Objetos

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Programación y Diseño Orientado a Objetos

(Curso 2019-2020)

Créditos

- Las siguientes imágenes e ilustraciones son libres y se han obtenido de:
  -  `https://pixabay.com/images/id-37254/`
  - Emojis, `https://pixabay.com/images/id-2074153/`
  -  `https://pixabay.com/images/id-2495144/`
  -  `https://pixabay.com/images/id-3687611/`
  -  `https://pixabay.com/images/id-29094/`
- El resto de imágenes e ilustraciones son de creación propia, al igual que los ejemplos de código

Objetivos

- Tener un primer contacto con el paradigma de la programación orientada a objetos
- Entender los siguientes conceptos:
  - Objeto
  - Clase
  - Identidad
  - Estado
  - Comportamiento
- Entender ejemplos sencillos

Contenidos

Concepto de Objeto

- Entidad perfectamente delimitada, que **encapsula estado y funcionamiento** y **posee una identidad** (OMG 2001)
- Elemento, unidad o entidad individual e identificable, real o abstracta, con un **papel bien definido en el dominio del problema** (Dictionary of Object Technology 1995)

- ¿Qué significa cada una de estas frases?
  - Soy un ejemplar de Lápiz
  - Soy único
  - Mi color es el verde
  - Como todos los lápices, puedo dibujar
  - Yo, además, puedo borrar

Clase

- La clase, entre otras cosas, actúa de **molde o plantilla** para la creación de objetos
  - En algunos lenguajes las clases son también objetos a todos los efectos
- Los **objetos** creados a partir de una clase se denominan **instancias** de esa clase
  - Esos objetos *pertenecen* o simplemente *son* de esa clase.
- **Una clase crea un tipo de dato**. Se pueden declarar variables de ese tipo o clase (si el lenguaje dispone de este mecanismo)

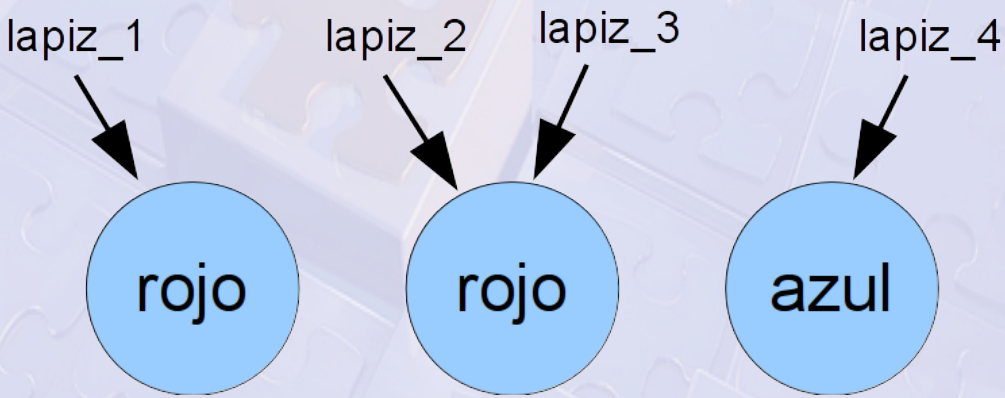
Java: Instanciando clases, creando objetos

```
1 Lapiz miLapiz = new Lapiz (Color.Rojo);
2 Lapiz tuLapiz = new Lapiz (Color.Verde);
```

Identidad

- **Cada instancia tiene su propia identidad**
- La identidad la define la **posición de memoria**
- Independientemente de su estado, objetos distintos residirán en zonas de memoria distintas

★ *En el ejemplo, ¿qué lápices son iguales a otros?, ¿se puede considerar más de un criterio de igualdad?, ¿cuáles?*



Estado y Comportamiento

- El **estado** de un objeto vendrá definido por los **valores de sus atributos**
  - Cada objeto tiene una **zona de memoria propia** para el almacenamiento de sus atributos
- Los objetos exhiben **comportamiento**
  - Disponen de una serie de **métodos** (funciones o procedimientos) que pueden ser llamados/invocados

Ejemplo en C++: Invocando métodos de objetos

```
1  Persona amparo("Amparo"), samuel("Samuel");
2  Persona *cristina = new Persona ("Cristina"); //Puntero C++
3  // ...
4  amparo.saluda()     // Se invoca al método: saluda
5  // "Hola, me llamo Amparo"
6  samuel.saluda()     // otra instancia distinta
7  // "Hola, me llamo Samuel"
8  cristina->saluda() // C++
9  // "Hola, me llamo Cristina"
```

Estado y Comportamiento

Paradigma de Programación Orientada a Objetos

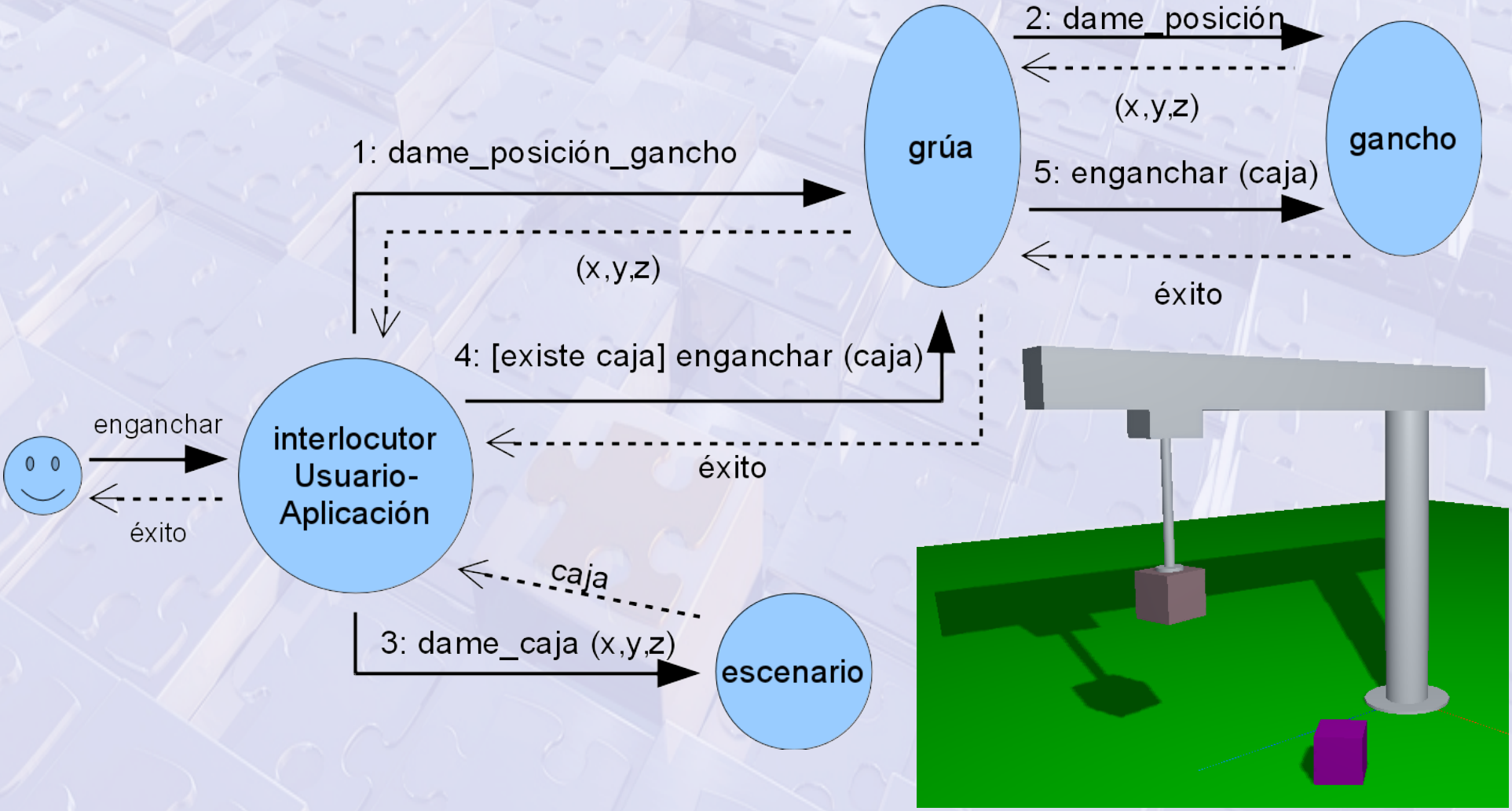
- **Paradigma**: Teoría o conjunto de teorías cuyo núcleo central se acepta sin cuestionar y que suministra la base y modelo para resolver problemas y avanzar en el conocimiento (R.A.E)
- **Paradigma de programación**: Conjunto de reglas que indican como desarrollar software
- Base de la **orientación a objetos**: Se unen los datos y el procesamiento en entidades denominadas objetos

Programación Orientada a Objetos (POO)

- **Los objetos son las entidades que *se manejan* en el software**
- Programar *consiste* en **modelar** el problema mediante un universo dinámico de objetos
- Cada objeto pertenece a una **clase** y como tal, tiene una **responsabilidad** dentro de la aplicación
- La funcionalidad del programa se obtiene haciendo que unos objetos *le pidan* a otros objetos (envío de mensajes) *que hagan cosas* (ejecución de métodos)
  - *Se deben programar los métodos de cada clase de manera que cada objeto se ocupe de lo suyo y no haga el trabajo de otro* 
- El objetivo es obtener **alta cohesión** y **bajo acoplamiento**

Ejemplo

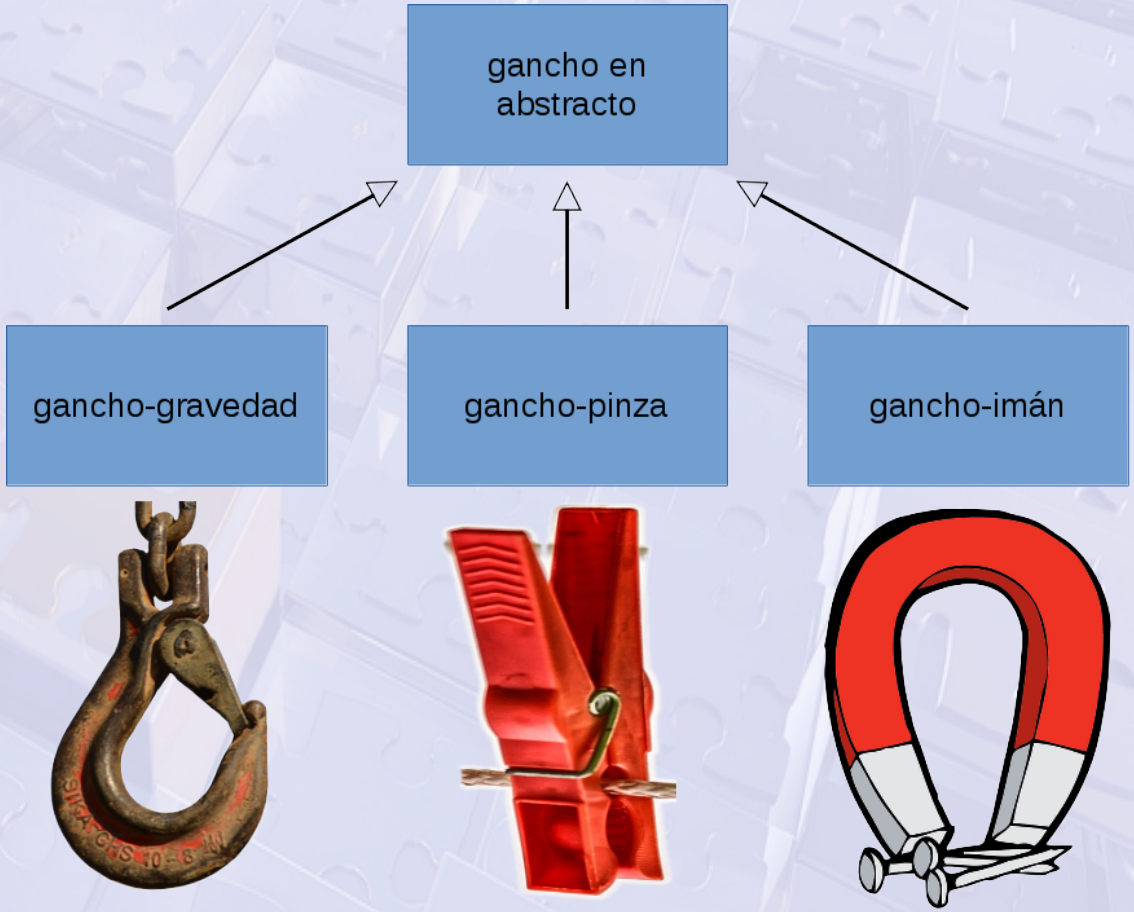
Simulador de entrenamiento de operario de grúa



POO

Conceptos básicos

- El paradigma se basa en los siguientes conceptos:
  - Clase
  - Objeto o instancia
  - Estado
  - Identidad
  - Mensaje
  - Herencia
  - Polimorfismo



Primeros Ejemplos

- Este es el aspecto de dos ejemplos de programas orientados a objetos escritos en los lenguajes de programación Java y Ruby
- Estos ejemplos servirán de ayuda para comenzar el trabajo de las prácticas de la asignatura. En ellos aparecen elementos que veremos con detalle más adelante

Java: Ejemplo básico

```
1 package basico;
2
3 //Enumerado con visibilidad de paquete
4 /*public*/ enum ColorPelo { MORENO, CASTAÑO, RUBIO, PELIROJO }
5
6   public class Persona { // Clase con visibilidad publica
7   private String nombre;   //Atributos de instancia privados
8   private int edad;
9   private ColorPelo pelo;
10
11  public Persona (String n,int e,ColorPelo p) { // Constructor público
12    nombre=n;
13    edad=e;
14    pelo=p;
15  }
16
17  void saluda() { // Visibilidad de paquete. Método de instancia
18    System.out.println("Hola, soy "+nombre);
19  }
20 }
21
22 public class Basico { //Clase con programa principal
23   public static void main(String[] args) {
24       Persona p=new Persona("Pepe",10,ColorPelo.RUBIO);
25       p.saluda();
26   }
27 }
```

Ruby: Ejemplo básico

```ruby
#encoding: UTF-8
module Basico
  module ColorPelo
    MORENO=     :moreno
    CASTAÑO=    :castaño
    RUBIO=      :rubio
    PELIROJO=   :pelirojo
  end

  class Persona
    def initialize(n,e,p) # "constructor"
      # Atributos de instancia (son privados)
      @nombre=n
      @edad=e
      @pelo=p
    end

    public # aunque los métodos son públicos por defecto
    def saluda # Método público de instancia
      puts "Hola, soy "+@nombre
    end
  end

  p=Persona.new("Pepe",10,ColorPelo::RUBIO)
  p.saluda
end
```

Java: Uso desde otro paquete

```java
// En otro paquete
package otroPaquete;

import basico.Persona;
import basico.ColorPelo;    // Error: ColorPelo no tiene visibilidad pública

// ...

Persona manolo = new Persona ("Manolo", 20, ColorPelo.MORENO);
// Error: No se reconoce el símbolo  ColorPelo

System.out.println (manolo.toString());
// basico.Persona@33909752
// Para que muestre información útil hay que redefinir  toString()
```

Ruby: Uso desde otro módulo

```ruby
# Fuera del módulo que hemos llamado "Basico"

manolo = Basico::Persona.new("Manolo", 20, Basico::ColorPelo::MORENO)

puts manolo.inspect
#< Basico::Persona:0x5571 @nombre="Manolo",@edad=20,@pelo=:moreno >
```

Objetos y clases → *Diseño* ←

- Determinar qué objetos (y por extensión, qué clases) van a modelar mejor el sistema no es una tarea fácil
  - Hablamos de “Diseñar Software”, algo que se empieza a aprender en la titulación, y no se termina de aprender nunca
- No obstante, las clases:
  - Deben tener una responsabilidad muy concreta
    - Si una clase se ocupa de muchas cosas, tal vez haya que crear varias clases y distribuir responsabilidades
  - Deben ser, en cierta medida, “autónomas”
    - Si una clase se ve muy afectada por cambios realizados en otras clases, tal vez esa clase tiene responsabilidades que no le corresponden
  - Deben ser “introvertidas” y “no altruistas”
    - El estado de una clase solo debe modificarse desde la propia clase
    - Ninguna clase debe hacer el trabajo que le corresponde a otra clase

Objetos

Dpto. Lenguajes y Sistemas Informáticos
Universidad de Granada

Programación y Diseño Orientado a Objetos

(Curso 2019-2020)