Ejercicio U5_B5A_E1:

debo crear una clase MasTres que produzca la salida anterior.

Ahora la estructura del proyecto debe ser:

Si añado MasTres de la siguiente forma

```
class MasTres {
    int inicio;
    int val;
    MasTres(){
        inicio=0;
        val=0;
    }
    public int obtenerSiguiente(){
        val +=3;
        return val;
    }
    public void restablecer(){
        inicio=0;
        val=0;
    }
    public void establecerInicio(int x){
        inicio=x;
        val=x;
    }
}
```

¡no implementamos el interface!

Para implementar el interface escribimos:

```
class MasTres implements Serie{
    int inicio;
    int val;
    MasTres(){
        inicio=0;
        val=0;
    }
    public int obtenerSiguiente(){
        val +=3;
        return val;
    }
    public void restablecer(){
        inicio=0;
        val=0;
    }
    public void establecerInicio(int x){
        inicio=x;
        val=x;
    }
}
```

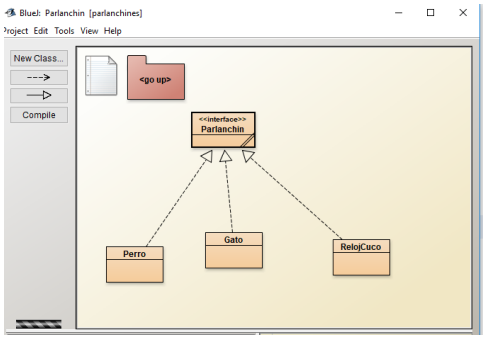
Y podemos comprobar que en ambos casos nuestro main() produce exactamente los mismos resultados(no sería así con otros main(), si se precisara usar capacidades de polimorfismo). Es mejor la solución de implementar el interface por dos razones:

- Si implementamos el interface el compilador nos avisará si nos olvidamos de implementar algún método, o de si nos equivocamos en algún parámetro o tipo de retorno de algún método.
- si MasTres no "implementa" el interfaz *Serie* perderemos capacidades de polimorfismo como veremos en un apartado posterior.

Ejercicio U5_B5A_E2:

```
//Parlanchin.java
package parlanchines;
public interface Parlanchin {
    public abstract void habla();
}

//Perro.java
package parlanchines;
public class Perro  implements Parlanchin{
    @Override
    public void habla(){
        System.out.println("¡Guau!");
    }
}

//Gato.java
package parlanchines;
public class Gato  implements Parlanchin{
    @Override
    public void habla(){
        System.out.println("¡Miau!");
    }
}



//RelojCuco.java
package parlanchines;
public class RelojCuco  implements Parlanchin{
    @Override
    public void habla(){
        System.out.println("¡Cucu, cucu, ..!");
    }
}


import parlanchines.*;
public class App {
    public static void main(String[] args){
        Gato g = new Gato();
        Perro p= new Perro();
        RelojCuco rc= new RelojCuco();

        g.habla();
        p.habla();
        rc.habla();
    }

}
```

Si te apetece puedes hacerlo en bluej para comprobar que "dibujito" hace bluej de la jerarquía



Ejercicio U5_B5A_E3:

Todo igual salvo que ahora Gato y Perro extienden a Animal, y Animal implementa Parlanchin

Observa que la clase Animal se escribe vacía pero que realmente hereda de Parlachin

public abstract void habla()

Que a su vez propaga a sus subclases.

```
package parlanchines;
public abstract class Animal implements Parlanchin{
        //hereda habla() de parlanchin
}

//Gato.java
package parlanchines;
public class Gato  extends Animal{
   @Override
   public void habla(){
      System.out.println("¡Guau!");
   }
}

//Perro.java
package parlanchines;
public class Perro  extends Animal{
   @Override
   public void habla(){
      System.out.println("¡Guau!");
   }
}
```