# **Tema 5** Herencia y polimorfismo

## Herencia

**1.** Defínase el siguiente enumerado en el paquete “clases.enumerados”

| PAQUETE: clases.enumerado<br>ENUMERADO: Licenciatura |
|---|
| MIEMBROS |
| Físicas = 0 |
| Químicas = 1 |
| Biología = 2 |
| Sociologia = 3 |
| Matemáticas = 4 |
| Telecomunicaciones = 5 |
| Industriales = 6 |
| Sicologia = 7 |
| Historia = 8 |
| Derecho = 9 |
| Navales = 10 |
| Aeronautico = 11 |
| Informatica = 12 |

**2.** Defínase la clase Profesor.

| PAQUETE: clases<br>CLASE: Profesor<br>EXTIENDE: Persona | | | |
|---|---|---|---|
| ACCESO | TIPO | MIEMBROS | |
| F publico | Licenciatura | LICENCIATURA | |
| privado | String | centro | “” |
| privado | int | experiencia | 0 |
| publico | String | getCentro() | |
| publico | int | getExperiencia() | |
| publico | String | setCentro(String) | |
| publico | int | setExperiencia(int) | |
| publico | | Profesor(String,int,double,double, Licenciatura, String, int) | |
| publico | | Profesor(String) | |
| publico | | Profesor(String, int, Licenciatura) | |
| publico | | Profesor(int, String, Licenciatura) | |
| publico | | Profesor(String, int, double, Licenciatura, String) | |
| REEMPLAZAMIENTO DE METODOS | | | |
| publico | String | toString() | |
| publico | int | compareTo(Persona) | |

| | | | |
|---|---|---|---|
| publico | boolean | equals(Object o) | |

- LICENCIATURA → Constante pública de tipo Licenciatura. Valor por defecto de "Fisicas"
- centro → Atributo de tipo cadena que almacena el centro actual donde trabaja el profesor. Inicializado a "".
- experiencia → Atributo de tipo entero que almacena los años de experiencia. Inicialmente a 0.
- getCentro() → devuelve el valor del atributo "centro".
- getExperiencia() → devuelve el valor del atributo "experiencia"
- setCentro(String) → Establece el valor del atributo centro. Lanzará un error de tipo "IllegalArgumentException" si el centro no fuera la cadena vacía o una palabra en mayúscula.
- setExperiencia(int) → Establece el valor del atributo experiencia. Lanzará un error de tipo "IllegalArgumentException" si la experiencia no fuera un valor entre 0 y 39, ambos inclusive.
- Profesor(String,int,double, double, Licenciatura, String, int) → Constructor para inicializar el nombre, la edad, la altura, el peso, la licenciatura, el centro y la experiencia de un profesor.
- Profesor(String) → Constructor para inicializar el nombre de un profesor.
- Profesor(String, Licenciatura) → Constructor para inicializar el nombre y la licenciatura de un profesor.
- Profesor(String, int, Licenciatura) → Constructor para inicializar el nombre, la edad y la licenciatura de un profesor
- Profesor(int, String, Licenciatura) → Constructor para inicializar la edad, el nombre y la licenciatura de un profesor.
- Profesor(String, int, double, Licenciatura, String) → Constructor para inicializar la edad, el nombre, la edad, el peso y la licenciatura de un profesor.
- toString() → Reemplaza dicho método devolviendo la cadena de la clase padre, a la cual se le añaden los valores actuales de los atributos de la clase "Porfesor" con el mismo formato que el de la clase padre.
- compareTo(Persona) → Reemplaza el método 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".
- equals() → Dos profesores son iguales si son iguales con respecto a la clase padre y tienen la misma experiencia.

Hágase una aplicación en la que se introducen dos profesores y una persona (nombre, edad, altura, peso, centro, experiencia y licenciatura). Compárese y dígase si son o no iguales los dos profesores, y cada uno de los profesores con la persona. ¿Es posible comparar la persona con cada profesor? ¿Por qué? (ProfesorAplicacion)

**3.** Un cuadrado es un tipo de "Rectangulo". Derívese la clase "Cuadrado" de "Rectángulo" con las siguientes características y métodos:

a) girado → Atributo booleano que indica si está girado o no 45º. Inicialmente vale 0.
b) getGirado() → Devuelve el valor del atributo "girado".
c) getLado() → Devuelve el valor del lado (ancho o alto)
d) setGirado(boolean) → Establece el valor del atributo "girado"
e) setLado(int) → Establece el valor del lado (ancho o alto) al valor pasado como argumento.
f) Cuadrado(int) → Cosntaructor para establecer el lado del cuadrado.
g) Cuadrado(int, boolean) → Constructor para establecer el lado del cuadrado y si está girado o no.
h) Cuadrado(int, char, char) → Cosntaructor para establecer el lado del cuadrado, el borde y el relleno.
i) Cuadrado(Cuadrado) → Constructor de copia.
j) setAncho(nt) → Reemplazar dicho método para que al modificar el ancho, al alto se le asigne el mismo valor de ser distinto.
k) setAlto(int) → Reemplazar dicho método para que al modificar el alto, al ancho se le asigne el mismo valor de ser distinto.
l) dibujar() → Reemplazar el método del siguiente modo: Si el cuadrado no está girado, se invoca el método dibujar de la clase padre; si está girado, se dibuja en forma de rombo.
m) dimension() → Se reemplaza mostrando el mensaje "lado = " con el valor del lado.
n) toString() → Se reemplaza añadiendo a la cadena obtenida por la clase padre, los atributos de la clase "Cuadrado" (lado y girado) con el mismo formato.
o) compareTo(Rectangulo) → Se reemplaza comparando dos cuadrados por su lado si el argumento es de tipo "Cuadrado"; si no es de tipo "Cuadrado" se comparan utilizando el método "compareTo" de la clase "Rectangulo".

Hágase una aplicación que realice lo siguiente:

a) Inicialice un cuadrado de lado 5.
b) Establezca como relleno '+'.
c) Muestre su dimensión.
d) Dibuje el cuadrado anterior.
e) Aumente en 1 unidad el ancho de cuadrado.
f) Muestre su dimensión.
g) Vuelva a dibujar el cuadrado.
h) Establézcase el valor del lado a 8.
i) Gire el cuadrado.
j) Vuelva a dibujar el cuadrado.

(CuadradoAplicacion)

**4.** Una caja es un rectángulo que tiene una medida más: el largo. Defínase la clase "Caja" que deriva de "Rectangulo" con las siguientes características y métodos:

a) UNO → Constante estática de tipo Caja que almacena la caja de 1 x 1 x 1.
b) DIEZ → Constante estática de tipo Caja que almacena la caja de 10 x 10 x 10
c) NUMERO_DE_CARAS → Constantes estática de tipo entera inicializada a 12.
d) LARGO_INICIAL → Constante de objeto de tipo entero
e) largo → Atributo de tipo entero inicializado a 2.
f) getLargo() → Devuelve el valor del atributo largo.
g) setLargo(int) → Establece el valor del atributo "largo" al valor pasado al método. Se lanzará una excepción del tipo "IllegalArgumentException" si el valor pasado fuera inferior a 2.
h) Caja() → Cosntructor por defecto.
i) Caja(int) → Constructor para inicializar el atributo "largo"
j) Caja(int,int,int) → Constructor para inicializar el atributo "ancho", "alto" y "largo"
k) Caja(int,int,int, char, char) → Constructor para inicializar el atributo "ancho", "alto", "largo", "borde" y "relleno"
l) Caja(Caja) → Constructor de copia
m) volumen() → Obtiene el volumen de la caja.
n) esCubo() → Devuelve verdadero si es un cuadrado (la base) y el largo coincide con el ancho.
o) sumar(Caja,Caja) → Constructor estático que obtiene una caja cuyo ancho, alto y largo son la suma de los anchos, altos y largos de las clases dadas.
p) plus() → Se reemplaza aumentado uno tanto el ancho, alto y largo, llamando al método de la clase padre si fuera preciso.
q) dimension() → Se reemplaza para mostrar la cadena en formato "ancho x alto x largo", llamando al método de la clase padre si fuera preciso.
r) dibujar() → Se reemplaza dibujando la base (se llama al método de la clase padre) con un mensaje de "BASES"; la cara lateral, con un mensaje previo de "CARA LATERAL" y la cara frontal, con un mensaje previo de "CARA FRONTAL".
s) restaurar() → Se reemplaza para restuarar la caja al ancho, alto y largo inicial, llamando al método de la clase padre si fuera preciso.
t) toString() → Se reemplaza añadiendo a la cadena obtenida por la clase padre, los atributos de la clase "Caja" (largo) con el mismo formato.
u) compareTo(Rectangulo) → Se reemplaza comparando dos cajas por su volumen si el argumento es de tipo "Caja"; si no es de tipo "Caja", se comparan utilizando el método "compareTo" de la clase "Rectangulo".
v) equals(Object) → Reemplacese de forma que dos cajas son iguales si tiene el mismo volumen si el argumento es de tipo "Caja"; si no es de tipo "Caja", pero es de tipo "Rectángulo" se comparan utilizando el método "equals" de

la clase "Rectangulo" comparando por el ancho y alto; si no es de tipo "Caja" ni "Rectángulo", se lanza un mensaje del tipo "IllegalArgumentException". Si el argumento es de tipo "null", se lanza la excepción "NullPointerException".

Hágase una aplicación que realice lo siguiente:

a) Inicialice una caja de 4 x 5 x 3.
b) Compruébese si es un cubo.
c) Compárese con la caja DIEZ.
d) Compruébese si es igual a la caja UNO.
e) Modifique su ancho en 2 unidades más, el alto en una más y el largo en 2 más.
f) Muéstrese la dimensión de la caja anterior.
g) Se establezca las dimensiones de la caja anterior a sus dimensiones iniciales, y muéstrese su dimensión.
h) Obténgase la suma de las cajas UNO, DIEZ y la inicial, y muéstrese su dimensión.

(CajaAplicacion)

**Lanzar eventos en clases derivadas**

**5.** Láncese eventos de cambio y cambiándose para cada uno de los atributos de las clase "Profesor", "Cuadrado" y "Caja"

**Excepciones de usuario internas**

**6.** Defínase en la clase "Rectangulo" las siguientes excepciones de usuario:
   a) FueraDeRangoException→ Se lanzará cuando un parámetro esté fuera de un determinado rango de valores
   b) Láncese dicha excepción en las clases "Rectangulo" y "Caja" cuando proceda.

Hágase una aplicación en la que el usuario lea un rectángulo (no se utilizarán los métodos de lectura definidos en dicha clase). Si alguno de los datos de la anchura, altura, borde y/o relleno es erróneo, entonces se mostrará un mensaje indicándolo, y se le pedirá otra vez la introducción de los datos. Una vez se haya introducido los datos correctamente, se dibujará el rectángulo. (RectanguloExcepciones)

**7.** Defínase en la clase "Persona" la siguiente excepción de usuario:
   a) NombreInvalidoException → Se lanzará cuando el nombre sea inválido.
   b) EdadFueraDeRangoException→ Se lanzará cuando la edad esté fuera de un rango.

c) AlturaFueraDeRangoException→ Se lanzará cuando la altura esté fuera de un rango.
d) PesoFueraDeRangoException→ Se lanzará cuando el peso esté fuera de un rango.
e) En una clase "Excepciones" en el paquete "clases.util" se definirá las siguientes excepciónes:
"ValorNegativoException" → Se lanzará cuando un valor sea negativo.
"NúmeroInvalidoDeDecimalesException → Se lanzará cuando el valor tiene más decimales que los precisados.
f) Láncese dicha excepción en la clase "Persona" cuando proceda.

Hágase una aplicación en la que el usuario lea una persona (no se utilizarán los métodos de lectura definidos en dicha clase). Si alguno de los datos del nombre, edad, altura y/o peso es erróneo, entonces se mostrará un mensaje indicándolo, y se le pedirá otra vez la introducción de los datos. Una vez se haya introducido los datos correctamente, se mostrará el objeto. (PersonaExcepciones)

**Métodos y clases finales**

**8.** Defínase en la clase "Rectangulo" los métodos "area", "perímetro" y "esCuadrado" como finales. (RectanguloFinal)

**9.** Defínase en la clase "Caja" los métodos "dibujar", "toString", "equals" y "sumar" como finales. (CajaFinal)

**10.** Defínase la clase "Cuadrado" como final. (CuadradoFinal)

**Polimorfismo**

**11.** Hágase una aplicación en la que se realice lo siguiente:
a) Defina un array de objetos de tamaño 5.
b) La primera componente tendrá un rectángulo de 3 x 7
c) La segunda componente tendrá un cuadrado de lado 5
d) La tercera componente tendrá un cuadrado de lado 4
e) La cuarta componente tendrá una caja de 4 x 2 x 9
f) La quinta componente tendrá un rectángulo por defecto.
g) Recórrase el array, y para cada componente se mostrará de qué clase es (con su paquete) y su dimensión.

(Polimorfismo)