UNIVERSITAT JAUME·I

1º Ingeniería Informática
1º Matemática Computacional

EI1008/MT1008 Programación II

Práctica 2: Programación orientada a objetos

Ejercicio 1

Implementa en lenguaje Java una clase `Punto` que permita representar puntos formados por dos coordenadas `x` e `y`. Ambas se deben guardar en atributos privados de tipo `double`. La especificación de la clase requiere que tenga los siguientes constructores y métodos públicos:

1. Un constructor `Punto(double x, double y)` que permita crear un nuevo punto dándole sus coordenadas.
2. Un constructor sin parámetros `Punto()` que permita crear un nuevo punto con coordenadas $(0, 0)$.
3. Un método `double getX()` que devuelva la coordenada `x` del punto.
4. Un método `double getY()` que devuelva la coordenada `y` del punto.
5. Un método `Punto desplazar(double desplazamientoX, double desplazamientoY)` que devuelva el nuevo punto obtenido al sumarle al punto los desplazamientos que se le pasen como parámetros. Ten en cuenta que la llamada `p.desplazar(a, b)` no debe modificar `p`.
6. Un método `double distancia(Punto otroPunto)` que permita calcular la distancia entre el punto y otro punto que se le pase como parámetro.

   Recuerda que la distancia entre (x_1, y_1) y (x_2, y_2) es $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. Para calcularla puedes utilizar `Math.sqrt` y `Math.pow`.
7. Un método `String toString()` que permita obtener una cadena formada por las coordenadas del punto, encerradas entre paréntesis y separadas por una coma y un blanco. Por ejemplo, para el punto de coordenadas 3,5 y 2,45 se devolvería la cadena `"(3.50, 2.45)"`.
8. Un método `boolean equals(Object otroObjeto)` que devuelva `true` si el punto tiene las mismas coordenadas que otro que se le pase como parámetro y `false` en caso contrario.

Debes implementar y comprobar el correcto funcionamiento de cada método, uno por uno, en el orden que consideres más apropiado.

Ejercicio 2

Implementa en lenguaje Java una clase `Restaurante` que permita representar restaurantes. De cada restaurante nos interesa almacenar su nombre (una cadena), su posición (un objeto de la clase `Punto` definida en el ejercicio anterior) y su valoración (un entero). Esos tres datos se deben guardar en atributos privados. La especificación de la clase requiere que tenga los siguientes constructores y métodos públicos:

1. Un constructor `Restaurante(String nombre, Punto posición, int valoración)` que permita crear un nuevo restaurante a partir de los datos recibidos.
2. Un método `String getNombre()` que devuelva el nombre del restaurante.
3. Un método `Punto getPosición()` que devuelva la posición del restaurante.

4. Un método `int getValoración()` que devuelva la valoración del restaurante.

5. Un método `double distancia(Punto p)` que devuelva la distancia entre el restaurante y el punto dado como parámetro.

6. Un método estático `Restaurante[] leeRestaurantes(String nombreFichero)` que construya y devuelva un vector de restaurantes cuyos datos se obtengan a partir del fichero de texto indicado. La primera línea de este fichero contiene un entero con la cantidad de restaurantes y cada una de las restantes líneas contiene, separados por espacios en blanco, las coordenadas `x` e `y` del punto que determina la posición de un restaurante, la valoración de ese restaurante y su nombre.

Debes implementar y comprobar el correcto funcionamiento de cada método, uno por uno, en el orden que consideres más apropiado.

Ejercicio 3

Escribe un programa que permita averiguar el restaurante más próximo a una posición indicada por el usuario. El programa debe pedir inicialmente el nombre del fichero de texto que almacena la información de los restaurantes y debe llamar al método estático `leeRestaurantes` de la clase `Restaurante` para obtener el vector de restaurantes correspondiente. A continuación, debe pedir la posición en la que se encuentra el usuario y debe mostrar en la salida estándar el nombre y la posición del restaurante más cercano en línea recta. Para ello, tu solución debe incluir un método estático `restauranteMásPróximo` que tenga como parámetros un vector de restaurantes, v, y un punto, p, y devuelva como resultado el restaurante de v más próximo a p.

Un ejemplo de ejecución del programa[1] es:

```
Indique su posición actual
Coordenada X: 21
Coordenada Y: 15
El restaurante más cercano a su posición es Casa Pepe, situado en el punto (20.0, 18.0)
```

Ejercicio 4

Escribe un programa similar al anterior pero que pida la distancia máxima que se quiere andar y devuelva el restaurante con mejor valoración situado a una distancia inferior o igual a la distancia dada.

La solución de este ejercicio debe incluir un método estático `restauranteMejorValorado` que tenga como parámetros un vector de restaurantes, v, un punto, p, y una distancia, d, y devuelva como resultado el restaurante de v mejor valorado cuya distancia a p sea menor o igual a d. En caso de que no haya ningún restaurante situado a la distancia máxima que se quiere andar, el método debe devolver `null`.

Ejercicio 5

Implementa en lenguaje Java una clase `Fecha` que permita representar fechas formadas por día, mes y año. Esos tres datos se deben guardar en atributos privados. La especificación de la clase requiere que tenga los siguientes constructores y métodos públicos:

1. Un constructor `Fecha(int día, int mes, int año)` que permita crear una nueva fecha. En este ejercicio puedes suponer que es responsabilidad del usuario de la clase proporcionar los datos de una fecha válida.

[1] En este ejemplo no se muestra la petición del nombre del fichero.

2. Un constructor `Fecha(Fecha otraFecha)` que permita crear una nueva fecha que sea una copia de la que se le pasa como parámetro.

3. Un método `String toString()` que permita obtener una cadena que representa la fecha en formato `día/mes/año`.

4. Un método `boolean equals(Object otroObjeto)` que devuelva `true` si la fecha es igual a la que se le pase como parámetro y `false` en caso contrario.

5. Un método `int compareTo(Fecha otraFecha)` que devuelva un valor negativo, cero o positivo según la fecha sea anterior, igual o posterior, respectivamente, a `otraFecha`.

6. Un método de acceso `int getDía()` que devuelva el día.

7. Un método de acceso `int getMes()` que devuelva el mes.

8. Un método de acceso `int getAño()` que devuelva el año.

9. Un método estático `boolean añoBisiesto(int año)` para comprobar si un año es bisiesto. Recuerda que un año es bisiesto si es divisible por 4 y no es divisible por 100, o si es divisible por 400.

10. Un método estático `int díasMes(int mes, int año)` que devuelva la cantidad de días de un determinado mes, teniendo en cuenta si es febrero y pertenece a un año bisiesto.

11. Un método estático `Fecha hoy()` que devuelva la fecha del sistema. Para ello, puedes ayudarte del siguiente bloque de código, tras importar la clase `java.util.Calendar`:

```
Calendar calendario = Calendar.getInstance();
int día = calendario.get(Calendar.DAY_OF_MONTH);
int mes = calendario.get(Calendar.MONTH) + 1;
int año = calendario.get(Calendar.YEAR);
```

12. Un método `Fecha díaSiguiente()` que cree y devuelva una nueva fecha correspondiente al día siguiente. Ten en cuenta que la llamada `f.díaSiguiente()` no debe modificar `f`.

Debes implementar y comprobar el correcto funcionamiento de cada método, uno por uno, en el orden que consideres más apropiado.

Ejercicio 6

Escribe un programa que le proponga al usuario adivinar tu fecha de nacimiento y al final le diga cuántos intentos ha necesitado. Cada vez que el usuario se equivoque, el programa debe ayudarle diciéndole si tu fecha de nacimiento es anterior o posterior a la que ha introducido. El programa solo debe aceptar fechas comprendidas entre el 1/1/1900 y el 1/1/2010.

Ejercicio 7

Modifica una copia del ejercicio anterior de forma que el programa detecte cuándo el usuario ha hecho un intento absurdo. Utiliza la variable `mínimaFechaPosterior` para almacenar la menor de las fechas posteriores a la de tu nacimiento introducidas hasta el momento por el usuario, y la variable `máximaFechaAnterior` para almacenar la mayor de las fechas anteriores a la de tu nacimiento que ha introducido el usuario. Cuando éste introduce una fecha posterior o igual a `mínimaFechaPosterior`, el programa debe indicarle que ese intento

no tiene sentido, puesto que ya le había dicho que lo que busca es anterior a `mínimaFechaPosterior`. El comportamiento debe ser análogo si el usuario introduce una fecha anterior o igual a `máximaFechaAnterior`.

Ejercicio 8

Añade a la clase `Fecha`[2] lo necesario para lanzar una excepción de tipo `ExcepcionFechaInvalida`[3] cuando los valores de `día`, `mes` y `año` que se pasen al constructor o al método `díasMes` no correspondan a una fecha válida (por ejemplo, si el mes es mayor que 12 o si el día es mayor que la cantidad de días del mes).

Si intentas utilizar los programas de los ejercicios 6 y 7 con esta nueva versión de la clase `Fecha` obtendrás errores de compilación, ya que estos programas no gestionan las posibles excepciones que puedan producirse. Modifica una copia de tus soluciones a estos ejercicios para que capturen la excepción que se produce en el caso de que el usuario introduzca datos incorrectos, de modo que se vuelvan a pedir los datos al usuario cuando esto suceda.

Ejercicio 9

Implementa en lenguaje Java una clase `Tarea` que permita representar tareas formadas por una fecha y una descripción. Estos datos se deben guardar en atributos privados. La especificación de la clase requiere que tenga los siguientes constructores y métodos públicos:

1. Un constructor `Tarea(Fecha fecha, String descripción)` que permita crear una nueva tarea a partir de los datos recibidos.
2. Un método `Fecha getFecha()` que devuelva la fecha asociada a una tarea.
3. Un método `String getDescripción()` que devuelva la descripción asociada a una tarea.
4. Un método `String toString()` que permita obtener una cadena formada por la fecha asociada a la tarea, el carácter dos puntos, un blanco y la descripción de la tarea. Por ejemplo, para una tarea con fecha 23/4/2018 y descripción *Control T2*, se devolvería la cadena `"23/4/2018: Control T2"`.

A continuación, implementa una clase `Agenda` que permita representar agendas de tareas pendientes. Las tareas se guardarán **ordenadas por fecha** en un vector privado de objetos de la clase `Tarea` antes definida. Para no consumir memoria innecesaria, la longitud de este vector deberá coincidir en todo momento con la cantidad de tareas almacenadas (en particular, la longitud será 0 cuando la agenda esté vacía). La especificación de la clase requiere que tenga los siguientes constructores y métodos públicos:

1. Un constructor sin parámetros `Agenda()` que permita crear una nueva agenda inicialmente vacía.
2. Un método `void añadir(Tarea tarea)` que añada la tarea dada a la agenda de modo que siga ordenada por fecha. Cuando la agenda ya contenga otras tareas con esa misma fecha, la nueva tarea se añadirá a continuación de ellas.
3. Un método `int cantidad()` que devuelva la cantidad de tareas almacenadas en la agenda.
4. Un método `Tarea[] consultar(Fecha fecha)` que devuelva un vector con todas las tareas correspondientes a la fecha dada. La longitud del vector devuelto será cero cuando la agenda no contenga ninguna tarea con esa fecha.
5. Un método `void borrarPasadas(Fecha fecha)` que borre todas las tareas anteriores a la fecha dada, sin incluirla.

[2] Para poder conservar la versión anterior y evitar conflictos de nombres entre las clases, crea un nuevo paquete para la nueva versión de la clase Fecha.

[3] Utiliza la clase `ExcepcionFechaInvalida` que tienes en el Aula Virtual de la asignatura.

6. Un método `void borrar()` que borre todas las tareas anteriores al día de hoy. Para ello, debes llamar al método `borrarPasadas`.

7. Un método `String toString()` que permita obtener una representación de la agenda en forma de cadena. En esta cadena aparecerá representada cada una de las tareas de la agenda (con el formato propio de la clase `Tarea`), seguida de un salto de línea, como se observa en el siguiente ejemplo:

```
26/2/2018: EI1007-C1
16/4/2018: EI1006
16/4/2018: EI1007-C2
7/5/2018: EI1007-C3
```

Ejercicio 10

El método `añadir` del ejercicio anterior presenta un inconveniente. La adición de una sola tarea a una agenda de n tareas implica crear un vector auxiliar de $n+1$ tareas y copiar en él todas las tareas que contiene la agenda, además de la nueva tarea. Se trata de una operación costosa. Con los métodos `borrarPasadas` y `borrar` sucede algo similar.

Para mejorar su eficiencia te proponemos modificar[4] la representación interna de una agenda para utilizar, además de un vector de tareas, un atributo privado que indique la cantidad de tareas de la agenda. De esta manera distinguiremos entre la capacidad del vector (la cantidad máxima de tareas que puede almacenar) y la ocupación del vector indicada por el nuevo atributo (la cantidad de tareas que realmente contiene).

Reimplementa todos los métodos de la clase `Agenda` que lo requieran para adaptarlos a la nueva representación interna. Debes modificar, entre otros:

- El constructor `Agenda()` para que cree un vector de tareas de una determinada capacidad inicial (es habitual emplear un valor 8 o 10) que represente una agenda vacía.
- El método `void añadir(Tarea tarea)` para que *redimensione* el vector de tareas únicamente cuando sea necesario (es decir, cuando su capacidad m coincida con el número de tareas n), de modo que su nueva capacidad sea el doble de la anterior.
- El método `void borrarPasadas(Fecha fecha)` para que, sin disminuir nunca la capacidad del vector[5] ni hacer uso de ningún otro vector auxiliar, suprima las tareas correspondientes.

[4] Debes conservar la clase `Agenda` del ejercicio anterior. conflictos de nombres entre ambas clases, debes crear un paquete para cada implementación de la clase.

[5] Aunque en este ejercicio no te pedimos que lo hagas, una estrategia habitual consiste en reducir a la mitad la capacidad m del vector cuando el número de tareas n llega a valer $m/4$.