Objects

v Objects are instances of classes
Book oneBook = new Book ();
Book otherBook = new Book ();
Book book3 = new Book ();
v All objects are manipulated through references
Person name1, name2;
name1 = new Person ("Manuel");
name2 = name1;
v All objects must be explicitly created.
Circle c1 = new Circle (p1, 5);
String s = "Book"; // Strings are an exception!

v In Java, objects are stored in heap memory and manipulated through a reference (variable), stored in the stack.
- Have been (attributes)
- Behave (methods)
- Have identity (the reference)

Methods

v Methods, messages, functions, procedures
v Invocation is always done through the dot notation.
oneBook.setTitle ("Tourism in Aveiro"); otherBook.setPubYear (2000);
v The recipient of the message is always on the left.
v The receiver is always a class or reference
for an object.
Math.sqrt (34); otherBook.setPubYear (2000);

Namespace – Package

v In Java, the management of the namespace is done through the concept of package.
v Why namespace management?
V Avoid class name conflicts
- We generally have no problem distinguishing the names of the classes we build.
- But how do we ensure that our Book class does not collide with another that may possibly already exist?

Package import

v Use

- Classes are referenced using their absolute names or using the import primitive.

```
import java.util.ArrayList import java.util. *
```

- The import clause must always appear on the first line of a program.

v When we write, import java.util. *;

- we are indicating a path for a package of classes allowing us to use them through simple names:

```
ArrayList <String> al = new ArrayList <> ();
```

v Otherwise we would have to write: java.util.ArrayList <String> al = new java.util.ArrayList <> ();

Create a package

v In the first line of code: package poo;

- ensures that the public class of this compilation unit will be part of the poo package.

v The namespace is based on a structure of sub-directories

- This package will correspond to a directory entry: $ CLASSPATH / poo
- Good practice to use inverted DNS: en.ua.deti.poo

v Its use will be in the form:

```
poo.Book sr = new poo.Book ();
```

- or

```
import poo. *
Book sr = new Book ();
```

---

toString

v All objects in Java understand the toString () message

Book oneBook = new Book (); oneBook.setTitle ("Tourism in Aveiro"); System.out.println (oneBook); // oneBook.toString ()

```
Book @ 33909752
```

v It is usually necessary to redefine this method to provide a more suitable result.

```
@Override
public String toString () {
return "Book: title =" + title + "; pubYear =" + pubYear;

}
```

```
Book: title = Tourism in Aveiro; pubYear = 0
```

Insecure Programming

v Many of the programming errors result from:

- uninitialized data - some programs / libraries need to initialize components and make the task dependent on the programmer.
- incorrect management of dynamic memory - "forgetting" to free memory, insufficient reserve, ...

v To solve these two problems, the Java language uses the concepts of:

- constructor
- garbage collector

Member initialization

v Within a class, initializing variables can be made in your statement.

```
class Measurement {int i = 25;
char c = 'K'; }
```

v However, this initialization is the same for all objects of the class

- The most common solution is to use a constructor. class Measurement {

```
int i;
char c;
Measurement (int im, char ch) {
i = im; c = ch; }
```

Constructor

v The initialization of an object can imply the simultaneous initialization of several types of data.

v A special member function, constructor, is invoked whenever an object is created.

v Instantiation is done using the new operator. Car c1 = new Car ();

v The constructor is identified by the same name as the class.

v This method can be overloaded from to allow different ways of initialization.

Car c2 = new Car ("Ferrari", "430");

v Does not return any value

v Always assume the class name

v Can have input parameters

v It is called only once: when creating the object

```
  public class Book {
    String title;
int pubYear;
public Book (String t, int py) {title = t;
pubYear = py; }
// ...
}
```

Default constructor

v A constructor without parameters is called the default constructor or the default constructor.
- It is automatically created by the compiler if no constructor is specified.

```
  class Book {
    String title;
    int pubYear;
}
Book m = new Book (); // OK
```

- If there is at least one constructor associated with a given class, the compiler no longer creates the default one.

```
class Book {
    String title;
int pubYear;
Book (int py) {pubYear = py; }
}
Book m = new Book (); // mistake!
```

Default values for primitive types

v If a variable is used as a member of a class the compiler takes care of initializing it by default
- This is not guaranteed in the case of local variables so we must always initialize all variables

Overloading

v We can use the same name in several functions - as long as they have different arguments and conceptually perform the same action

```
    void sort (int [] a);
    void sort (Book [] b);
```

v The static link checks the signature of the function (name + arguments)
v You cannot distinguish functions by the return value
- because it is allowed to invoke, e.g., void f () or int f () in the form f () where the return value is not used

Overlapping constructors

v Allow different ways to start an object of a given class.

```
public class Book {
public Book (String title, int pubYear) {...} public Book (String title) {...}
public Book () {...}
}
Book c1 = new Book ("The stone raft", 1986); Book c2 = new Book ("Galveias");
Book c3 = new Book ();
```

This reference

v The this reference can be used within each object to reference that same object

```
   public class Book {
       String title;
int pubYear;
public Book (String title, int pubYear) {
this.title = title;
this.pubYear = pubYear; }
}
class Faucet {
void closes () {/ * ... * /}
void locks () {closes (); / * or this.close () * /}
```

v Another use of the this reference is to return, in a given method, the reference to that object.
- can be used in a chain (lvalue).

```
public class Contador { int i = 0;
Contador increment() {

    i++; return this;
  }

void print() { System.out.println("i = " + i);

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

Contador x = new Contador();

x.increment().increment().increment().print(); }

}
```

Invoke one constructor within another

v When we write several constructors, we can call one within the other.
- the reference this allows to invoke another constructor on the same object.

```
public Book (String title, int pubYear) {this.title = title;
this.pubYear = pubYear;
}
public Book (String title) {
      this (title, 2000);
  }
```

v This form can only be used within constructors;
- in this case, this must be the first instruction to appear;
- it is not possible to invoke more than one this constructor.

The static concept

v Static methods have no associated this reference.
v Therefore, it is not possible to invoke non-static methods from static methods.
v You can invoke a static method without objects of that class.
v Static methods have the semantics of global functions (they are not associated with objects).

Static elements

v Static variables, or class variables, are common to all objects in that class.
v Your declaration is preceded by static.
v The invocation is made on the class identifier

```
class Test {
public static int a = 23;
public static void someFunction () {...} // ...
}
Test.someFunction (); // invoked on the Test class s1 = new Test ();
Test s2 = new Test (); System.out.println (Test.a);
Test.a ++; // s1.a and s2.a will be 24
```

Initialization of static members

v If static members are initialized, they take priority over all others.
v A static member is only initialized when the class is loaded (and only then)
- when the first object of that class is created or when it is used for the first time.
v We can use a special block - initializer
static - to group the initializations of
static members

```
class Circle {
static private double list [] = new double [100]; static {// static initializer
// list initialization []}
}
```

Object Vectors

v A vector in Java represents a set of references
- previous rules apply to default values int [] a = new int [10]; // 10 int
Book [] xC = new Book [10]; // 10 refs! Not 10 Books !!

Range / Scope

v An automatic variable can be used as long as it is defined until the end of that context
v Each block can have its own objects

```
{intk;
{inti;
      } // ‘i’ is not visible here
   } // ‘k’ is not visible here
```

v Illegal example

```
{intx = 12;
{intx = 96; / * error! * /}
}
```

Range of references and objects

v Example with references and objects {
Book b1 = new Book ("Elephant Memory"); }
// ‘b1’ is no longer visible here
v In this case, reference b1 is released (removed) and the object can no longer be used
- Will be removed by the Garbage collector

Encapsulation

v Key POO ideas
- Information Hiding (Encapsulation)
- Inheritance
- Polymorphism

v Encapsulation
- Separation between what cannot be changed (interface) and what can change (implementation)
- Interface visibility control (public, default, protected, private)

v It allows you to create different levels of access to the data and methods of a class.
v The levels of access control that we can use are, from the highest to the lowest access:
- public - can be used in any class
- "default" - visible within the same package
- protected - visible within the same package and derived classes
- private - only visible within the class

v A method of a class has access to all the information and methods of that class

---

Modifiers / Selectors

v Encapsulation allows you to hide an object's internal data
- But, sometimes it is necessary to access this data directly (reading and / or writing).
v Important rules!
- All attributes must be private.
- Access to the internal information of an object (private part) must always be done, through functions of the public interface.

v Selector
- Returns the current value of an attribute

```
public float getRadius () {// or public float radius () return radius;
}
```

v Modifier
- Modify the state of the object

```
public void setRadius (float newRadius) {
// or public void radius (float newRadius)
this.radius = newRadius; }
```

Private methods

v Internally a class can have several private methods that are only used internally by other methods of the class.

```
// example of auxiliary functions in a class class Screen {
    private int row ();
    private int col ();
private int remainingSpace ();
// ...};
```

What can a class contain

v The definition of a class can include:
- zero or more data attribute declarations
- zero or more method definitions
- zero or more constructors
- zero or more static initialization blocks
- zero or more class definitions or internal interfaces
v These elements can only occur within the ‘class ClassName {...}’ block
- "everything belongs" to some class
- only ‘import’ and ‘package’ can occur outside of a ‘class’ (or ‘interface’) statement

Good practice

v The construction semantics of an object must
make sense
Personalap = newPessoa (); L
Person p = new Person ("António Nunes"); K
Pessoa p = new Pessoa ("António Nunes", 12244, dataNasc); J
v We must give a minimum of public visibility when accessing an object
- Only what is strictly necessary
v Sometimes it makes more sense to create a new object than
that changing existing attributes
Point p1 = new Point (2,3); p1.set (4,5); L

v Join members of the same type
- Do not mix static methods with instance methods
v Declaring variables before or after methods - Do not mix methods, constructors and variables
v Keep constructors together, preferably at the beginning
v If you need to define static blocks, define only one at the beginning or end of the class.
v The order of the members is not important, but following conventions improves the readability of the code

---

Class Relations

v Part of the class modeling process consists of:
- Identify entities applying for classes
- Identify relationships between these entities
v Relationships between classes are easily identified using some real models.
- For example, a Digital Clock and an Analog Clock are both types of Clock (specialization or inheritance).
- A Digital Clock, on the other hand, contains a Battery (composition).
v Relations: - IS-A
- HAS-A

Inheritance (IS-A)

v IS-A indicates specialization (inheritance), that is, when a class is a subtype of another class.

v For example:
- Pinheiro is an (IS-A) tree.
- A Digital Clock is an (IS-A) Clock.

```
class Watch {/ * ... * /
}
class Digital Watch extends Watch {/ * ... * /
}
```
