CLASE OBJECT

Dentro del árbol de herencia de Java, la clase Object está situada en la parte más alta.

Todas las clases Java heredan directa o indirectamente de la clase Object.

Aunque el concepto de herencia lo estudiaremos un poco más adelante, es necesario introducirlo aquí para entender porqué las clases que creamos pueden utilizar métodos como equals o toString y la forma en la que estos métodos funcionan.

La clase Object define el comportamiento básico que todos los objetos deben tener. Para ello proporciona una serie de métodos que son heredados y por lo tanto están disponibles en todas las clases Java.

Algunos de estos métodos son:

1. Método toString()

```
public String toString(){
        ......
}
```

El método toString() devuelve una cadena que representa el objeto. El contenido de esta cadena depende del objeto sobre el que se esté aplicando.

El método toString() **se invoca de forma automática cuando se muestra por pantalla el contenido de un objeto mediante la instrucción System.out.print o System.out.println**.

Esto quiere decir que las siguientes instrucciones son equivalentes:

```
System.out.println(unObjeto);
System.out.println(unObjeto.toString());
```

Ejemplo de uso de toString: Disponemos de una clase llamada Coordenadas

```
package ejemplotostring1;
public class Coordenadas {
    private double x;
    private double y;

    public Coordenadas() {
    }
    public Coordenadas(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public double getX() {
        return x;
    }
    public void setX(double x) {
        this.x = x;
    }
    public double getY() {
        return y;
    }
    public void setY(double y) {
        this.y = y;
    }
}
```

Un programa que utilice la clase podría ser este: En el método main de la clase principal creamos un objeto de tipo Coordenadas utilizando el constructor con parámetros y después vamos a mostrar por pantalla el objeto creado usando el método toString().

```
package ejemplotoString1;
public class EjemploToString1 { //Clase principal
    public static void main(String[] args) {
        Coordenadas punto1 = new Coordenadas(6.8,15.22);
        System.out.println(punto1);
    }
}
```

En la instrucción

```
System.out.println(punto1);
```

Se hace una llamada al método toString() de la clase Coordenada. Aunque este método no está contenido en la clase, lo podemos utilizar porque lo hemos heredado de Object. Este método devuelve un String que será el que se muestre por pantalla.

En este caso, el método toString() devuelve una cadena similar a esta, que es lo que se muestra por pantalla:

```
ejemplotostring1.Coordenadas@4a5ab2
```

Vemos que el método toString() heredado por defecto de Object devuelve un String con el nombre del package seguido del nombre de la Clase y de un número.

Lo que obtenemos no nos es muy útil ya que queremos que muestre el contenido del objeto, pero este método **lo podemos redefinir o sobrescribir** para representar nuestros objetos de forma más adecuada.

Cuando se sobrescribe un método heredado hay que mantener el prototipo o encabezado del método tal cual se hereda. Si se escribe el método de forma distinta (por ejemplo, añadiendo un parámetro) estaremos sobrecargando el método, no sobrescribiéndolo.

En este caso lo vamos a modificar para que muestre el valor de las coordenadas de modo que cuando escribamos la instrucción:

```
System.out.println(punto1);
```

Se muestre por ejemplo:

```
Coordenadas: x = 6.8 y = 15.22
```

Para esto tenemos que escribir el método toString() dentro de la clase de la siguiente forma:

```
@Override
public String toString() {
       return "Coordenadas: " + "x = " + x + " y = " + y;
}
```

```
@Override
public String toString() {
      return "Coordenadas: " + "x = " + x + " y = " + y;
}
```

Dentro del método toString escribiremos todo el código necesario para formar el String deseado.
El encabezado del método debemos escribirlo tal cual.
No lo podemos modificar

La anotación:

```
@Override
```

Es necesario escribirla antes del método e indica que estamos modificando el método toString() heredado de Object.

Después de añadir el método a la clase, la instrucción:

```
System.out.println(punto1);
```

Mostrará por pantalla:

```
Coordenadas: x = 6.8 y = 15.22
```

2. Método equals()

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

Este método compara dos objetos para ver si son o no iguales.

El método comprueba si los objetos pertenecen a la misma clase y además contienen los mismos datos. Devuelve true si los objetos son iguales y false si no lo son.

Comparar dos objetos con equals es distinto a hacerlo con el operador ==.

Este operador == compara si dos referencias a objetos *apuntan* al mismo objeto.

Igual que ocurre con el método toString() en una clase podemos **sobrescribir el método equals()** para adaptarlo a nuestras necesidades.

El método equals que genera NetBeans de forma automática para la clase Coordenadas es:

```
@Override
public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Coordenadas other = (Coordenadas) obj;
        if (Double.doubleToLongBits(this.x) != Double.doubleToLongBits(other.x)) {
            return false;
        }
        if (Double.doubleToLongBits(this.y) != Double.doubleToLongBits(other.y)) {
            return false;
        }
        return true;
}
```

En este caso este método equals nos sirve para realizar la comparación y no hay que hacer ninguna modificación.

Vemos que lo primero que se comprueba es si la referencia del objeto que quiero comparar es la misma que la del objeto actual. En ese caso son dos variables que apuntan al mismo objeto y por lo tanto se trata del mismo objeto.

Después se comprueba que se reciba un objeto para compararlo con el que llama al método, o sea, si la referencia obj es igual a null significa que no apunta a ningún objeto y por lo tanto la comparación es falsa.

Lo siguiente es comprobar que ambos objetos son de la misma clase. **La clase a la que pertenece un objeto la podemos obtener mediante el método getClass()**. Si no lo son la comparación es falsa.

Por último nos queda comprobar los atributos de cada objeto para ver si contienen los mismos valores. Para ello el objeto que se recibe como parámetro se *convierte* a objeto de la clase Coordenadas y se comparan los atributos x e y de ambos.

En este caso hemos dejado el método tal cual nos lo ha generado NetBeans. En algunos casos será necesario modificarlo para adaptarlo a nuestras necesidades a la hora de determinar cuando dos objetos son iguales. En esos casos modificaremos esta última parte del método en la que se comprueban los valores de los atributos siendo recomendable dejar el resto de código sin modificar.

```
@Override
public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Coordenadas other = (Coordenadas) obj;
        if (Double.doubleToLongBits(this.x) != Double.doubleToLongBits(other.x)) {
            return false;
        }
        if (Double.doubleToLongBits(this.y) != Double.doubleToLongBits(other.y)) {
            return false;
        }
        return true;
}
```

Código que podemos modificar.
El resto de código aunque se puede modificar
es recommendable dejarlo como está.

Un ejemplo de uso del método equals podría ser este:

```
public class ToString1 {
    public static void main(String[] args) {
        Coordenadas punto1 = new Coordenadas(6.8, 15.22);
        Coordenadas punto2 = new Coordenadas(6.8, 10.12);
        if(punto1.equals(punto2)){
            System.out.println("Coordenadas iguales");
        }else{
            System.out.println("Coordenadas distintas");
        }
    }
}
```