

Curso de Java a distancia

Clase 33: Anotaciones Java predefinidas: @Override, @Deprecated y @SuppressWarnings

Las anotaciones son una forma de agregar metadatos al código fuente de Java.

Las anotaciones se utilizan para:

Brindar información al IDE (por ejemplo Eclipse) para detectar por ejemplo errores previos a su compilación.

Acceder a información de los metadatos en tiempo de ejecución del programa.

Las anotaciones comienzan con el caracter @

Podemos crear anotaciones propias, pero ahora veremos algunas anotaciones predefinidas básicas:

- @Override : Aplicamos esta anotación a un método para indicar que el mismo sobrescribe el método heredado de una clase o interfaz:

```
@Override
public boolean equals(Object obj) {
    Persona persona = (Persona) obj;
    if (dni.equals(persona.dni))
        return true;
    else
        return false;
}
```

En el caso que nos equivoquemos en la firma del método a sobrescribir el entorno Eclipse puede identificar que hemos cometido un error:

```

14
15 @Override
16 public boolean EQUALS(Object obj) {
17     Persona persona = (Persona) obj;
18     if (dni.equals(persona.dni))
19         return true;
20     else
21         return false;
22 }
23

```

Sin la anotación @Override el entorno de Eclipse intuye que queremos crear un método llamado

```

14
15 public boolean EQUALS(Object obj) {
16     Persona persona = (Persona) obj;
17     if (dni.equals(persona.dni))
18         return true;
19     else
20         return false;
21 }

```

'EQUALS':

- @Deprecated : Una anotación de este tipo indica que un método, atributo, clase etc. está obsoleto y no se recomienda su uso.

Clase: Persona

```

public class Persona {
    private String nombre;
    private int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    @Deprecated
    public String retornarNombre() {
        return this.nombre;
    }

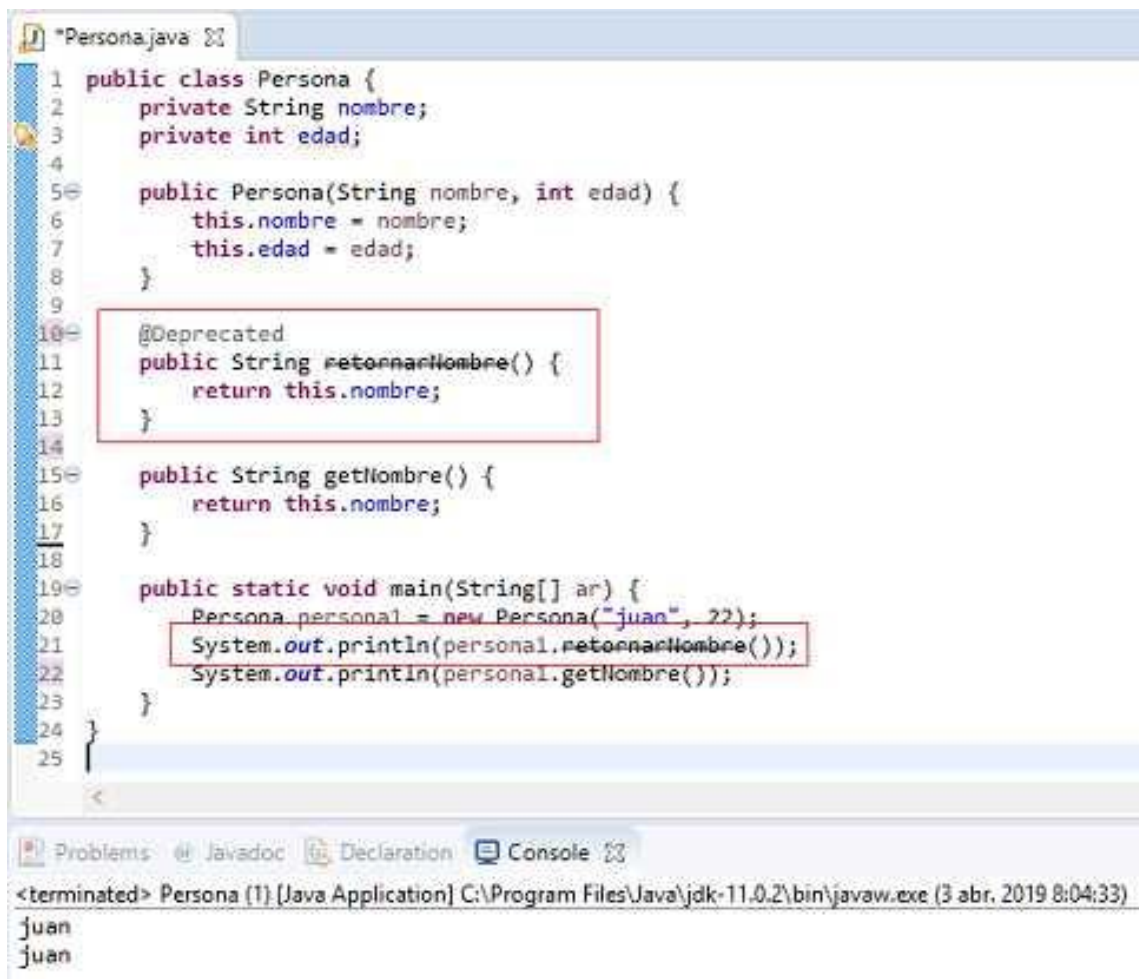
    public String getNombre() {
        return this.nombre;
    }

    public static void main(String[] ar) {
        Persona persona1 = new Persona("juan", 22);
        System.out.println(persona1.retornarNombre());
        System.out.println(persona1.getNombre());
    }
}

```

Por ejemplo si queremos recuperar el nombre de la persona mediante el método 'getNombre' y en futuras versiones eliminar el método 'retornarNombre' podemos agregar la anotación @Deprecated al método obsoleto para que los programadores utilicen 'getNombre' en vez de 'retornarNombre'.

Los entornos de programación nos pueden mostrar información a partir del metadato que agrega la anotación:



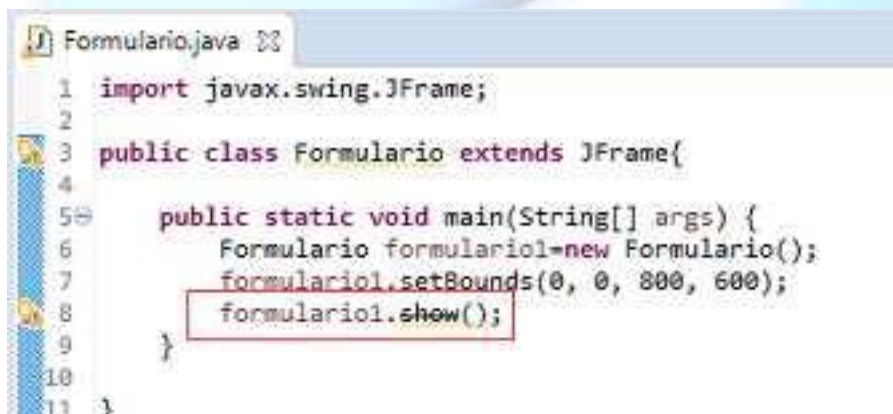
```
1 public class Persona {
2     private String nombre;
3     private int edad;
4
5     public Persona(String nombre, int edad) {
6         this.nombre = nombre;
7         this.edad = edad;
8     }
9
10    @Deprecated
11    public String retornarNombre() {
12        return this.nombre;
13    }
14
15    public String getNombre() {
16        return this.nombre;
17    }
18
19    public static void main(String[] ar) {
20        Persona personal = new Persona("juan", 22);
21        System.out.println(personal.retornarNombre());
22        System.out.println(personal.getNombre());
23    }
24 }
25
```

Problems | Javadoc | Declaration | Console

<terminated> Persona (1) [Java Application] C:\Program Files\Java\jdk-11.0.2\bin\javaw.exe (3 abr. 2019 8:04:33)

juan
juan

El API de Java ha tenido una serie de cambios a lo largo del tiempo, si analizamos las miles de clases veremos que muchos métodos se desaconsejan su uso:



```
1 import javax.swing.JFrame;
2
3 public class Formulario extends JFrame{
4
5     public static void main(String[] args) {
6         Formulario formulario1=new Formulario();
7         formulario1.setBounds(0, 0, 800, 600);
8         formulario1.show();
9     }
10 }
11
```

Como sabemos para visualizar un JFrame debemos llamar al método 'setVisible' en lugar de 'show':

Clase: Formulario

```
import javax.swing.JFrame;
```

```
public class Formulario extends JFrame{
```

```

public static void main(String[] args) {
    Formulario formulario1=new Formulario();
    formulario1.setBounds(0, 0, 800, 600);
    formulario1.setVisible(true);
}

```

```

}

```

- @SuppressWarnings : Evita que el entorno de desarrollo y compilador muestren advertencias sobre nuestro código:

```

import javax.swing.JFrame;

```

```

@SuppressWarnings("serial")
public class Formulario extends JFrame{

```

```

    @SuppressWarnings("deprecation")
    public static void main(String[] args) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0, 0, 800, 600);
        formulario1.show();
    }

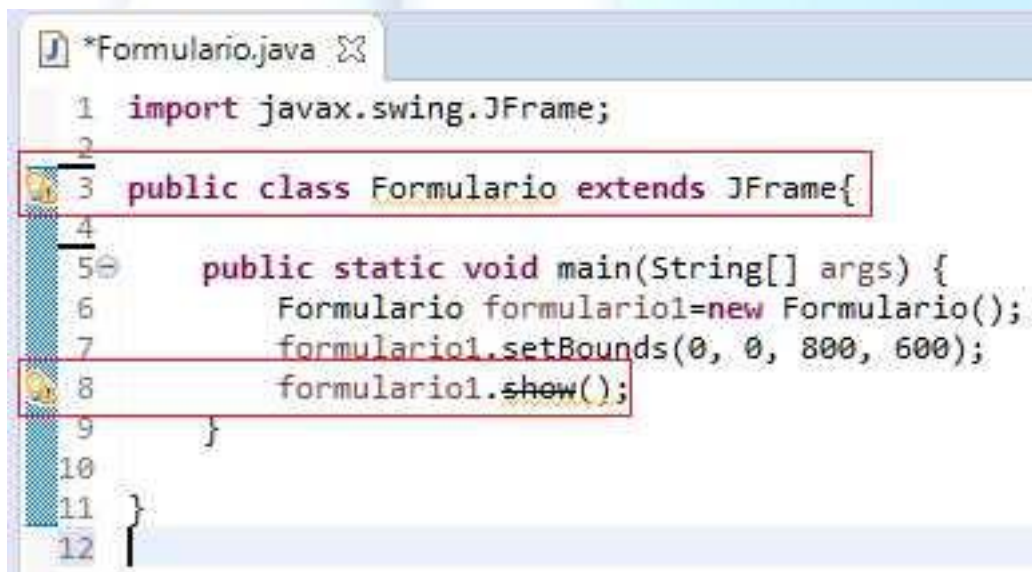
```

```

}

```

Previo a insertar la anotación:



Luego de agregar la anotación @SuppressWarnings desaparecen los Warnings:


```
Formulario.java
1 import javax.swing.JFrame;
2
3 @SuppressWarnings("serial")
4 public class Formulario extends JFrame{
5
6     @SuppressWarnings("deprecation")
7     public static void main(String[] args) {
8         Formulario formulario1=new Formulario();
9         formulario1.setBounds(0, 0, 800, 600);
10        formulario1.show();
11    }
12
13 }
```

Hay que tener en cuenta que primero tenemos que intentar evitar los warnings modificando el código fuente, pero en algunas situaciones por ejemplo proyectos antiguos puede ser una solución ocultar warnings.

Los Ides de Java (Eclipse, NetBeans, IntelliJ IDEA etc.) nos ayudan a generar en forma automática las anotaciones:

```
*Formulario.java
1 import javax.swing.JFrame;
2
3 public class Formulario extends JFrame{
4
5     public static void main(String[] args) {
6         Formulario formulario1=new Formulario();
7         formulario1.setBounds(0, 0, 800, 600);
8         formulario1.show();
9     }
10
11 }
12 }
```

The method show() from the type Window is deprecated
2 quick fixes available:
@ Add @SuppressWarnings 'deprecation' to 'main()'
Configure problem severity
Press 'F2' for focus

Generación automático de constructores, getters y setters con Eclipse

Cuando queremos tener acceso a un atributo de la clase lo más común es implementar un método que lo retorne y si queremos cambiar su valor implementamos otro método que lo modifique.

Como esta actividad es muy común para toda clase los Ides de Java (Eclipse, NetBeans, IntelliJ IDEA etc.) nos brindan estas funcionalidades en forma automática.

Veamos como crear los métodos de acceso a atributos y constructores en forma automática con Eclipse.

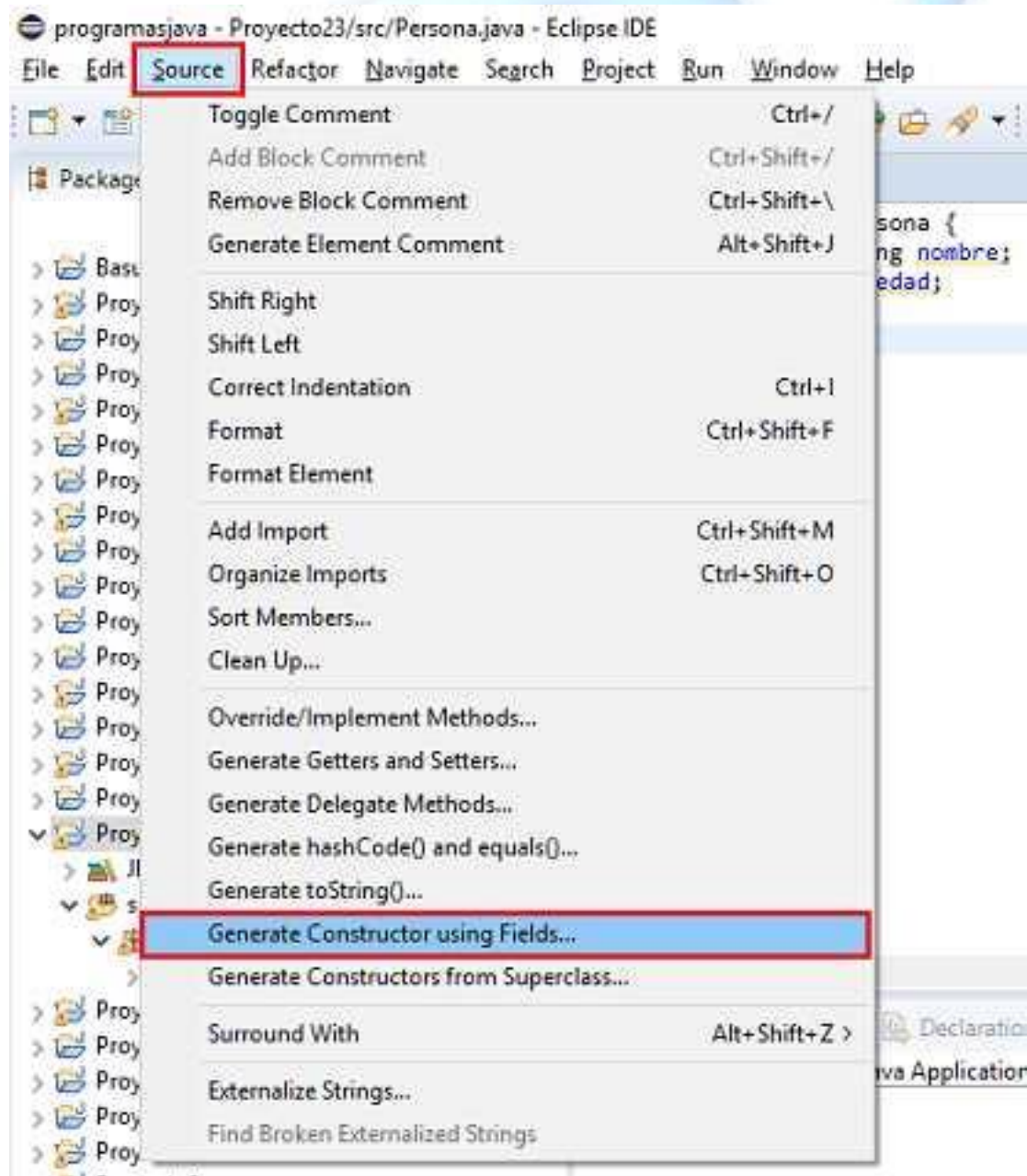
Problema:

Plantear un clase llamada 'Persona' definir los atributos nombre y edad. Crear el constructor y los métodos para modificar y recuperar los valores de los atributos mediante las herramientas que provee Eclipse.

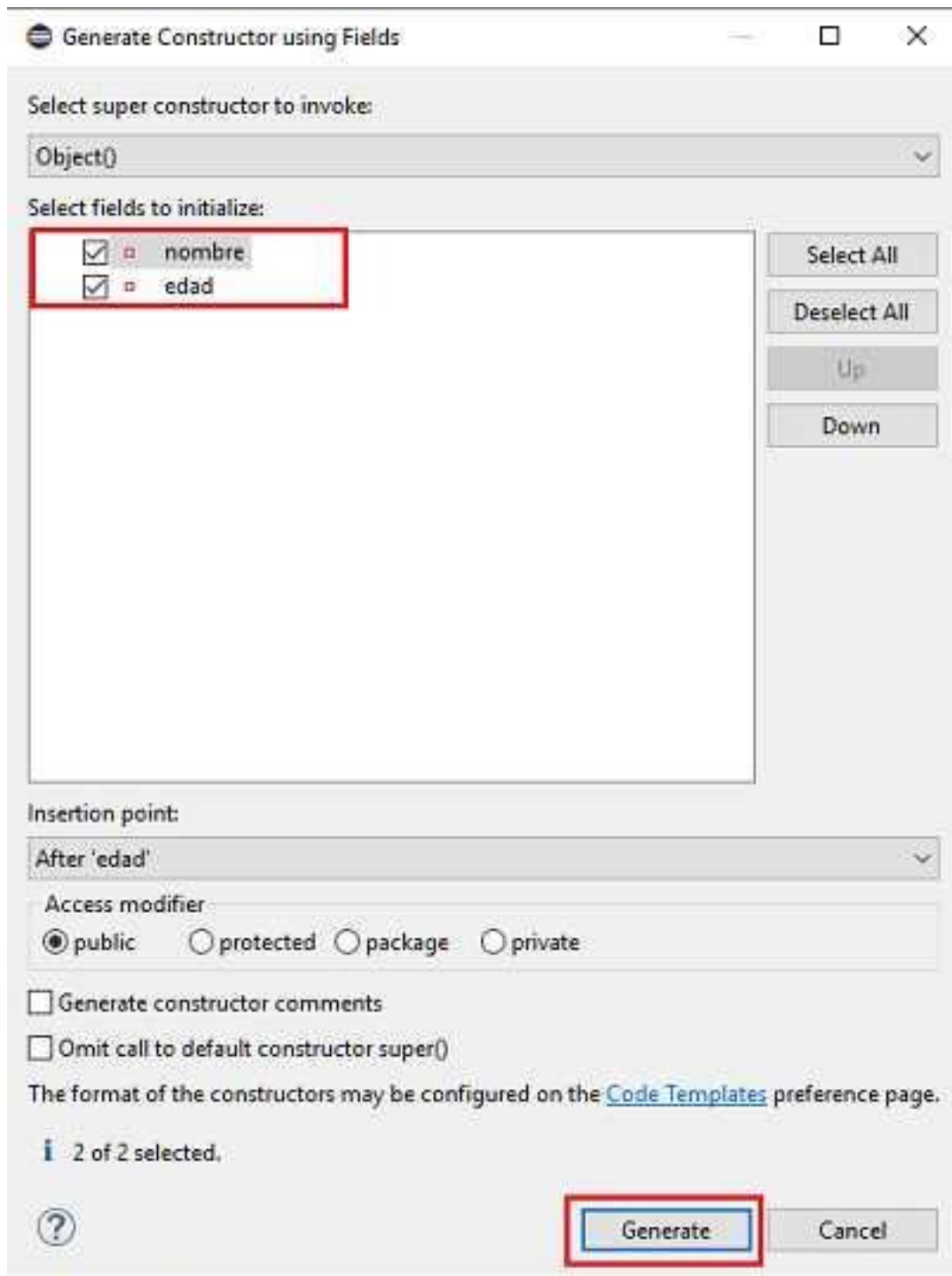
Clase: Persona

```
public class Persona {  
    private String nombre;  
    private int edad;  
}
```

Una vez que declaramos la clase con los atributos nombre y edad procedemos a elegir del menú de opciones de Eclipse: Source -> Generate constructor using Fields...:



Luego de elegir la opción se abre un diálogo para que seleccionemos que atributos queremos inicializar en el constructor:



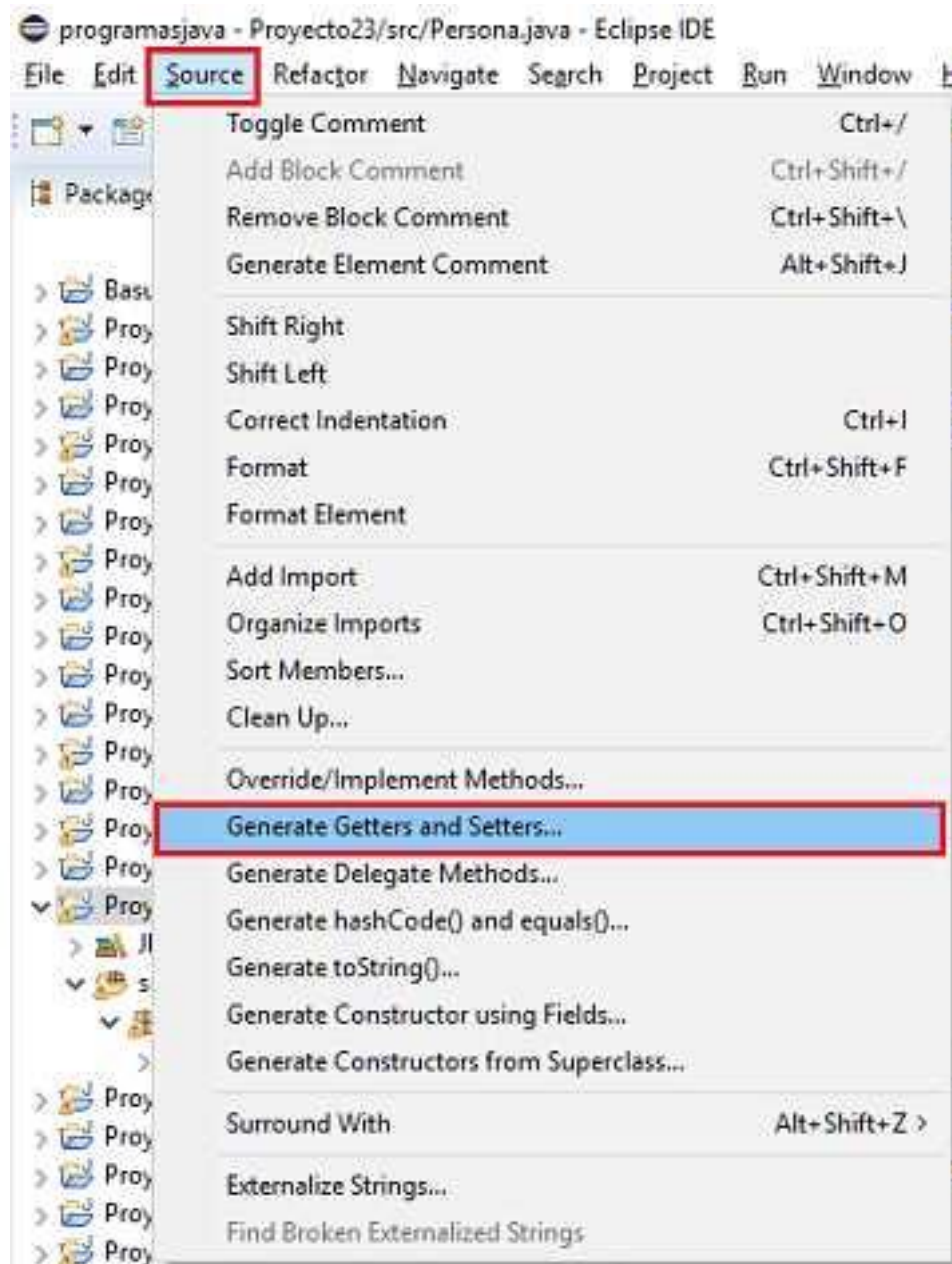
El código del constructor se agrega donde se encuentra el cursor actualmente.

Clase: Persona

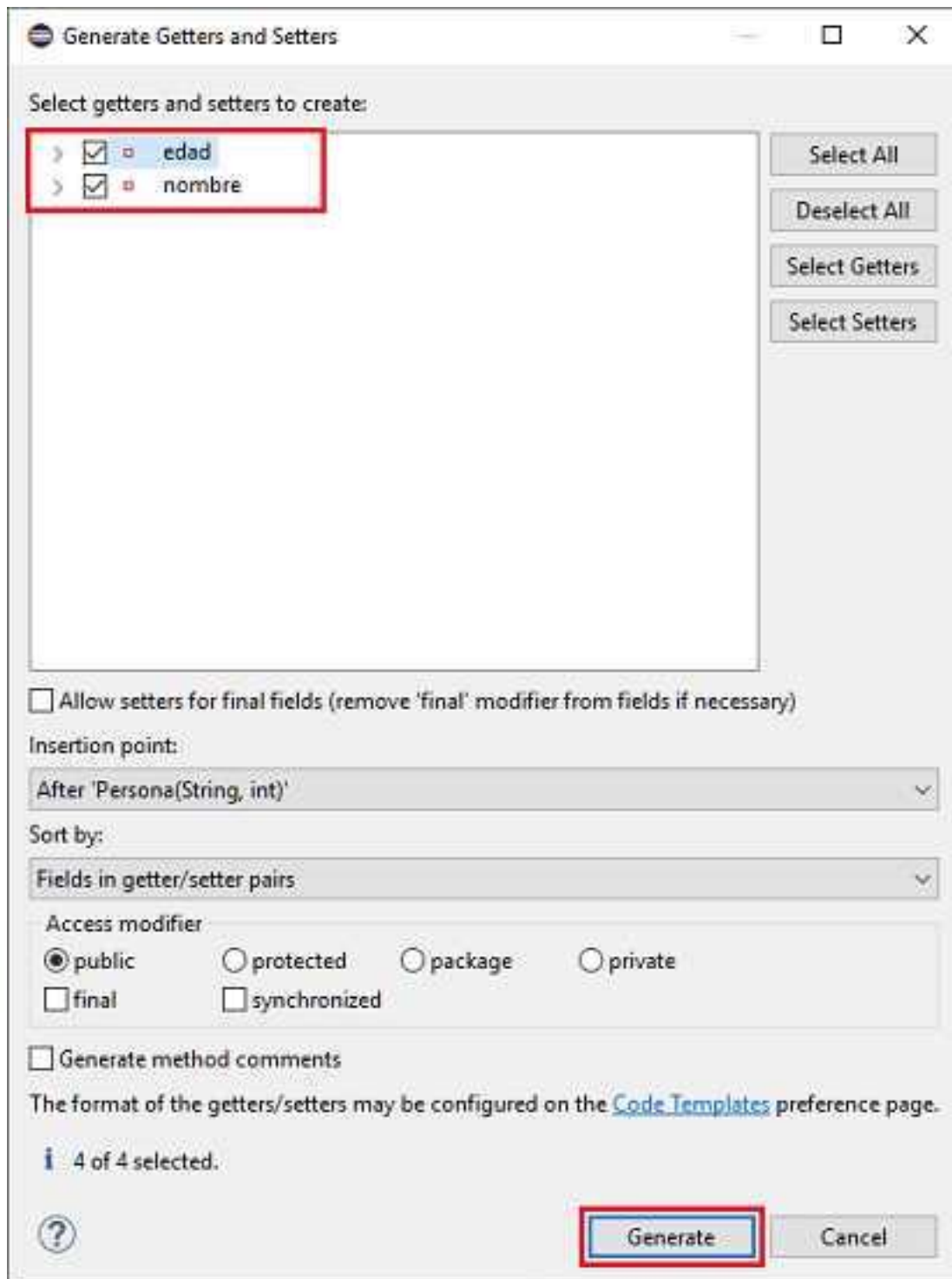
```
public class Persona {  
    private String nombre;  
    private int edad;  
    public Persona(String nombre, int edad) {  
        super();  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
}
```

La llamada al constructor de la clase padre mediante `super()` es opcional ya que de todos modos se hace en forma automática si no la disponemos.

Ahora procederemos a crear los getters y setters para la clase `Persona`, elegimos la opción: `Source -> Generate Getters and Setters...` (Recordar que los métodos se insertan donde se encuentre el cursor actualmente):



En el diálogo siguiente especificamos para que campos queremos generar el getter y el setter:



Ahora ya tenemos codificado el constructor de la clase 'Persona' y los setters y getters:

```
public class Persona {  
    private String nombre;  
    private int edad;  
    public Persona(String nombre, int edad) {  
        super();  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
    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;  
}  
}
```

Las herramientas que nos proveen Eclipse nos ahorra mucho tiempo con tareas repetitivas.

Ahora podemos ver porque es tan común llamar 'int getEdad()' a un método que retorne el campo edad en lugar de 'int retornarEdad()'.

Los diálogos que vimos para generar constructores, getters y setters tienen muchas opciones como pueden ser fijar si el método es public, private, protected, que se generen solo los getters o setters, seleccionar no todos los campos etc.

Problema propuesto

1. Confeccionar una clase que represente un empleado. Definir los campos nombre y sueldo. Crear dos constructores, uno que llegue el nombre y el sueldo y otro que solo llegue el nombre y se inicialice el sueldo en 1000. Crear getters y setters de los campos nombre y sueldo.

Solución

```
public class Empleado {
    private String nombre;
    private float sueldo;

    public Empleado(String nombre, float sueldo) {
        super();
        this.nombre = nombre;
        this.sueldo = sueldo;
    }

    public Empleado(String nombre) {
        super();
        this.nombre = nombre;
        this.sueldo = 1000;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public float getSueldo() {
        return sueldo;
    }

    public void setSueldo(float sueldo) {
        this.sueldo = sueldo;
    }
}
```

Generación automático de los métodos toString y equals con Eclipse

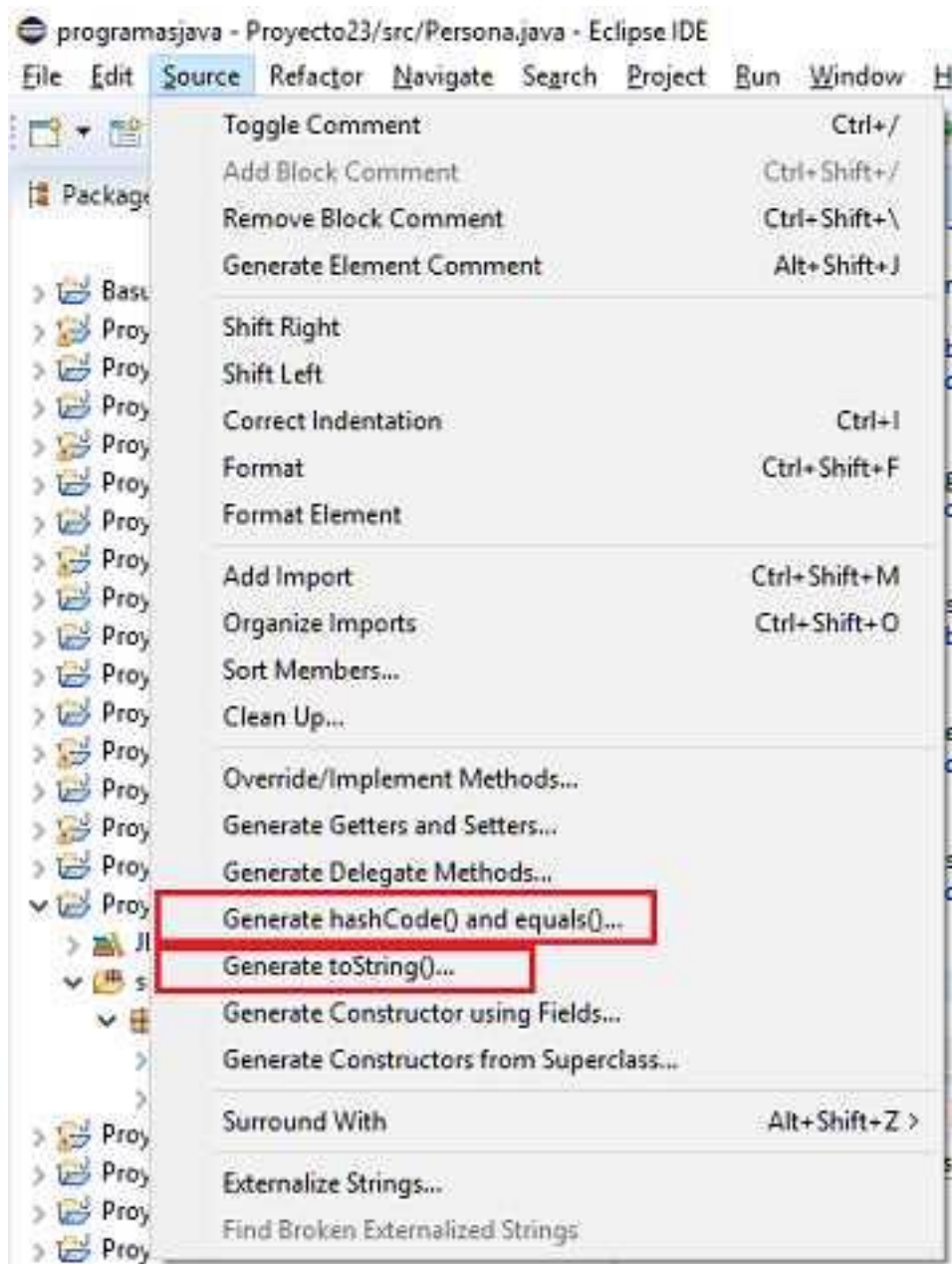
Vimos en conceptos anteriores que generalmente sobreescribimos los métodos toString y equals heredados de la clase Object.

Al ser una actividad tan común los Ide (Integrated Development Enviroment) de Java (Eclipse, NetBeans, IntelliJ IDEA etc.) nos brindan estas funcionalidades en forma automática.

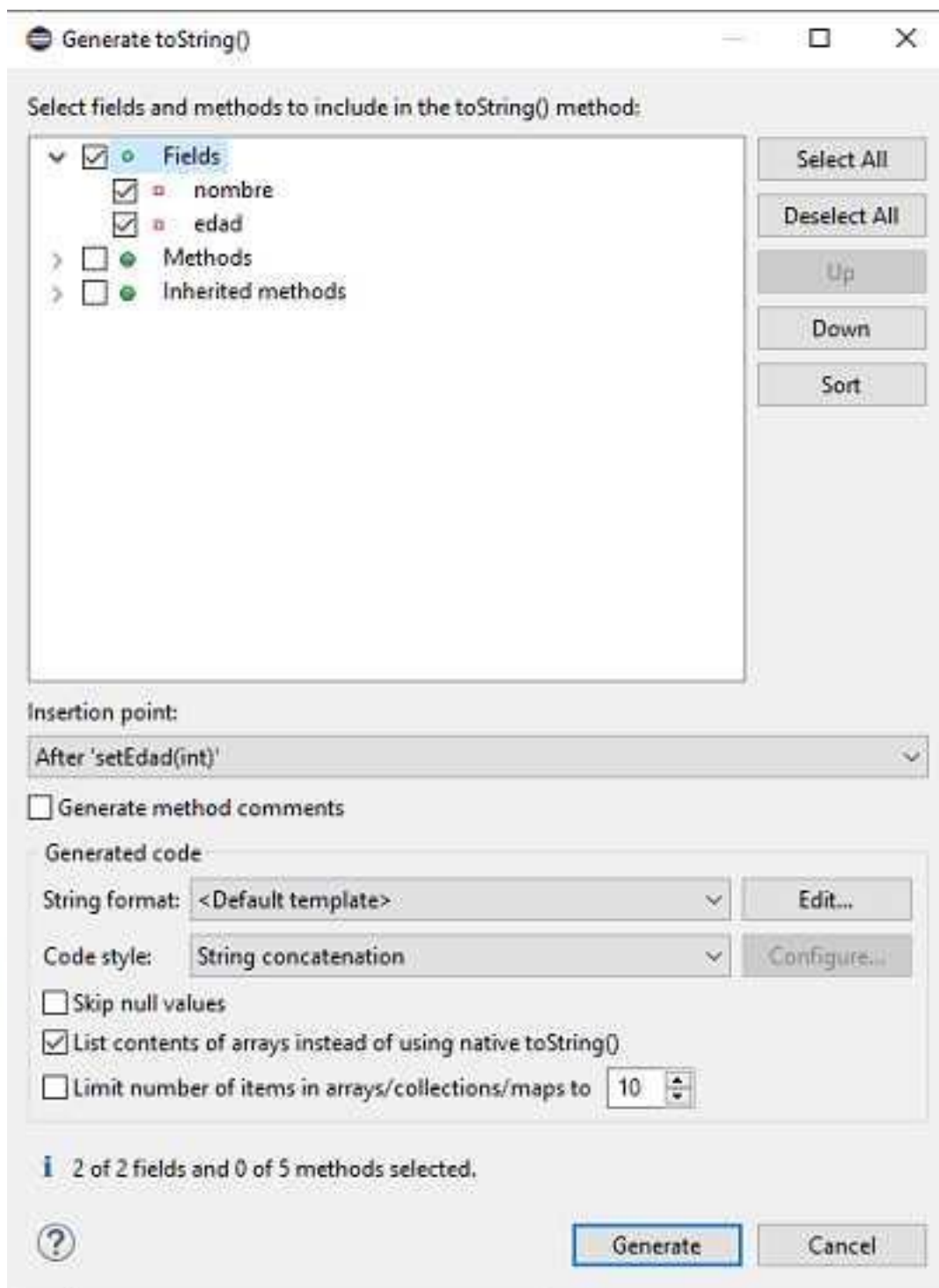
Problema:

Plantear un clase llamada 'Persona' definir los atributos nombre y edad. Crear el constructor y los métodos para modificar y recuperar los valores de los atributos, además implementar los métodos toString y equals mediante las herramientas que provee Eclipse.

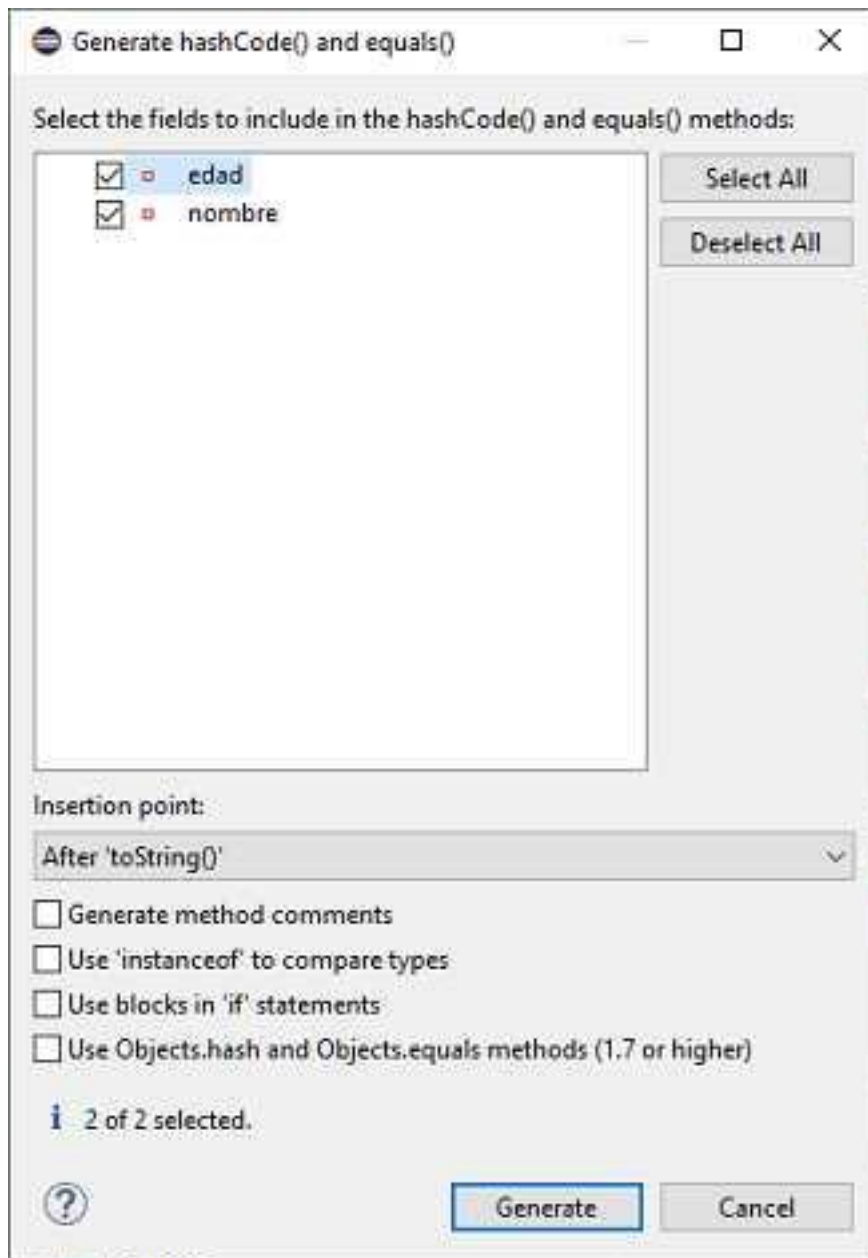
En la opción 'source' de Eclipse debemos seleccionar:



Cuando elegimos generar 'toString' podemos hacer algunas configuraciones:



Cuando generamos el método 'equals' también se genera el método 'hashCode':



En el diálogo podemos seleccionar que campos intervienen en el método 'equals'.

El método hashCode es importante para un manejo eficiente de colecciones de la clase Persona, por ejemplo ArrayList, LinkedList, HashSet, HashMap etc. Si dejamos por defecto el método hashCode heredado de la clase Object nuestro programa no será tan eficiente trabajando con estas colecciones.

Clase: Persona

```
public class Persona {  
    private String nombre;  
    private int edad;  
  
    public Persona(String nombre, int edad) {  
        super();  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
}
```

```

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;
}

@Override
public String toString() {
    return "Persona [nombre=" + nombre + ", edad=" + edad + "]";
}

@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + edad;
    result = prime * result + ((nombre == null) ? 0 : nombre.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Persona other = (Persona) obj;
    if (edad != other.edad)
        return false;
    if (nombre == null) {
        if (other.nombre != null)
            return false;
    } else if (!nombre.equals(other.nombre))
        return false;
    return true;
}
}

```

Problema propuesto

1. Desarrollar una clase que represente un Cuadrado que tenga como atributo su lado y los siguientes métodos: constructor, getLado, setLado, toString, equals y hashCode. Utilizar las herramientas de Eclipse para generarlos.

Solución

```
public class Cuadrado {
    private int lado;

    public Cuadrado(int lado) {
        super();
        this.lado = lado;
    }

    public int getLado() {
        return lado;
    }

    public void setLado(int lado) {
        this.lado = lado;
    }

    @Override
    public String toString() {
        return "Cuadrado [lado=" + lado + "]";
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + lado;
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Cuadrado other = (Cuadrado) obj;
        if (lado != other.lado)
            return false;
        return true;
    }
}
```

Muchas gracias hasta la próxima clase.

Alsina 16 [B1642FNB] San Isidro | Pcia. De Buenos Aires |Argentina |

TEL.: [011] 4742-1532 o [011] 4742-1665 |

www.institutosanisidro.com.ar info@institutosanisidro.com.ar