92 - Creación de Interfaces en Java

Listado completo de tutoriales

El objetivo de una interfaz es declarar una serie de métodos sin su implementación. Luego una o varias clases pueden implementar dicha interfaz.

Una interfaz es similar a una clase abstracta con todos sus métodos abstractos. En Java no existe la herencia múltiple por lo que las interfaces son ampliamente utilizadas.

Mediante una interfaz indicamos que debe hacerse pero no como se debe implementar. Veremos con un ejemplo la creación de una interfaz y la implementación de la misma en dos clases distintas.

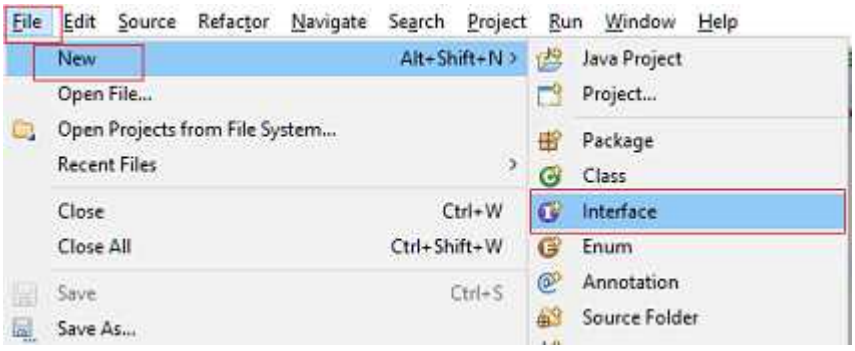
Problema:

Se desea implementar dos juegos distintos.
El primero que permita ingresar el nombre de dos jugadores, tirar dos dados y mostrar cual de los dos ganó.
El segundo juego permita a un jugador adivinar un número entre 1 y 100 que elige la computadora al azar.

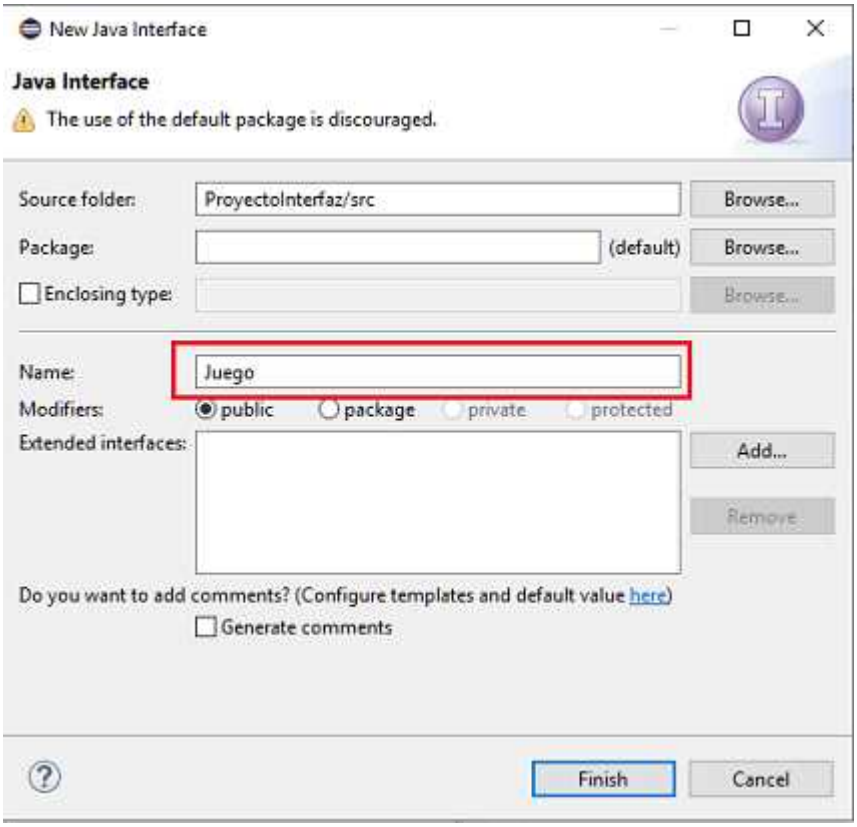
Podemos decir que todo juego tiene un inicio, estado de juego propiamente dicho y un fin. Si queremos estandarizar estos dos juegos y posibles nuevos juegos podemos declarar una interface llamada 'Juego' que declare los métodos: iniciar, jugar y finalizar.

Luego cada vez que creemos un juego debe respetar esta interface implementándola.

Para crear una interfaz en Eclipse elegimos la opción File -> New -> Interface:



En el diálogo siguiente indicamos el nombre de la interfaz, en nuestro ejemplo la llamaremos 'Juego':



Interface: Juego

```
public interface Juego {
    void iniciar();
    void jugar();
    void finalizar();
}
```

Se utiliza la palabra clave 'interface' y seguidamente el nombre de la interface.

Entre llaves declaramos los métodos, pero no su implementación. No se requiere que definamos que son public y abstract, ya que lo son por defecto.

No podemos crear un objeto de la interfaz 'Juego'. Solo se declaran para que otras clases los implementen.

Veamos ahora las clases que implementan la interfaz en nuestro problema:

Clase: JuegoDeDados

```
import java.util.Scanner;

public class JuegoDeDados implements Juego {
    private int dado1, dado2;
    private String jugador1;
    private String jugador2;
    private Scanner teclado;
```

```
    public JuegoDeDados() {
        teclado = new Scanner(System.in);
    }

    public void iniciar() {
        System.out.print("Ingrese el nombre del primer jugador:");
        jugador1 = teclado.nextLine();
        System.out.print("Ingrese el nombre del segundo jugador:");
        jugador2 = teclado.nextLine();
    }

    public void jugar() {
        dado1 = 1 + (int) (Math.random() * 6);
        dado2 = 1 + (int) (Math.random() * 6);
        System.out.println(jugador1 + " le salió el valor " + dado1);
        System.out.println(jugador2 + " le salió el valor " + dado2);
    }

    public void finalizar() {
```

Para indicar que la clase 'JuegoDeDados' implementará la interfaz 'Juego' se especifica luego de la palabra clave 'implements':

```
public class JuegoDeDados implements Juego {
```

Una clase puede implementar más de una interface, para ello indicamos los nombre de las interfaces a implementar separadas por coma, por ejemplo:

```
public class JuegoDeDados implements Juego, ActionListener, WindowList
```

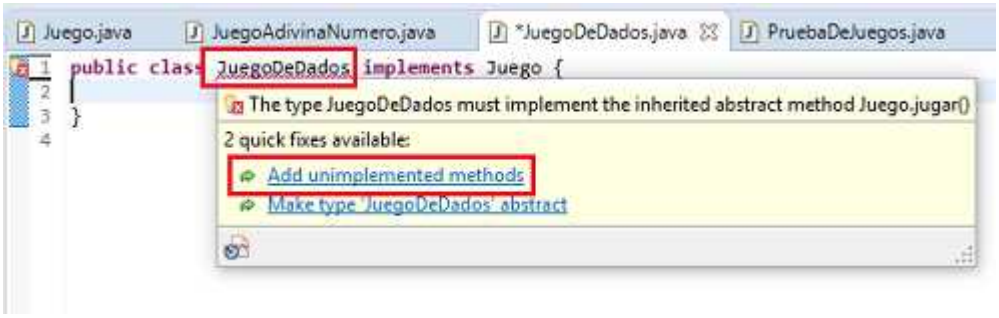
Luego de indicar que la clase 'JuegoDeDados' implementará la interfaz 'Juego' estamos obligados a codificar los algoritmos de los tres métodos contenidos en la interfaz:

```
    public void iniciar() {
        System.out.print("Ingrese el nombre del primer jugador:");
        jugador1 = teclado.nextLine();
        System.out.print("Ingrese el nombre del segundo jugador:");
        jugador2 = teclado.nextLine();
    }

    public void jugar() {
        dado1 = 1 + (int) (Math.random() * 6);
        dado2 = 1 + (int) (Math.random() * 6);
        System.out.println(jugador1 + " le salió el valor " + dado1);
        System.out.println(jugador2 + " le salió el valor " + dado2);
    }
```

```
    public void finalizar() {
        if (dado1 > dado2)
            System.out.println("Gano " + jugador1 + " con un valor de
        else if (dado2 > dado1)
            System.out.println("Gano " + jugador2 + " con un valor de
        else
            System.out.println("Empataron los dos jugadores con el val
    }
```

Con el entorno de Eclipse podemos fácilmente escribir los tres métodos eligiendo la opción 'Add unimplements methods':



De forma similar creamos la clase 'JuegoAdivinaNumero'

Clase: JuegoAdivinaNumero

```
import java.util.Scanner;

public class JuegoAdivinaNumero implements Juego {
    private int numAdivina;
    private Scanner teclado;
    private int intentos;

    public JuegoAdivinaNumero() {
        teclado = new Scanner(System.in);
    }

    public void iniciar() {
        numAdivina = 1 + (int) (Math.random() * 100);
    }

    public void jugar() {
        int numero;
        do {
            System.out.print("Adivina un número entre 1 y 100:");
            numero = teclado.nextInt();
            if (numAdivina < numero)
                System.out.println("El número a adivinar es menor");
            else if (numAdivina > numero)
                System.out.println("El número a adivinar es mayor");
            intentos++;
        } while (numero != numAdivina);
    }
```

Si bien son juegos muy distintos, la implementación de la interfaz 'Juego' nos estandariza los métodos mínimos que debe cumplir todo juego.

Par probar el funcionamiento codificaremos una tercer clase donde crearemos un objeto para cada uno de los juegos.

Clase: PruebaDeJuegos

```
public class PruebaDeJuegos {

    public static void main(String[] args) {
        JuegoDeDados juego1 = new JuegoDeDados();
        juego1.iniciar();
        juego1.jugar();
        juego1.finalizar();

        JuegoAdivinaNumero juego2 = new JuegoAdivinaNumero();
        juego2.iniciar();
        juego2.jugar();
        juego2.finalizar();
    }

}
```

Creamos un objeto de la clase 'JuegoDeDados' y llamamos a los métodos: iniciar, jugar y finalizar:

```
        JuegoDeDados juego1 = new JuegoDeDados();
        juego1.iniciar();
        juego1.jugar();
        juego1.finalizar();
```

De forma similar creamos el objeto de la clase 'JuegoAdivinaNumero'.

Herencia en interfaces

Una interfaz puede heredar de otra, luego la clase que la implementa debe codificar todos los métodos.

Interfaz: Interface1

```
public interface Interface1 {
    void metido1();
}
```

Interfaz: Interface2

```
public interface Interface2 extends Interface1 {
    void metodo2();
    void metodo3();
}
```

Clase: Interface2

```
public class PruebaInterfaces implements Interface2 {

    public void metido1() {

    }

    public void metodo2() {

    }

    public void metodo3() {

    }

}
```

Como podemos comprobar la clase 'PruebaInterfaces' al implementar la interfaz 'Interface2' debe codificar los tres métodos.

Herencia múltiple en interfaces

A diferencia de las clases que pueden heredar solo de una única clase, las interfaces pueden heredar de múltiples interfaces.

Interfaz: A

```
public interface A {
    void metodo1();
}
```

Interfaz: B

```
public interface B {
    void metodo2();
}
```

Interfaz: C

```
public interface C extends A, B {
    void metodo3();
```

```
}
```

Clase: PruebaInterfaces

```
public class PruebaInterfaces implements C{

    public void metodo1() {

    }

    public void metodo2() {

    }

    public void metodo3() {

    }

}
```

Como podemos comprobar la interface 'C' hereda de las interfaces 'A' y 'B':

```
public interface C extends A, B {
    void metodo3();
}
```

Retornar