

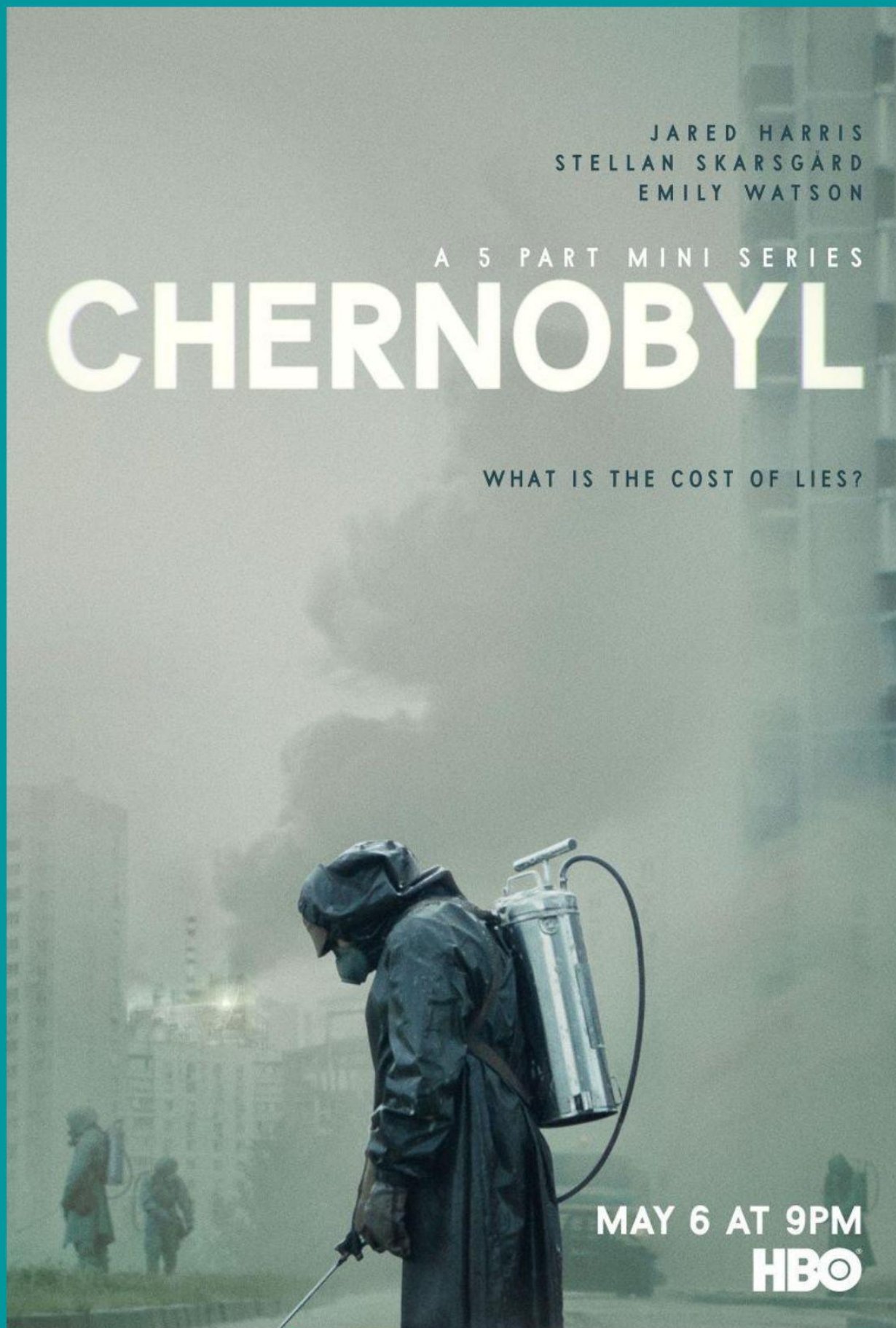
# ENTREGA 1

JARED HARRIS  
STELLAN SKARSGÅRD  
EMILY WATSON

A 5 PART MINI SERIES

# CHERNOBYL

WHAT IS THE COST OF LIES?



MAY 6 AT 9PM  
**HBO**

Á. Enrique Pineda Navas  
([quiquepineda@informatica.iesvalledeljerteplasencia.es](mailto:quiquepineda@informatica.iesvalledeljerteplasencia.es))

CURSO 2022/2023

*“Un reactor RBMK usa Uranio-235 como combustible. Cada átomo del U-235 es como una bala; que viaja casi a la velocidad de la luz atravesando cuanto encuentra a su paso: madera, metal, acero, carne. (...) El viento trasladará las partículas radiactivas por todo el continente y la lluvia las verterá encima de nosotros dejando 3 millones de millardos de trillones de balas en el aire que respiramos, el agua que bebemos, la comida que comemos.”*

Valery Legásov (Jared Harris), en la serie “Chernobyl”..

# índice

Chernobyl	4
Las clases abstractas	5
Estructura de las clases	7
Llave	7
Personaje	7
Operador	8
Minero	8
Bombero	8
Científico	9
Kgb	9
Oficial	9
Voluntario	10
Programa principal	11
Commits	13
Ejecución	14
Entrega	15
Calificación	16

# Chernobyl

Un año más, los alumnos de Programación del IES Valle del Jerte se enfrentarán al análisis y desarrollo de una práctica que englobe todos los contenidos del módulo de Programación. En esa ocasión, la práctica tratará el desastre nuclear más conocido de la historia: el accidente de la central nuclear de Chernobyl.

El alumno, a lo largo del curso 2022-2023, desarrollará una única práctica de manera individual. Esta práctica se desarrollará mediante un proceso secuencial, iniciando su andadura en el primer trimestre y ampliándola a medida que vaya avanzando el curso. Es por eso que, tal y como se especificó en el enunciado de presentación, el proyecto se dividirá en tres partes con sus respectivas entregas.

En este primer enunciado nos centraremos en el análisis y la implementación de la jerarquía de *Personajes* de la práctica y de la clase *Llave*, fundamental para que nuestros personajes puedan escapar de la central nuclear.

# Las clases abstractas

Una clase abstracta es una clase en la que alguno de sus métodos está declarado pero no está definido, es decir, se especifica su nombre, parámetros y tipo de devolución, pero no incluye código. A estos métodos sin código se les llama abstractos. (como a las clases). Además, cuando una clase es abstracta, no podemos crear objetos de estas clases.

Para indicar que una clase es abstracta, escribimos *abstract* entre el modificador de acceso y class. Ejemplo:

```
public abstract class Figura
```

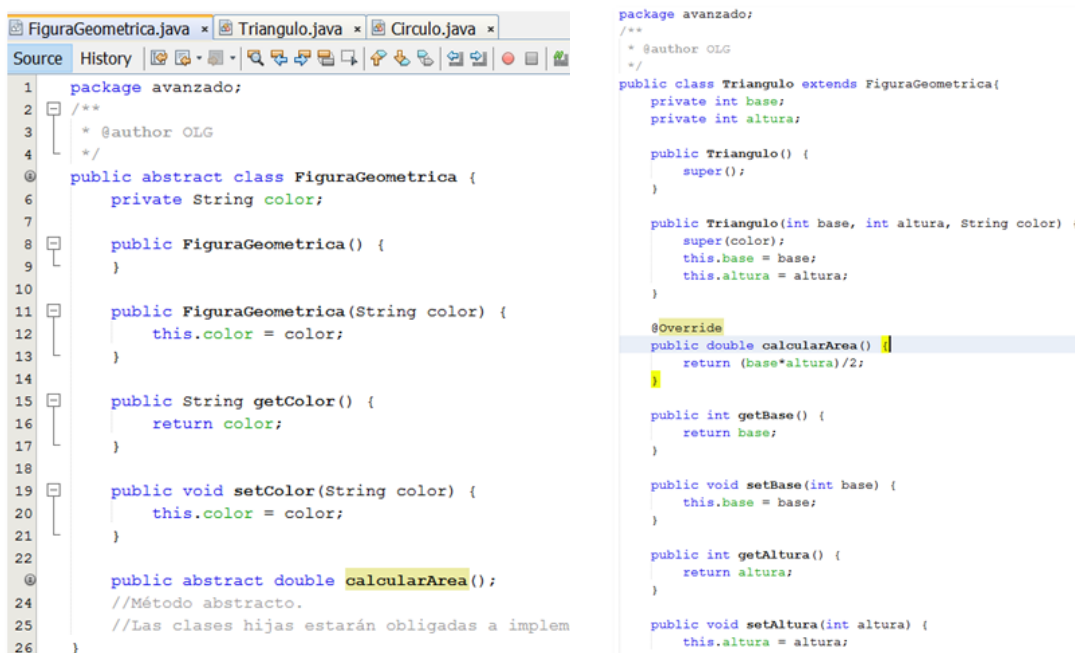
Definamos un método abstracto (sin implementar), por ejemplo, el método `calcularArea`, para que cada clase hija OBLIGATORIAMENTE implemente su propio método de forma diferente:

```
public abstract void calcularArea (); //no lleva llaves { }
```

```
//también incluimos la palabra abstract (igual que en la clase)
```

El uso de una clase abstracta toma sentido cuando queremos crear una clase padre que sirva como “plantilla” para las clases hijas, ya que la hijas están obligadas a sobrescribir todos los métodos abstractos que heredan.

Veamos un ejemplo completo:



```
package avanzado;

/**
 * @author OLG
 */
public abstract class FiguraGeometrica {
    private String color;

    public FiguraGeometrica() {
    }

    public FiguraGeometrica(String color) {
        this.color = color;
    }

    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public abstract double calcularArea();
    //Método abstracto.
    //Las clases hijas estarán obligadas a implem
}

package avanzado;

/**
 * @author OLG
 */
public class Triangulo extends FiguraGeometrica {
    private int base;
    private int altura;

    public Triangulo() {
        super();
    }

    public Triangulo(int base, int altura, String color) {
        super(color);
        this.base = base;
        this.altura = altura;
    }

    @Override
    public double calcularArea() {
        return (base*altura)/2;
    }

    public int getBase() {
        return base;
    }

    public void setBase(int base) {
        this.base = base;
    }

    public int getAltura() {
        return altura;
    }

    public void setAltura(int altura) {
        this.altura = altura;
    }
}
```

```
package avanzado;
/**
 * @author OLG
 */
public class Circulo extends FiguraGeometrica {

    private int radio;

    public Circulo() {
        super();
    }

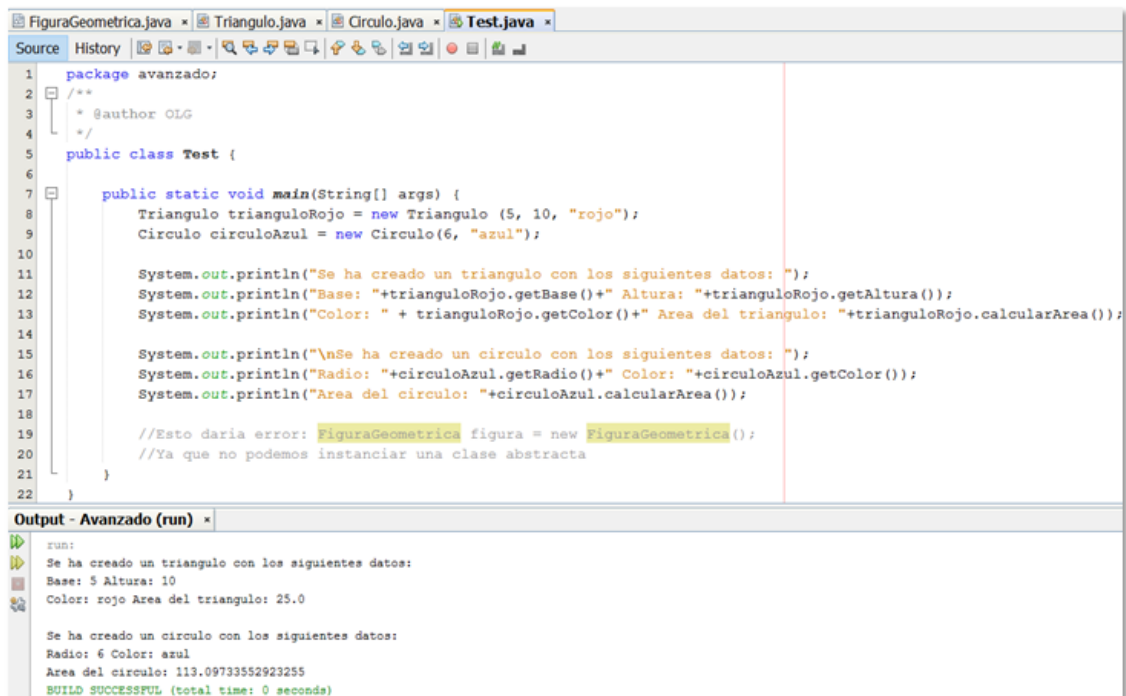
    public Circulo(int radio, String color) {
        super(color);
        this.radio = radio;
    }

    @Override
    public double calcularArea() {
        return Math.PI*radio*radio;
    }

    public int getRadio() {
        return radio;
    }

    public void setRadio(int radio) {
        this.radio = radio;
    }
}
```

Una vez hemos definido la jerarquía de clases, pasamos a ver el programa principal:



The screenshot shows an IDE with four tabs: `FiguraGeometrica.java`, `Triangulo.java`, `Circulo.java`, and `Test.java`. The `Test.java` tab is active, displaying the following code:

```
1 package avanzado;
2 /**
3  * @author OLG
4  */
5 public class Test {
6
7     public static void main(String[] args) {
8         Triangulo trianguloRojo = new Triangulo (5, 10, "rojo");
9         Circulo circuloAzul = new Circulo(6, "azul");
10
11         System.out.println("Se ha creado un triangulo con los siguientes datos: ");
12         System.out.println("Base: "+trianguloRojo.getBase()+" Altura: "+trianguloRojo.getAltura());
13         System.out.println("Color: " + trianguloRojo.getColor()+" Area del triangulo: "+trianguloRojo.calcularArea());
14
15         System.out.println("\nSe ha creado un circulo con los siguientes datos: ");
16         System.out.println("Radio: "+circuloAzul.getRadio()+" Color: "+circuloAzul.getColor());
17         System.out.println("Area del circulo: "+circuloAzul.calcularArea());
18
19         //Esto daria error: FiguraGeometrica figura = new FiguraGeometrica();
20         //Ya que no podemos instanciar una clase abstracta
21     }
22 }
```

Below the code editor, the `Output - Avanzado (run)` window shows the execution results:

```
run:
Se ha creado un triangulo con los siguientes datos:
Base: 5 Altura: 10
Color: rojo Area del triangulo: 25.0

Se ha creado un circulo con los siguientes datos:
Radio: 6 Color: azul
Area del circulo: 113.09733552923255
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Estructura de las clases

En este apartado se especifican los atributos y métodos que tendrán cada una de las clases, y deberás analizar cada una de ellas para estructurar correctamente la jerarquía. Es probable que, en algunos casos, no se especifiquen explícitamente los atributos o métodos, por lo que tendrás que leer con atención.

## Llave

A lo largo de la Central Nuclear de Chernobyl nos encontraremos distintas llaves para poder abrir las puertas de salida de la central. Estas llaves las tendrán que ir recogiendo nuestros operadores nucleares a medida que se las vayan encontrando por el camino. Todas las llaves estarán identificadas por una cadena de caracteres.

### Atributos

- idLlave. De tipo entero.
- celdaActual. De tipo entero.

### Métodos

Ninguno especialmente reseñable.

## Personaje

Será la clase padre a partir de la cual cuelgue toda la jerarquía. Esta clase no podrá ser instanciada por ningún objeto.

Debemos tener en cuenta que en *Chernobyl* disponemos, de manera genérica, de dos tipos de personajes: *Operadores Nucleares* y miembros de la *KGB*. Estos dos tipos de personajes tendrán unas características comunes: un nombre (no habrá nombres repetidos), turno en el que entran en acción, identificador de la celda en la que se encuentran en cada momento y la marca (símbolo, de tipo char, con el que se mostrarán por pantalla).

Además de las características comunes anteriormente mencionadas, nos encontraremos también con métodos que serán comunes tanto a los operadores como a los miembros de la KGB.

### Métodos

- *realizarAcciones*. método de tipo void que no recibe ningún parámetro. Este método se implementará en las clases hijas.
- *mover*. no retorna ni recibe nada. Para esta primera entrega, el método mostrará un mensaje con el nombre del Personaje que se está moviendo y el texto “se está moviendo”.

Ejemplo: “Shcherbina se está moviendo.”

## Operador

Los operadores nucleares son los personajes que, de una manera u otra, trabajan dentro de la central nuclear. En *Chernobyl* nos encontramos con 3 tipos de operadores: *Minero*, *Bombero* y *Científico*.

Cuando un miembro de la KGB se encuentre con un operador nuclear, éstos serán catalogados. Pero si no se encuentran con ninguno, no serán catalogados nunca. Parece lógico, entonces, que no podamos crear ningún operador nuclear que haya sido catalogado...

### Métodos

- *realizarAcciones*. Esta clase también dispondrá del método *realizarAcciones* que se implementará en las clases hijas.

## Minero

Los mineros se encargarán de retirar los escombros derivados de la explosión del núcleo. Esto, al igual que los kilos de escombros que van retirando en cada turno, se detallará en el próximo enunciado. En esta primera entrega los mineros no dispondrán de ninguna característica en concreto.

### Atributos

Nada destacable.

### Métodos

Implementaremos el método *realizarAcciones*. En esta primera entrega mostrará el mensaje "Minero [nombre] realizando sus acciones."

Ejemplo: "Minero Glújov realizando sus acciones."

## Bombero

El objetivo de los bomberos es el de refrigerar las zonas por las que vayan moviéndose. En esta clase necesitaremos almacenar un entero, que nos indicará cuántos turnos faltan para que se agote la batería del refrigerador.

### Atributos

- *bateriaRefrigerador*. De tipo entero. Estará siempre inicializada a 5.

### Métodos

Implementaremos el método *realizarAcciones*. En esta primera entrega mostrará el mensaje "Bombero [nombre] realizando sus acciones."

Ejemplo: "Bombero Ignatenko realizando sus acciones."

La clase Bombero dispondrá, además, de los siguientes métodos:

- *recargarRefrigerador*. Este método no podrá ser ejecutado desde fuera de esta clase y su único cometido será el de poner a 5 el atributo *bateriaRefrigerador*.



- refrigerar. Para esta primera parte de la práctica, sólo tendrá que mostrar el mensaje “El bombero [nombre] está refrigerando la zona.”.

Ejemplo: “El bombero Ignatenko está refrigerando la zona.”

## Científico

Los científicos, en la simulación de *Chernobyl*, serán los encargados de ir recogiendo las llaves que se vayan encontrando a su paso para abrir la puerta de salida. En esta primera entrega no tendrán ningún atributo específico, pero en próximos enunciados se detallará cómo almacenar todas las llaves que vayan recopilando.

### Métodos

Se implementará el método `realizarAcciones` tal y como se especifica para el resto de clases hijas de `Operador`.

Ejemplo: “Científico Legásov realizando sus acciones.”

## Kgb

Son los enemigos de los operadores. *Chernobyl* dispone de dos tipos de miembros de la KGB, la temida policía secreta de la Unión Soviética: *Voluntarios* y *Oficiales*. Al ser los dos miembros del KGB tienen puntos en común (ambos catalogarán a los operadores nucleares que se encuentran a lo largo de la simulación).

### Atributos

Nada destacable.

### Métodos

- *realizarAcciones*. Esta clase también dispondrá del método `realizarAcciones` que se implementará en las clases hijas.

- catalogar. Debemos implementar aquella acción común a los dos miembros de la KGB: La de catalogar a los operadores nucleares que hayan reconocido.

Ejemplo: “Anatoli está catalogando a un nuevo Operador Nuclear.”

## Oficial

Los *Oficiales* son los miembros de la KGB de mayor rango y, por tanto, los más peligrosos. De hecho, estos miembros de la KGB, en próximas entregas, desarrollarán la capacidad de destruir las llaves que hayan cogido los científicos, impidiendo a todos los operadores nucleares escapar de la central nuclear.

### Métodos

Implementaremos el método `realizarAcciones`:

Ejemplo: “Oficial Vladimir realizando sus acciones.”

## Voluntario

Los otros enemigos de nuestros operadores nucleares son los voluntarios. Estos son aspirantes a Oficiales, por lo que no son tan peligrosos.

### Métodos

Implementaremos el método realizarAcciones:

Ejemplo: "Voluntario Yuri realizando sus acciones."

**Nota importante:** Todas las clases deberán implementar constructores por defecto, parametrizados, *getters* y *setters*. Las clases "nietas", así como la clase Llave, dispondrán de su correspondiente método *toString*.

# Programa principal

Debes utilizar el programa principal que te facilita el profesor. El código fuente del programa principal es el siguiente:

```
public static void main(String[] args) {  
  
    //OPERADORES NUCLEARES  
  
    Personaje p1 = new Cientifico("Legásov", 1, 10, 'L');  
    Personaje p2 = new Cientifico("Shcherbina", 2, 1, 'S');  
    Personaje p3 = new Minero("Glújov", 4, 0, 'G');  
    Personaje p4 = new Bombero();  
    p4.setNombre("Ignatenko");  
    p4.setTurno(7);  
    p4.setIdCeldaActual(12);  
    p4.setMarca('I');  
  
    //KGB  
  
    Personaje p5 = new Oficial("Vladimir", 1, 11, 'V');  
    Personaje p6 = new Voluntario("Yuri", 1, 34, 'Y');  
    Personaje p7 = new Voluntario("Anatoli", 1, 35, 'T');  
  
    //LLAVES  
  
    Llave llave1 = new Llave("001", 31);  
    Llave llave2 = new Llave("002", 44);  
  
    //MOSTRANDO INFORMACIÓN  
  
    System.out.println(""); //Línea en blanco  
    System.out.println("OPERADORES NUCLEARES");  
    System.out.println(p1.toString());  
    System.out.println(p2.toString());  
    System.out.println(p3.toString());  
    System.out.println(p4.toString());  
    System.out.println(""); //Línea en blanco
```

```
System.out.println("KGB");

System.out.println(p5.toString());

System.out.println(p6.toString());

System.out.println(p7.toString());

System.out.println(""); //Línea en blanco


System.out.println("LLAVES");

System.out.println(llave1.toString());

System.out.println(llave2.toString());

System.out.println(""); //Línea en blanco


System.out.println("SIMULACIÓN");

p1.realizarAcciones();

p2.realizarAcciones();

p3.realizarAcciones();

p4.realizarAcciones();

p5.realizarAcciones();

p6.realizarAcciones();

p7.realizarAcciones();

}
```

# Commits

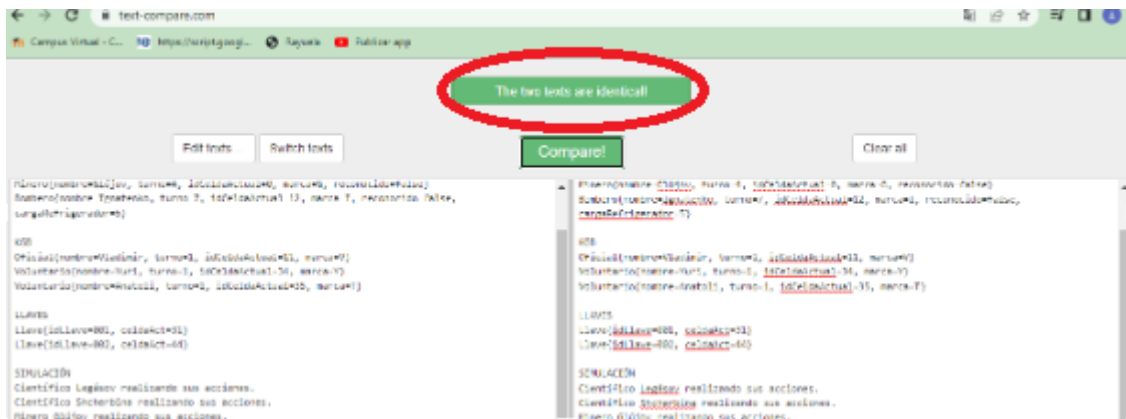
Nuestra práctica Chernobyl debe estar alojada, obligatoriamente, en un repositorio de GitHub. Deberás realizar, mínimo, los siguientes commits en tu repositorio:

nº Commit	Comentario
1	<i>Clase Llave</i>
2	<i>Clase Personaje</i>
3	<i>Clase Operador</i>
4	<i>Clase Minero</i>
5	<i>Clase Bombero</i>
6	<i>Clase Científico</i>
7	<i>Clase Kgb</i>
8	<i>Clase Oficial</i>
9	<i>Clase Voluntario</i>
10	<i>Finalizada Entrega1</i>

# Ejecución

Recuerda que tu ejecución debe coincidir completamente con la ejecución del profesor. En caso de no coincidir completamente, la calificación de tu práctica será de 3 sobre 10.

Para comparar el resultado de la ejecución, se recomienda al alumno la utilización de la aplicación web [text-compare.com](https://text-compare.com) (En una columna pegaremos el resultado de nuestra ejecución y en la otra columna pegaremos el resultado de la ejecución del profesor).



# Entrega

La *Entrega 1: Jerarquía de clases* se hará a través de una tarea que se habilitará días antes de la finalización del trimestre en la plataforma Moodle.

Para cada entrega, el alumno entregará un archivo comprimido (.zip) con el nombre *EntregaX\_Apellidos\_Nombre.zip*.

Ejemplo: Entrega1\_PinedaNavas\_AngelEnrique.zip

En cuanto a las fechas de entrega serán, aproximadamente, dos semanas antes de la finalización de cada trimestre. Una vez vencida la fecha de entrega, el alumno deberá realizar la defensa de la misma.

Para corregir la práctica es imprescindible que compile correctamente y ejecute. Además, toda práctica entregada fuera de plazo se considerará como no entregada.

# Calificación

De forma general, los puntos a tener en cuenta para la evaluación del proyecto y su defensa son:

## **Bloque 0: Evaluación previa.**

- Fecha de entrega.
- GitHub.
- Interacción entre el profesor y el alumno.

## **Bloque 1: Aspectos formales y estéticos del código.**

- Limpieza (No deja código que no se utilice, ...).
- Sangría.
- Documentación interna.

## **Bloque 2: Código fuente.**

- Constructores.
- Variables.
- Estructuras condicionales y repetitivas.
- Métodos (paso de parámetros, return, ...)
- Jerarquía de clases (atributos, métodos, uso abstract, ...).

## **Bloque 3: Funcionamiento.**

- Ejecución de la práctica.

## **Bloque 4: Defensa.**

- Consistirá en una modificación de tu práctica. La calificación será de *Apto* o *No Apto*.