2. Programación Orientada a Objetos (POO)
2.2. Definición de clases

Programación Modular y Orientada a Objetos

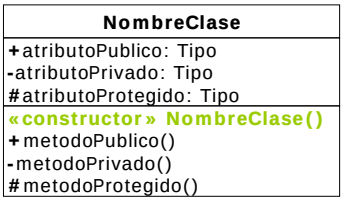
Felipe Ibañez y Juan Miguel Lopez
felipe.anfurrutia@ehu.es
juanmiguel.lopez@ehu.es
Dpto. de Lenguajes y Sistemas Informáticos
UPV/EHU

Contenido

- Definición de clase:
  - Atributos
  - Constructores
  - Métodos
  - La palabra clave this
  - Atributos y métodos de clase (static)
    - Estado de clase vs Estado de objeto
- Tipos de datos
  - Tipos primitivos vs. Tipos referencia
  - Tipos enumerados vs. Tipos Clase
  - Clases Wrapper (Envoltorio)

Estructura básica de la clase

UML



Java

```
public class NombreClase {
    Atributos
    Constructores
    Metodos
}
```

Modificadores de Visibilidad

\+ means public visibility
\- means private visibility
\# means protected visibility

Atributos

- Almacenan valores de un objeto
  - Definen el estado del objeto
- También se conocen como *variables de instancia*

```
public class TicketMachine {
    private int price;
    private int balance;
    private int total;
    //Further details omitted.
}
```

- Nota: En BlueJ se puede ver con *Inspect*

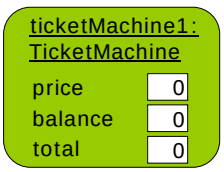
Constructores (definición)

- Tienen el mismo nombre que la clase
- Sirven para inicializar un objeto
  - Asignan valores iniciales a los atributos
  - Pueden tener parámetros para estos valores
- Se utilizan para crear objetos en el programa cliente
- Pueden ser implicitos o explicitos

Constructor Implicito

- No se define ningún constructor, pero el JVM utilizará el de defecto
  - Ej. public TicketMachine()
- Asignará valores por defecto a todos los atributos según su tipo

Uso: Ej. TicketMachine ticketMachine1 = new TicketMachine();



Constructores Explicitos

- Se puede definir el de defecto (sin parámetros)
  - Para asignar valores fijos a los atributos

```
public TicketMachine(){
   price = 10;
   balance = 0;
   total = 0;
}
```

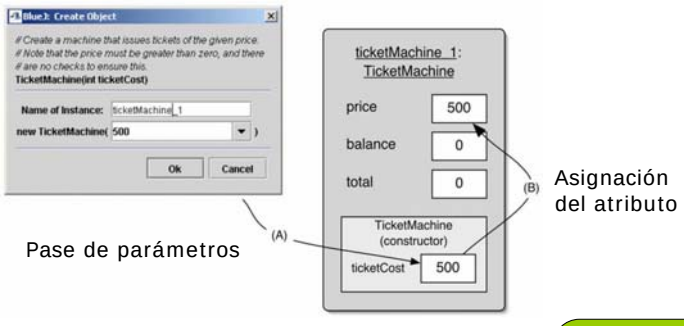
- Constructor sobrecargado (con parámetros)
  - Algunos de los valores a asignar a los atributos se pasan mediante parámetros

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

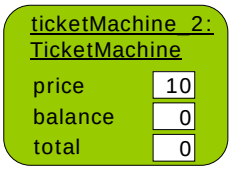
Regla: Si se ha definido algún constructor sobrecargado y NO se ha definido el de defecto, éste no podrá ser utilizado

Constructores (uso)

Ej. TicketMachine ticketMachine_1 = new TicketMachine(500);



Ej. TicketMachine ticketMachine_2 =
new TicketMachine();

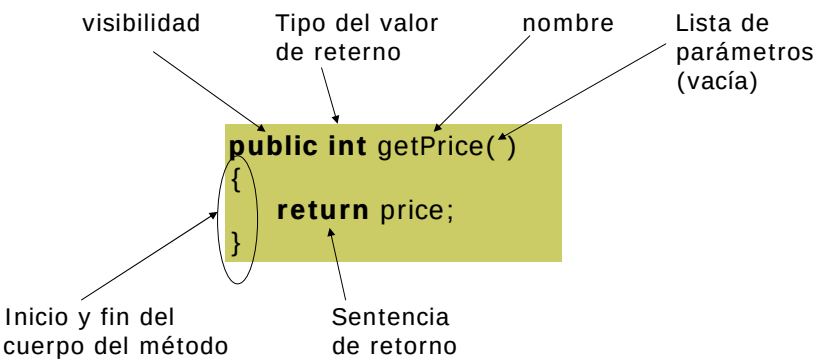


Métodos

- Hay muchos tipos
  - Consulta (Acceso)
  - Modificación
  - Acciones
  - Constructores
  - Destructores
- Tienen
  - Cabecera (*signatura*): nombre, parámetros, valor de retorno
  - Cuerpo (implementación): conjunto de sentencias

Métodos de consulta (acceso)

- Proporcionan información sobre un objeto
- También conocidos como getters



Métodos de modificación

- Sirven para cambiar el estado del objeto
  - Cambiando el valor de una o más variables
  - Utilizan normalmente sentencias de asignación
  - Suelen recibir parámetros de entrada
- También conocidos como setters
  - Ej.

```
public void setPrice (int newPrice) {
    price = newPrice; }
```

Atributo que se modifica

```
public void insertMoney(int amount) {
    if(amount > 0) {
        balance = balance + amount;
    }
    else {
        System.out.println("Use a positive amount: "+amount);
    }
}
```

Métodos de acción: impresion

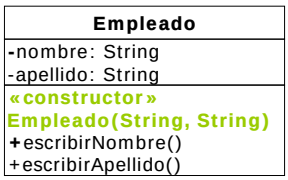
- Realizan algún cálculo o acción (visual)

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

Ejemplo sencillo

- Crear una clase empleado que tenga:
  - dos características
    - el nombre
    - el apellido
  - dos comportamientos
    - escribirNombre()
    - escribirApellido()
- Rellenar el cuerpo de los métodos que tienen el comportamiento para imprimir por pantalla.

Ejemplo sencillo (Empleado.java)



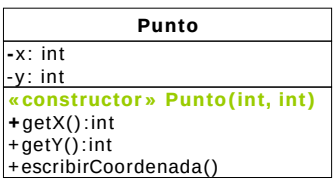

```
public class Empleado {
  private String nombre;
  private String apellido;

  public Empleado(String pNom, String pApell) {
    nombre = pNom;
    apellido = pApell;
  }
  public void escribirNombre() {
    System.out.println(nombre);
  }
  public void escribirApellido() {
    System.out.println(apellido);
  }
}
```

Ejercicio

- Crea una clase Punto que modele un punto en un espacio bidimensional.
  - Tendrá dos atributos, x e y, que guardan las coordenadas.
  - Habrá un constructor sin parámetros que crea un punto en (0, 0) y otro al que se le pueden pasar las coordenadas del punto.
  - También habrá métodos para obtener las coordenadas y para imprimir el punto con el formato (x,y).

Ejercicio (Punto.java)




```
public class Punto {
  private int x;
  private int y;

 public Punto() {
     x = 0;
     y = 0;
  }
  public Punto(int pX, int pY) {
     x = pX;
     y = pY;
  }
  public int getX() {
      return x;
  }
  public int getY() {
      return y;
  }
  public void escribirCoordenada() {
      System.out.println("("+x+","+y+")");
  }
}
```

Contenido

- Definición de clase:
  - Atributos
  - Constructores
  - Métodos
  - La palabra clave this
  - Atributos y métodos de clase (static)
    - Estado de clase vs Estado de objeto
- Tipos de datos
  - Tipos primitivos vs. Tipos referencia
  - Tipos enumerados vs. Tipos Clase
  - Clases Wrapper (Envoltorio)

La palabra clave this

- Se usa en la implementación de los métodos de instancia ó constructores
  - Para solucionar ambigüedades ó
  - Para reutilizar código
- Referencia el objeto actual:
  - el que ha recibido el mensaje ó
  - el que se está construyendo
- Mas información:
  - http://download.oracle.com/javase/tutorial/java/javaOO/thiskey.html

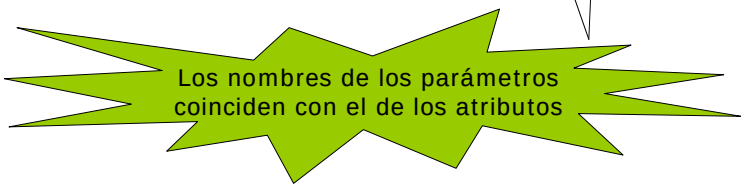
La palabra clave this (solución a la ambigüedad)

- Sin utilizar this

```
public class Punto {
   private int x;
   private int y;
   public Point(int pX, int pY){
         x = pX;
         y = pY;
   }
}
```

- Utilizando this

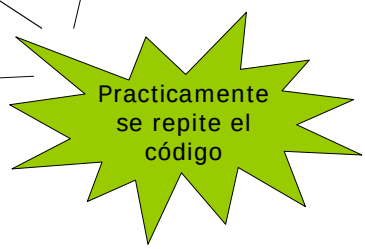
```
public class Point {
   public int x;
   public int y;
   public Point(int x, int y) {
         this.x = x;
         this.y = y;
   }
}
```



Problema de eficiencia

```
public class Rectangle {
    private int x, y;
    private int width, height;
    public Rectangle() {
          this.x = 0;
          this.y = 0;
          this.width = 0;
          this.height = 0;
    }
    public Rectangle(int width,
                     int height) {
          this.x = 0;
          this.y = 0;
          this.width = width;
          this.height = height;
    }
```

```
    public Rectangle(int x, int y,
                     int width, int height) {
          this.x = x;
          this.y = y;
          this.width = width;
          this.height = height;
    }
    ...
}
```



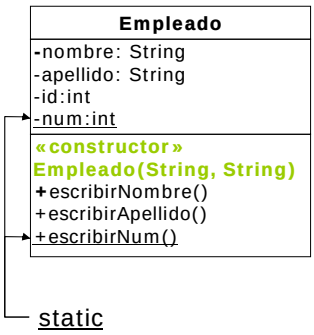
La palabra clave this (reutilización: solución a la eficiencia)

```
public class Rectangle {
    private int x, y;
    private int width, height;
    public Rectangle() {
        this(0, 0, 0, 0);
    }
    public Rectangle(int width, int height) {
        this(0, 0, width, height);
    }
    public Rectangle(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }
    ...
}
```

Atributos y métodos de clase (static)

- A veces algunas características son compartidas por las instancias de una clase
  - Ej: todos los coches tiene 4 ruedas + repuesto
- Métodos y atributos estáticos
  - Identidicados mediante la palabra clave: **static**
  - **El valor de un atributo estático se almacena en la clase, no en la instancia**
  - Los métodos estáticos sólo pueden acceder a los atributos estáticos
- Más información:
  - http://download.oracle.com/javase/tutorial/java/javaOO/classvars.html

Atributos y métodos de clase (Ejemplo)



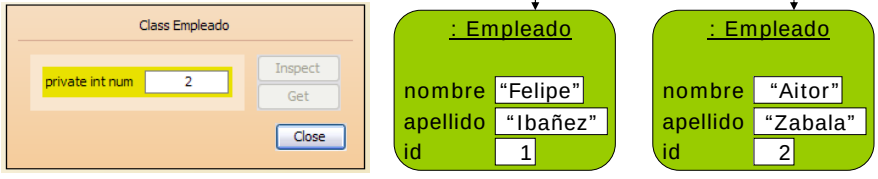

```
public class Empleado {
  private String nombre;
  private String apellido;
  private int id;
  private static int num = 0;
  public Empleado(String pNom, String pApell) {
    nombre = pNom;
    apellido = pApell;
    num++;
    id = num;
  }
  public void escribirNombre() {
    System.out.println(this.nombre);
  }
  // Se puede utilizar desde la clase
  public static void escribirNum() {
     System.out.println(num); // No se puede utilizar THIS
  }
  ...
}
```

Estado de clase vs. de objeto

```
...
Empleado empleado1 = new Empleado("Felipe", "Ibañez");
Empleado empleado2 = new Empleado("Aitor", "Zabala");
empleado1.escribirNombre();
Empleado.escribirNum();
...
```

Pantalla: Felipe
2

Estado de la memoria



Atributos y métodos de clase (Ejemplo)

- http://download.oracle.com/javase/tutorial/java/javaOO/classvars.html

Contenido

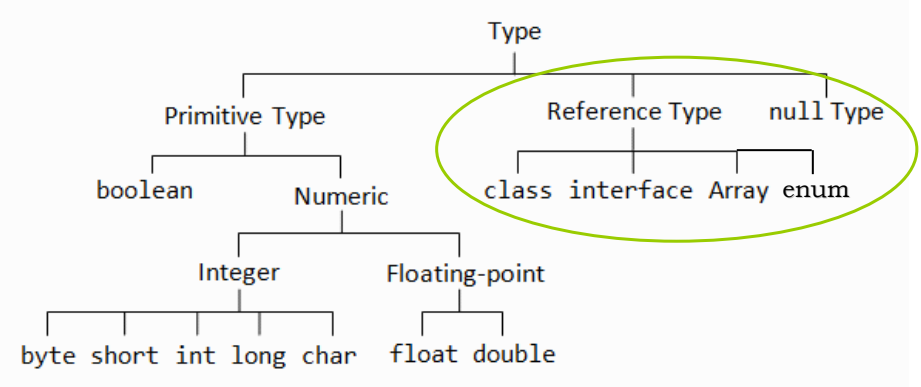
- Definición de clase:
  - Atributos
  - Constructores
  - Métodos
  - La palabra clave this
  - Atributos y métodos de clase (static)
    - Estado de clase vs Estado de objeto
- Tipos de datos
  - Tipos primitivos vs. Tipos referencia
  - Tipos enumerados vs. Tipos Clase
  - Clases Wrapper (Envoltorio)

Tipos de datos

- Definen un rango de valores y conjunto de operaciones
- Pueden ser definidos por el lenguaje de programación o el programador
- Se utilizan en la definición de las variables y parámetros
- Distintos tipos:

Tipo	Definidos por	Nombre	Rango de valores	Operaciones
Primitivo	Lenguaje	*int* (entero)	$[-2^{31},2^{31}-1]$	+, -, *, /, %
Clase (referencia)	Lenguaje / programador	*Coche*	Indefinido	arrancar(), frenar(), …
Enumerado (referencia)	Lenguaje / programador	*DiaSemana*	LUNES, MARTES, …, DOMINGO	numeroDia()
Null	Lenguaje		null	

Tipos de datos en Java



Tipos primitivos vs. Tipos referencia

Operaciones | **Estado de la memoria**

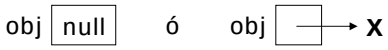
Tipo primitivo

declaración → `int i;`

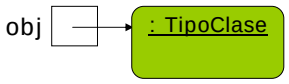
asignación → `i = 22;`

Tipo clase (referencia)

declaración → `TipoClase obj;`



creación y asignación → `obj = new TipoClase();`



Tipos primitivos vs. Tipos referencia

Operaciones | **Estado de la memoria**

Tipo primitivo

declaración → `int a, b;`

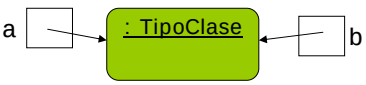
asignación → `a = 22;`

copiar → `b = a;`

Tipo clase (referencia)

declaración → `TipoClase a, b;`

creación y asignación → `a = new TipoClase();`



Tipos enumerados

- Los tipos enumerados sirven para restringir el contenido de una variable a una serie de valores predefinidos.
- Esto suele ayudar a reducir los errores en nuestro código.
- Pueden ser simples o complejos
- Se pueden definir fuera o dentro de una clase
- En Java la definición es parecida a las clases

Tipos enumerados (ejemplo simple)

- Definición:

```
public enum DiaSemana {
        LUNES, MARTES, MIERCOLES, JUEVES,
                VIERNES, SABADO, DOMINGO;
}
```

Los distintos valores se separan mediante coma

- Uso:

```
...
DiaSemana dia = DiaSemana.LUNES;
switch (dia) {
        case DOMINGO:
                System.out.println("Ma√±ana a currar");
        case SABADO:
                System.out.println("Hoy fiesta!!"); break;
        default:
                System.out.println("A currar");
}
```

Tipos enumerados (Ejemplo complejo)

- Son "clases especiales", por lo tanto, se pueden usar atributos, constructores y métodos para su definición

```
public enum Vaso {
  // Tipos de vaso disponibles. Pasan al constructor su capacidad en cc.
  JARRA(500), TUBO(250), TERCIO(333), CAÑA(200);
  // Variable interna donde almacenaremos la capacidad
  private final int cc;
  // Nuestro constructor nos fuerza a pasar parámetros al definir un nuevo valor
   Vaso(int pCC) { cc = pCC; }
   // Devuelve la capacidad del vaso
   public int getCentimetrosCubicos() {
         return cc;
   }
}
```

Devuelve el nombre del valor predefinido

```
 public static void main(String[] pArgs) {
    Vaso vaso = Vaso.JARRA;
    System.out.println("Este vaso es de tipo " + vaso.name() +
       " y su capacidad es de " + vaso.getCentimetrosCubicos());
}
```

Tipos enumerados (Ejemplo complejo)

```
public enum Planetas {
   MERCURIO (3.303e+23, 2.4397e6),
   VENUS    (4.869e+24, 6.0518e6),
   TIERRA   (5.976e+24, 6.37814e6),
   MARTE    (6.421e+23, 3.3972e6),
   JUPITER  (1.9e+27,   7.1492e7),
   SATURNO  (5.688e+26, 6.0268e7),
   URANO    (8.686e+25, 2.5559e7),
   NEPTUNO  (1.024e+26, 2.4746e7);

  private final double masa;
  private final double radio;

  Planeta(double pMasa, double pRadio) {
       masa = pMasa;
       radio = pRadio;
}
…
```

Los valores predefinidos inicializan sus atributos mediante el constructor correspondiente

Tipos enumerados (Ejemplo complejo)

```
    public double masa()   {
        return masa;
    }

    public double radio() {
        return radio;
    }

    public static final double G = 6.67300E-11;

    public double gravedadSuperficie() {
        return G * masa / (radio * radio);
    }

    public double pesoSuperficie(double pMasa) {
        return pMasa * gravedadSuperficie();
    }
}
```

Tipos enumerados (Ejemplo complejo)

Devuelve en un array todo el rango de valores predefinidos por el Tipo enumerado

Devuelve el nombre del valor predefinido

```
public static void main(String[] pArgs) {
  double pesoTierra = Double.parseDouble(args[0]);
  double masa = pesoTierra/Tierra.gravedadSuperficie();
  for (Planeta p: Planeta.values())
      System.out.println("Tu peso en el planeta " + p.name() +
                         " es " + p.pesoSuperficie(masa));
}
```

- Itera sobre la colección de valores.
- Sigue el patrón de **for-each** object **in** a collection

Tipos enumerados (Ejemplos)

- http://francho.org/lab/519/java-ejemplo-de-uso-de-tipos-enumerados-enum/
- http://monillo007.blogspot.com/2008/01/los-tipos-enumerados-enums-en-java.html
- http://download.oracle.com/javase/tutorial/java/javaOO/enum.html

Tipos enumerados vs. Tipos Clase

- Se definen mediante atributos, constructores y métodos
- El rango de valores está prefijado en la definición
- NO se utiliza el operador new
- Las variables de tipo enumerado sólo pueden contener uno de los valores prefijados

- Se definen mediante atributos, constructores y métodos
- Se pueden crear todos los objetos (valores) que se quieran
- Para crear objetos se utiliza el operador new
- Las variables de tipo clase pueden contener un objeto de la misma clase o de una de sus subclases

Clases Wrapper (Envoltorio)

- Los tipos primitivos (ej. int) no son clases
- Pero algunas veces, necesitamos utilizar tipos primitivos en un contexto dónde requiere manipular objetos, no datos primitivos
  - e.g. en estructuras de datos genéricas, de Objects
- Java proporciona un conjunto de clases wrapper para tratar datos primitivos como objeto
- Proporciona una clase wrapper específica para cada tipo primitivo
- Estan en el paquete java.lang, por lo tanto se pueden utilizar universalmente

Clases Wrapper

Clase	*corresponde a*	**Tipo Primitivo**
Boolean		**boolean**
Character		**char**
Byte		**byte**
Short		**short**
Integer		**int**
Long		**long**
Float		**float**
Double		**double**

Cada uno:

• nos permite manipular datos primitivos como objetos

• contiene métodos útiles de conversión

Ej. `Integer` contiene

Definición: `static Integer valueOf(String s)`

Uso: `Integer.valueOf("27")`

devuelve una instancia de tipo Integer, cuyo contenido es el int: 27

Clases Wrapper

Utilizar wrappers para conversiones entre objetos y primitivas:

```
// crear e inicializar un int
int i = 7;

// crear un objeto Integer y convertir el int en él
Integer intObject = new Integer( i );

// recuperar el int desenvolviéndolo(unwrapping it) desde el objeto
System.out.println( intObject.intValue() );

// convertir un string en un objeto Integer
String strS = "27";
Integer intObject;
intObject = Integer.valueOf(strS);

// después obtener el int
i = intObject.intValue();
```