

Programación

Práctica 1: POO Programación Orientada a Objetos

Josué Ferri Pascual

jferri@dsic.upv.es

Objetivos

→ Ser capaz

- Generar tipos de datos propios tipos, basándose en las características de la POO
- Crear diferentes instancias de tipos propios

→ Ser capaz de diferenciar entre los conceptos de

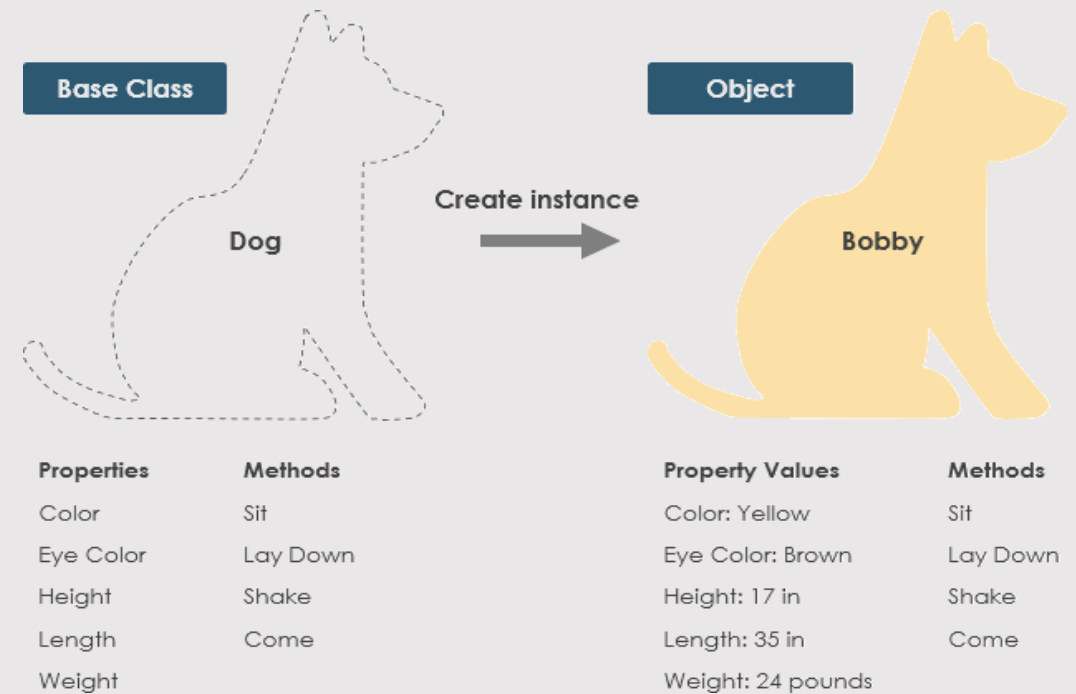
- Clase
- Instancia
- Encapsulación

→ Ser capaz de diferenciar entre

- Miembro de instancia
- Miembro de clase

→ Ser capaz de desarrollar:

- Una aplicación completa,
- Organizada en diferentes paquetes y ficheros



POO

**Recordatorio
programación
orientada a objetos
Uso con IDE**

- Ocultación, herencia y polimorfismo
- Clases e instancias
- Tipos primitivos y referencias
- Ocultación Getters y Setters
- Ciclo de vida
- Variables de clase vs. instancia

Programación orientada a objetos

→ Los lenguajes POO, como Java, C++, Python, C# o Dart, ofrecen características como las siguientes:

→ **Ocultación:**

- La capacidad de un tipo de ocultar al usuario parte de su funcionamiento, de modo que el usuario solo conoce qué puede hacer con el objeto, pero no cómo funciona internamente (para más información, ver interfaces)

➤ **Herencia:**

- La posibilidad de reaprovechar código entre clases mediante relaciones de tipo is-a

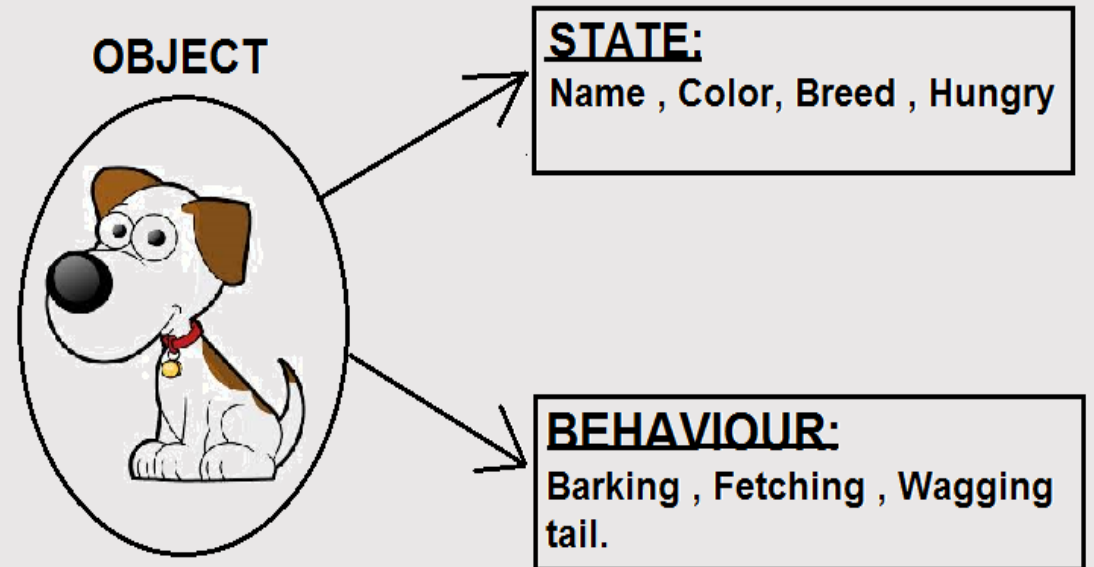
➤ **Polimorfismo:**

- Según el tipo de una instancia, frente a una misma operación, el objeto se puede comportar de una forma u otra

Clases

→ Clases

- Una **clase** permite que el usuario pueda definir un tipo propio en base a:
 - Un estado (variables)
 - Comportamiento (métodos).
- Una **clase** podemos verla como una **plantilla** que recoge las características del tipo declarado.



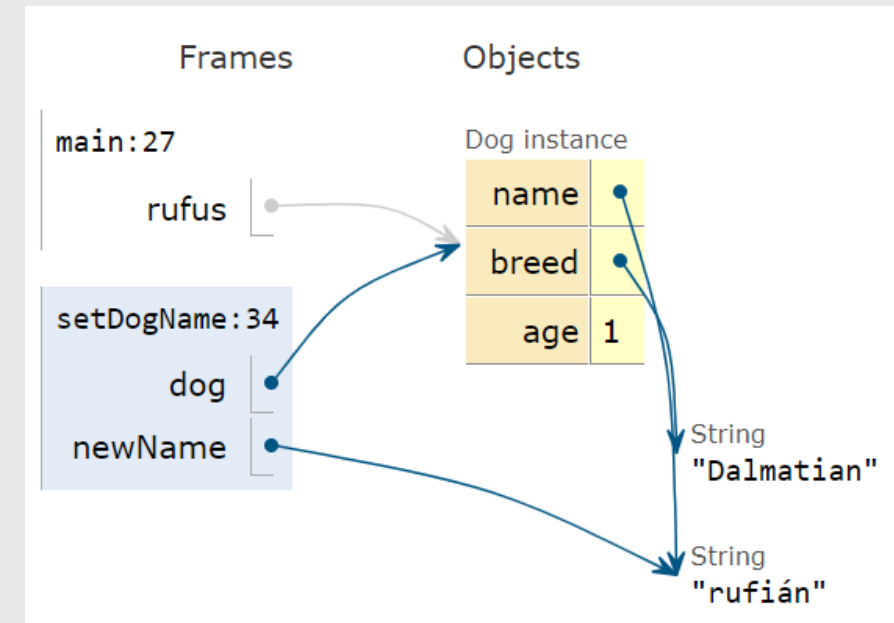
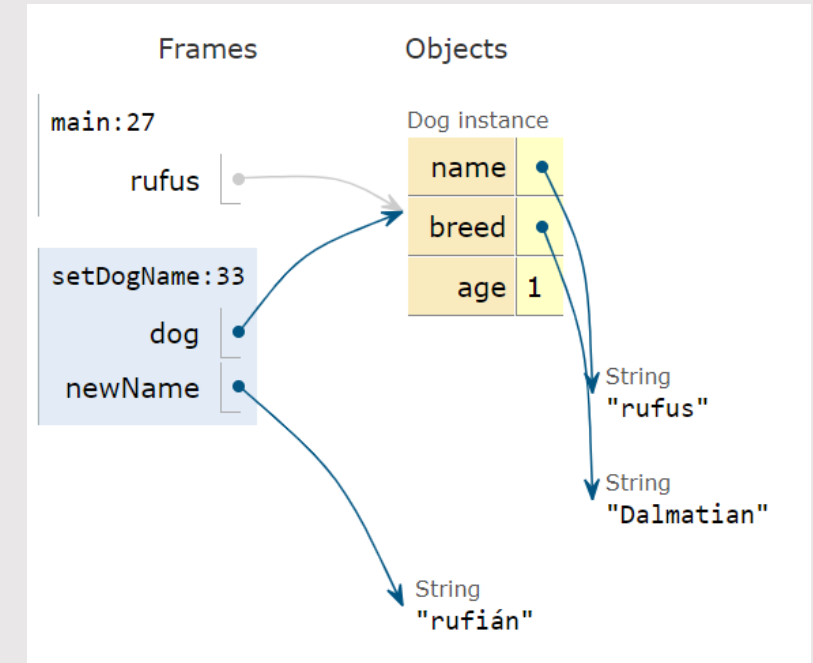
Clases e instancias

```
public class Dog {  
    // State  
    public String name;  
    public String breed;  
    public int age;  
    // Behaviour  
    public void bark() {  
        System.out.println("barking");  
    }  
    public void wagTail() {  
        System.out.println("wagging tail");  
    }  
}
```

```
public class Demo {  
    public static void main(String[] args) {  
        Dog rufus = new Dog();  
        rufus.name = "rufus";  
        rufus.breed = "Dalmatian";  
        rufus.age = 1;  
        rufus.bark();  
    }  
}
```

Tipos primitivos y referencias

```
public class Demo {  
    public static void main(String[] args) {  
        Dog rufus = new Dog();  
        rufus.name = "rufus";  
        rufus.breed = "Dalmatian";  
        rufus.age = 1;  
        setDogName(rufus, "rufián");  
        rufus.bark();  
    }  
    public static void setDogName(Dog dog, String newName) {  
        dog.name = newName;  
    }  
}
```



Ocultación Getters y Setters

```
public class Dog {  
    // State  
    private String name;  
    private String breed;  
    private int age;  
    // getters y setters  
    public String getName() {  
        return this.name;  
    }  
    public void setName(String name) {  
        this.name = name;  
    }  
    // Behaviour  
    public void bark() {...}  
    public void wagTail() {...}  
}
```

```
public class Demo {  
    public static void main(String[] args) {  
        Dog rufus = new Dog();  
        // Esto dará error  
        // rufus.name = "rufus";  
        rufus.setName("rufus");  
        System.out.println(rufus.getName());  
        rufus.bark();  
    }  
}
```


Clase: constructor, getters, setters, comportamiento

```
public class Dog {  
    // Ciclo de vida  
    private String name;  
    private String breed;  
    private int age;  
  
    // Constructor  
    public Dog(String name, String breed, int age) {  
        this.name = name;  
        this.breed = breed;  
        this.age = age;  
    }  
}
```

```
// getters y setters  
public String getName() {  
    return this.name;  
}  
  
public void setName(String name) {  
    this.name = name;  
}  
  
// Behaviour  
public void bark() {  
    System.out.println("barking");  
}  
  
public void wagTail() {  
    System.out.println("wagging tail");  
}  
}
```

Ciclo de vida

→ Objeto, ciclo de vida desde su reserva hasta que se libera

- Instanciación: Reserva de memoria
- Inicialización: Preparación del estado del objeto para su uso
- Uso: Realización operaciones
- Finalización: Liberar posibles dependencias (e.g., cerrar un fichero)
- Destrucción: Liberación de memoria del propio objeto

```
public class MainCiclo {  
    public static void main(String[] args) {  
        // Fase 1 => new; Fase 2 => Constructor  
        // Dog rufus=new Dog();  
        Dog rufus = new Dog("Rufus", "Dalmatian", 2);  
        // Fase 3 usamos el objeto  
        rufus.setName("rufián");  
        System.out.println(rufus.getName());  
        rufus.bark();  
    } // Fase 4 => El objeto sale de su ámbito y no hay  
    // referencias al mismo  
} // Fase 5 => Se libera la memoria del mismo
```

Variables de clase vs. instancia

→ Cada instancia tiene sus propias variables

→ Variables de clase son compartidas por todas las instancias

```
public class Dog {  
    // State  
    private String name;  
    private String breed;  
    private int age;  
    private static int numDogs = 0; // variable de clase  
    // Constructor  
    public Dog(String name, String breed, int age) {  
        this.name = name;  
        this.breed = breed;  
        this.age = age;  
        numDogs++; // incrementar numDogs  
    }  
    // Getters & Setters  
    public static int getNumDogs() {  
        return numDogs;  
    }  
}
```

Variables de clase vs. instancia

→ Cada instancia tiene sus propias variables

→ Variables de clase son compartidas por todas las instancias

→ Se puede acceder:

- Directamente indicando la clase
- Desde la instancia

```
public class Demo {  
    public static void main(String[] args) {  
        System.out.println(Dog.getNumDogs()); // 0  
        Dog rufus = new Dog("Rufus", "Dalmatian", 2);  
        System.out.println(Dog.getNumDogs()); // 1  
        System.out.println(rufus.getNumDogs()); // 1  
        Dog gaby = new Dog("Gaby", "Golden Retriever", 4);  
        System.out.println(Dog.getNumDogs()); // 2  
        Dog linda = new Dog("Linda", "Bulldog", 1);  
        System.out.println(Dog.getNumDogs()); // 3  
    }  
}
```

Práctica 1

Tareas a realizar

- Tarea 1 - ClothingItem
- Tarea 2 - Inventory
- Tarea 3 - Main, parte 1
- Tarea 4 - SalesRegister
- Tarea 5 - Main, parte 2

Objetivos

→ Desarrollo software de gestión de una tienda

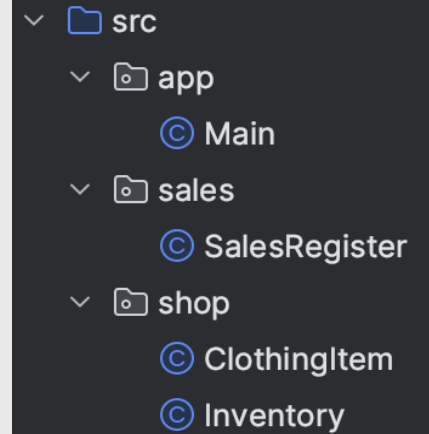
- Utilizado por un empleado de la tienda
- Realizar diferentes acciones sobre un inventario de prendas
- Gestión de registro de ventas

→ POO

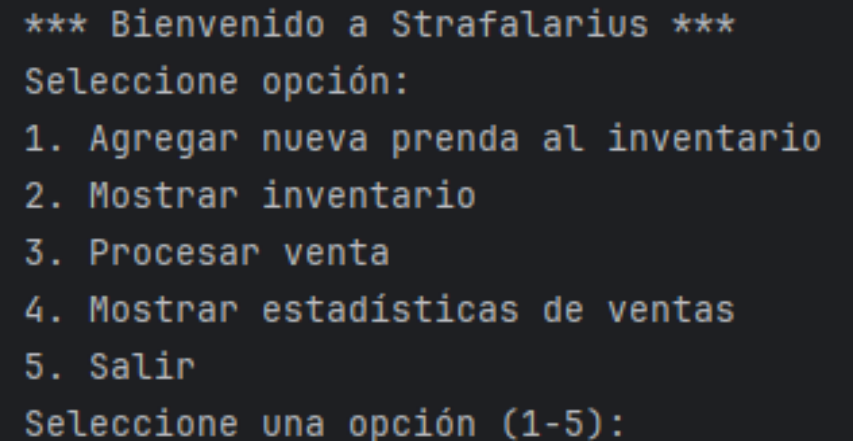
- Estructura de paquetes
- Organizado según instrucciones

→ Autoevaluación y seguimiento

- Código para autoevaluación
- Uso Herramienta Github



```
src
├── app
│   └── Main
├── sales
│   └── SalesRegister
├── shop
│   ├── ClothingItem
│   └── Inventory
```



```
*** Bienvenido a Strafalarious ***
Seleccione opción:
1. Agregar nueva prenda al inventario
2. Mostrar inventario
3. Procesar venta
4. Mostrar estadísticas de ventas
5. Salir
Seleccione una opción (1-5):
```

<https://www.youtube.com/watch?v=INoFJVWqWK0>

Tarea 1 - ClothingItem

- **Crea un paquete shop**
- **Crea clase ClothingItem dentro de shop**
- **Variables de la clase**
 - name tipo String
 - price tipo double
 - size tipo char (solo podrá tener los valores 'S', 'M' o 'L')
- **Declara constructor**
 - Mismo orden
- **Aplica principio ocultación**
 - Evitar acceso directo a variables
 - Getters
 - Setters
- **Sobrescribir toString()**
 - Devolverá la información de las variables

Tarea 2 - Inventory

→ Crea clase Inventory dentro de shop

- Array de la clase ClothingItem
- Variable de instancia itemLength nos indicará cuantas prendas hay
- Variable de instancia, constante, MAX_SIZE, capacidad máxima

→ Declara constructor

- Indicando MAX_SIZE

→ Declara métodos

- getItemCount para obtener número de items en inventario
- addItem permitirá añadir prendas al inventario
- checkStock comprobar si hay stock para una prenda
- removeItem para eliminar una prenda del inventario
- extractItem para encontrar una prenda, dados su name y size, devolverá el objeto ClothingItem

→ Sobrescribir toString()

- Presentar información tabulada

```
Inventario: => itemLength=3, MAX_SIZE=3
```

Nombre	Precio	Talla

Socks	5.99	S
Hat	14.99	M
Gloves	9.99	S

Tarea 3 - Main, parte 1

→ Esta clase representa el punto de entrada de la aplicación

→ Utilizaremos para testear

- Crea una clase Main en el paquete app, con el método main
- En esta clase, declara una variable de clase de tipo Inventory

→ Comprueba los diferentes métodos implementados

- Añadir varias prendas
- Comprobar stock
- Comprobar eliminar prenda
- Imprimir inventario mediante toString()
- Comprobar extractItem
- Verifica que MAX_ITEM limita el número de artículos

→ Verificar con herramienta test

- Crea una clase TestInventory en paquete test.
- Llama desde main a dicha clase según indicaciones del guión

```
Verificando la clase: shop.ClothingItem
Constructor verificado.
Parámetros del constructor verificados.
Atributos verificados.
Getter y setter verificados para: name
Getter y setter verificados para: price
Getter y setter verificados para: size
Método toString verificado.
Atributos privados verificados.
Puntuación total Test.TestClothingItem: 10.0 / 10.0
```

Tarea 4 - SalesRegister

→ Crea clase SalesRegister en paquete sales

- Mantiene un registro (numérico) de las ventas procesadas
- Todos sus miembros serán de clase, no existirán instancias de esta clase.

→ Declara variables

- Variable de clase de tipo long para contar cuantos artículos se han vendido totalSalesCount
- Variable de clase de tipo double para sumar las ventas totales que se han realizado, totalSalesAmount

→ Declara métodos

- processSale, método de clase que procese una venta.
- de clase getTotalSalesAmount() para conocer el recuento de ventas que se han realizado
- de clase getTotalSalesAmount() para conocer cuantas ventas se han procesado
- getBalance() que nos muestre un balance usando los dos anteriores
- ponga a cero la cuenta de productos vendidos resetTotalSalesCount()
- ponga a cero las ventas resetTotalSalesAmount()

→ Verificar con herramienta test

- Crea una clase TestSalesRegister en paquete test.

Tarea 5 - Main, parte 2

- Declara un Scanner como variable de clase e inicialízalo (System.in)
- Declara un método que
 - Imprima las posibles opciones del menú
 - Devuelva la opción elegida por el usuario (teclado)
- Muestra el menú en la estructura de iteración do-while

```
*** Bienvenido a Strafalarious ***  
Seleccione opción:  
1. Agregar nueva prenda al inventario  
2. Mostrar inventario  
3. Procesar venta  
4. Mostrar estadísticas de ventas  
5. Salir  
Seleccione una opción (1-5):
```

```
Seleccione una opción (1-5): 1  
Ingrese los detalles de la prenda:  
Nombre: Camisa  
Precio: 50.9  
Talla (S/M/L): L  
Prenda añadida al inventario correctamente.
```

```
Seleccione una opción (1-5): 2  
Inventario: => itemLength=1, MAX_SIZE=100  
Nombre                Precio                Talla  
-----  
Camisa                50.9                L
```

Práctica 1: POO Programación Orientada a Objetos

Josué Ferri Pascual

jferri@dsic.upv.es