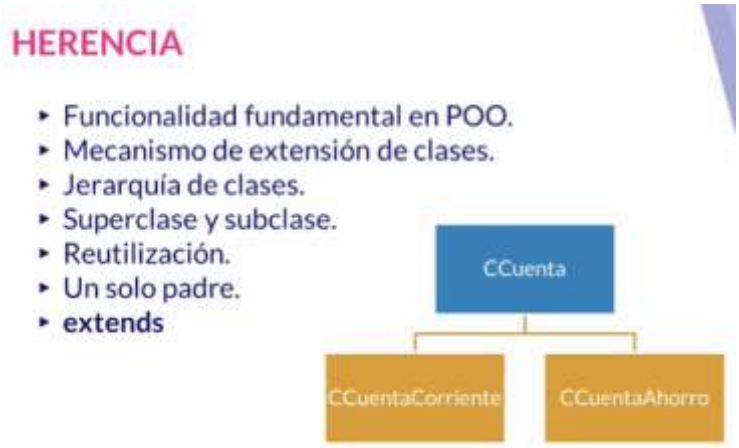
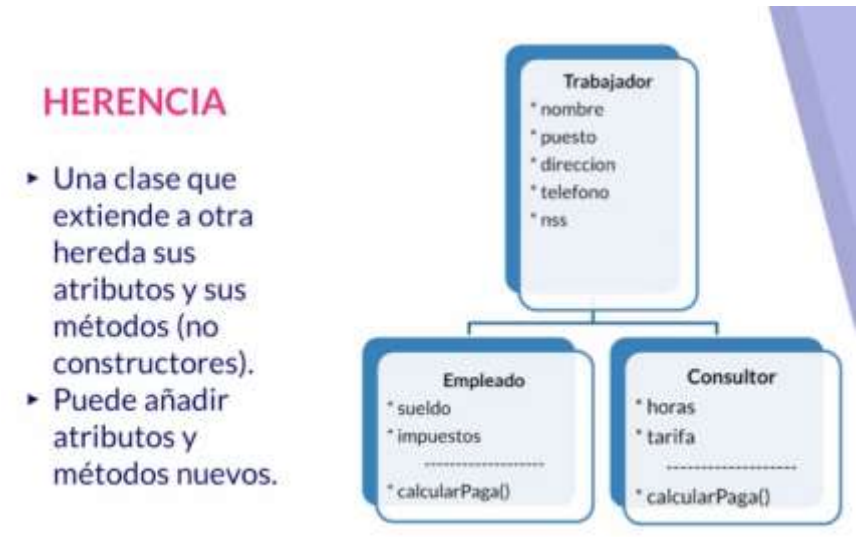
Herencia

Es un mecanismo de cualquier lenguaje orientado a objeto. Nos permite asociar unas clases con otras mediante relaciones de jerarquía (padre e hijo, o superclase y subclase) podéis encontrarlo con varios nombres. En java solo se permite heredar de una clase (una clase solo tiene un padre), no existe la herencia múltiple (si en c++ o paiton), manejarla es muy complejo.



Una clase hereda de otra, es lo mismo que decir que extiende de otra. Se usa la palabra reservada extends. En el ejemplo, una clase tiene más de un hijo, cada hijo solo tiene un padre. Sirve para modelar objetos que se basan en otros objetos y poder reutilizar partes de sus variables y de su código.



Una clase hereda atributos y métodos, nunca el constructor. En el ejemplo:

La clase trabajador tiene sus atributos, esos atributos los hereda Empleado que añade sus atributos y sus métodos. Consultor hereda los atributos de trabajador y tiene los suyos propios (horas, tarifa,…) también sus propios métodos calcularPaga().

Un consultor es un trabajador (clase extendida es a clase base). Un empleado es un trabajador. Una clase que herede de otra solo puede acceder a los métodos y atributos que sean públicos o protegidos.

HERENCIA y ACCESO

- Una clase que extiende a otra hereda todos sus atributos y métodos.
- Solamente puede acceder a los que sean *public* y *protected* (y por defecto si está en el mismo paquete).
- ***protected*** está poco recomendado para las propiedades. Mejor ***private*** y acceso a través de métodos ***public***.

Como hasta ahora atributos private y métodos públicos.

HERENCIA y SOBREESCRITURA

- Una clase que extiende a otra hereda puede añadir tantos métodos o atributos como necesite.
- Si un nuevo atributo (o método) se llama igual que otro de una superclase, lo solapa, y ya no puede accederse al de la clase padre.

Si añadimos atributos o métodos que se llamen igual que en la clase base no podemos acceder a los de la clase base, accedemos a los de la extendida.

HERENCIA y `final`

- Si no queremos que nadie pueda heredar de una de nuestras clases, podemos marcarla como **final** en su definición.

```
public final class ClaseFinal {

}
```

EJEMPLO:

Tenemos la clase trabajador, de la que va a extender luego empleado.

```
public class Trabajador {

    private String nombre;
    private String puesto;
    private String direccion;
    private String telefono;
    private String nSS; //Número Seguridad Social


    public Trabajador(String nombre, String puesto, String direccion, String telefono, String nSS)
        this.nombre = nombre;
        this.puesto = puesto;
        this.direccion = direccion;
        this.telefono = telefono;
        this.nSS = nSS;
    }
```

Observa la clase trabajador, como extiende de empleado tiene los atributos y métodos de empleado (nunca el constructor) y además los suyos propios.

Tiene además sueldo, impuestos y una variable que es una constante, pagas.

```
public class Empleado extends Trabajador {

    private double sueldo;
    private double impuestos;

    private final int PAGAS = 14;
```

Para construir un empleado se construye primero la parte propia del trabajador y después la del empleado (veremos eso más adelante, el uso de super, pero observa cómo se hace):

```
    public Empleado(String nombre, String puesto, String direccion, String telefono, String nSS, doub
        //Profundizamos en "super" en las próximas lecciones
        super(nombre, puesto, direccion, telefono, nSS);
        this.sueldo = sueldo;
        this.impuestos = impuestos;
    }

    public double getSueldo() {
        return sueldo;
    }

    public void setSueldo(double sueldo) {
        this.sueldo = sueldo;
```

Setter and getter de los atributos propios de la clase. Y otros, por ejemplo, calcular paga:

```
    public double calcularPaga() {
        return (sueldo-impuestos) / PAGAS;
    }
```

Otro ejemplo un consultor (trabajador externo a la empresa, también es un trabajador), tiene el mismo padre que el empleado:

```
public class Consultor extends Trabajador {

    private int horas;
    private double tarifa;


    public Consultor(String nombre, String puesto, String direccion, String telefono, Stri
        super(nombre, puesto, direccion, telefono, nSS);
        this.horas = horas;
        this.tarifa = tarifa;
    }
```

Get y set, y además calcular paga:

```
    public double calcularPaga() {
        return horas*tarifa;
    }
```

Al crear las instancias, podemos crear trabajadores, empleados y consultores. Ejemplo:

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

    Trabajador trabajador;
    Empleado empleado;
    Consultor consultor;

    trabajador = new Trabajador("Bill Gates", "Presidente", "Redmond", "", "");
    empleado = new Empleado("Larry Ellison", "Presidente", "Redwood", "", "", 100000.0, 1000.0);
    consultor = new Consultor("Steve Jobs", "Consultor Jefe", "Cupertino", "", "", 20, 1000.0);

    System.out.println(trabajador);
    System.out.println(empleado);
    System.out.println(empleado.calcularPaga());
    System.out.println(consultor);
    System.out.println(consultor.calcularPaga());
```

Cuando generamos el método toString, podemos acceder a los atributos del trabajador a través de los métodos getter, ya que los atributos son private, se accede a sus valores a través de los métodos.

```
@Override
public String toString() {
    return "Empleado [sueldo=" + sueldo + ", impuestos=" + impuestos + ", getNombre()=" + getNombre(
            + ", getPuesto()=" + getPuesto() + ", getDireccion()=" + getDireccion() + ", getTelefono
            + getTelefono() + ", getnSS()=" + getnSS() + "]";
}
```