Ministerio de Producción
Presidencia de la Nación

Ministerio de Educación y Deportes

Subsecretaría de Servicios Tecnológicos y Productivos

Programa 111 mil
VOS PODÉS SER UNO.

La clase Object y sobrecarga de métodos

Programación

- La clase Object y los métodos equals() y toString()
- Sobrecarga de métodos y de constructores
- Ejercicios prácticos en PC

Clase Object

- La clase **Object** es la clase padre (directa o indirectamente) de cualquier clase en Java

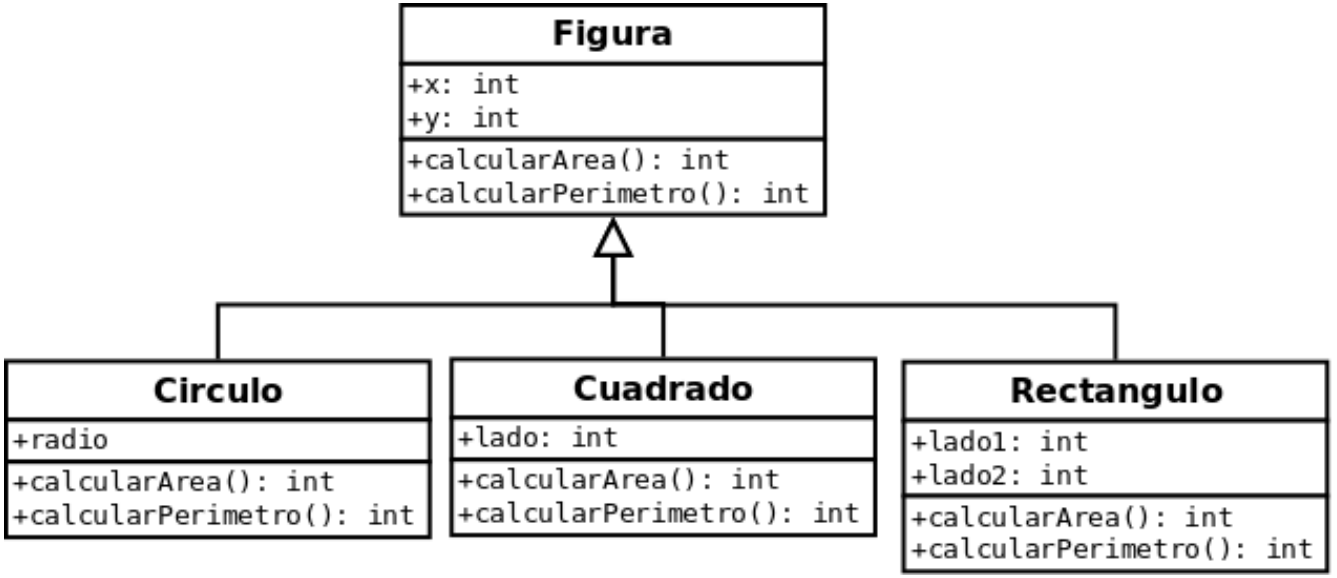
- Posee dos métodos importantes: **equals** y **toString**:

```
public boolean equals(Object obj){...}
public String toString(){...}
```

- Cualquier objeto puede responder a la invocación de estos métodos. Es importante que cada una de las clases lo **sobreescriba** con el comportamiento deseado

Repaso: Sobreescritura de métodos

- La sobreescritura nos permite que una clase hija pueda *redefinir* métodos de su clase padre (o ancestro)
- Los métodos redefinidos deben respetar el nombre, parámetros y tipo de retorno
- Ejemplo:



Ahora sí… cómo funciona equals()?

La versión original **public boolean equals(Object obj){...}**:

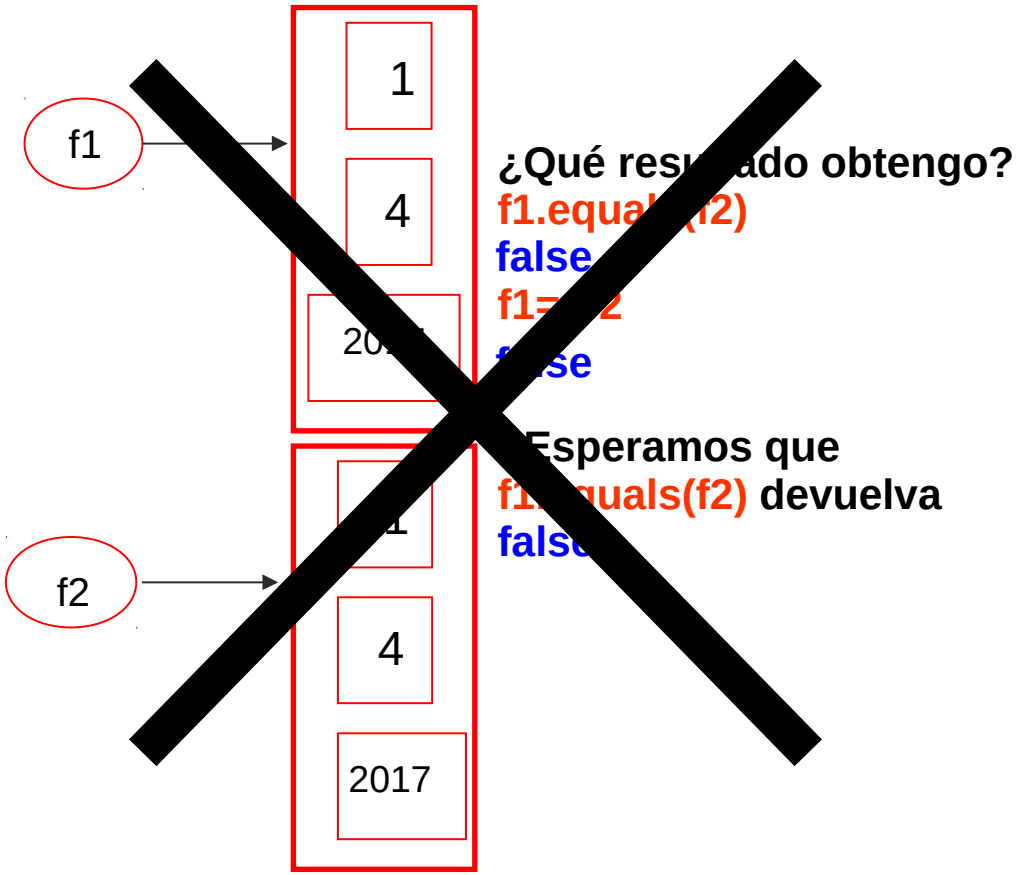
- Compara referencias, es decir devuelve **true** si la referencia del objeto recibido como argumento es igual a la del objeto receptor del mensaje (o sea, las referencias apuntan a la misma instancia de objeto)
- NO compara el contenido de los objetos
- Es equivalente a usar el operador **==**

```
Object objeto1 = new Object();
Object objeto2 = new Object();

System.out.print(objeto1.equals(objeto2));

System.out.print(objeto1.equals(objeto1));

System.out.print(objeto1 == objeto2);
```

Cuál es la salida en cada caso?

equals(): Otro ejemplo

```
public class Fecha
extends Object {
   private int dia=1;
   private int mes=4;
   private int año=2017;
   // métodos de instancia
}

//creo dos fechas
Fecha f1 = new Fecha();
Fecha f2 = new Fecha();
```



Solución: Sobreescribir equals()

Recuerden: El **casting** permite conversión entre tipos. **Fecha** por defecto extiende de **Object**, el cual no define los atributos **dia**, **mes**, **año** para realizar la comparación!

```
publi
 priv
 priv
 priv

 public boolea          (Object o) {
  boolean resu          = false;
  // Esto castea un objeto al tipo Fecha
  Fecha f = (Fecha)o;
  if ((f.dia == this.dia) && (f.mes == this.mes) && (f.año == this.año))
   resultado = true;
  return resultado;
 }

 public static void main(String args[]){
  Fecha f1 = new Fecha();
  Fecha f2 = new Fecha();
  System.out.println(f1==f2);
  System.out.println(f1.equals(f2));
 }
}
```

Cuál es la salida?

false
true

Y si invoco equals() con parámetro “null” u otro objeto?

```
public class Fecha extends Object {
 private int dia = 1;
 private int mes = 4;
 private int año = 2017;

 public boolean equals(Object o) {
  boolean resultado = false;
  // Esto “castea” un objeto al tipo Fecha
  Fecha f = (Fecha)o;
  if ((f.dia == this.dia) && (f.mes == this.mes) && (f.año == this.año))
   resultado = true;
  return resultado;
 }
```

```
 public static void main(String args[]){
  Fecha f1 = new Fecha();
  Objeto objeto1 = new Objeto();

  System.out.println(f1.equals(null));
  System.out.println(f1.equals(objeto1));
 }
}
```

He aquí la solución...

```
public class Fecha extends Object {
 private int dia = 1;
 private int mes = 4;
 private int año = 2017;

 public boolean equals(Object o) {
  // instanceof verifica que "o" sea instancia de la clase Fecha
  if ((o == null) || !(o instanceof Fecha))
    return false;
  boolean resultado = false;
  // Esto castea un objeto al tipo Fecha
  Fecha f = (Fecha)o;
  if ((f.dia == this.dia) && (f.mes == this.mes) && (f.año == this.año))
   resultado = true;
  return resultado;
 }

 public static void main(String args[]){
  Fecha f1 = new Fecha();
  Objeto objeto1 = new Objeto();

  System.out.println(f1.equals(null));
  System.out.println(f1.equals(objeto1));
 }
}
```

La salida es:
false
false

Cómo funciona toString()

- Otro método que todas las clases heredan de la clase Object es el método **toString()**
- El método produce una representación textual y legible del contenido de un objeto:

  public String toString(){}

- La versión original produce un string formado por el nombre de la clase, seguido del símbolo "@" y de un valor hexadecimal calculado usando el objeto
- Ej. invocado sobre una instancia de Fecha: **Fecha@25edc9**
- De la misma manera que el método **equals()**, es importante que las clases lo **sobreescriban** con el comportamiento deseado

Invocando a toString()

```
public class Fecha extends Object {
 private int dia = 1;
 private int mes = 4;
 private int año = 2017;

 // métodos de instancia
}
```

Si tenemos un fragmento de código como el siguiente:

```
Fecha f1 = new Fecha();
System.out.println(f1.toString());
```

¿Cuál es la salida?

Fecha@19821f

No es información muy útil!!!

Invocando a toString(): Notas adicionales

- No es necesario invocar al método **toString()** ya que los métodos **println()** y **print()** convierten a strings sus argumentos invocando al método **toString()**:

System.out.println(f1.toString()); equivale a **System.out.println(f1);**

System.out.print(f1.toString()); equivale a **System.out.print(f1);**

- En caso que los argumentos sean datos primitivos se convierten a string usando los métodos **String.valueOf(int)** / **String.valueOf(long)** / **String.valueOf(boolean)**, etc.

Veamos un ejemplo:

```
int valor = 2;
String valorComoString = String.valueOf(valor);
System.out.println(valorComoString);
```

Ejercicio de toString()

- Teniendo en cuenta el código de la clase Fecha que figura debajo, implemente el método **toString()** para mostrar por pantalla la fecha en formato “dia/mes/año”

```
public class Fecha extends Object {
 private int dia = 1;
 private int mes = 4;
 private int año = 2017;

 // métodos de instancia
}
```

Ejercicio de toString(): Solución

```
public class Fecha extends Object {
 private int dia = 1;
 private int mes = 1;
 private int año = 2017;

 public String toString() {
  String separador = "/";
  String diaComoString = String.valueOf(dia);
  String mesComoString = String.valueOf(mes);
  String añoComoString = String.valueOf(año);
  String resultado = diaComoString + separador +
             mesComoString + separador +
             añoComoString;
  return resultado;
 }

 // otros métodos de instancia

}
```

```
// Para invocar:

Fecha f1 = new Fecha();
System.out.println(f1);
```

Ejercicio de toString(): Mejora

```
public class Fecha extends Object {
 private int dia = 1;
 private int mes = 1;
 private int año = 2017;
 public static final String separador = "/";

 public String toString() {
  String diaComoString = String.valueOf(dia);
  String mesComoString = String.valueOf(mes);
  String añoComoString = String.valueOf(año);
  String resultado = diaComoString + separador +
             mesComoString + separador +
             añoComoString;
  return resultado;
 }

 // otros métodos de instancia

}
```

Otro ejemplo de toString(): Clase Persona

```
public class Persona extends Object{
 private String nombre  = "Federico";
 private int edad = 25;
 public String toString(){
  return "Nombre: " + this.getNombre() +
         " y edad: " + this.getEdad();
 }
 public String getNombre(){
  return nombre;
 }
 public void setNombre(String nombre){
  this.nombre=nombre;
 }
 public int getEdad(){
  return edad;
 }
 public void setEdad(int edad){
  this.edad=edad;
 }
}
```

¿Cuál es la salida?

Federico
25
Nombre: Federico y edad: 25

```
public class TestPersona{
 public static void main (String args[]){
  Persona p = new Persona();
  System.out.println(p.getNombre());
  System.out.println(p.getEdad());
  System.out.println(p);
 }
}
```

Sobrecarga de métodos

- En Java, se pueden definir en una misma clase múltiples métodos con el mismo nombre pero con argumentos diferentes. A esto se le llama **sobrecarga** de métodos
- Los métodos sobrecargados tienen el mismo nombre y el mismo tipo de retorno, pero varían en la cantidad, tipo y orden de sus argumentos. Tienen **firmas** diferentes. Ejemplos:

public void metodo(int arg1, long arg2) → public void metodo(int arg1)
public void metodo(int arg1, long arg2) → public void metodo(int arg1, int arg2)
public void metodo(int arg1, long arg2) → public void metodo(long arg1, int arg2)

- Un método sobrecargado tiene implementaciones diferentes dependiendo de sus argumentos. Es decir, la sobrecarga permite definir múltiples versiones de un mismo método
- La **sobrecarga** ocurre dentro de una misma clase y la **sobreescritura** entre una clase y sus subclases

Ejemplo: Calculadora

- Supongamos una clase Calculadora con un método para sumar dos números, como sigue a continuación:

Sumamos 2 int

Sumamos 2 float

Sumamos 1 int y 1 float

```
public class Calculadora {
    public int sumar (int num1, int num2){
        System.out.println("método 1");
        return num1 + num2;
    }
    public float sumar (float num1, float num2){
        System.out.println("Método 2");
        return num1 + num2;
    }
    public float sumar (int num1, float num2) {
        System.out.println("método 3");
        return num1 + num2;
    }
}
```

Ejemplo: Calculadora

```
public class Calculadora {
   public int sumar (int num1, int num2){
       System.out.println("método 1");
       return num1 + num2;
    }
    public float sumar (float num1,
                                float num2){
       System.out.println("Método 2");
       return num1 + num2;
    }
    public float sumar (int num1, float num2) {
       System.out.println("método 3");
       return num1 + num2;
    }
}
```

```
public class CalculadoraTest {
  public static void main(String[] args)
        Calculadora c = new Calculadora();
         int total1 = c.sumar(2,3);
         System.out.println(total1);
         float total2 = c.sumar(2,12.85F);
         System.out.println(total2);
         float total3 = c.sumar(15.99F,12.85F);
         System.out.println(total3);
  }
}
```

¿Cuál es la salida en cada caso?

método 1
método 3
método 2

Ejemplo: Calculadora

El método println() de la clase PrintStream está por definición sobrecargado...

En este ejemplo,

como parámetro recibe un int

y también un float

```
public class CalculadoraTest {
  public static void main(String[] args)
      Calculadora c = new Calculadora();
       int total1 = c.sumar(2,3);
       System.out.println(total1);
       float total2 = c.sumar(2,12.85F);
       System.out.println(total2);
       float total3 = c.sumar(15.99F,12.85F);
       System.out.println(total3);
   }
}
```

- Alternativamente, definir métodos **printlnFloat**, **printlnInt**, **printlnLong**, etc. nos obliga a llamar al método correspondiente dependiendo del argumento que le pasemos en cada invocación!

Sobrecarga en la API de Java

Notar que el método **println()** de la clase **PrintWriter** es **sobrecargado** y permite imprimir en pantalla objetos, strings, datos primitivos y arreglos

El método **println()** internamente llama al método **toString()** del objeto receptor.

En el caso de tipos primitivos, los convierte a string usando el método String.valueOf()

Es un objeto **PrintWriter**, permite escribir en pantalla

Variaciones del Método `System.out.println`.

Método	Uso
`void println()`	Termina la línea corriente escribiendo el caracter
`void println(boolean x)`	Despliega un valor `boolean` y finaliza la línea.
`void pirntln(char x)`	Despliega un caracter y finaliza la línea.
`void println(char[] x)`	Despliega un array de caracteres y finaliza la
`void println(double x)`	Despliega un valor `double` y finaliza la línea.
`void println(float x)`	Despliega un valor `float` y finaliza la línea.
`void println(int x)`	Despliega un valor `int` y finaliza la línea.
`void println(long x)`	Despliega un valor `long` y finaliza la línea.
`void println(Object x)`	Despliega una objeto `Object` y finaliza la línea.
`void println(String x)`	Despliega un `string` y finaliza la línea

Sobrecarga en la API de Java: Invocación

¿Qué método se invoca en cada caso?

```
Fecha f1 = new Fecha();
String f1s = f1.toString();
System.out.println(f1s);
System.out.println(f1);
System.out.println(
    f1.toString());
```

Variaciones del Método `System.out.println`.

Método	Uso
`void println()`	Termina la línea corriente escribiendo el caracter
`void println(boolean x)`	Despliega un valor `boolean` y finaliza la línea.
`void pirntln(char x)`	Despliega un caracter y finaliza la línea.
`void println(char[] x)`	Despliega un array de caracteres y finaliza la
`void println(double x)`	Despliega un valor `double` y finaliza la línea.
`void println(float x)`	Despliega un valor `float` y finaliza la línea.
`void println(int x)`	Despliega un valor `int` y finaliza la línea.
`void println(long x)`	Despliega un valor `long` y finaliza la línea.
`void println(Object x)`	Despliega una objeto `Object` y finaliza la línea.
`void println(String x)`	Despliega un `string` y finaliza la línea

Sobreescritura versus sobrecarga

```
public class Fecha extends Object {
private int dia = 1;
private int mes = 4;
private int año = 2017;
public boolean equals(Fecha f){
  boolean result=false;
  if (f != null)
   if ((f.dia==this.dia)&&(f.mes==this.mes)&& (f.año==this.año))
    result=true;
  return result;
}
public static void main(String args[]){
  Fecha f1, f2;
  f1 = new Fecha();
  f2 = new Fecha();
  System.out.println(f1.equals(f2));
}
```

¿Qué mecanismo estamos usando?

¿Sobreescritura o Sobrecarga?

Sobrecarga!

¿Porqué?

La clase Fecha posee 2 versiones del método **equals()**, una con argumento Fecha y otra que heredamos de Object con argumento Object

Para sobreescribir el método equals() debemos escribir un método con la misma firma que el definido en la clase Object

Ejercicio de sobreescritura versus sobrecarga

```
public class A {
 void do()
 {...}
 void do(int i)
 {...}
 void do(float x)
 {...}
 void do(float x, float y)
 {...}
}
```

```
public class B extends A {
 void do (int i)
 {…}
 void do (float x, float y)
 {…}
 void do (float x, int k)
 {…}
}
```

```
public class C extends B {

void do (int i)
{...}

}
```

- Los métodos sobrecargados de la clase A son:
  void do(), void do(int i), void do(float x), void do(float x, float y)
- ¿Qué métodos de la clase B sobrescriben a los de la clase A?
  void do(int i), void do(float x, float y)
- Los métodos sobrecargados de la clase B son:
  void do(float x, float y), void do(float x, int k)
- ¿Qué métodos de la clase C sobrescriben a los definidos B?
  void do (int i)

Constructores

- Como se ha visto, en Java creamos instancias de una clase mediante el operador **new** invocando un código especial llamado **constructor**:

```
Fecha f1 = new Fecha();
```

- Los constructores permiten asignar al objeto valores específicos a los atributos al momento de la creación:

```
public class Fecha {
  int dia, mes, año; // creados sin valor inicial
  public Fecha() { // constructor
    dia = 1;
    mes = 4;
    año = 2017;
  }
}
```

- Entonces, el nombre del constructor es idéntico al nombre de la clase, el constructor NO tiene valor de retorno, y los constructores NO son métodos (no se pueden invocar sin usar **new**)

Más sobre constructores

- El constructor nulo puede ser **sobrecargado**, pudiendo definir constructores con diferente cantidad y tipo de argumentos dentro de la misma clase:

```
public class Fecha {
   int dia, mes, año; // creados sin valor inicial
   public Fecha(){ // constructor
      dia = 1;
      mes = 4;
      año = 2017;
   }
   public Fecha(int diaFecha, int mesFecha, int añoFecha) {
      dia = diaFecha;
      mes = mesFecha;
      año = añoFecha;
   }
}
```

```
public class FechaTest {
  public static void main(String[] args){
    Fecha f1 = new Fecha();
    Fecha f2 = new Fecha(31, 12, 2017);
    // ¿Qué valor tiene cada fecha?
  }
}
```

Sobrecarga de constructores: Otro ejemplo

```
public class Vehiculo {
private String nroPatente="";
private String propietario="SinDueño";
  public Vehiculo(){
  }
  public Vehiculo(String nroPat){
     nroPatente = nroPat;
  }
  public Vehiculo(String nroPat,String prop){
     nroPatente = nroPat;
     propietario = prop;
  }
}
```

La sobrecarga de constructores permite declarar múltiples versiones del constructor de la clase y de esta manera podemos crear e inicializar objetos de diferentes maneras. Java determina el constructor a invocar a partir de la lista de argumentos...

```
public class TestVehiculo {
  public static void main(String[] args){
     Vehiculo a1 = new Vehiculo();
     Vehiculo a2 = new Vehiculo("AA123AA");
     Vehiculo a3 = new Vehiculo("AA124AA","Juan Perez");
  }
}
```

Para tener en cuenta...

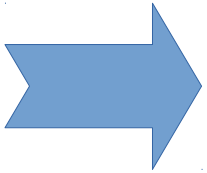
- Sólo es posible utilizar los constructures **definidos** en una clase para crear instancias de la misma:

```
public class Vehiculo {
private String nroPatente="";
private String propietario="SinDueño";
  public Vehiculo(String nroPat){
    nroPatente = nroPat;
  }
}
```

```
public class TestVehiculo {
  public static void main(String[] args){
    Vehiculo a1 = new Vehiculo("AA123AA");
    Vehiculo a2 = new Vehiculo(); // Error!
  }
}
```

- Si no se agrega ningún constructor a una clase, Java agrega un constructor vacío (también llamado **nulo**):

```
public class Vehiculo {
  private String nroPatente="";
  private String propietario="SinDueño";
}
```



```
public class Vehiculo {
  private String nroPatente="";
  private String propietario="SinDueño";
  public Vehiculo(){
  }
}
```

Práctica de toString(), equals() y sobrecarga

Implementar en PC los ejercicios del práctico adjunto...