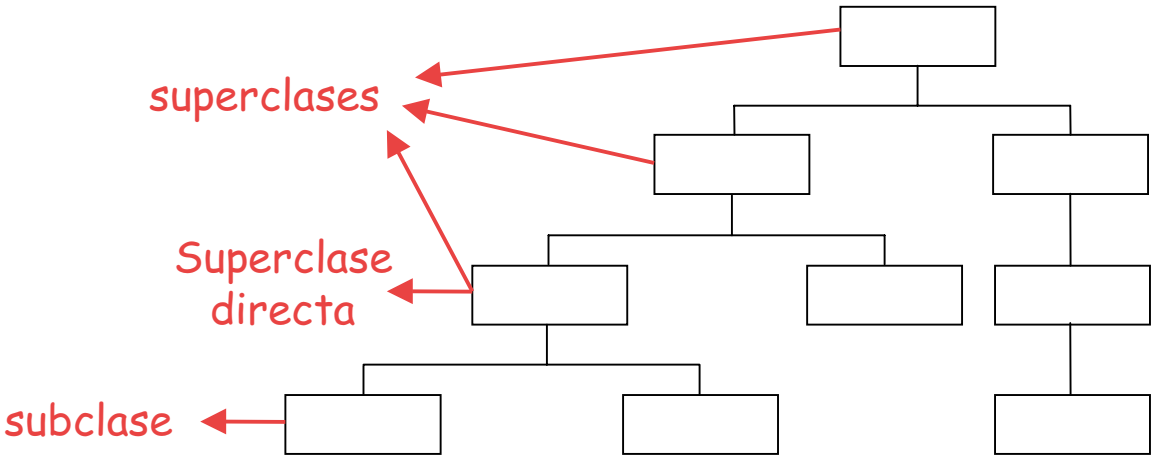
Herencia (I)

Mecanismo que permite construir a partir de una clase (**superclase**) otra clase (**subclase**) que *hereda* los miembros (variables y funciones) de la primera.

Herencia (II)

- **En Java la herencia es simple**, es decir, una clase sólo puede tener una superclase (directa).

- La herencia define una **JERARQUÍA DE CLASES** en la que cada clase tiene una superclase (directa), y cero o más subclases.
  Además de la superclase directa, cualquier clase puede tener múltiples superclases indirectas (todas sus antecesoras).
  Las clases situadas en la parte inferior de la jerarquía son más especializadas que las que están en la parte superior.



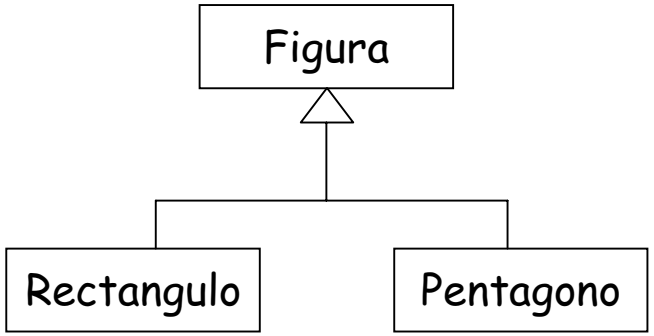
Definición de subclases

- La sintaxis para definir en Java una subclase es la siguiente:

```
class Subclase extends Superclase
```

- Por ejemplo, la jerarquía de la figura se definiría de la siguiente manera:

```
class Rectangulo extends Figura
{
    ...
}
class Pentagono extends Figura
{
    ...
}
```



- **Si** en la definición de una clase **no se especifica** la cláusula extends, se entiende que **su superclase es la clase Object**.
  La clase **Object** es la raíz de la jerarquía de clases de Java.

Miembros heredados (I)

- Una subclase **hereda todos los miembros** de su superclase **excepto los constructores**.

- **La herencia** de un miembro **no implica el acceso al mismo**. Las reglas de acceso son:
  - Una subclase no tiene acceso directo a los miembros **private** de su superclase.
  - Un subclase sí tiene acceso directo a los miembros **public** y **protected** de su superclase.
  - Si una subclase pertenece al mismo paquete de su superclase, también tiene acceso a los miembros sin calificar (**default**).

	public	protected	private	default
Desde una subclase del mismo paquete	SÍ	SÍ	NO	SÍ
Desde una subclase de otro paquete	SÍ	SÍ	NO	NO

Miembros heredados (II)

- **Ejemplo:** acceso a las variables miembro heredadas

```
public class Superclase
{
    public String atributoPublic = "atributoPublic";
    String atributoDefault = "atributoDefault";
    private String atributoPrivate = "atributoPrivate";
    protected String atributoProtected = "atributoProtected";
}

public class Subclase extends Superclase
{
    public String atributoPrueba;

    public Subclase()
    {
        atributoPrueba = atributoPublic;
        atributoPrueba = atributoDefault;
        atributoPrueba = atributoProtected;
        atributoPrueba = atributoPrivate;
    }
}
```

Definición de la superclase con atributos de distinto nivel de acceso

Acceso permitido

Acceso no permitido.
ERROR de compilación "Variable atributoPrivate in Superclase not accessible from Subclase"

Miembros heredados (III)

- **Ejemplo:** acceso a las funciones miembro heredadas

```
public class Superclase
{
      public void metodoPublic(){}
      void metodoDefault() {}
      protected void metodoProtected() {}
      private void metodoPrivate() {}
}
```

Definición de la superclase con métodos de distinto nivel de acceso

```
public class Subclase extends Superclase
{
public void pruebaDeMetodos()
{
   metodoPublic();
   metodoDefault();
   metodoProtected();
   metodoPrivate();
}
}
```

Acceso permitido

Acceso no permitido.
ERROR de compilación "No method matching metodoPrivate() found in Subclase"

Miembros heredados (IV)

- **Si una subclase** tuviera **acceso** a los **miembros privados**, entonces **siempre se podría acceder** a los miembros privados de las clases con **tan solo crear una subclase**, y esta situación **viola** el principio de **encapsulación**.
  Una subclase **puede acceder** a los miembros privados de su superclase **a través de la interfaz de la superclase**, es decir, a través de los métodos que proporciona la superclase para acceso a sus miembros privados.

- La estructura interna de los datos (variables miembro) de un objeto de una subclase se compone de los datos que ella defina y de los datos que herede de su superclase.

- Los miembros heredados por una subclase, son también heredados por sus subclases (**propagación de herencia**).

Miembros heredados (V)

- Si una superclase define **un atributo** `static`, se mantendrá **una única copia** para los objetos de la superclase y todas sus subclases.

Atributos con el mismo nombre (I)

- Una subclase **puede acceder directamente** a los atributos **públicos, protegidos y no modificados** en su acceso (*default* o de pertenencia al mismo paquete) de su superclase.
- Si una **subclase añade un atributo con un nombre que coincide con el nombre** de algún miembro heredado y accesible, entonces el **miembro heredado queda oculto** para la subclase, ya no puede acceder directamente a él (aunque **sí se puede acceder** utilizando la palabra reservada **super**).

- **Ejemplo:**
  - Se define la clase *ClaseA* con un atributo *x* y dos métodos que acceden al atributo.
  - Se define la clase *ClaseB* que hereda de la clase *ClaseA* y que añade un atributo con el mismo nombre (*x*). La clase ClaseB también define un método que accede al atributo.
  - Se crea un objeto de la clase *ClaseB* y se comprueba que el atributo que define con el mismo nombre (x) oculta al atributo heredado.

Atributos con el mismo nombre (II)


```
public class ClaseA
{
    public int x = 1;
    public int obtenerX() { return x; }
    public int obtener10X () { return 10*x;}
}
public class ClaseB extends ClaseA
{
    public int x = 5;
    public int obtenerX() { return x; }
}
public class MainClass
{
    public static void main(String [] args)
    {
        ClaseB objetoClaseB = new ClaseB();
        System.out.println(objetoClaseB.x);
        System.out.println(objetoClaseB.obtenerX());
        System.out.println(objetoClaseB.obtener10X());
    }
}
```

Definición de la superclase *ClaseA*

Definición de la subclase *ClaseB*

Accede al atributo *x* declarado en *ClaseB*

Accede al atributo *x* declarado en *ClaseB*

Accede al atributo *x* declarado en *ClaseA*

Atributos con el mismo nombre (III)

- **Ejemplo:**
  - Se modifica el ejemplo anterior de manera que se utiliza la palabra reservada **super** para que la clase *ClaseB* acceda desde su método *obtenerX* al atributo *x* de su superclase.

```
public class ClaseB extends ClaseA
{
    public int x = 5;
    public int obtenerX() { return super.x; }
}
```

- El **control de acceso de los atributos** que una subclase define con el **mismo nombre que atributos heredados** se puede **modificar** en cualquier sentido (más restrictivo o menos).

Redefinición de métodos (I)

- **Redefinir un método** heredado es **volver a escribirlo** con el **mismo nombre**, los mismos **parámetros** y el mismo tipo de **retorno**.

- Cuando **una subclase redefine un método de una superclase**, se **oculta el método** de la superclase, **pero no las sobrecargas** que pudieran existir de dicho método en la superclase
  (**si** el método se refedine en la subclase con **distinto número** o **tipo de parámetros**, el método de la superclase **no se oculta**, sino que se comporta como una **sobrecarga**).

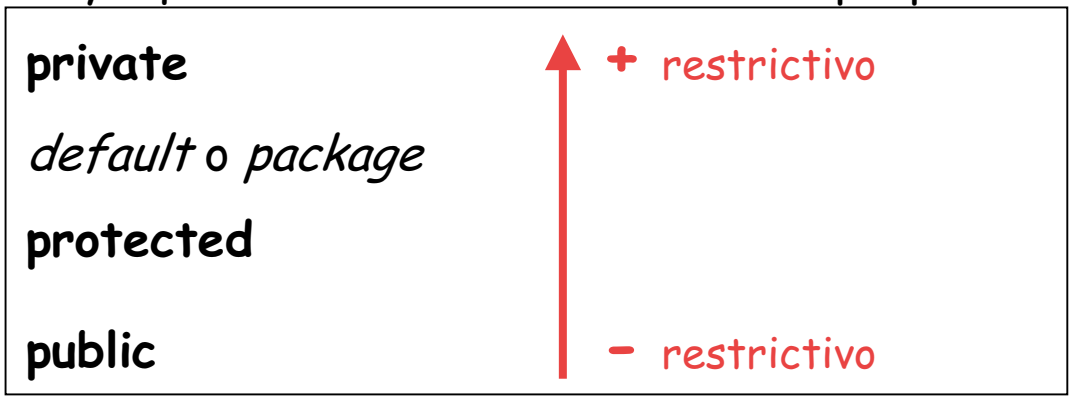
- Si una subclase **hereda un método abstracto**, tiene que **redefinirlo o** bien **ser** también **abstracta**.

- Utilizando la palabra reservada **super** se puede **acceder a un método de la superclase** que ha sido redefinido en la subclase.

Redefinición de métodos (II)

- Una subclase que redefina un método heredado **sólo tiene acceso a su propia versión y a la de su superclase** directa.

- **No se puede redefinir un método** en una **subclase** y hacer que el control de acceso sea **más restrictivo** que en la superclase.
  - Si un método en la superclase es **public**, solo se puede redefinir como **public**.
  - Si un método en la superclase es **protected**, se puede redefinir como **protected** o como **public**.
  - Si un método en la superclase es **private**, no tiene sentido hablar de redefinición ya que solo es accesible desde su propia clase.



Referencias a superclases y subclases (I)

- Una **referencia a un objeto de una superclase** puede **apuntar a objetos de cualquiera de sus subclases**. Lo que ocurre es una conversión implícita de una referencia a un objeto de una subclase en una referencia a su superclase directa o indirecta.

```
Superclase referenciaSuperclase;
referenciaSuperclase = new Subclase();
```

Referencias a superclases y subclases (II)

- Una **referencia a un objeto de una subclase** puede **apuntar** a un **objeto de una superclase** sólo **cuando** el objeto al que se tiene acceso a través de la referencia a la superclase **es un objeto de la subclase**. Además, la **conversión** tiene que ser siempre **explícita** (*cast*).

```
Superclase referenciaSuperclase = new Superclase();
Subclase referenciaSubclase;
referenciaSubclase = referenciaSuperclase;
```

ERROR DE COMPILACIÓN:
"Explicit cast needed to convert Superclase toSubclase"

```
Superclase referenciaSuperclase = new Superclase();
Subclase referenciaSubclase;
referenciaSubclase = (Subclase)referenciaSuperclase;
```

ERROR DE EJECUCIÓN:
Excepción del tipo ClassCastException

```
Superclase referenciaSuperclase = new Subclase();
Subclase referenciaSubclase;
referenciaSubclase = (Subclase)referenciaSuperclase;
```

SIN ERROR

Referencias a superclases y subclases (III)

- Cuando se **accede a un objeto de una subclase** por medio de una **referencia a su superclase**, **sólo** se pueden invocar los **métodos de la superclase**. Es decir, los métodos que se pueden invocar sobre un objeto dependen del tipo de la variable que se utiliza para referenciarlo, y no de la clase del objeto.

- **Ejemplo:**
  - Se define la clase *Superclase* con dos métodos *metodoSup1* y *metodoSup2*.
  - Se define la clase *Subclase* que hereda de la clase *Superclase* y que también define dos métodos *metodoSub1* y *metodoSub2*.
  - Se declara una variable de tipo *Superclase* y se le asigna un nuevo objeto de la clase *Subclase*. Se comprueba que desde la variable de tipo *Superclase* no se puede acceder a los métodos propios de la clase *Subclase*.

Referencias a superclases y subclases (IV)


```
public class Superclase
{
    public void metodoSup1() {}
    public void metodoSup2() {}
}
public class Subclase extends Superclase
{
    public void metodoSub1() {}
    public void metodoSub2() {}
}
public class MainClass
{
    public static void main(String [] args)
    {
        Superclase referenciaSup;
        referenciaSup = new Subclase();
        referenciaSup.metodoSup1();
        referenciaSup.metodoSup2();
        referenciaSup.metodoSub1();
        referenciaSup.metodoSub2();
    }
}
```

Definición de la superclase *Superclase*

Definición de la subclase *Subclase*

Variable utilizada para referenciar

Creación de un objeto d la clase *Subclase*

Acceso permitido por ser métodos de *Superclase*

ERROR de comp.: "Method metodoSub1 not found in Superclase"

ERROR de comp.: "Method metodoSub2 not found in Superclase"

Referencias a superclases y subclases (V)

- Cuando se invoca un método mediante una referencia a la superclase, y que está definido en ella y redefinido en sus subclases, la versión que se ejecuta depende de la clase del objeto referenciado, no de la variable que lo referencia

  = POLIMORFISMO

- **Ejemplo:**
  - Se define la clase *Superclase* con el método *unMetodo.*
  - Se definen las subclases *SubclaseA* y *SubclaseB.*
  - Se redefine el método *unMetodo* en las dos subclases.
  - Se declara una variable de tipo *Superclase* y se le asigna un nuevo objeto de la subclase *SubclaseA*. Se comprueba que si desde esta variable de tipo *Superclase* se invoca al método unMetodo, la versión que se ejecuta es la implementada en la subclase *SubclaseA*.
    Se realiza la misma comprobación con la subclase *SubclaseB*.

Referencias a superclases y subclases (VI)


```
public class Superclase
{
    public String unMetodo() { return "unMetodo de Superclase";}
}
public class SubclaseA extends Superclase
{
   public String unMetodo() { return "unMetodo en SubclaseA";}
}
public class SubclaseB extends Superclase
{
   public String unMetodo() { return "unMetodo en SubclaseB";}
}
public class MainClass
{
    public static void main(String [] args)
    {
        Superclase referenciaSup;
        referenciaSup = new SubclaseA();
        System.out.println(referenciaSup.unMetodo());
        referenciaSup = new SubclaseB();
        System.out.println(referenciaSup.unMetodo());
    }
}
```

Definición de la superclase *Superclase*

Definición de la subclase *SubclaseA* con la redefinición del método *unMetodo*

Definición de la subclase *SubclaseB* con la redefinición del método *unMetodo*

Variable para referenciar los objetos de las subclases

Prueba de la *SubclaseA*

Prueba de la *SubclaseB*