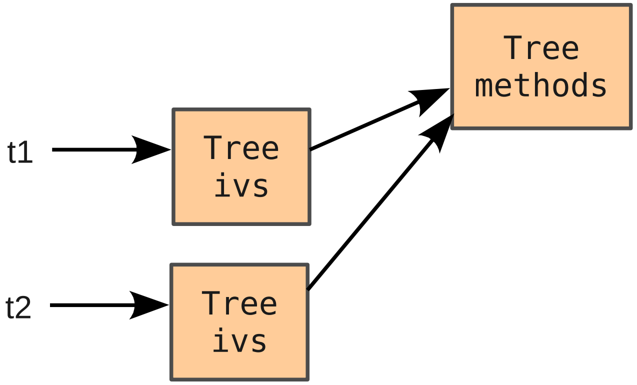
Creation of several objects

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

Tree t2 = new Tree ();
```

2 new -> 2 objects

What happens in memory

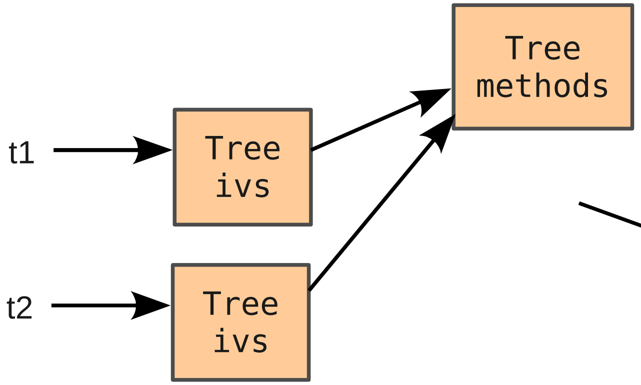
- 2 new: 2 memory reservations for the instance variables of the 2 objects
  (their dbh may be different)
- the methods of the 2 objects are shared in memory
- each new returns a reference to the corresponding object
- we assign them to 2 different references 't1' and 't2'



Instances

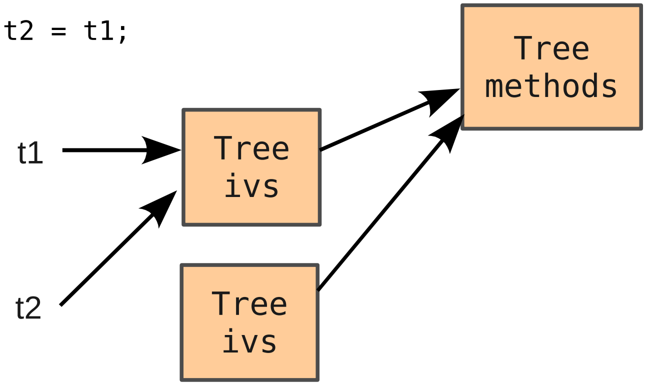
Using the references

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

Tree t2 = new Tree ();
```



t1
t2
Tree
ivs
Tree
ivs
Tree
methods

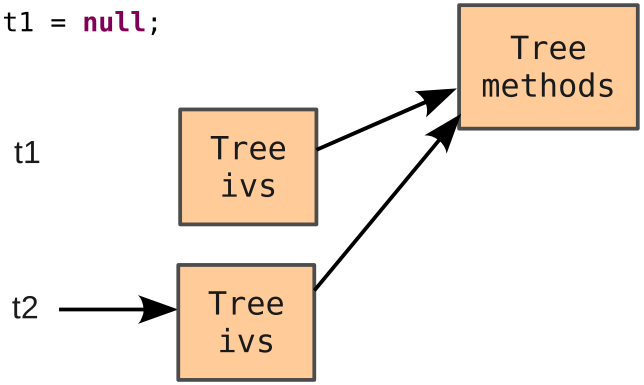
```
t2 = t1;
```



t1
t2
Tree
ivs
Tree
ivs
Tree
methods

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

```
t1 = null;
```



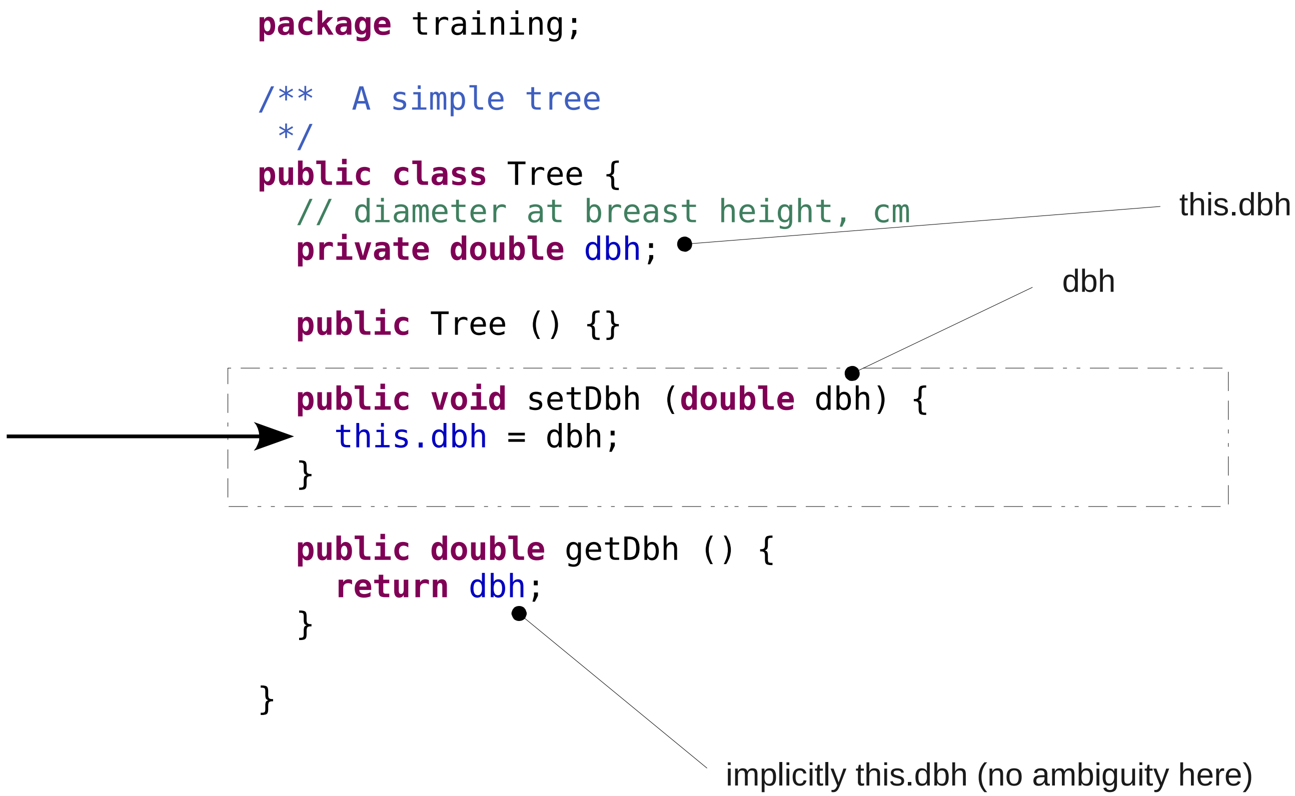
t1
t2
Tree
ivs
Tree
ivs
Tree
methods

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

Specific references

A keyword for the reference to the current object: this

- to remove ambiguities



```
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;
  }

}
```

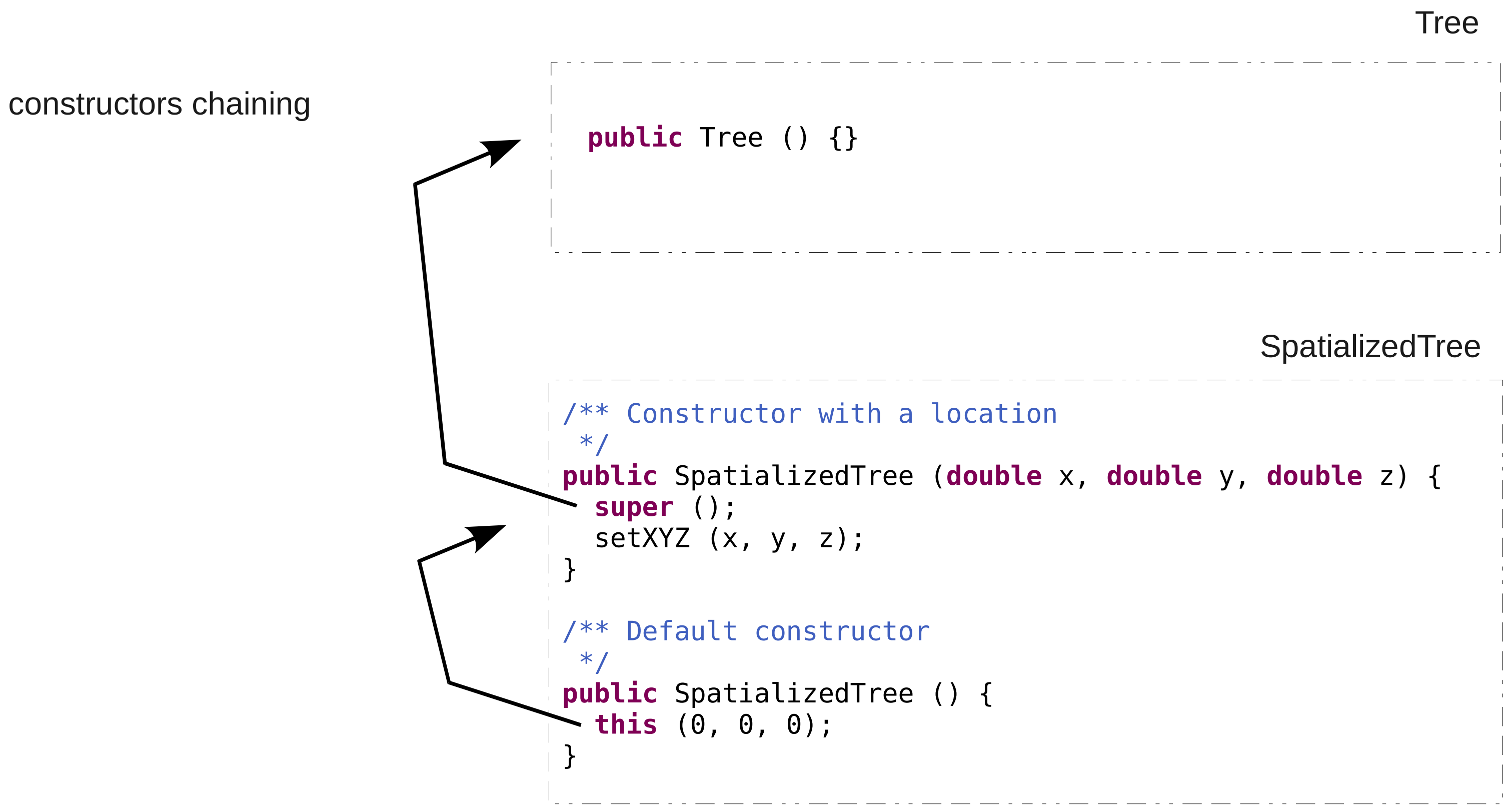
this.dbh

dbh

implicitly this.dbh (no ambiguity here)

Constructor

- **particular method** called at instanciation time (new)
- **same name** than the class
- **no return type**
- deals with instance variables **initialisation**
- **several** constructors may coexist if they have different parameter types
- **chain the constructors** to optimise the code (never duplicate)



constructors chaining

Tree

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

SpatializedTree

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

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

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, z of the base of the trunk (m)
  private double x;
  private double y;
  private double z;

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

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

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

}
```

constructors are particular methods without a return type

setXYZ method: 3 parameters

getSomething () is called an accessor

Method overloading / overriding

Overload (surcharge)
- in the same class
- several methods with same name
- and different parameter types

some class

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

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

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

Tree

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

TreeWithCrown

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

tell the compiler
-> it will check

TreeWithCrown **extends** Tree

Static method / static variables

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 value or an object

example in class **Tree**

```
static public double method1 (double param1, double param2) {
  return param1 * param1 + param2 ;
}
```

- param1 and param2 are the parameters

- their names have a local scope: they are only available in the method

```
double r = Tree.method1 (12d, 5d);
```

A common variable shared by all the instances

- used for the constants: Math.PI

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

- can be a variable

  public **static** int counter;

Calling the methods

Syntax

- reference.methodName (parameters);

- returnType = reference.methodName (parameters);

- parameters may be empty
- or a list of ',' separated 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 trees
Tree t1 = new Tree ();
Tree t2 = new Tree ();
Tree t3 = new Tree ();

// set their diameter
t1.setDbh (12);
t2.setDbh (14.5);
t3.setDbh (15);

t1.getDbh ();  // 12

t1 = t2;
t1.getDbh ();  // 14.5

double d1 = t1.getDbh ();

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

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 ();
```

Summary: objects have complex types

Java manipulates Objects

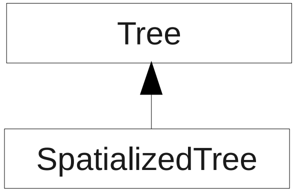
- an object is **a concept** (i.e. a particular data structure)
- objects are instantiated with the keyword **new**
- a variable contains **a reference** to an object
- assignation is a **reference copy** (the variables points to the same object)
- objects are **destroyed** when there is no more reference on them (garbage collecting)
- by default an object variable is set to **null**
- objects have properties accessible with **the '.' operator**

no object created yet

```
// declare two references
MyObject o1, o2;  // null

// create an object (instanciation)
o1 = new MyObject ();

// the object can be used
o1.value;
o1.sum ();

// assignment
o2 = o1 ;

// set both references to null
o1 = null;
o2 = null;  // the object will be destroyed by the garbage collector
```

Inheritance

UML notation



Reuse a class to make more specific classes
- e.g. a tree with a crown, a tree with leaves, etc.
- inheritance corresponds to a **'is a' relation**
- a **sub-class** 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)

```
package training;

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

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

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

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

}
```

extends

calls constructor of
the superclass

new methods

```
// SpatializedTree
SpatializedTree t3 = new SpatializedTree ();
t3.setXYZ (1, 1, 0);

t3.setDbh (15.5);

t3.getX ();    // 1
t3.getDbh ();  // 15.5
```

inherited method

Abstract class

An incomplete superclass with common methods

- class 'template' containing **abstract methods** to be implemented in all subclasses
- useful to share common methods in an inheritance graph
- each subclass implements the abstract methods
- can not be instanciated directly

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

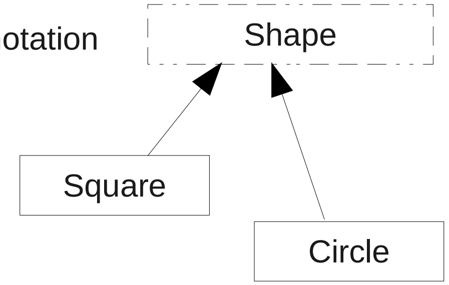
```
abstract class Shape {
    abstract public float area ();  // m2
}
```

an abstract method: no body

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

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

Two subclasses: they implement the abstract method

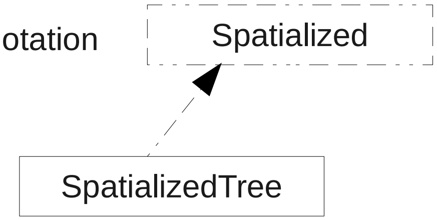
UML notation



Interface

Like an abstract class...
- **all** the methods are abstract ('abstract' is not required)
- **makes sure** that a class implements a number of methods
- a kind of **contract**
- classes extend other classes
- classes **implement** interfaces
- implementing several interfaces is possible

UML notation



```
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

```
/**   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.y = y;
  }

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

}
```

method body required in the classes

Enums

A type for enumerations (a kind of class)

- an enum is a type with a limited number of value
- better than using integer or constant values

Declaration

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

An exemple of use

```
private Day day;
...

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

Nested class

A class within another class

- not public

- static class / interface (no access to the ivs)

- member class (like a method)

- local class (in a method)

- anonymous class (on the fly)

May be complex, not explained in details here...

Polymorphism

Write generic code to be executed with several types

- more abstract and general implementations

```java
abstract class Shape {
    abstract public float area ();  // m2
}

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

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

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

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

  }
  return s;
}
```

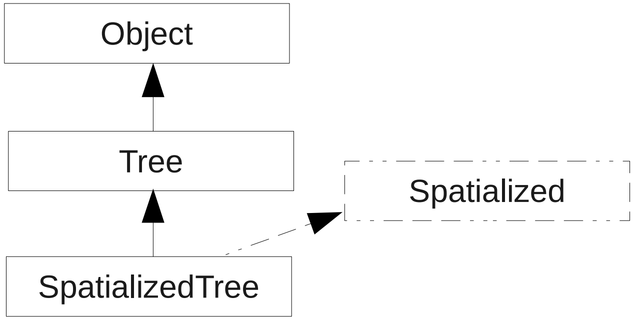
this code is generic
works with all shapes

several classes, all Shapes

Example of use

```java
// ...
Shape[] a = {new Square (), new Circle (), new Square ()};
float total = totalArea (a);
// ...
```

The instanceof operator



All classes inherit the Object class

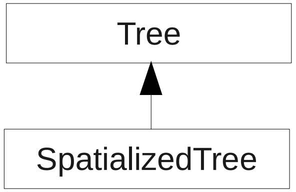
- instanceof tests the type of an object

```
SpatializedTree t1 = new SpatializedTree ();

t1 instanceof SpatializedTree;  // true
t1 instanceof Tree;             // true
t1 instanceof Object;           // true

t1 instanceof Spatialized;  // true
```

also with an interface

```
Tree t2 = new Tree ();

t2 instanceof Tree;             // true
t2 instanceof SpatializedTree;  // false
```

Cast



In an inheritance graph
e.g. SpatializedTree 'is a' Tree

- It is possible to use the supertype for a reference...

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

Tree instead of SpatializedTree

- and call the methods defined by the supertype...

```
t.setDbh (10);  // ok
```

- but if we want to call the methods of the subtype...

```
t.setXYZ (2, 1, 0);  // compilation error
```

Tree does not define setXYZ ()

- we must cast from the supertype to the subtype

```
SpatializedTree s = (SpatializedTree) t;  // cast

s.setXYZ (2, 1, 0);  // ok
```

SpatializedTree does define setXYZ ()

- with instanceof

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

- also on numbers

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

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

	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			

A focus on the collection framework

Collections are dynamic containers: like an array without a size limitation

- contain objects references of a specific type (or subtypes)

- have a specific behaviour
  - 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)

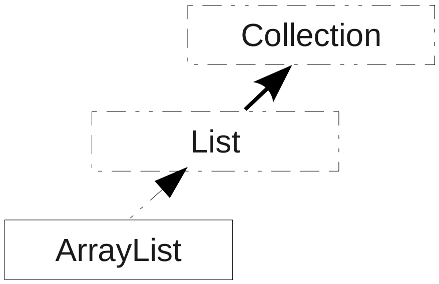
- all collections implement **the Collection interface**

The Collection interface

All collections implement an interface: **Collection**

```
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
public Iterator iterator ();         // a way to iterate
public Object[] toArray();    // an array containing all the objects
...
```

ArrayList



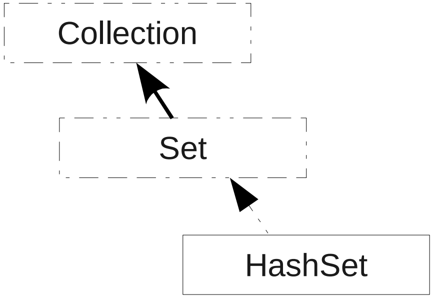
Implements the **Collection** interface
- variable size (grows automatically)

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
public Object get (int index);            // returns the object at the given index
public int indexOf (Object o);            // returns the index of o
public int lastIndexOf (Object o);        // returns the last index of o
public Object remove (int index);         // removes the object at the given index
public Object set (int index, Object o);  // sets o at the given index
public List subList (int fromIndex, int toIndex);  // sub list between the 2 indices
...
```

```
List<String> l = new ArrayList<String> ();
l.add ("Robert");
l.add ("Brad");
l.add ("Georges");

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

List<Integer> l2 = new ArrayList<Integer> ();
l2.add (23);  // autoboxing -> new Integer (23)
l2.add (12);

int i = l2.get (1);  // unboxing with Integer.intValue () -> 12
```

HashSet



Implements the **Collection** interface
- variable size (grows automatically)

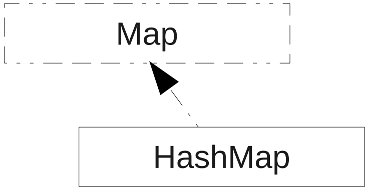
HashSet
- implements the **Set** interface
- does **not** keep insertion order
- does **not** accept duplicates

- same methods than Collection

```java
Set<Tree> s1 = new HashSet<Tree> ();
s1.add (new Tree (1));
s1.add (new Tree (2));
s1.add (new Tree (2));  // duplicate, ignored

int n1 = s1.size ();  // 2



Set s2 = new HashSet ();
// i.e. set<Object> s2 = new HashSet<Object> ();
s2.add ("one");
s2.add ("two");

s2.contains ("one");    // true
s2.contains ("three");  // false
```

```java
package training;

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

  public Tree () {this (0);}
  public Tree (int id) {this.id = id;}

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

  public void setId (int id) {this.id = id;}
  public int getId () {return id;}

  @Override
  public int hashCode () {return id;}

  @Override
  public boolean equals (Object o) {
    if (!(o instanceof Tree)) {return false;}
    return id == ((Tree) o).getId ();
  }

}
```

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 object references

```
Map<String,Color> m = new HashMap<String,Color> ();
m.put ("Black", new Color (0, 0, 0));
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 = m.get ("Red");  // returns a color object
m.containsKey ("black");  // true
m.keySet ();  // set of keys: Black, Red, Green, Blue
```

The tools in the Collections class

```
List l = Collections.singletonList("Matt")
```

Tools for the collections are proposed in a class: Collections

empty collections & maps

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

single element

```
public static Set singleton(Object o)
public static List singletonList(Object o)
public static Map singletonMap(Object key, Object value)

public static List nCopies(int n, Object o)
```

sorting

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

changing elements order

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

changing contents

```
public static void copy(List dest, List src)
public static void fill(List list, Object obj)
public static boolean replaceAll(List list, Object oldVal, Object newVal)
```

searching

```
public static int binarySearch(List list, Object key
public static Object min(Collection coll)
public static Object max(Collection coll)
```

non modifiable collections
and maps

```
public static List unmodifiableList(List list)
public static Set unmodifiableSet(Set s)
public static Map unmodifiableMap(Map m)
```

How to iterate on objects in collections

```
// List of Tree
List<Tree> l1 = new ArrayList<Tree> ();
l1.add (new Tree (1));
l1.add (new Tree (2));
l1.add (new Tree (3));

// Set dbh
for (Tree t : l1) {
  int id = t.getId ();
  t.setDbh (id * 0.7);
}

// Remove small trees
for (Iterator i = l1.iterator (); i.hasNext ();) {
  Tree t = (Tree) i.next ();
  if (t.getDbh () < 1) {i.remove ();}
}

// Print the result
for (Tree t : l1) {
  System.out.println ("Tree id: "+t.getId ());
}
```

'for each' syntax since java 1.5

an Iterator + a cast is possible

the iterator can remove the
current element from the list

```
// List of objects
List l2 = new ArrayList ();
l2.add (new Tree (1));
l2.add (new Tree (2));

// Set dbh
for (Object o : l2) {
  // Cast needed
  Tree t = (Tree) o;

  int id = t.getId ();
  t.setDbh (id * 0.7);
}
```

list without a type

a cast is needed at iteration time

How to iterate on objects in maps

```
Map<String,Color> m = new HashMap<String,Color> ();
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 (String key : m.keySet ()) {
  //...
}

for (Color value : m.values ()) {
  //...
}

for (Map.Entry<String,Color> entry : m.entrySet ()) {
  String key = entry.getKey ();
  Color value = entry.getValue ();
  //...
}
```

iterate on keys

iterate on values

iterate on entries

Java online documentation and libraries

Java Standard Edition technical documentation
http://download.oracle.com/javase/

Javadoc 1.6
http://download.oracle.com/javase/6/docs/api/

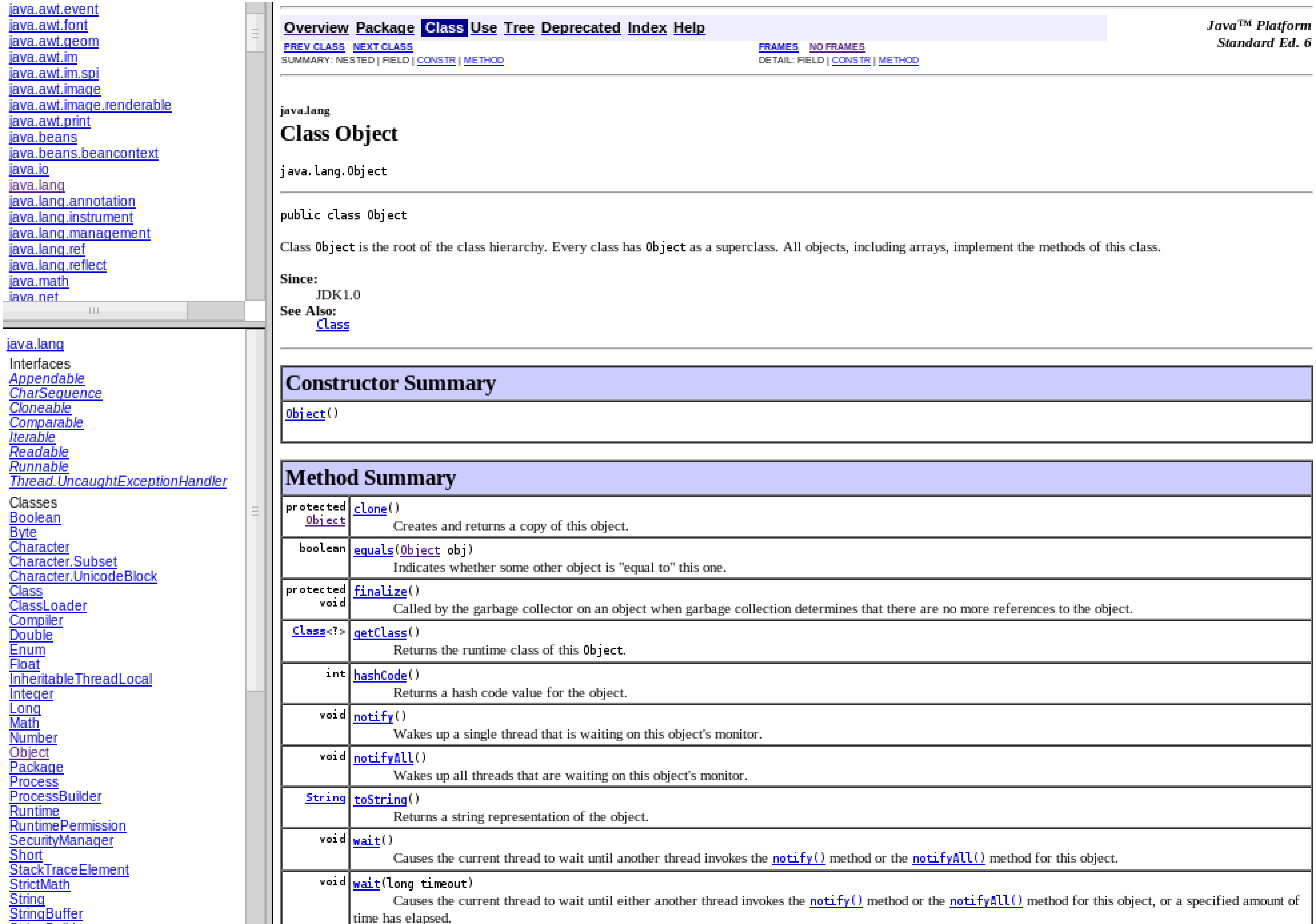
Libraries

- javax.swing: gui
- java.math
- java.util: collections
- java.io: input / output
- java.net: networking
- multiThreading
- database
- intropection

...

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.

Links to go further

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

Creating a graphical user interface
    http://download.oracle.com/javase/tutorial/uiswing/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, 5th Edition (english), 4me ed. (francais)
    David Flanagan - O'Reilly - 2005