OBJECT ORIENTED PROGRAMMING
UNIT2: OO PROGRAMMING. OBJECTS

JESUITAS
Sagrado Corazón
Jesuitas · Logroño

Index

Ticket machine-Internal Overview

A2.3: Open "naive-ticket-machine" project using Bluej. Test application…

- Interacting with an object gives us clues about its behavior.
- Looking inside allows us to determine how that behavior is provided or implemented.
- All Java classes have a similar-looking internal view.

Basic class structure


```
public class TicketMachine
{
    Inner part of the class omitted.
}
```

The outer wrapper of TicketMachine

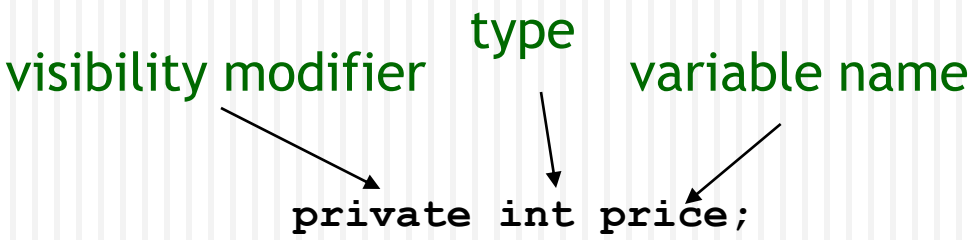
```
public class ClassName
{
    Fields
    Constructors
    Methods
}
```

The contents of a class

Fields

- Fields store values for an object.
- They are also known as instance variables.
- Use the *Inspect* option to view an object's fields.
- Fields define the state of an object.

```
public class TicketMachine
{
    private int price;
    private int balance;
    private int total;

    (…)
}
```



Constructors

- Constructors initialize an object.
- They have the same name as their class.
- They store initial values into the fields.
- They often receive external parameter values for this.

```
public TicketMachine(int ticketCost)
{
    price = ticketCost;
    balance = 0;
    total = 0;
}
```

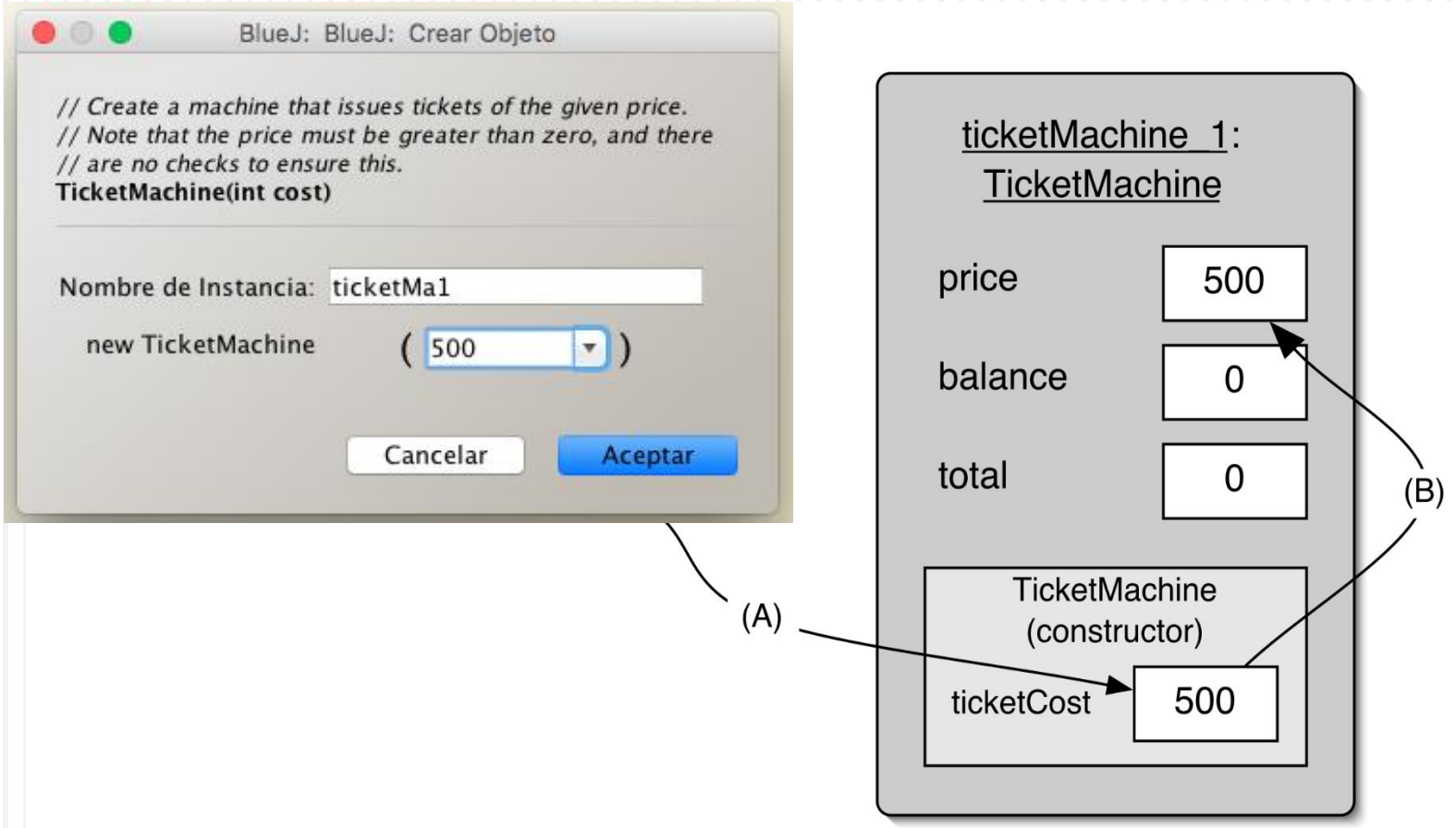
Constructors

- Example:

It is possible to create Pajaro objects two ways→it is said that the constructor is *OVERLOADED*. *(Lesson 4)*

```
public class Pajaro {
    // Attributes or properties
    private char color;
    private int edad;

    // Constructors
    Pajaro(){color='v'; edad=0;} // constructor of Pajaro class
    Pajaro(char c, int e){color=c;edad=e;} // constructor of Pajaro class

    // The rest of the Methods
    public void setEdad (int e){edad =e;}
    public void printEdad () {System.out.println(edad);}
    public void setColor (char c) {color=c;}
}
```

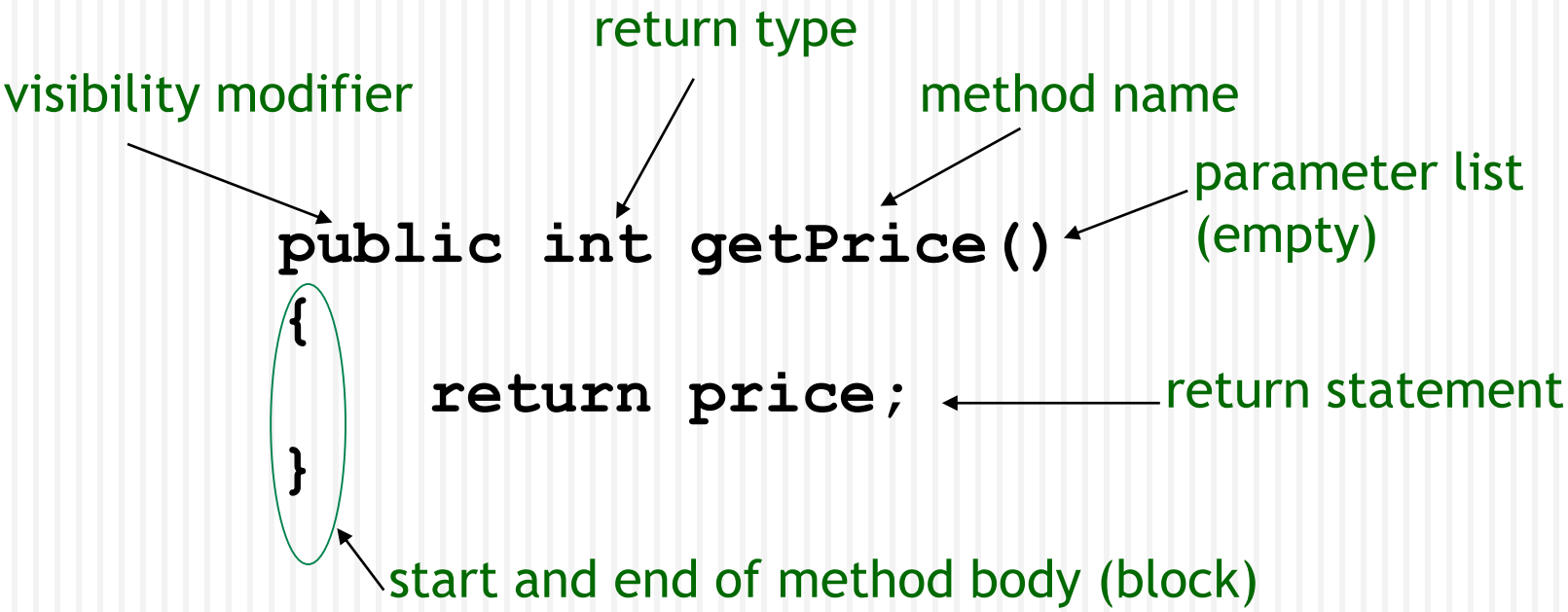
Passing data via parameters



Accessor methods

- Methods implement the behaviour of objects.
- Accessors provide information about an object (also known as getters).
- Methods have a structure consisting of a header and a body.
- The header defines the method's signature:
  - `public int getPrice()`
- The body encloses the method's statements.

Accessor methods



Accessor methods

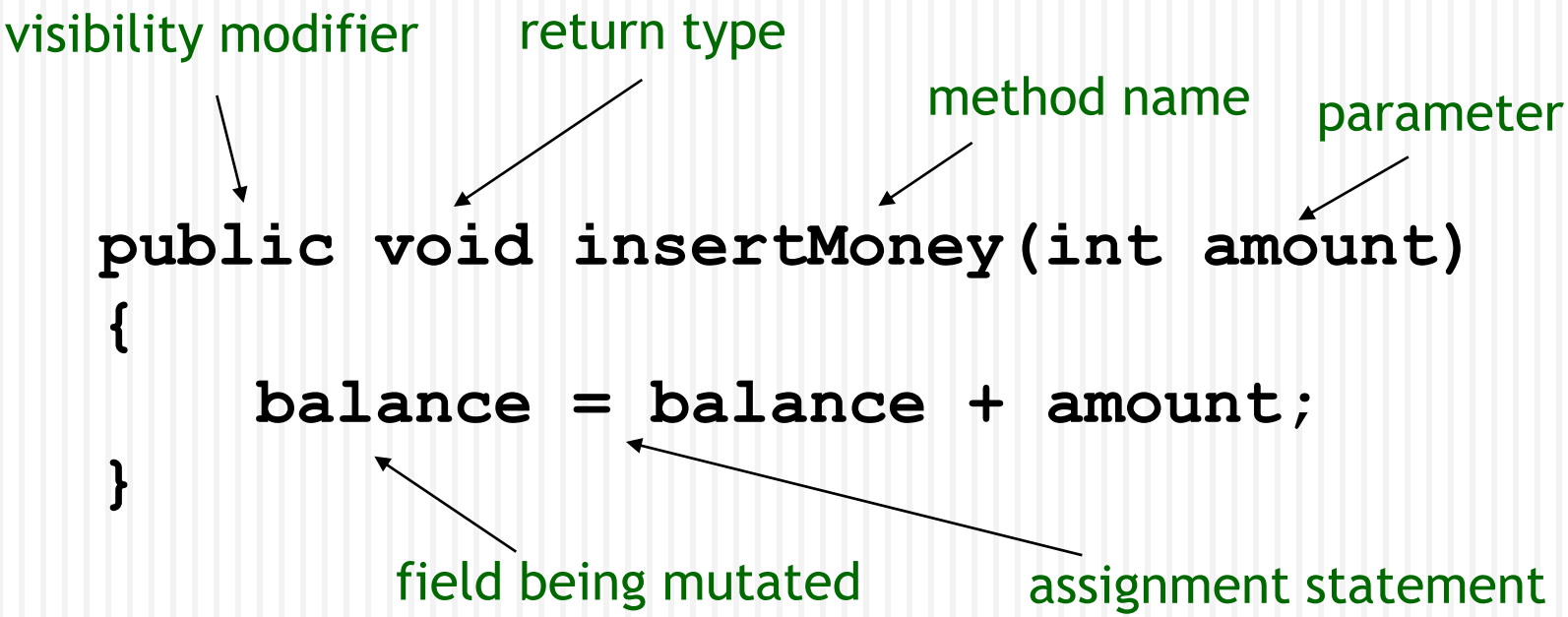
A2.4: Find the errors (there are 5!!)

```
public class CokeMachine
{
   int
   private price;

   public CokeMachine()
   {
      price = 300;
   }

   public int getPrice()
   {
      return Price;
   }
}
```

Mutator methods

- Have a similar method structure: header and body.
- Used to mutate (i.e. change) an object's state.
- Achieved through changing the value of one or more fields.
  - Typically contain assignment statements.
  - Typically receive parameters.

Mutator methods



Mutator methods


```
public void printTicket()
{
    // Simulate the printing of a ticket.
    System.out.println("####################");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " cents.");
    System.out.println("####################");
    System.out.println();

    // Update the total collected with the balance.
    total = total + balance;
    // Clear the balance.
    balance = 0;
}
```

Practice

A2.5: Modify your TicketMachine class following next instructions:

1) Complete the body of the *setPrice* method so that it assigns the value of its parameter to the price field. Its signature looks like this:
public void setPrice(int cost)
Hint: This is well-known kind of method, usually called **"setter"**. What is the difference between a getter and a setter?

Practice

A2.5: (cont.):

2) Complete the following method, whose purpose is to subtract the value of its parameter from a field named price.

```
/**
* Reduce price by the given amount.
*/
public void discount(int amount)
{
...
}
```

3) Add a method called *prompt* to the TicketMachine class. This should have a void return type and take no parameters. The body of the method should print the following single line of output:

Please insert the correct amount of money.

Practice

A2.5: (cont.):
4) Add a *showPrice* method to the *TicketMachine* class. This should have a void return type and take no parameters. The body of the method should print:

The price of a ticket is xyz cents.

where *xyz* should be replaced by the value held in the *price* field when the method is called.

Practice

A2.5: (cont.):

5) Give the class two constructors. One should take a single parameter that specifies the price, and the other should take no parameter and set the price to be a default value of your choosing. Test your implementation by creating machines via the two different constructors.

6) Implement a method called *empty* that simulates the effect of removing all money from the machine. This method should have a *void* return type, and its body should simply set the *total* field to zero.

Does this method need to take any parameters? Test your method by creating a machine, inserting some money, printing some tickets, checking the total, and then emptying the machine.

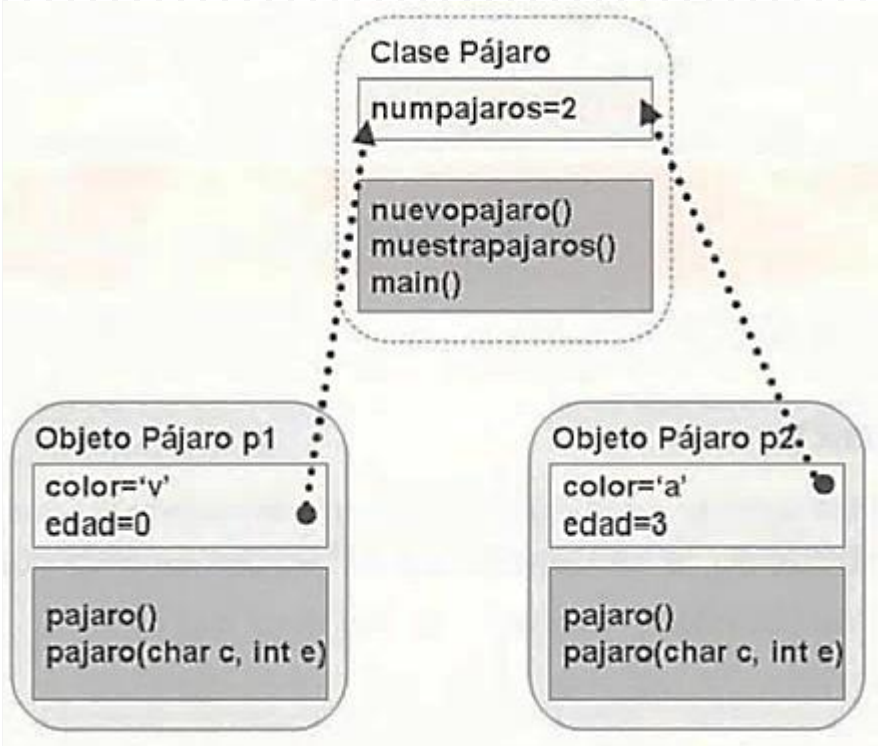
Is the empty method a mutator or an accessor?

Practice

- Thoughts about our ticket machine:
  - Their behavior is inadequate in several ways:
    - No checks on the amounts entered.
    - No refunds.
    - No checks for a sensible initialization.
  - How can we do better?
    - We need more sophisticated behavior… (but not for the moment)

Static and dynamic methods/attributes

- *Static*: Declaring a method or attribute as static, means that there is going to be just one instance of that method or attribute.



Static and dynamic methods/attributes

```
public class Pajaro {
    // Attributes or properties
    private static int numpajaros=0;
    private char color;
    private int edad;

    // Methods
    Pajaro(){color='v'; edad=0;nuevoPajaro();} // constructor of Pajaro class
    Pajaro(char c, int e){color=c;edad=e; nuevoPajaro();} // constructor of Pajaro class

    // The rest of the Methods
    static void nuevoPajaro() {numpajaros++;}
    static void muestrapajaros(){System.out.println(numpajaros);}

    public static void main (String[] args){
        Pajaro p1= new Pajaro();
        Pajaro p2= new Pajaro('a',3);
        Pajaro.muestrapajaros();
        Pajaro.muestrapajaros();
    }

}
```

Libraries = packages

- A library or a package is a group of classes that are related:
  - Example: *java.io* is a package that groups classes that provides a program with input and output functionality.
  - The classes inside a library do not all have the same superclass.
  - A package can have more packages inside.
  - Using packages you avoid conflicts. Example: In the same packages you cannot have two classes with the same name.
  - Classes that belong to the same package have privileges in order to access to attributes or methods of other classes inside the package.

Libraries = packages

- *import* Statement:
  - If we want to use a class inside a package, you can use the import statement:
    - We can import just a class. Example:

      import java.lang.System;

    - Or we can import all the classes inside a package/library:

      import java.awt.*;

      (…)

      Frame fr = new Frame ("Panel de ejemplo");

  - Another way to use a class (less "comfortable"):

    java.awt.Frame *fr =new java.awt.Frame ("Panel de ejemplo");*

Libraries = packages

Paquete o librería	Descripción
java.io	Librería de Entrada/Salida. Permite la comunicación del programa con ficheros y periféricos.
java.lang	Paquete con clases esenciales de Java. No hace falta ejecutar la sentencia import para utilizar sus clases. Librería por defecto.
java.util	Librería con clases de utilidad general para el programador.
java.applet	Librería para desarrollar *applets*.
java.awt	Librerías con componentes para el desarrollo de interfaces de usuario.
java.swing	Librerías con componentes para el desarrollo de interfaces de usuario. Similar al paquete awt.
java.net	En combinación con la librería java.io, va a permitir crear aplicaciones que realicen comunicaciones con la red local e Internet.
java.math	Librería con todo tipo de utilidades matemáticas.
java.sql	Librería especializada en el manejo y comunicación con bases de datos.
java.security	Librería que implementa mecanismos de seguridad.
java.rmi	Paquete que permite el acceso a objetos situados en otros equipos (objetos remotos).
java.beans	Librería que permite la creación y manejo de componentes *javabeans*.

Practice

A2.6: Extract the source code of the class Math from the JDK source code " ⇒ "src.zip" ⇒ "Math.java" under folder "java.lang"). Study how constants such as E and PI are defined. Also study how methods such as abs(), max(), min(), toDegree(), etc, are written.

Practice

A2.7: You'll create a simple class called Person with:

- Attributes: name, age, dni, gender (M for man, W for woman), weight and height. These attributes will not be directly accessible.
- Implement 3 different constructors:
  - A default constructor, giving attributes default values (gender M by default).
  - Another one with name, age and gender by parameter and the others with default values.
  - The last constructor takes every attribute by parameter.
- Implement these methods:
  - isAdult: returns if the person is an adult or not.
  - idealWeight: returns if the person has an ideal weight (IMC>=20 and IMC <=25)
  - toString: prints all the information of the person.

Now, create an executable class with a main method where:

- Create 3 objects using the 3 type of constructors.
- Use the methods implemented with the 3 objects just created.