

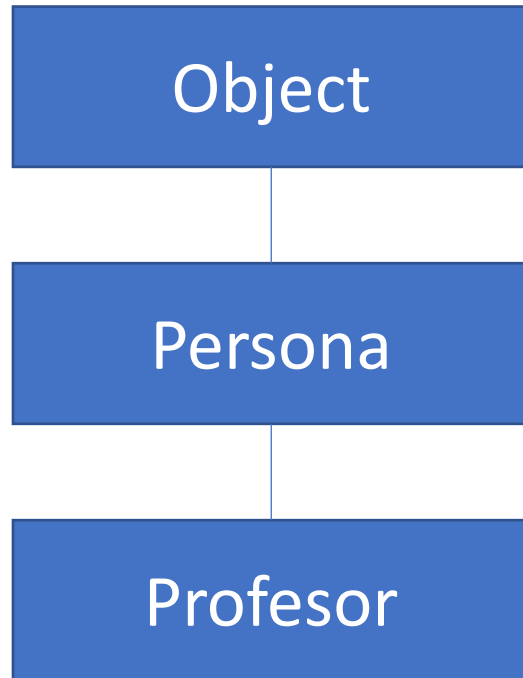
# Herencia y polimorfismo

- 1) Saber derivar de una clase
- 2) Uso del método getClass()
- 3) Uso de super()
- 4) Reemplazamiento de métodos y uso de super
- 5) Modificador protected
- 6) Lanzar eventos en clases derivadas
- 7) Excepciones de usuario internas y externas
- 8) Métodos y clases finales

# Derivar clases

Una clase se puede derivar de otra clase

No es necesario importar Persona ya que están en el mismo paquete



```
package classes;
public class Profesor extends Persona
{
}
```

La clase Profesor hereda todos los atributos y métodos de la clase Persona

Al compilar se lanza un error debido a que no se puede referenciar ningún constructor de la clase Persona (no está definido el constructor por defecto)

# El método “getClass”

El método “getClass” de la clase “Object” aplicado a un objeto devuelve un objeto de tipo “Class” con el tipo de dato de dicho objeto.

La clase “Class” tiene dos métodos, “getName()” y “getSimpleName()”, para obtener el nombre de la clase con y sin su paquete.

```
Persona p = new Persona("PEDRO",34,1.8,67);  
String s1=p.getClass().getName();  
String s2=p.getClass().getSimpleName();  
System.out.println(s1);  
System.out.println(s2);
```



Salida  
consola

clases.Persona  
Persona

# Uso de super()

Se puede definir un constructor en la clase Profesor para inicializar atributos de la clase Persona

```
public Profesor(String nombre, int edad, double altura, double peso)
{
    super(nombre, edad, altura, peso);
}
```

Se invoca el constructor de la clase Persona para inicializar el nombre, la edad, la altura y el peso

- El método super() tiene que ser la primera instrucción
- No se puede utilizar a la vez this() (invocar un constructor de la clase) y super()

# Enumerado LICENCIATURA

En el paquete “clases.enumerados”  
se define el enumerado Licenciatura



```
package clases.enumerados;  
public enum Licenciatura  
{  
    Fisicas,  
    Quimicas,  
    Biologia,  
    Sociologia,  
    Matemáticas,  
    Telecomunicaciones,  
    Industriales,  
    Sicologia,  
    Historia,  
    Derecho,  
    Navales,  
    Aeronautico,  
    Informatica  
}
```

```
package clases;  
import clases.enumerados.Licenciatura;  
public class Profesor extends Persona  
{
```

```
    final public Licenciatura LICENCIATURA;  
    private String centro = "";  
    private int experiencia;
```

← Atributos de la clase Profesor

```
    public Profesor(String nombre, int edad, double altura, double peso)
```

```
    {  
        super(nombre, edad, altura, peso);
```

En el constructor se inicializa el atributo constante

```
        LICENCIATURA = Licenciatura.Fisicas;
```

```
    }  
}
```

# Métodos get y set en la clase Profesor

```
public String getCentro()  
{  
    return centro;  
}
```

```
public Profesor setCentro(String centro)  
{  
    this.centro = centro;  
    return this;  
}
```

Se valida que el centro es una palabra en mayúsculas

```
public int getExperiencia()  
{  
    return experiencia;  
}
```

```
public Profesor setExperiencia(int experiencia)  
{  
    this.experiencia = experiencia;  
    return this;  
}
```

Se valida que el valor de la experiencia sea entre 0 y 39, ambos inclusive

# Constructores en la clase Persona

## Constructor para inicializar el nombre y la licenciatura de un profesor

```
public Profesor(String nombre, Licenciatura licenciatura)
{
    super(nombre);
    LICENCIATURA = licenciatura;
}
```

← En la clase Persona existe un constructor para inicializar el nombre



# Constructores en la clase Persona

**Constructor para inicializar el nombre, la edad, la altura, el peso, el centro, la experiencia y la licenciatura de un profesor**

```
public Profesor(String nombre, int edad, double altura, double peso,  
    String centro, int experiencia, Licenciatura licenciatura)  
{  
    super(nombre, edad, altura, peso);  
    setCentro(centro);  
    setExperiencia(experiencia);  
    LICENCIATURA = licenciatura;  
}
```

Se invoca el constructor de la clase Persona para inicializar el nombre, edad, altura y peso

La clase Profesor se encarga de inicializar los atributos definidos en Profesor y la clase Persona se encarga de inicializar los atributos definidos en Persona

# Reemplazamiento de métodos y uso de super

En la clase Profesor se puede reemplazar el método “toString” del siguiente modo:

```
@Override
public String toString()
{
    return super.toString()+
        "\nCentro: "+getCentro()+
        "\nExperiencia: "+getExperiencia()+
        "\nLicenciatura: "+LICENCIATURA;
}
```

Se invoca el primer método “toString” que haya en la cadena de ascendentes o el de Object si no hubiera ninguno entre los antecesores de Profesor

Si en el método “toString()” de la clase Persona se invocará un método que ha sido reemplazado en Profesor, se ejecutará este último método

# Reemplazamiento del método equals

Dos profesores son iguales si son iguales como personas y tienen la misma experiencia. Un profesor se compara con otro objeto que no sea profesor por el criterio de igualdad definido en Persona

@Override

```
public boolean equals(Object o)
{
    if(o instanceof Profesor)
    {
        Profesor pr = (Profesor) o;
        return super.equals(pr) && getExperiencia()==pr.getExperiencia();
    }
    return super.equals(o);
}
```

# Comparación de profesores

La clase Profesor no tiene que implementar la interfaz Comparable

Se reemplaza el método “compareTo” comparando dos profesores por su experiencia, si el argumento es de tipo “Profesor”; si no, se compararán las dos “Personas” del mismo modo que se hace en la clase “Persona”.

```
@Override
public int compareTo(Persona p)
{
    if(p instanceof Profesor)
    {
        Profesor pr = (Profesor) p;
        if(getExperiencia()<pr.getExperiencia()) return -1;
        if(getExperiencia()>pr.getExperiencia()) return 1;
        return 0;
    }
    return super.compareTo(p);
}
```

# Modificador protected

El modificador “protected” aplicado a un miembro de una clase, permite que dicho miembro sea accesible en sus clases derivadas.

**Al destruirse un objeto de tipo Profesor se mostrará por consola el último valor que tenía el atributo experiencia**

```
protected Finalizador fin = new Finalizador(this);  
static class Finalizador implements Runnable  
{  
    private int edad;  
    protected int experiencia;  
    public void run()  
    {  
        Persona.totalEdad -= edad;  
        System.out.println(experiencia);  
    }  
}
```



**Clase Persona**

**Clase Profesor**



```
public Profesor setExperiencia(int experiencia)  
{  
    this.experiencia = experiencia;  
    fin.experiencia = experiencia;  
    return this;  
}
```

Falta validación

# Lanzar eventos en clases derivadas

Lanzar eventos de cambios en el atributo experiencia

```
public Profesor setExperiencia(int experiencia)
{
    int experienciaAnterior = this.experiencia;
    ...
    this.experiencia = experiencia;
    ...
    getCambios().firePropertyChange("experiencia", experienciaAnterior, this.experiencia);

    return this;
}
```

Se accede al gestor de eventos de cambios con el método público getCambios



# Excepciones de usuario

Una excepción de usuario es una clase que deriva de la clase “RuntimeException”

```
package clases.excepciones;  
public class PesoFueraDeRangoException extends RuntimeException  
{  
    public PesoFueraDeRangoException(String mensaje)  
    {  
        super(mensaje);  
    }  
}
```




# Excepciones de usuario (b)

## Lanzar la excepción `PesoFueraDeRangoException`

```
public Persona setPeso(double peso)
{
    if(peso<=0 || peso>140)
        throw new PesoFueraDeRangoException("El peso tiene que ser
        mayor que 0 y menor o igual que 140");
    this.peso = peso;

    return this;
}
```



Hay que importar en `Persona` la clase  
"`clases.excepciones.PesoFueraDeRnagoException`"



# Excepciones de usuario (c)

Hágase un programa que cree una persona, lea un peso y se lo asigne a dicha persona. Se capturará el posible error y se mostrará el mensaje de error lanzado desde la clase

Salida  
consola



Peso: 156

El peso tiene que ser mayor que 0 y menor o igual que 140

```
Persona p = new Persona("PEDRO",34,1.8,56.7);  
double peso = in.leerDouble("Peso: ");
```

```
try
```

```
{
```

```
    p.setPeso(peso);
```

```
}
```

```
catch(PesoFueraDeRangoException e)
```

```
{
```

```
    System.out.println(e.getMessage());
```

```
}
```

Hay que importar "classes.Persona" y  
"classes.excepciones.PesoFueraDeRangoException"

# Excepciones de usuario internas

Las excepciones de usuario se pueden definir dentro de una clase como una clase estática

## Clase Persona

```
public static class PesoFueraDeRangoException extends RuntimeException
{
    public PesoFueraDeRangoException(String mensaje)
    {
        super(mensaje);
    }
}
```



En la clase Persona se lanza como ya sabemos la excepción  
PesoFueraDeRangoException en el método setPeso sin tener que importarla

# Excepciones de usuario interna (b)



Hágase un programa que cree una persona, lea un peso y se lo asigne a dicha persona. Se capturará el posible error y se mostrará el mensaje de error lanzado desde la clase

Salida  
consola



Peso: 156

El peso tiene que ser mayor que 0 y menor o igual que 140

```
Persona p = new Persona("PEDRO",34,1.8,56.7);  
double peso = in.leerDouble("Peso: ");
```

Sólo hay que importar "classes.Persona"

```
try
```

```
{
```

```
    p.setPeso(peso);
```

```
}
```

```
catch(Persona.PesoFueraDeRangoException e)
```

```
{
```

```
    System.out.println(e.getMessage());
```

```
}
```

La clase se referencia como  
cualquier miembro estático



# Métodos y clases finales

Los métodos finales son métodos que no se pueden reemplazar en clases derivadas. Las clases finales son clases que no se pueden extender

```
final public class Profesor extends Persona
```

```
{
```

```
    @Override
```

```
    final public String toString()
```

```
{
```

```
        return super.toString()+
```

```
        "\ncentro: "+getCentro()+
```

```
        "\nexperiencia: "+getExperiencia()+
```

```
        "\nlicenciatura: "+LICENCIATURA;
```

```
}
```

```
}
```

A partir de la clase Profesor no se pueden derivar otras clases. Se dice que Profesor es una **clase sellada**

El método "toString" no se puede reemplazar en clases que derivan de Profesor



# Polimorfismo

Observa el siguiente código de una aplicación con el método “mostrar”

```
Object objeto = new Object();  
Object persona = new Persona("PEDRO",34,1.8,56.7);  
Object profesor = new Profesor("JUAN",23,1.7,60,"Retamar",3,Licenciatura.Fisicas);
```

Se puede asignar a una variable de tipo Object un objeto que deriva de Object

mostrar(objeto);

mostrar(persona);

mostrar(profesor);

```
public static void mostrar(Object o)  
{  
    System.out.println(o.toString());  
}
```

java.lang.Object@27fa135a

Nombre: PEDRO

Edad: 34

Altura: 1.8

Peso: 56.7

Nombre: JUAN

Edad: 23

Altura: 1.7

Peso: 60.0

centro: Retamar

experiencia: 3

licenciatura: Fisicas

El método “toString” ha sido reemplazado en la clase Persona y Profesor

El método “toString” que se ejecuta depende del tipo de objeto almacenado en “o”

# Herencia en UML

[Derivar clases](#)

