

UNIDAD 8:

HERENCIA

CLASE 2: CLASE OBJETO

HERENCIA

SUPERCLASE

PERSONA

- dni
- nombre
- apellidos
- fecha de nacimiento



SUBCLASE

ALUMNO

- clases que atiende

Alumno a = new Alumno(...);

Persona p = a;

HERENCIA

```
Alumno a = new Alumno(...);
```

```
Persona p = a;
```

a podrá acceder a todos los miembros de Persona y además también a los miembros de Alumno

p podrá acceder **sólo** a los miembros de Persona

HERENCIA

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;
```

Alumno a = new Alumno(...);
Persona p = a;

p.apellido; ?
a.apellido; ?

HERENCIA

Clase Persona

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

↑
p.apellido;

Clase Alumno

```
public class Alumno extends Persona {  
    String apellido;
```

↑
a.apellido;

HERENCIA

Clase Persona

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

Clase Alumno

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

Alumno a = new Alumno(...);

Persona p = a;

p.getNombreCompleto(); ?

a.getNombreCompleto(); ?

HERENCIA

Clase Persona

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

Clase Alumno

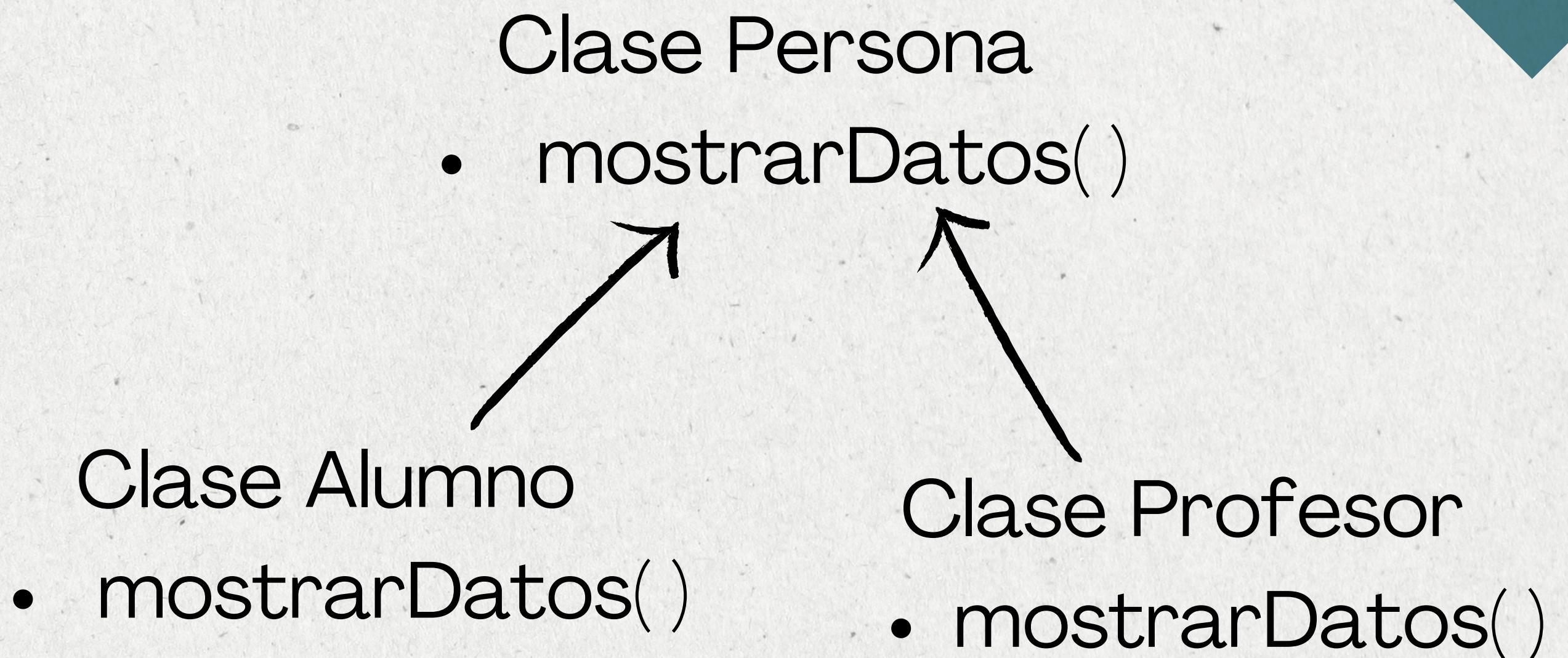
```
@Override public String getNombreCompleto() {  
    String cad = "";  
    cad += "Nombre: "+nombre+", ";  
    cad += "Apellido: "+apellido+", ";  
    return cad;  
}
```

p.getNombreCompleto();
a.getNombreCompleto();



POLIMORFISMO

Selección de métodos en tiempo de ejecución.



POLIMORFISMO

```
Persona p;  
p = new Persona();  
p.mostrarDatos(); //El método de persona  
p = new Alumno();  
p.mostrarDatos(); //El método de alumno  
p = new Profesor();  
p.mostrarDatos(); //El método de profesor
```

CLASE OBJECT

Todas las clases
creadas por el usuario
heredan de Object

Clase Object



Clase Persona



Clase Alumno

CLASE OBJECT

```
public class Persona extends Object{  
    private int edad;  
    double altura;  
    String nombre;  
    double peso;  
    final String dni;
```

CLASE OBJECT

VENTAJAS

- Todas las clases implementan un conjunto de métodos universales.
- Referenciar cualquier clase de Java como una instancia de Object.

TOSTRING()

Devuelve una cadena que representa el objeto con la información que se deseé mostrar.

```
public String toString()
```

TOSTRING()

El método predeterminado de Object devuelve el paquete al que pertenece la clase, el nombre de la misma y después la referencia del objeto.

```
Persona p = new Persona("12345678")
System.out.println(p.toString());
```

Paquete1.Persona@4517d9a3

CLASE OBJECT

TOSTRING()

```
@Override public String toString() {  
    return "DNI: "+dni+"\nNOMBRE: "+nombre+  
           "\nAPELLIDO: "+apellido+"\nFECHA DE NACIMIENTO: "  
           +fecha_nac;  
}
```

DNI: 12345678K
NOMBRE: Sara
APELLIDO: García
FECHA DE NACIMIENTO: 10/09/2004

CLASE OBJECT

TOSTRING()

```
System.out.println(p);
```

```
System.out.println(p.toString());
```

DNI: 12345678K

NOMBRE: Sara

APELLIDO: García

FECHA DE NACIMIENTO: 10/09/2004

CLASE OBJECT

EQUALS()

Object uno == Object otro - No va a funcionar

De esta manera compara sus referencias y no se fija en el contenido de las clases

```
Persona a = new Persona("Laura", 8, 1.20);
Persona b = new Persona("Laura", 8, 1.20);
a == b -> Será falso
```

CLASE OBJECT

EQUALS()

Compara dos objetos y decide si son iguales, devuelve true si lo son y false si no.

public boolean equals (Object otro)

CLASE OBJECT

EQUALS()

```
@Override public boolean equals(Object otro){  
    return true;  
}
```

Override del método
de Object

```
public boolean equals(Persona otro){  
    return true;  
}
```

Nuevo método con el mismo
nombre (sobreescritura)

```
@Override public boolean equals(Persona otro){  
    return true;  
}
```

Erróneo, no recibe el mismo parámetro
que la versión de object por lo que no
es overriding

CLASE OBJECT

EQUALS() - CAST

```
@Override public boolean equals(Object otro){  
    return true;  
}
```



Cast consiste en convertir una variable de un tipo en otro que esté relacionado con él.

(int) char o (char) int // (int) double

CLASE OBJECT

EQUALS() - CAST

```
@Override public boolean equals(Object otro){  
    boolean resultado = false;  
    Persona p = (Persona) otro;  
    return resultado;  
}
```

CLASE OBJECT

EQUALS()

```
@Override public boolean equals(Object otro){  
    Persona p = (Persona) otro;  
    if(this.dni == p.dni && this.fecha_nac.equals(p.fecha_nac)) {  
        return true;  
    }else {  
        return false;  
    }  
}
```

CLASE OBJECT

EQUALS()

```
public class main {  
    public static void main(String[] args) {  
        Persona p1 = new Persona("12345678K", "Sara",  
                                  "García", "10/09/2004");  
        Persona p2 = new Persona("12345678K", "Sara",  
                                  "Marquez", "10/09/2004");  
        System.out.println(p1.equals(p2));  
    }  
}
```

true

CLASE OBJECT

GETCLASS()

Este método puede ser invocado por cualquier objeto y devuelve la clase del elemento que lo invoca.

`Object.getClass();`

`Object.getClass().getName();`

CLASE OBJECT

GETCLASS()

```
Object o1 = "Palabra";
Object o2 = new Persona("12345678K", "Sara",
    "García", "10/09/2004");
Object o3 = 4.3;
System.out.println(o1.getClass());
System.out.println(o2.getClass());
System.out.println(o3.getClass());
```

```
class java.lang.String
class Paquete1.Persona
class java.lang.Double
```

CLASE OBJECT

GETCLASS()

```
Object o1 = "Palabra";
Object o2 = new Persona("12345678K", "Sara",
    "García", "10/09/2004");
Object o3 = 4.3;
System.out.println(o1.getClass().getName());
System.out.println(o2.getClass().getName());
System.out.println(o3.getClass().getName());
```

java.lang.String
Paquete1.Persona
java.lang.Double

GETSUPERCLASS()

```
Object o1 = new Alumno("12345678K", "Sara",
    "García", "10/09/2004", 6);
Class a1 = o1.getClass();
System.out.println(a1.getSuperclass().getName());
```

Paquete1.Persona

CLASE OBJECT

CLONE()

Permite crear una copia del objeto

- No es lo mismo que crear una nueva referencia

Persona p // a = p ! a = p.clone()

- Sus clases si serán las mismas

Persona p // p.getClass() == p.clone().getClass()

- Sus valores de los atributos si que serán iguales

Persona p // p.equals(p.clone()) = true

EJERCICIOS

Ejercicio 1:

Crea la clase **Hora** que tendrá:

- Los atributos **hora** (0 a 23) y **minutos** (0 a 59).
- Los métodos:
 - **Hora(hora,minutos)**: Construye el objeto con los parámetros pasados siempre y cuando estén entre los valores máx y min de cada atributo. Sino, coloca Hora 0 y Minuto 0.
 - **void inc()**: Incrementa la hora un minuto (ten en cuenta los límites)
 - Métodos **get/set** para ambas variables (ten en cuenta los límites)
 - **toString()**: Devuelve un String con la hora en formato: *hora:minuto*

EJERCICIOS

Crea la clase **HoraExacta** que será hija de la clase **Hora**, además de los métodos heredados de **Hora**:

- El atributo **segundo** (0 a 59).
- Los métodos:
 - **HoraExacta(hora,minutos,segundos)**: Construye el objeto con los parámetros pasados siempre y cuando estén entre los valores máx y min de cada atributo. Sino, coloca Segundo a 0.
 - **Sobreescribe void inc()**: Incrementa la hora un segundo (ten en cuenta los límites)
 - Métodos **get/set** para segundo (ten en cuenta los límites)
 - **Sobreescribe toString()**: Devuelve un String con la hora en formato: *hora:minuto:segundos*

EJERCICIOS

- **Sobreescibe equals ()**: Compara la hora pasada por parámetro con el objeto invocante. Devuelve un boolean.

Después crea un objeto clase **Hora** e incrementa su tiempo una hora (mediante un bucle), crea un objeto clase **HoraExacta** y crea una copia de la misma. Al primer objeto aumenta la hora 1 minuto (mediante un bucle) y después compara los objetos Hora Exacta y muestra el resultado del equals por pantalla. Después muestra las 3 horas por pantalla.