

UNIDAD 8:

HERENCIA

CLASE I: DEFINICIÓN DE HERENCIA

HERENCIA

Permite que las características pasen de padres a hijos.

PROFESOR

- dni
- nombre
- apellidos
- fecha de nacimiento
- clases que imparte

ALUMNO

- dni
- nombre
- apellidos
- fecha de nacimiento
- clases que atiende

HERENCIA

Permite que las características pasen de padres a hijos.

PROFESOR

- **dni**
- **nombre**
- **apellidos**
- **fecha de nacimiento**
- **clases que imparte**

ALUMNO

- **dni**
- **nombre**
- **apellidos**
- **fecha de nacimiento**
- **clases que atiende**

HERENCIA

PERSONA

- **dni**
- **nombre**
- **apellidos**
- **fecha de nacimiento**

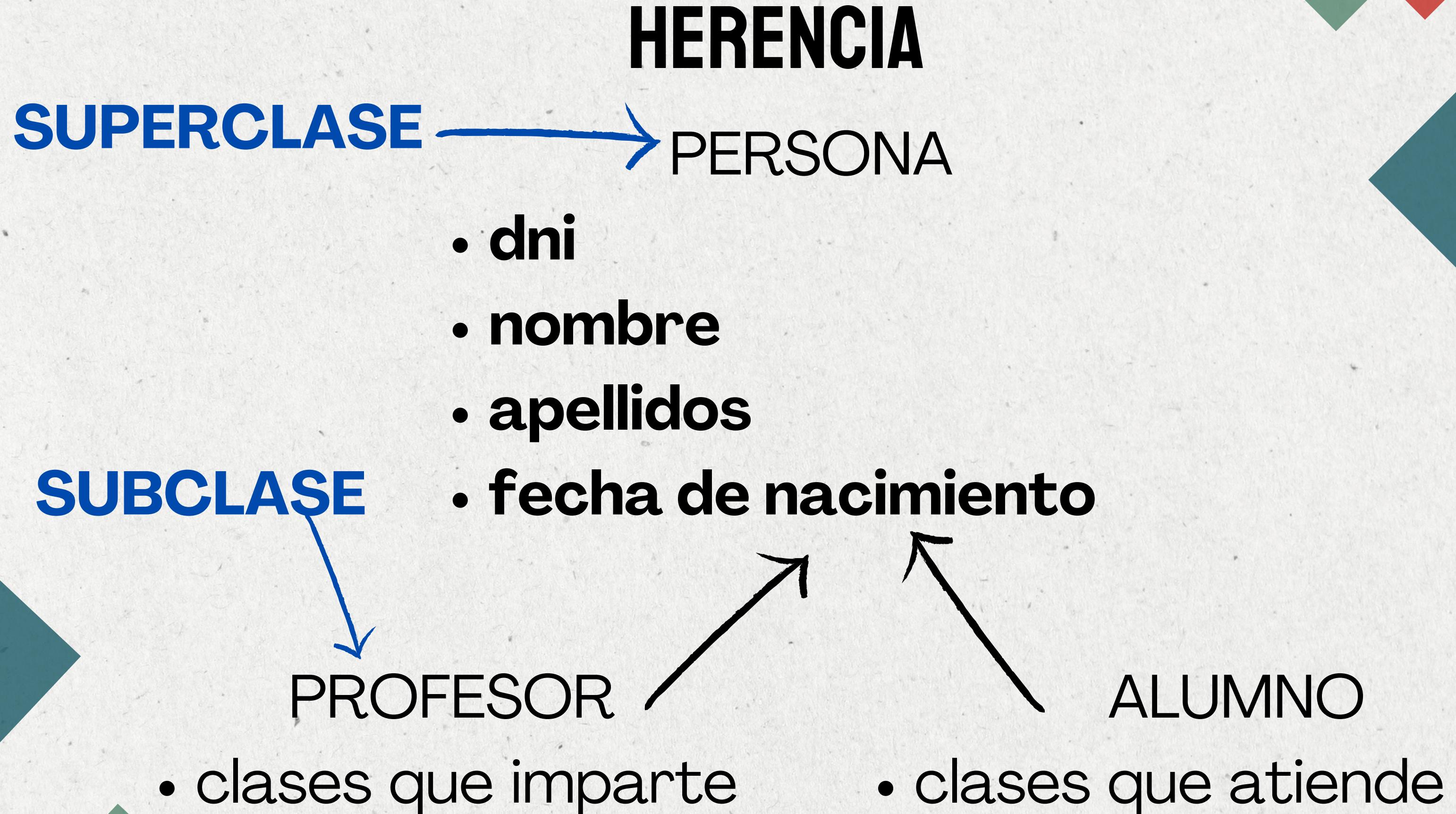
PROFESOR

- clases que imparte

ALUMNO

- clases que atiende





SUBCLASE

PERSONA

- **dni**
- **nombre**
- **apellidos**
- **fecha de nacimiento**



ALUMNO

- **(dni)**
- **(nombre)**
- **(apellidos)**
- **(fecha de nacimiento)**
- **clases que atiende**

SUBCLASE

class OBJECT (java)



class PERSONA



class ALUMNO

CREACIÓN DE UNA SUBCLASE

```
public class Persona {  
    String dni;  
    String nombre;  
    String apellido;  
    String fecha_nac;
```

Persona(dni, nombre, apellido, fecha_nac);

CREACIÓN DE UNA SUBCLASE

```
public class Alumno extends Persona {  
    int clases_atiende;
```

Alumno(dni, nombre, apellido, fecha_nac,
 clases_atiende);

CREACIÓN DE UNA SUBCLASE

```
public class main {  
    public static void main(String[] args) {  
        Alumno a = new Alumno("12345678K", "Sara", "García", "10/09/2004", 6);  
        System.out.println(a.apellido);  
        System.out.println(a.dni);  
    }  
}
```

SUBCLASES

En C++ existe la
herencia múltiple

LIBRETA

- hojas escritas
- hojas disponibles

CALENDARIO

- dia
- mes
- año

AGENDA

- notas

MODIFICADORES DE ACCESO Y HERENCIA

```
public class Persona{  
    private String dni;  
    private String nombre;  
    private String apellido;  
    private String fecha_nac;
```

¿ALUMNO PODRÁ ACCEDER
SIENDO HIJO?

NO

MODIFICADORES DE ACCESO Y HERENCIA

SOLUCIÓN I: ACCESO A TRAVÉS DE GET/SET

Clase Persona

```
public String getDNI() {  
    return dni;  
}
```

Clase Alumno

```
public String mostrarInfo() {  
    return "El dni del alumno es "+getDNI();  
}
```

```
Alumno a = new Alumno("12345678K", "Sara", "García", "10/09/2004", 6);  
System.out.println(a.mostrarInfo());
```

MODIFICADORES DE ACCESO Y HERENCIA

SOLUCIÓN 2: ACCESO PROTEGIDO

Clase Persona

```
public class Persona{  
    protected String dni;  
    protected String nombre;  
    private String apellido;  
    private String fecha nac;
```

Clase Alumno

```
public String mostrarInfo() {  
    return "El dni del alumno es "+dni;  
}
```

PROGRAMACIÓN ORIENTADA A OBJETOS

MODIFICADORES DE ACCESO PARA MIEMBROS

	LA PROPIA CLASE	CLASES VECINAS	CLASES HIJAS	CLASES EXTERNAS
PRIVATE	SI			
SIN MODIFICADOR	SI	SI		
PROTECTED	SI	SI	SI	
PUBLIC	SI	SI	SI	SI

REDEFINICIÓN DE MÉTODOS

Redefinir un atributo - Ocultación

Redefinir un método - Sustitución u overriding

Declarar un miembro igual (atributo o método) en la clase hija como en el padre. El miembro padre queda oculto por el hijo.

REDEFINICIÓN DE MÉTODOS

Clase Persona

```
public String getNombreCompleto() {  
    String cad = "";  
    cad += "Nombre: "+nombre+", ";  
    cad += "Apellido: "+apellido+", ";  
    return cad;  
}
```

Clase Alumno

```
String apellido;  
  
@Override public String getNombreCompleto() {  
    String cad = "";  
    cad += "Nombre: "+nombre+", ";  
    cad += "Apellido: "+apellido+", ";  
    return cad;  
}
```

SUPER Y SUPER ()

this - Referencia a la propia clase que lo ejecuta

super - Referencia a la clase padre desde el código de la clase hija

SUPER Y SUPER ()

Clase Persona

```
public class Persona{  
    protected String dni;  
    protected String nombre;  
    protected String apellido;  
    private String fecha_nac;
```

Clase Alumno

```
public class Alumno extends Persona {  
    String apellido;  
  
    public String mostrarInfo() {  
        return "El dni del alumno es "+dni;  
    }  
    +" y sus apellidos son "+apellido+super.apellido;
```

SUPER Y SUPER ()

Clase Persona

```
public class Persona{  
    protected String dni;  
    protected String nombre;  
    protected String apellido;  
    private String fecha_nac;  
    Persona(String d, String n, String a, String f){  
        dni = d;  
        nombre = n;  
        apellido = a;  
        fecha_nac = f;  
    }  
}
```

SUPER Y SUPER ()

Clase Alumno

```
public class Alumno extends Persona {
    int clases_atiende;
    Alumno(String d, String n, String a, String f,int c){
        super(d,n,a,f);
        clases_atiende = c;
    }
}
```

SUPER Y SUPER ()

Clase Alumno

```
Alumno(String d, String n, String a, String f,int c){  
    clases atiende = c;  
    super(d,n,a,f);  
}
```

EJERCICIOS

Ejercicio 1:

Crea la clase Mueble que tendrá:

- Los atributos identificador, nombre, una cantidad disponible, un precio y un enumerado que tendrá los valores “sin_stock o con_stock”.
- Los métodos get/set para todos los atributos y constructor con parámetros.
 - AumentaCantidad y disminuyeCantidad que reciben una cantidad y la suman o restan a cantidad disponible.
 - mostrarInfo() que será sobrescrito por las clases hijas. En esta versión mostrará: *El mueble **nombre** que está **stock** y vale **precio**.*

Después, crea dos clases hijas con los datos:

- Sofa: Tendrá el atributo sitios que almacena cuantos asientos tiene.
 - Método get/set para su variable y constructor con parámetros.
 - mostrarInfo() mostrará: *El sofá de **sitios** plazas de nombre **nombre**, está **stock** y vale **precio**.*

EJERCICIOS

- Silla: Tendrá el atributo esConjunto que almacena si es parte de un conjunto y cantConjunto que será la cantidad de sillas que forman el conjunto.
 - Método get/set para su variable y constructor con parámetros.
 - mostrarInfo() mostrará si es Conjunto: *El conjunto de **cantConjunto** sillas de nombre **nombre**, está **stock** y vale **precio**.* Si no es conjunto: *La silla de nombre **nombre**, está **stock** y vale **precio**.*
 - disminuyeCantidad() también será sobreescrito para que si es un conjunto sólo disminuya cantidades multiples a la cantidad del conjunto.

ARRAYS CON CLASES Y HERENCIA

```
public class Profesor extends Persona{  
    int clases_imparte;  
    @Override public String getNombreCompleto() {  
        String cad = "";  
        cad += "Nombre: "+nombre+" "+apellido;  
        return cad;  
    }  
  
    Profesor(String d, String n, String a, String f,int c){  
        super(d,n,a,f);  
        clases_imparte = c;  
    }  
}
```

ARRAYS CON CLASES Y HERENCIA

```
public static void main(String[] args) {
    Persona p1 = new Persona("12345678K", "Sara",
        "García", "10/09/2004");
    Persona p2 = new Persona("12345678K", "Pepe",
        "Márquez", "10/09/2004");
    Alumno o1 = new Alumno("12345678K", "Jose",
        "González", "10/09/2004", 6);
    Profesor pr1 = new Profesor("12345678K", "Marta",
        "García", "10/09/2004", 3);
    Persona[] personas = new Persona[4];
    personas[0] = p1;
    personas[1] = p2;
    personas[2] = o1;
    personas[3] = pr1;
    for(Persona p: personas) {
        System.out.println(p.getNombreCompleto());
    }
}
```

ARRAYS CON CLASES Y HERENCIA

Nombre: Sara, Apellido: García,
Nombre: Pepe, Apellido: Márquez,
Nombre: Jose, Apellido: null,
Nombre: Marta García

```
@Override public String getNombreCompleto() { }  
    String cad = "";  
    cad += "Nombre: "+nombre+", ";  
    cad += "Apellido: "+apellido+", ";  
    return cad;  
}
```

```
public String getNombreCompleto() {  
    String cad = "";  
    cad += "Nombre: "+nombre+", ";  
    cad += "Apellido: "+apellido+", ";  
    return cad;
```

```
@Override public String getNombreCompleto() {  
    String cad = "";  
    cad += "Nombre: "+nombre+" "+apellido;  
    return cad;  
}
```

Después crea un main que cree un array de Muebles (podrá almacenar Muebles, Sillas y Sofás) de longitud 5.

Al abrir el programa deberá aparecer un menú que pida al usuario de qué tipo quiere crear el mueble (1. Mueble sencillo, 2. Sofá, 3. Silla) 5 veces para llenar el array. Después crea un nuevo menú que tenga las diferentes opciones:

- Listar muebles (recorrerá el array y mostrará la información de todos los muebles)
- Comprar silla (se deberá escribir por teclado la cantidad a comprar y el nombre de la silla para buscar en el array. Mostrará por teclado “Quedan **getCantidad** sillas de **nombre** en stock”)
- Comprar sofá (se deberá escribir por teclado la cantidad a comprar y el nombre del sofá para buscar en el array. Mostrará por teclado “Quedan **getCantidad** sofás de **nombre** en stock”)

- Comprar mueble (se deberá escribir por teclado la cantidad a comprar y el nombre del muebles para buscar en el array. Mostrará por teclado “Quedan **getCantidad** muebles de **nombre** en stock”)
- Aumentar cantidad (se escribirá por teclado una cantidad que se añadirá a cada uno de los elementos del array)
-