

# Repaso Java

//Parte 2

IT BOARDING

**BOOTCAMP**



# Índice



**01** Herencia,  
sobreescritura y sobrecarga

**02** Identidad vs. Igualdad

**03** Clase Abstracta

**04** Polimorfismo



1

Herencia,  
sobreescritura y  
sobrecarga

IT BOARDING

**BOOTCAMP**





# Herencia

- La relación de herencia se identifica como “**es una especie de**”.
- En Java la herencia se indica con la keyword **extends**.
- Todas las clases heredan de la clase madre “Object”.

## Ejemplos:

**Teléfono Celular** es una especie de **Teléfono**.

**Paloma** es una especie de **Ave**.

**Ave** es una especie de **Animal**.

✖ ✖ ✖

✖ ✖ ✖

✖ ✖ ✖

# Sobreescritura (override)

- Sobreescribir es **reemplazar** la implementación de un método heredado.
- El método se debe declarar con la firma EXACTA del método padre.
- La anotación **@Override** fuerza ese chequeo de sobreescritura en tiempo de compilación. Si bien no es necesaria, su utilización se considera una buena práctica.

```
public class Numero
{
    private int valor;

    @Override
    public String toString()
    {
        return Integer.toString(valor);
    }

    // ...
}
```

# Sobreescritura (overload)

- Sobrecargar es ofrecer distintas firmas para el mismo método

## Distinta cantidad de argumentos

```
public class Multiplier {  
  
    public int multiply(int a, int b) {  
        return a * b;  
    }  
  
    public int multiply(int a, int b, int c) {  
        return a * b * c;  
    }  
}
```

## Distintos tipos de argumentos

```
public class Multiplier {  
  
    public int multiply(int a, int b) {  
        return a * b;  
    }  
  
    public double multiply(double a, double b) {  
        return a * b * c;  
    }  
}
```



# 2

Identidad vs.  
igualdad

IT BOARDING

**BOOTCAMP**



# Igualdad vs. igualdad

- **Identidad.** Un objeto es idéntico a otro si y sólo si **SON EL MISMO OBJETO.**
- **Igualdad.** Un objeto es igual a otro si se definen con **IGUAL LÓGICA e IGUAL COMPORTAMIENTO.**

*¿Cuál será la salida por consola de este fragmento de código?*

```
Numero n = new Numero(5);
Numero m = new Numero(5);

if( n == m )
{
    System.out.println("SON EL MISMO OBJETO!");
}

if( n.equals(m) )
{
    System.out.println("SON OBJETOS IGUALES!");
}
```



# Equals vs. hashCode

- Todos los objetos tienen estos dos métodos, se heredan de “Object”.
- Son los encargados de establecer la **igualdad e identidad** de dos objetos.
- Por defecto el método **equals** solo considera **Identidad**.

```
Numero n = new Numero(5);
Numero m = new Numero(5);

if( n == m )
{
    System.out.println("SON EL MISMO OBJETO!");
}

if( n.equals(m) )
{
    System.out.println("SON OBJETOS IGUALES!");
}
```

# Sobreescribiendo equals

- **Reflexión:** Un objeto debe ser igual a sí mismo. `x.equals(x) == true`.
- **Simetría:** Si `a.equals(b)` entonces `b.equals(a)`.
- **Transitividad:** Si `a.equals(b)` y `b.equals(c)` entonces `a.equals(c)`.
- **Consistencia:** El resultado de `equals()` solo debe cambiar si alguna de las propiedades del objeto contenidas en el `equals`, cambia.

```
@Override
public boolean equals(Object o) {
    return ((Numero) o).getValor().equals(this.valor);
}
```

# Sobreescribiendo hashCode

- Si se sobreescribe equals, se debe sobreescribir hashCode.
- **Consistencia con equals:** Objetos que son equals entre ellos, deben devolver idéntico hashCode.
- **Colisiones:** Objetos con diferente equals, no pueden devolver mismo hashCode
- **Consistencia interna:** El resultado de hashCode() solo debe cambiar si alguna de las propiedades del objeto contenidas en el equals, cambia.

```
@Override  
public int hashCode() {  
    return this.valor;  
}
```

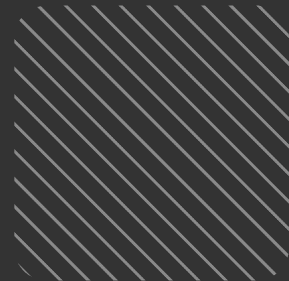


# 3

## Clases Abstractas

IT BOARDING

**BOOTCAMP**





# Clase abstracta

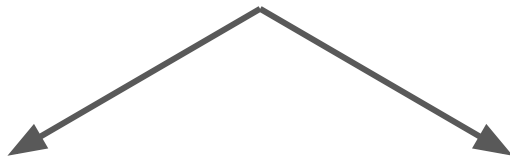
- En java se define con el keyword “**abstract**”.
- No se pueden instanciar, pero si heredar.
- Podrán tener métodos abstractos (sin implementar).
- Una clase concreta que la herede deberá proveer una implementación de los métodos abstractos.



## Fruta <Abstract>

```
public void reproducirse() {caerSemillaAlSuelo(); germinar(); etc}  
public void pudrirse() {implementacion;}
```

```
public abstract Color obtenerColor();  
public abstract boolean esAcida();
```



## Banana <concrete>

```
public Color obtenerColor() {return Color.Yellow}  
public boolean esAcida() {rerurn false;}
```

## Tomate <concrete>

```
public Color obtenerColor() {return Color.Red}  
public boolean esAcida() {rerurn true;}
```



4

Polimorfismo

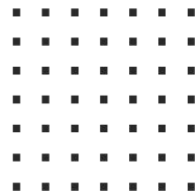
IT BOARDING

**BOOTCAMP**



# Poliformismo

- Un objeto puede representarse en distintas formas simultáneamente.
- Permite ejecutar una acción en diferentes maneras.
- Hay 2 tipos de polimorfismo:
  - En tiempo de compilación (sobrecarga/overload).
  - En tiempo de ejecución (sobreescripción/override).
- Ej. de la vida real: Una persona puede ser al mismo tiempo Padre, Esposo, Amigo, Alumno, Empleado, dependiendo del contexto, variará su comportamiento.





# Poliformismo

## overload

```
public class Multiplier {  
  
    public int multiply(int a, int b) {  
        return a * b;  
    }  
  
    public double multiply(double a, double b) {  
        return a * b;  
    }  
}  
  
class Main {  
    public static void main(String[] args)  
    {  
  
        System.out.println(Multiplier.multiply(2, 4));  
  
        System.out.println(Multiplier.multiply(5.5, 6.3));  
    }  
}
```

## override

```
class Parent {  
  
    void Print()  
    {  
        System.out.println("parent class");  
    }  
}  
  
class subclass1 extends Parent {  
  
    void Print()  
    {  
        System.out.println("subclass1");  
    }  
}  
  
class subclass2 extends Parent {  
  
    void Print()  
    {  
        System.out.println("subclass2");  
    }  
}  
  
class TestPolymorphism3 {  
    public static void main(String[] args)  
    {  
  
        Parent a;  
  
        a = new subclass1();  
        a.Print();  
  
        a = new subclass2();  
        a.Print();  
    }  
}
```

¿Dudas?  
¿Preguntas?

IT BOARDING

**BOOTCAMP**





# Gracias.

IT BOARDING

**BOOTCAMP**

