Interfaces y Clases Abstractas

Interfaces y Clases Abstractas

INTERFACES

Son estructuras de datos, que tienen solamente las firmas de los métodos, sin implementación (métodos abstractos)

```
public interface Lavable{
    void lavar();
    public void secar();
}
```

Interfaces y Clases Abstractas

Los métodos deben implementarse en las clases que implementan la interface

```
public interface Lavable{
    void lavar();
    public void secar();
}
```

```
public class Peluche
implements Lavable{

}
```

ERROR DE COMPILACIÓN - No implementa los métodos de la interface

Interfaces y Clases Abstractas

Los métodos deben implementarse en las clases que implementan la interface

```
public interface Lavable{
    void lavar();
    public void secar();
}
```

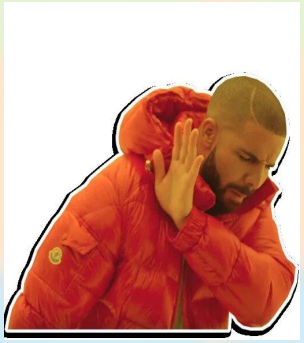
```
public class Peluche
implements Lavable{
    void lavar(){}
    public void secar(){}

}
```

IMPLEMENTA LOS MÉTODOS

Interfaces y Clases Abstractas

En las CLASES se pueden tener métodos abstractos, es decir sin implementación

```
public class Comible{
    public void comer();
}
```

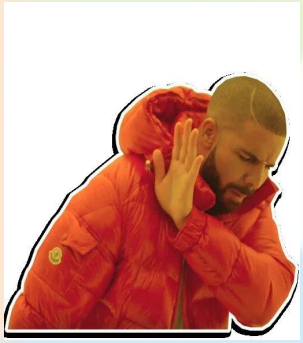


ERROR DE COMPILACIÓN!! Debe agregar el modificador abstract

Interfaces y Clases Abstractas

En las CLASES se pueden tener métodos abstractos, es decir sin implementación

```
public class Comible{
    public abstract void comer();
}
```

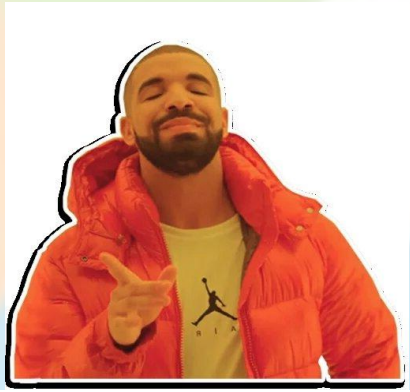


ERROR DE COMPILACIÓN!! Si una CLASE tiene un método abstracto, la clase debe ser abstract

Interfaces y Clases Abstractas

En las CLASES se pueden tener métodos abstractos, es decir sin implementación

```
public abstract class Comible{
    public abstract void comer();
}
```



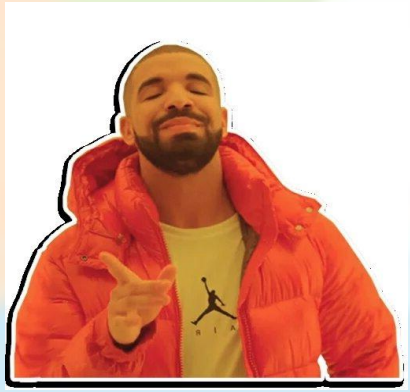
En una clase los métodos abstractos deben tener el modificador **abstract**

Si una clase tiene métodos abstractos, la clase debe ser **abstract**

Interfaces y Clases Abstractas

Una clase abstracta puede tener métodos abstractos y métodos implementados

```
public abstract class Comible{
    public abstract void comer();
    public void terminar(){ }
}
```

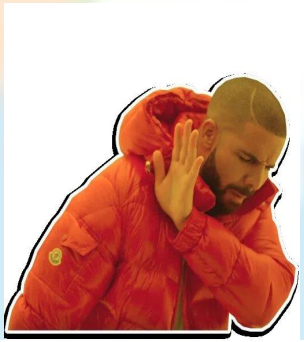


Interfaces y Clases Abstractas

Los métodos de las clases abstractas se implementan en las clases hijas

```
public abstract class Comible{
    public abstract void comer();
    public void terminar(){ }
}
```

```
 public  class Res extends Comible{
 }
```



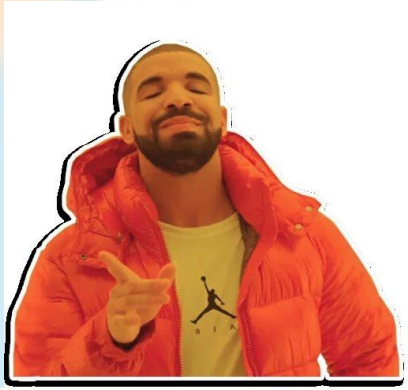
ERROR DE COMPILACIÓN!! La hija debe implemenar los métodos abstractos

Interfaces y Clases Abstractas

Los métodos de las clases abstractas se implementan en las clases hijas

```
public abstract class Comible{
    public abstract void comer();
    public void terminar(){ }
}
```

```
 public  class Res extends Comible{
    public void comer(){}
 }
```



Implementa el método abstracto

INTERFACES

Todos los métodos deben ser abstractos

Los métodos no necesitan el modificador abstract

No hace falta poner el modificador abstract a la interface

CLASES ABSTRACTAS

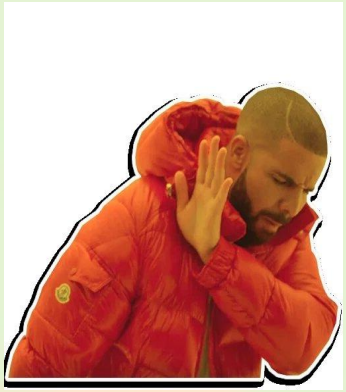
Puede tener métodos abstractos e implementados

Si el método no tiene implementación se debe poner el modificador abstract

Si hay un método abstracto en la clase, se debe poner el modificador abstract a la clase

Interfaces y Clases Abstractas

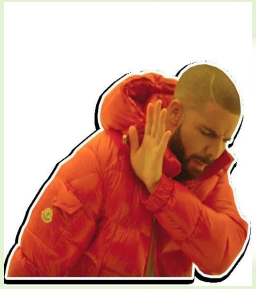
```
public interface Calculable{
    public void calcular() {  }
}
```



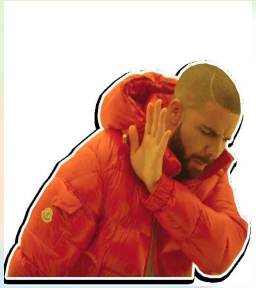
```
public interface Calculable{
    public void calcular();
}
```

Interfaces y Clases Abstractas

```
public class Expandible{
    public void expandir();
}
```



```
public class Expandible{
    public abstract void expandir();
}
```



```
public abstract class Expandible{
    public abstract void expandir();
}
```

Interfaces y Clases Abstractas

Cuando una clase hereda de otra o implementa una interface, HEREDA los métodos. Si hereda los métodos abstractos tiene 2 opciones:

1) Implementar los métodos
2) Quedarse con los métodos abstractos, y por lo tanto si tiene métodos abstractos, debe definirse como **abstract**

Interfaces y Clases Abstractas

```
public interface Lavable{
    void lavar();
    public void secar();
}
```

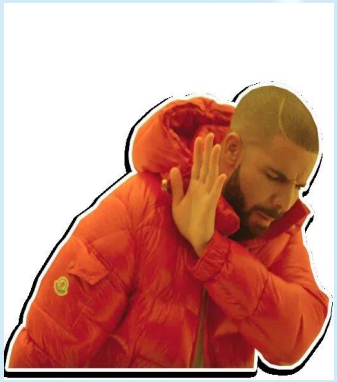
```
public class Dinero implements Lavable{
    public void lavar() { }
    public void secar() { }
}
```

Interfaces y Clases Abstractas

```
public interface Lavable{
    void lavar();
    public void secar();
}
```

```
public abstract class Dinero implements Lavable{
}
```

```
public class Dolar extends Dinero {
}
```



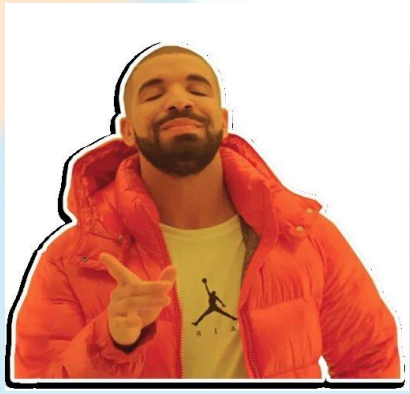
Interfaces y Clases Abstractas

```
public interface Lavable{
    void lavar();
    public void secar();
}

public abstract class Dinero implements Lavable{
}

public abstract  class Dolar extends Dinero {
}
```

Interfaces y Clases Abstractas

```
public abstract class Comible{
    public abstract void comer();
    public void terminar(){ }
}
```

```
public class Res extends Comible{
    public void comer(){ }
}
```

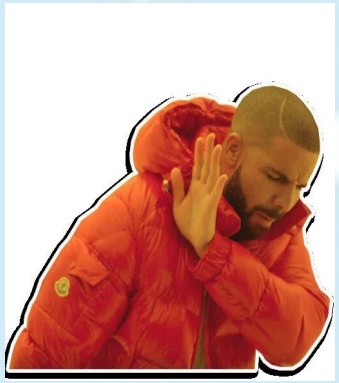


Interfaces y Clases Abstractas

```
public abstract class Comible{
    public abstract void comer();
    public void terminar(){ }
}

public abstract  class Pollo extends Comible{

}
```

Interfaces y Clases Abstractas

```
public abstract class Comible{
    public abstract void comer();
    public void terminar(){ }
}

public abstract  class Pollo extends Comible{

}

 public  class PolloFrito extends Pollo{

 }
```



Interfaces y Clases Abstractas

```
public abstract class Comible{
    public abstract void comer();
    public void terminar(){ }
}

public abstract  class Pollo extends Comible{

}

 public  class PolloFrito extends Pollo{
     public void comer(){ }
 }
```

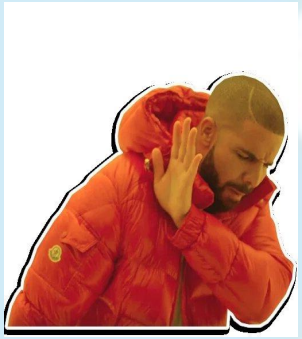
Interfaces y Clases Abstractas

```
public abstract class Comible{
    public abstract void comer();
    public void terminar(){ }
}
```

```
public interface Lavable{
    void lavar();
    public void secar();
}
```

Una clase puede heredar de otra (solo de una) y puede implementar una o más interfaces

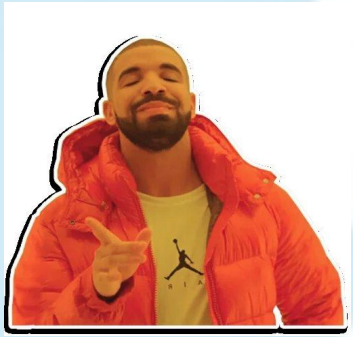
```
class Res extends Comible implements Lavable, Comparable{
}
```



Interfaces y Clases Abstractas

```
abstract class Res extends Comible implements Lavable,
Comparable{     }
```

```
class Res extends Comible implements Lavable, Comparable{
   public void lavar(){}
   public void secar(){}
    public void comer(){}
}
```



Interfaces y Clases Abstractas

```
class Animal{
    public void comer(){
        System.out.print("animal comiendo");
    }
}

  class Gato extends Animal{
     public void maullar(){
        System.out.print("gato maullando");
     }
  }
```

Interfaces y Clases Abstractas

```
class Animal{
    public void comer(){
        System.out.print("animal comiendo");
    }
}

  class Gato extends Animal{
    public void maullar(){
        System.out.print("gato maullando");
    }
  }
```

El hijo hereda los métodos del Padre, excepto los private, esos no se heredan

Interfaces y Clases Abstractas

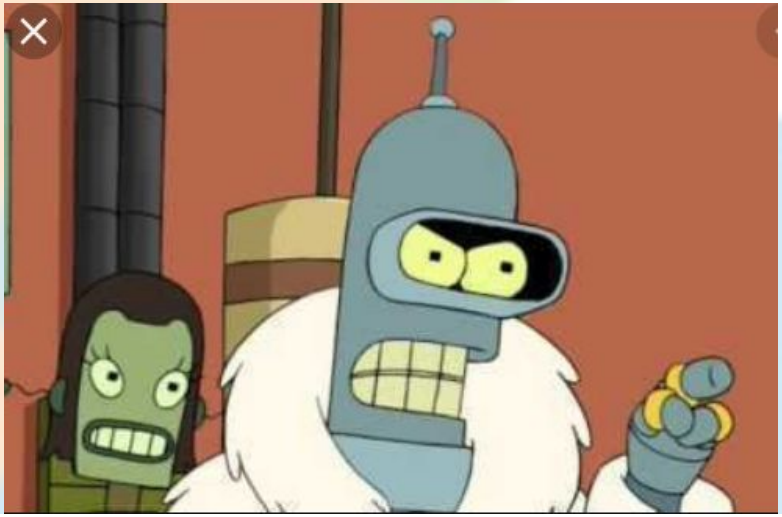
```
Gato g = new Gato();
g.maullar();
g.comer();
g.toString();
g.equals("s");
```

Interfaces y Clases Abstractas

```
Gato g = new Gato();
g.maullar();//de Gato
g.comer();//de Animal
g.toString(); //de Object
g.equals("s"); //de Object
```

Si la implementación del método que heredó del padre, no le sirve, la clase puede **SOBREESCRIBIR** el método.

Actualmente Gato ya tiene un método comer que lo heredó del padre, pero se decide **SOBREESCRIBIRLO**



Sobreescribiré mi propio método con juegos de azar y mujerzuelas!!

Interfaces y Clases Abstractas

```
class Animal{
    public void comer(){
        System.out.print("animal comiendo");
    }
}
    class Gato extends Animal{
        public void comer(){
            System.out.print("gato comiendo");
        }
        public void maullar(){
            System.out.print("gato maullando");
        }
    }
```

Interfaces y Clases Abstractas

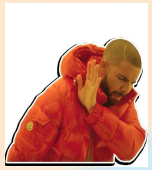
Cuando se sobreescribe un método, no se lo puede hacer más restrictivo

public
protected
default
private

Interfaces y Clases Abstractas

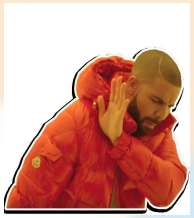
```
class Padre{
    public void m1() { }
}
```

```
class Hijo extends Padre{
    private void m1() { }
}
```

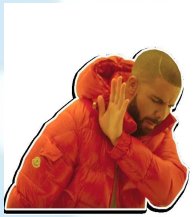


```
class Hijo extends Padre{
    public void m1() { }
}
```

```
class Hijo extends Padre{
    void m1() { }
}
```



```
class Hijo extends Padre{
    protected void m1() { }
}
```



Interfaces y Clases Abstractas

```
class Padre{
    void m1() { }
}
```

```
class Hijo extends Padre{
    private void m1() { }
}
```

```
class Hijo extends Padre{
    public void m1() { }
}
```

```
class Hijo extends Padre{
    void m1() { }
}
```

```
class Hijo extends Padre{
    protected void m1() { }
}
```