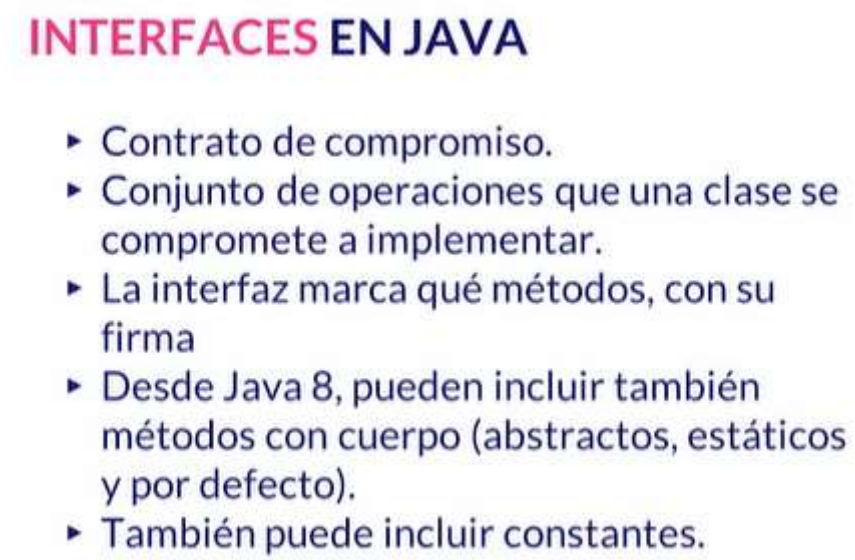
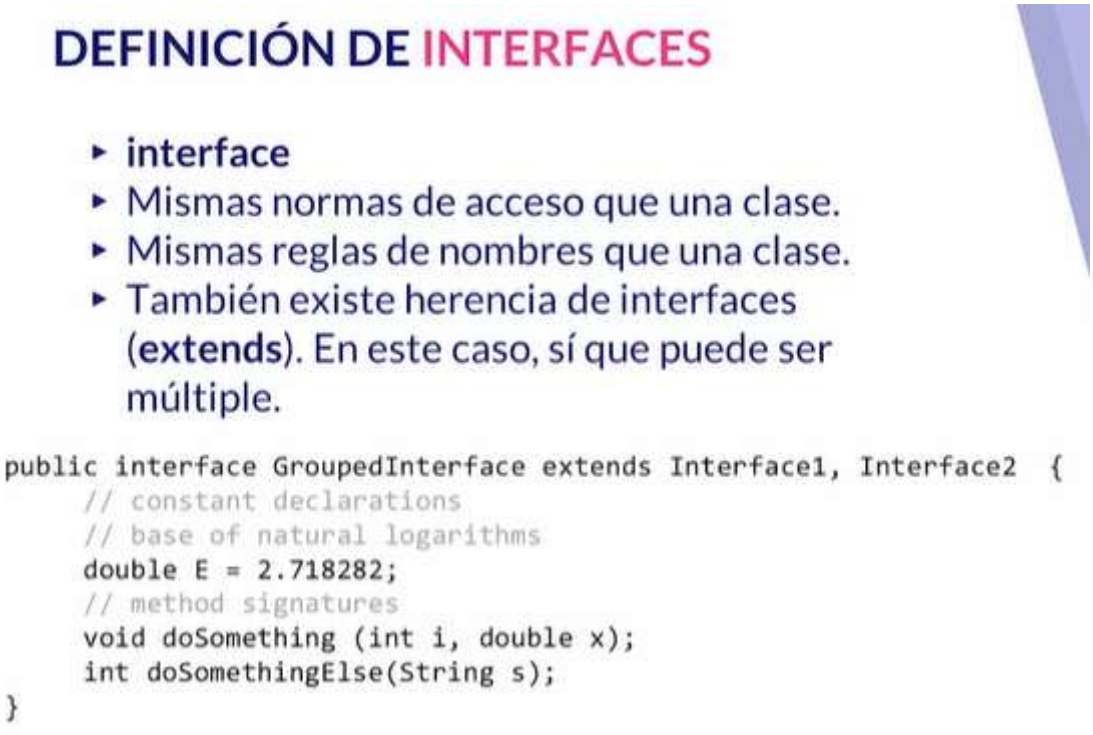
INTERFACES: Hemos visto parte de este concepto cuando hablamos de clases abstractas.

Observa que tipo de métodos pueden tener, atributos,…





Observa como en el ejemplo la interface extiende de otras, hereda de otras, puede ser herencia multiple. Una interface extiende de otras dos en el ejemplo. Y en su cuerpo tiene constantes y métodos abstractos, no pongo abstrac porque estoy en una interface y no hace falta ponerlo.

IMPLEMENTACIÓN DE INTERFACES

- **implements**
- Una clase puede implementar más de una interfaz

```
public class RectanglePlus implements Relatable {
//...
    public int isLargerThan(Relatable other) {
        RectanglePlus otherRect = (RectanglePlus)other;
        if (this.getArea() < otherRect.getArea())
            return -1;
        else if (this.getArea() > otherRect.getArea())
            return 1;
        else
            return 0;
    }
}
```

Ahora una clase implementa una o varias interfaces.

INTERFACES COMO TIPOS

- Una interfaz puede ser el tipo de dato a usar para crear una instancia de un objeto.
- La clase del objeto a crear debe implementar dicha interfaz.
- Muy útil, sobre todo, para recibir argumentos en métodos.

```
RectanglePlus rectangleOne = new RectanglePlus(10, 20);
Relatable rectangleTwo = new RectanglePlus(20, 10);
```

Fíjate bien en el ejemplo, puedo instanciar objetos tanto haciendo referencia a la clase, como haciendo referencia a la interfaz.

MÉTODOS POR DEFECTO

- Novedad en Java 8
- **default.**
- Un método puede tener una implementación por defecto, descrita en la interfaz.

```
public interface Interfaz {

     default public void metodoPorDefecto() {
         System.out.println("Este es uno de los nuevos
                                    métodos por defecto");
     }

}
```

Si una clase implementa de esa interface y no necesita añadir nada a ese método por defecto, no es necesario que lo implemente (de sobre escribirlo). Si quisiera modificar ese método puede implementarlo, sobre escribirlo. Y cuando llamemos a ese método no usará el de por defecto, sino el nuevo que ha implementado la clase.

Una interface puede tener métodos estáticos, para poder hacer operaciones auxiliares, para comparar clases,… están disponible en muchas interfaces de java.

MÉTODOS ESTÁTICOS

- Novedad en Java 8
- **static.**
- Misma sintaxis que los métodos estáticos en clases

```
public interface Interfaz {

     public static void metodoEstatico() {
         System.out.println("Método estático en un interfaz");
     }
}
```

Ejemplo:

Para crear una interfaz, File--- new---interface y ponemos nombre.

```
public interface Relatable {

    /*
     * Método que nos va a permitir si un objeto
     * de este tipo es más grande que otro
     *
     */
    public int isLargerThan(Relatable other);

}
```

Una clase, que implemente esta interface, tiene un método que comprueba si dos objetos son uno más grande que otro. Por ejemplo, un círculo con un triángulo. Por ejemplo, esta clase, que tiene varios métodos, pero que obligatoriamente tiene que implementar el método que le obliga la interface que implementa.

```
public class RectanglePlus implements Relatable {


    public int width = 0;
    public int height = 0;
    public Point origin;

    // four constructors
    public RectanglePlus() {
        origin = new Point(0, 0);
    }
    public RectanglePlus(Point p) {
        origin = p;
```

Método que hay que implementar:

```
    // a method required to implement
    // the Relatable interface
    public int isLargerThan(Relatable other) {
        RectanglePlus otherRect = (RectanglePlus)other;
        if (this.getArea() < otherRect.getArea())
            return -1;
        else if (this.getArea() > otherRect.getArea())
            return 1;
        else
            return 0;
    }
```

Ejemplo, para llamar al método:

```java
public static void main(String[] args) {

    RectanglePlus rectangleOne = new RectanglePlus(10, 20);
    Relatable rectangleTwo = new RectanglePlus(20, 10);

    switch (rectangleOne.isLargerThan(rectangleTwo)) {
    case -1:
        System.out.println("Es menor");
        break;
    case 0:
        System.out.println("Son iguales");
        break;
    case 1:
        System.out.println("Es mayor");
        break;
    }
```

Creo dos objetos, referencias (uno a la clase, otro a la interface), pero los constructores son de la clase, recuerda, no puedo crear objetos de la interface. Un objeto llama al método y le pasa como parámetro otro objeto, para compararlos.

Otro ejemplo:

```java
public interface Interfaz {

    public void metodo();

    default public void metodoPorDefecto() {
        System.out.println("Este es uno de los nuevos métodos por defecto");
    }

    public static void metodoEstatico() {
        System.out.println("Método estático en un interfaz");
    }

}
```

Esta interface tiene un método abstracto (sin código), otro por defecto (puede tener código) y otro estático que también tiene código.

Observa cuales son obligatorio implementar:

Una clase que implementa esa interfaz, con solo el método que es obligatorio:

```java
public class Clase implements Interfaz {

    @Override
    public void metodo() {
        System.out.println("método");
    }

}
```

Otra que tiene el obligatorio, y sobreescribe el por defecto:

```java
public class Clase2 implements Interfaz {

    @Override
    public void metodo() {
        System.out.println("Otro método");
    }

    @Override
    public void metodoPorDefecto() {
        System.out.println("Mi propia implementación del método por defecto");
    }
```

Llamadas a los diferentes métodos:

```java
public class InterfacesPorDefecto {

    public static void main(String[] args) {

        Clase c1 = new Clase();

        c1.metodo();
        c1.metodoPorDefecto();

        Clase2 c2 = new Clase2();

        c2.metodo();
        c2.metodoPorDefecto();

        Interfaz.metodoEstatico();
```

Recuerda que para llamar a los métodos estáticos no uso los objetos, sino la clase, en este caso la interfaz.

Ejecución: