METODOLOGÍA DE LA PROGRAMACIÓN

Guión de prácticas de laboratorio

Profesores: Jorge Puente y Ramiro Varela

TEMA 1. Programación Orientada a Objetos. HERENCIA

Material para dejar en el Campus Virtual: 6 ficheros zip con otros tantos proyectos que contienen las siguientes clases con sus correspondientes ejemplos de prueba

- **`00_Bicycle.zip`**: clase **`Bicycle`**
- **`Fig_9_5_CommissionEmployee.zip`**: clase **`CommisionEmployee`**
- **`Fig_9_6_BasePlusCommissionEmployee.zip`**: clase **`BasePlusComissionEmployee`**
- **`Fig_9_8_BasePlusCommissionEmployee.zip`**: Clases **`CommisionEmployee`** y **`BasePlusComissionEmployee`** definidas mediante herencia
- **`CirculosGUI.zip`**: clase **`MiVentana`** que se utilizará para representar círculos en modo gráfico. Incluye también la clase auxiliar **`Dado`** que permite generar números aleatorios
- **`Jeroquest_1.0.zip`**: Jerarquía de clases **`Personaje, Barbaro`** y **`Momia`**

PRACTICA 1.1. Uso de clases simples

Objetivo: Familiarizarse con las clases básicas **`Bicycle, CommisionEmployee`** y **`BasePlusComissionEmployee,`** y hacer algunas modificaciones en clases que ya están programadas**.**

Ejercicio 1.1. Uso y modificación de la clase **`Bicycle`**

a) Crear un proyecto a partir del fichero 00_Bicycle 1.0.zip
b) Modificar la clase **`Bicycle`** para que cumpla los requisitos del principio de encapsulación. Es decir, que cada dato tenga un observador y un único modificador y que el resto de métodos accedan a los datos a través de estos observadores y modificadores.
c) Añadir operaciones al ejemplo de uso. Por ejemplo, crear otra bicicleta con la misma **`gear`** que **`bike1`** y con el doble de **`speed`** de **`bike2`**.
d) Añadir un constructor de copia y poner ejemplos de uso.
e) Implementar el método **`toString()`** en la clase **`Bicycle`** y utilizarlo para mostrar objetos en modo texto.

Ejercicio 1.2. Añadir operaciones a los ejemplos de uso de **`CommissionEmployee`**

a) Crear un proyecto a partir del fichero Fig_9_5_CommissionEmployee.zip y ejecutar el ejemplo de prueba que incluye.
b) Crear un vector de trabajadores de la clase **`CommissionEmployee`**, y mostrar los datos del trabajador que más gana.
c) Un poco más complicado que el anterior. Mostrar por consola todos los trabajadores, pero ordenados por su salario de menor a mayor.

Ejercicio 1.3. Lo mismo que el ejercicio anterior, pero con la clase **`BasePlusComissionEmployee`**, es decir

a) Crear un proyecto a partir del fichero Fig_9_6_BasePLusCommissionEmployee.zip y ejecutar el ejemplo de prueba que incluye.
b) Crear un vector de trabajadores de la clase **`BasePlusCommissionEmployee`**, y mostrar los datos del trabajador que más gana.
c) Mostrar por consola todos los trabajadores, pero ordenados por su salario de menor a mayor.

PRACTICA 1.2. Creación de proyectos que utilizan más de una clase

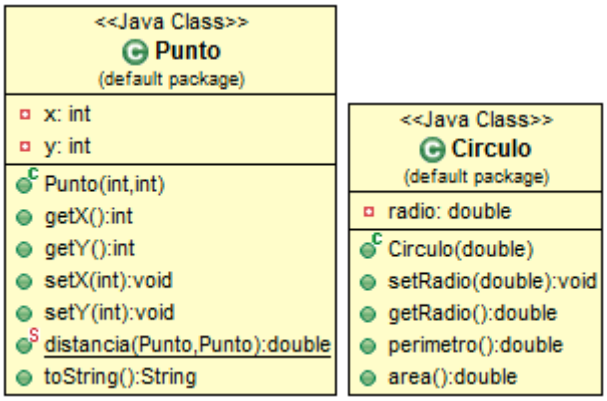
Objetivo: Saber crear proyectos con más de una clase, y programar algunas clases sencillas.

Ejercicio 1.2.1. Crear un proyecto que utilice las dos clases **`CommissionEmployee`** y **`BasePlusCommissionEmployee`**

a) Crea un proyecto nuevo que contenga a las dos clases anteriores, así como un programa de prueba.
b) Crea dos vectores de trabajadores, uno de cada clase, y calcular (y mostrar por consola) el trabajador que más gana (indicando a qué clase pertenece).
c) Un poco más complicado que el anterior. Mostrar por consola todos los trabajadores de los dos vectores, pero ordenados por su salario de menor a mayor. Así, los trabajadores de las dos clases pueden aparecer intercalados. Este ejercicio requiere ordenar los dos vectores y luego recorrerlos simultáneamente para ir mostrando los trabajadores ordenados. Es un algoritmo de "mezcla" de dos estructuras ordenadas.

Ejercicio 1.2.2. Programar dos clases simples **`Circulo`** y **`Punto`** (iguales o similares a las que se utilizaron en Introducción a la Programación)

a) Crea un proyecto que contenga dos clases: **`Circulo`** y **`Punto`,** y un programa que utilice las dos clases, con algunos ejemplos de uso.
b) La clase **`Circulo`** tiene como único dato el **`radio`** y debe tener métodos para ver y modificar el radio, calcular la superficie, así como los constructores apropiados: al menos un constructor a partir de un entero y el constructor de copia.
c) La clase **`Punto`** tiene dos datos **`x`** e **`y`** que representan las coordenadas en el plano. Debe incluir un constructor a partir de dos enteros y los métodos observadores y modificadores apropiados, así como un método para calcular la distancia de un punto a otro punto.



PRACTICA 1.3. Relaciones de composición

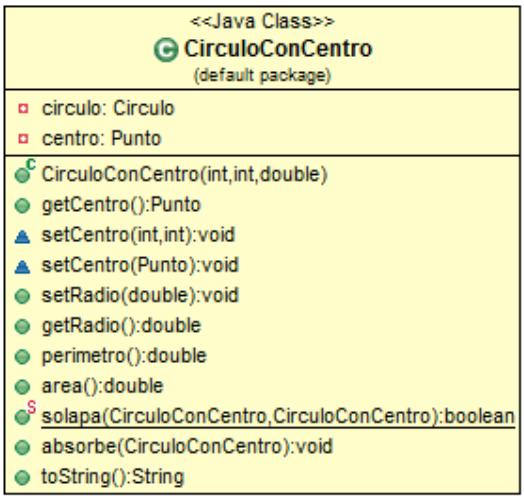
Objetivo: Entender bien la relación de composición para distinguirla después de la relación de herencia.

Ejercicio 1.3.1. Definir la clase **`BasePlusCommissionEmployee`** por composición de **`CommissionEmployee`**

a) Define la clase **`BasePlusCommissionEmployee`**, con un dato de tipo **`CommissionEmployee`**, y definir adecuadamente los métodos para que la clase mantenga la misma funcionalidad que en las implementaciones anteriores.
b) Comprueba que el resultado de los ejemplos de uso de la práctica anterior funcionan y producen los mismos resultados.
c) Comenta las ventajas y/o desventajas que tienen las dos formas de definir las clases **`BasePlusCommissionEmployee`** y **`CommissionEmployee`** , es decir definirlas de forma independiente, o bien utilizando la relación de composición.

Ejercicio 1.3.2. Definir la clase **`CirculoConCentro`** a partir de las clases **`Circulo`** y **`Punto`** mediante relación de composición.

a) Define la clase **`CirculoConCentro`** haciendo la interpretación de que un circulo con centro "tiene un" círculo y "tiene un" centro. Incluir los métodos observadores y modificadores y los constructores apropiados.



b) Incluye, en las clases apropiadas, métodos para modificar la posición del centro, para aumentar o disminuir el radio, para calcular el perímetro y el área de un círculo y para indicar si un círculo se solapa con otro o no.

PRACTICA 1.4. Relaciones de HERENCIA

Objetivo: Entender bien los conceptos relacionados con la herencia, y distinguir la herencia de la composición

Ejercicio 1.4.1. Uso y modificación de las clases **`BasePlusCommissionEmployee`** y **`CommissionEmployee`** definidas mediante herencia. Cada vez que modifiques un método, coloca la implementación anterior en un comentario para poder compararla luego con la nueva implementación.

a) Crea un proyecto a partir del fichero Fig_9_8_BasePlusCommissionEmployee.zip y comprueba que los programas de prueba de las clases **`BasePlusCommissionEmployee`** y **`CommissionEmployee`** de las prácticas anteriores funcionan igual cuando estas clases están definidas a través de la relación de herencia.
b) Modifica el constructor de la subclase **`BasePlusCommissionEmployee`** de modo que no utilice la llamada **`super(first,last,ssn,sales,rate)`** para invocar al constructor de la superclase. Para ello tiene que asignar valores a los 5 datos definidos en la superclase de la forma apropiada.
c) En la implementación del método **`earnings`** de la subclase cambia las llamadas a los métodos observadores **`getCommissionRate()`** y **`getGrossSales()`** por los nombres de los campos de datos correspondientes. Observa el problema que se produce. Cambia el calificador **`private`** por **`protected`** en los campos de la superclase y observa que en todo funciona otra vez.
d) Modifica la implementación del método **`earnings`** de la subclase de modo que se utilice **`super`** para invocar al método de la superclase.
e) Toma nota de las tres formas de implementar el método **`earnings`** en la subclase (la original, la del apartado c) y la del apartado d)) y escribe un pequeño comentario sobre las ventajas y/o inconvenientes de cada uno de ellos, y cuál te parece la forma más razonable de implementarlo.

Ejercicio 1.4.2. Define la clase **`CirculoConCentro`** a partir de las clases **`Circulo`** y **`Punto`** utilizando relaciones de herencia.

a) Define la clase **`CirculoConCentro`** considerando que un círculo con centro "es un" círculo que "tiene un" centro. La clase debe incluir la misma funcionalidad que la definida en la práctica anterior (Ejercicio 3.2). Comprueba que el programa de prueba de la práctica anterior funciona correctamente.

PRACTICA 1.5. Uso de clases predefinidas

Objetivo: Saber crear una estructura de clases compleja utilizando la funcionalidad de clases ya definidas, aunque los detalles de implementación de estas clases no se conozcan bien, como es el caso de la interfaz gráfica para la clase **`CirculoConCentro`**

Ejercicio 1.5.1. Crear un proyecto a partir de Circuloszip añadiendo las clases que representan a los círculos y comprueba cómo funciona la representación gráfica de los círculos.

a) Crea un nuevo proyecto a partir del fichero CirculosGUI.zip. Este proyecto contiene el fichero MiVentana que implementa varias clases que permiten mostrar en modo gráfico un vector de círculos con centro. Para ello solamente hay que añadir al proyecto la clases **`CirculoConCentro`**, **`Circulo`** y **`Centro`**. La estructura de clases del proyecto se muestra en la figura de la página siguiente.
b) Ejecuta el programa de prueba y observa lo que ocurre. Haz también algún cambio en el programa de prueba, por ejemplo haz que el programa muestre otro conjunto de círculos después de pulsar enter y que este proceso se repita un número (por ejemplo 10) de veces.

Ejercicio 1.5.2. Añadir operaciones al proyecto

a) Crea un nuevo método en la clase CirculosTest llamado **`mueveCirculos`**. Este método recibirá como argumentos un vector de círculos con centro y un entero positivo k. El método modificará aleatoriamente cada uno de los círculos de modo que para cada uno de ellos se modificará en una cantidad aleatoria comprendida en [0..k] la componente x del centro, o bien la componente y del centro, o bien el radio. La modificación será positiva o negativa con probabilidad ½ (obviamente los valores de estas componentes y el radio serán siempre positivos). Para la generación de números aleatorios debes utilizar la clase **`Dado`** que está incluida en CirculosGUI.zip
b) Haz que desde el método **`main()`** se llame de forma iterativa al método **`mueveCirculos`** de modo que cada vez que se pulsa una tecla se haga una nueva llamada y se visualice el estado resultante de los círculos. ¿Cuándo terminará de ejecutarse este programa?

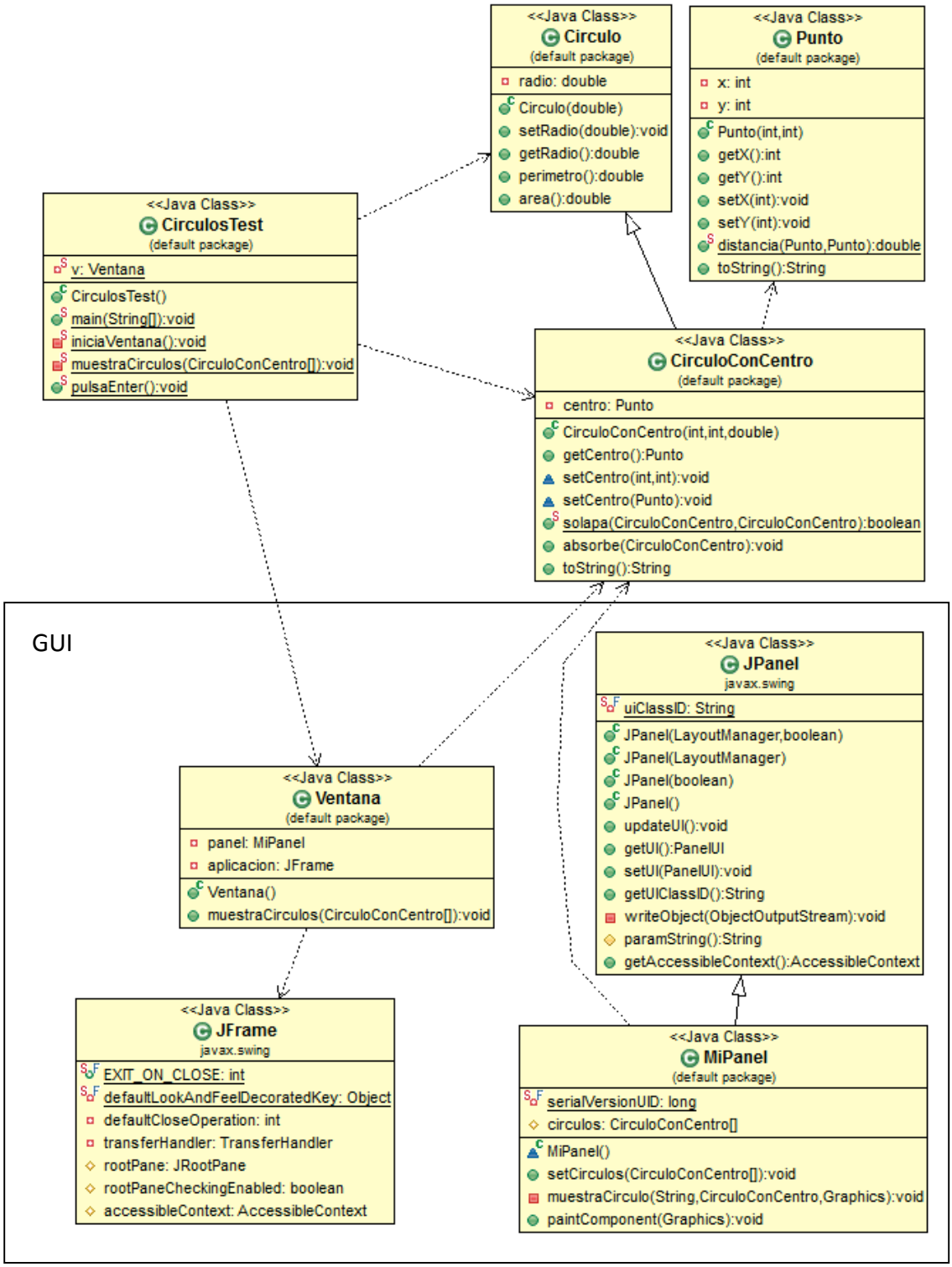


Diagrama de clases del proyecto CirculosGUI (no incluye los métodos de la clase **`JFrame`**). Las flechas muestran dos tipos de relaciones: a) Relaciones de Herencia (flechas en línea continua), y b) Relaciones de dependencia (las discontinuas). Las relaciones de dependencia significan que una clase hace uso de la otra clase, por ejemplo definiendo un dato o utilizando un método de la otra clase. Observad que las relaciones de dependencia incluyen a las de composición.

PRACTICA 1.6. Diseño de algunos algoritmos con círculos con centro

Objetivo: Ser capaz de diseñar algunos algoritmos con composiciones iterativas y condicionales que manipulan círculos y vectores de círculos.

Ejercicio 1.6.1. Diseñar los algoritmos que se proponen a continuación e implementalos con los métodos adecuados.

a) Añade a la clase **`CirculoConCentro`** una operación para que un círculo absorba a otro. La forma de utilizar esta operación será **`c1.absorbe(c2)`** y el resultado será que **`c2`** se anula, de forma que su radio vale 0 y su centro (0,0), y **`c1`** absorbe a **`c2`** de modo que a la superficie de **`c1`** se le suma la de **`c2`**, y el centro de **`c1`** pasa a ser un punto situado en el eje que une a los centros originales de **`c1`** y de **`c2`** y que está a una distancia de cada uno que es inversamente proporcional a la superficie original de cada círculo.
b) Ahora se trata de hacer un algoritmo para procesar un vector de círculos de la siguiente forma: define un vector de círculos y luego haz un recorrido del vector y con cada círculo **`c1`** no nulo comprueba si se solapa o no con el siguiente circulo **`c2`** no nulo en el vector (un vector es nulo si su radio es 0 y su centro (0,0)). Si se solapan entonces haz que el círculo **`c1`** absorba al círculo **`c2`**. Haz que se repita este recorrido desde el principio del vector tantas veces como sea preciso hasta que ninguno de los círculos no nulos que quedan se solape con otro. Después de cada iteración muestra el resultado.
c) Dados dos círculos con centro **`c1`** y **`c2`**, se trata de calcular la superficie de la intersección de los dos círculos. El cálculo exacto no es simple y se propone el siguiente método para calcular esta superficie de forma aproximada: si no se solapan, entonces la superficie de la intersección es 0, si se solapan entonces, para estimar la superficie de la intersección, se generan un total de N de puntos de forma que estén uniformemente distribuidos en la superficie de uno de los círculos, digamos **`c1`**, contamos cuántos de estos N puntos están también en el interior de **`c2`**, y a partir de este número de puntos podemos estimar la superficie de la intersección de los dos círculos con centro.
   Diseña e implementa las operaciones auxiliares que consideres que te facilitarán la resolución del problema, por ejemplo: generar un punto al azar dentro de un círculo, saber si un punto está dentro de un círculo, etc. La utilización de estas operaciones simplificará la definición y comprensión de la solución.

PRACTICA 1.7. Ejercicios con la jerarquía básica del Jeroquest : `Personaje, Barbaro y Momia`

Objetivo: Entender y saber utilizar una jerarquía de clases no simple como la de personajes del Jeroquest.

Ejercicio 7.1. Añadir operaciones al programa de prueba de la jerarquía de clases **`Personaje, Barbaro`** `y` **`Momia`**

a) Crea un proyecto a partir del fichero Jeroquest_1.0.zip
b) Crea en el proyecto el fichero **`clasesJeroquest.ucls`** y muesta la jerarquía compuesta por las tres clases del proyecto
c) Define un bárbaro y una momia y hacer que uno ataque al otro un número máximo de veces o hasta que se muera (el otro).
d) Define un vector de bárbaros y otro de momias, los dos del mismo tamaño, y haz que alternativamente un bárbaro ataque a una momia y viceversa, siguiendo el protocolo que se indica a continuación: se realiza un número entero N de turnos, en cada turno todos los bárbaros y todas las momias atacan en orden desde el primero al último; cuando ataca un bárbaro *i* elige a una momia *j* aleatoriamente; si el bárbaro *i* o la momia *j* no están vivos, entonces no hay ataque, en otro caso el bárbaro *i* ataca a la momia *j*. Análogamente cuando ataca la momia *i*. Es decir, primero ataca el bárbaro 0 a una momia aleatoria, luego ataca la momia 0 a un bárbaro aleatorio, luego ataca el bárbaro 1, y así sucesivamente. El juego termina cuando se consumen los N turnos, o bien cuando todos los personajes de una clase se mueran.
e) Lo mismo que en el ejercicio del apartado c), pero haciendo que el personaje atacado sea siempre el que más puntos de vida tiene.
f) Muestra los personajes de cada clase al final del juego. Aparecerán ordenados según el valor del campo cuerpo, de mayor a menor.

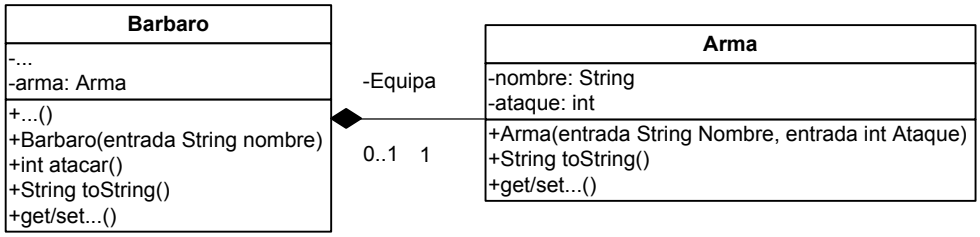
PRACTICA 1.8. Ejercicios con la jerarquía básica del Jeroquest : modificación de algunas clases `Personaje, Barbaro y Momia`

Objetivo: Saber modificar las clases de una jerarquía para cambiar la funcionalidad de un programa.

Ejercicio 1.8.1. Equipando a los bárbaros por *composición*.

Todos los héroes del juego pueden tener un arma para atacar. En este caso un héroe lanzará tantos dados como indique el arma (en lugar de los que indica su atributo **`ataque`**). Si en algún momento pierde el arma volverá a atacar con los dados indicados por su atributo **`ataque`**. El arma inicial de los bárbaros es una espada ancha como la de la figura de la derecha.

La siguiente figura muestra los cambios a introducir en la jerarquía de clases.



Lo que hay que hacer en este ejercicio es lo siguiente:

a) Para representar armas crea una nueva clase **`Arma`**. Esta clase contendrá la descripción del arma y el número de dados a utilizar cuando se ataque con ella.

b) Modifica la clase **`Barbaro`** para que contenga un nuevo atributo **`arma`** para representar su arma actual (si es que tiene alguna).

c) Modifica el constructor para que cree una espada ancha para el bárbaro.

d) Modifica el método **`getAtaque`** para que tenga en cuenta el arma utilizada. Observa que en este caso el método observador no devuelve simplemente el valor del campo **`ataque`**, sino que tiene que calcular los dados de ataque teniendo en cuenta si el barbaro tiene o no tiene arma.

e) En ambas clases se crea los observadores y modificadores necesarios y sobrecarga el método **`toString`** para mostrar la nueva información.

PRACTICA 1.9. Ejercicios con la jerarquía básica del Jeroquest : ampliación de la jerarquía de clases

Objetivo: Saber cómo añadir clases de la forma más adecuada a una jerarquía de clases.

Ejercicio 1.9.1. Ampliar la jerarquía de personajes del juego.

a) Define un nuevo tipo de personaje **`Enano`** e incorporarlo a la jerarquía de clases. **El enano ataca y se defiende igual que un bárbaro**. Amplia los ejemplos de prueba anteriores añadiendo un nuevo vector de enanos.

b) Define un nuevo tipo de personaje **`MomiaRabiosa`** e incorporarlo a la jerarquía de clases. Las momias rabiosas atacan **como las normales** pero hacen el doble de daño; sin embargo cuando las atacan no se defienden. Ampliar los ejemplos de prueba anteriores para incluir personajes de esta clase.

c) Teniendo en cuenta futuros cambios en estos arquetipos de personajes, valora las ventajas de que la clase **`MomiaRabiosa`** herede de la clase **`Momia`** o herede directamente de **`Personaje`**.

Ejercicio 1.9.2. Define la clase **`Heroe`** como subclase de **`Personaje`** y con dos subclases **`Barbaro`** y **`Enano`**. Además, considera que todos los héroes son manejados por algún jugador, define dicho atributo **`jugador`** (como una cadena de texto) en este tipo de personajes. Comprueba que los ejemplos de prueba anteriores siguen funcionando igual. ¿Dónde resulta más adecuado ubicar ahora el método **`atacar`**?

DISTRIBUCIÓN TEMPORAL

Sesión 1: PRACTICA 1.1.

Sesión 2. PRACTICA 1.2.

Sesión 3. PRÁCTICA 1.3.

Sesión 4. PRÁCTICA 1.4.

Sesión 5. PRÁCTICA 1.5.

Sesión 6. PRÁCTICA 1.6.

Sesión 7. PRÁCTICA 1.7.

Sesión 8: PRACTICA 1.8.

Sesión 9. PRACTICA 1.9.