control2SaludMayo21v2.pdf

Cerebrandex

Programación Orientada a Objetos

1° Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga

DEPARTAMENTO LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN	**PROGRAMACIÓN ORIENTADA A OBJETOS** (Prueba realizada el 10 de mayo de 2021)	
APELLIDOS, Nombre	TITULACIÓN Y GRUPO	
	MÁQUINA	

NOTAS PARA LA REALIZACIÓN DEL EJERCICIO:

- Al inicio del contenido de cada fichero deberá indicarse **el nombre del alumno, titulación, grupo y código del equipo** que está utilizando**.**
- La evaluación tendrá en cuenta la claridad de los algoritmos, del código y la correcta elección de las estructuras de datos, así como los criterios de diseño que favorezcan la reutilización.
- Los diferentes apartados tienen una determinada puntuación. Si un apartado no se sabe hacer, el alumno *no debe pararse en él indefinidamente*. Puede abordar otros.
- **Está permitido:**
  - Consultar la API y los apuntes.
- **No está permitido**:
  - Utilizar otra documentación electrónica o impresa.
  - Intercambiar información con otros compañeros.
  - Utilizar soportes de almacenamiento o teléfonos móviles.
- Una vez terminado el ejercicio subir, a la tarea creada en el campus virtual para ello, un solo archivo con los ficheros **fuente** (.java) que hayáis realizado.

Se desea desarrollar una aplicación para gestionar el aforo y la asistencia a las playas durante la época estival, con el objetivo de controlar la masificación en tiempos de coronavirus. Para ello, se creará un proyecto `prPlayas` con las clases `Playa`, `PlayaRestringida`, `PlayaException` y `GestionPlayas` en el paquete `playas`, y la clase distinguida `PruebaPlayas`, que se proporciona con el enunciado, en el paquete por defecto.

1) **(0.25 ptos.)** Clase `PlayaException`, que será una excepción **no comprobada** para manejar las situaciones excepcionales que puedan producirse en las siguientes clases.

2) **(2 ptos.)** La clase `Playa` mantendrá información sobre cada playa a gestionar. En concreto, su nombre (`String`), localidad a la que pertenece (`String`), latitud y longitud de un punto representativo de la misma (`double`), aforo máximo permitido (`int`) y una variable denominada `numPersonas` que indica el número de personas (o aforo actual) que en un momento concreto se encuentran en la playa (`int`). Además, incluirá una variable `estado` (`String`), que indicará el nivel de aforo de esa playa y podrá tener cualquiera de estos valores: `"VACIO"`, `"LEVE"`, `"MODERADO"`, `"SATURADO"` o `"SIN ESPACIO"`. También dispondrá de:
   a. *(0.25 ptos.)* Un constructor dados valores para el nombre, localidad, latitud, longitud y aforo máximo. Si el aforo máximo es un número negativo se lanzará una excepción de tipo `PlayaException`. Inicialmente, el número de personas que se encuentran en una playa es 0 y el estado de la playa es `VACIO`.
   b. *(0.25 ptos.)* Métodos `double getLongitud()`, `double getLatitud()`, `int getAforoMaximo()` y `String getEstado()`, que permiten obtener los valores almacenados en los atributos `longitud`, `latitud`, `aforoMax` y `estado`, respectivamente.
   c. *(0.5 ptos.)* Un método `void setNumPersonas(int)`, que modificará el número de personas existentes en la playa en ese momento. Si el parámetro de entrada es negativo se lanzará una excepción de tipo `PlayaException`. Además, en función

del número de personas y el aforo máximo se establecerá el nivel de aforo o estado de la playa de la siguiente forma:

- 0% del aforo máximo -> estado: VACIO
- Entre el 0% y el 25% (no incluido) del aforo máximo -> estado: LEVE
- Entre el 25% y el 75% (no incluido) del aforo máximo -> estado: MODERADO
- Entre el 75% y el 100% (no incluido) del aforo máximo -> estado: SATURADO
- Más del 100% del aforo máximo -> estado: SIN ESPACIO

d. *(0.75 ptos.)* Métodos `equals` y `hashcode`, teniendo en cuenta que dos playas son iguales sólo si lo son su latitud y longitud.

e. *(0.25 ptos.)* La representación textual de una playa (`toString`) incluye el nombre, la localidad, la latitud, la longitud, el aforo máximo, el número de personas y el estado, según el formato que se indica en el ejemplo siguiente:

```
(El Candado, Málaga, 36.715, -4.344, 65, 0 - Nivel de aforo: VACIO)
```

Obsérvese que toda la información está encerrada entre paréntesis.

3) **(6.25 ptos.)** La clase `GestionPlayas` almacenará la información de la colección de playas que pertenecen o están asociadas a una provincia en el array denominado `playas`. Además, se almacenará la provincia en cuestión en la variable de instancia `provincia` (`String`). La variable de instancia `numPlayas` (`int`) sirve para conocer el número de playas que hay almacenadas en el array y para gestionar el array, ya que los elementos se mantienen al principio del mismo (en posiciones `0...numPlayas-1`). La clase dispondrá de:

a. *(0.25 ptos.)* Un constructor con un argumento (`String`) que se corresponderá con el nombre de la provincia. Se utiliza el valor almacenado en la constante de clase `CAP_INICIAL` (`int`) como tamaño inicial del array. El valor de la constante `CAP_INICIAL` será 5. Inicialmente el array se encuentra vacío y no se dispone de playas registradas.

b. *(1.25 ptos.)* Un constructor con dos argumentos, uno de tipo `String` con el nombre de la provincia, y un array de `String` en el que cada elemento del array contendrá toda la información para crear una playa con el siguiente formato (deben aparecer siempre los cinco tokens separados por ;)[1]:

<nombre_playa>;<localidad>;<latitud>;<longitud>;<aforo_máximo>

Por ejemplo:

```
Los Álamos;Torremolinos;36.635;-4.486;400
```

Así, para cada elemento del array se deberá crear, si es posible, la playa con el nombre, localidad, latitud, longitud y aforo máximo y almacenarlo en un array de playas. Se puede reutilizar el método `incluye`, descrito posteriormente en el enunciado. Dependiendo del tipo de error que pueda producirse se mostrará en la consola el mensaje informativo correspondiente: "ERROR. Valor no númerico", "ERROR: Faltan datos" o "ERROR:Aforo negativo".

c. *(0.25 ptos.)* Un método **privado** `int posicion(Playa)` para buscar una playa en la colección dado un objeto de tipo `Playa`. Debe devolver la posición en la que se

[1] NOTA para Scanner: Dependiendo de la configuración de la máquina, es posible que el scanner no funcione correctamente con números en el formato decimal habitual. Para poder leer números decimales con el separador punto (p.ej., 7.1), se puede enviar al objeto Scanner el mensaje `useLocale`, por ejemplo, `sc.useLocale(Locale.ENGLISH)`.

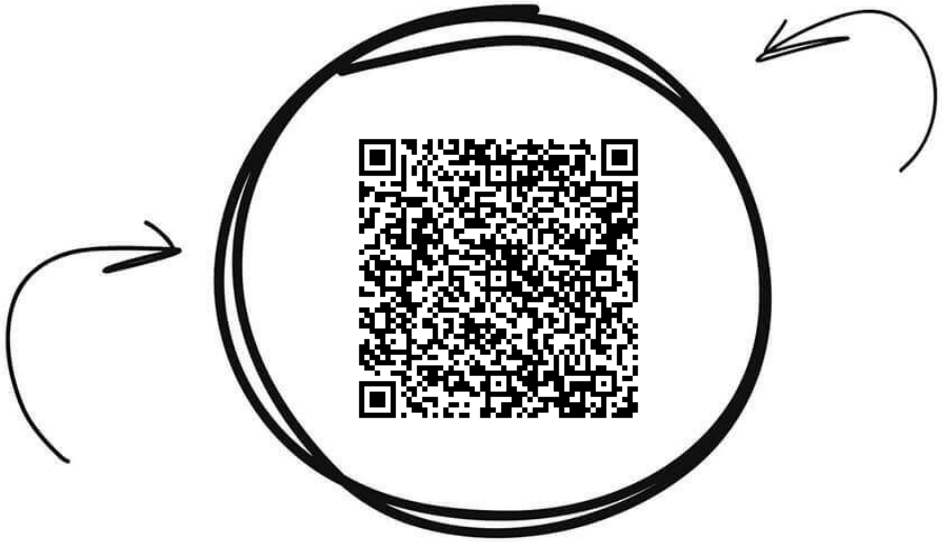
Dia

ACTIVAR EL MODO DIABLO ES ESTUDIARTE 5 TEMAS EN UNA NOCHE.

O beberte una de estas, que es más fácil.

*Dia recomienda el consumo responsable de bebidas energéticas

Programación Orientada a Obj...



Banco de apuntes de la

WUOLAH

Comparte estos flyers en tu clase y consigue más dinero y recompensas

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

encuentra el objeto `Playa` correspondiente o -1 en caso de que no se encuentre en el array.

d. *(0.75 ptos.)* Un método `void incluye(Playa)` para añadir un objeto de tipo `Playa` en el array de playas. Si la playa que se desea añadir ya está en la colección, esta sustituirá a la encontrada en el array. En caso de tener que añadir la playa y el array esté lleno, se duplicará su tamaño automáticamente. Puedes usar el método privado `posicion` anterior para la implementación de este método.

e. *(0.5 ptos.)* Un método **privado** `Playa buscar(double, double)` para buscar una playa en la colección dada las coordenadas de latitud y longitud. Debe devolver el objeto `Playa` correspondiente o lanzar una excepción de tipo `PlayaException` en caso de que no se encuentre en el array. Puedes usar el método `posicion` anterior para la implementación de este método.

f. *(2 ptos.)* Un método `void registrarOcupacion(String)` para actualizar el número de personas que están en cada playa en un momento dado. La información del número de personas vendrá dada en un fichero, cuyo nombre se pasa como argumento, que será proporcionado junto con este enunciado `(datos.txt)`, en que se incluirá cada playa en una línea y en el siguiente formato (deben aparecer siempre los tres tokens separados por ;):

<latitud>;<longitud>;<num_personas>

Por ejemplo, una línea del fichero sería:

```
36.490;-4.775;130
```

Por cada línea con formato y datos correctos se deberá buscar en el array la playa con dicha latitud y longitud, actualizando su número de personas. Para buscar la playa se podrá usar el método `buscar` anteriormente descrito. Si los datos de una línea no son correctos se dará un mensaje informativo indicativo de la situación excepcional que se haya producido y se pasará a la siguiente línea.

NOTA: Se puede modularizar el código en diferentes métodos.

g. *(0.5 ptos.)* Un método `Playa primeraPlayaConEstado(String)` para obtener la primera playa disponible dado un estado. Debe devolver el objeto `Playa` correspondiente o lanzar una excepción de tipo `PlayaException` en caso de que no se encuentre en el array.

h. *(0.75 ptos.)* La representación como cadena de caracteres de los objetos de esta clase muestra toda la información contenida en la misma, esto es, el nombre de la provincia y la representatión de cada una de sus playas. El formato que se ha de utilizar es el que se indica en el siguiente esquema. Se debe utilizar `StringBuilder` o `StringJoiner`.

```
NombreProvincia Playas: [playa1, playa2, ...]
```

4) **(1.5 ptos.)** La clase `PlayaRestringida` tiene un comportamiento similar al de la clase `Playa`, pero a una playa restringida es posible reducirle su aforo máximo en un porcentaje determinado. Dicho porcentaje se guardará en una variable `porcentaje` de tipo `int`. Por ejemplo, si el aforo máximo de una playa es 100 pero

su uso está restringido al 30%, su aforo máximo restringido será de 30. Se debe implementar:

a. *(0.25 ptos.)* Un constructor con los mismos argumentos del constructor de la clase `Playa` que inicializará sus variables de instancia de forma correspondiente e inicializará su porcentaje de restricción a 0.

b. *(0.25 ptos.)* Un método `setPorcentaje(int)` que actualice el porcentaje de máxima ocupación con el valor pasado como argumento. En caso de que el porcentaje sea menor que 0 o mayor que 100 se lanzará una excepción de tipo `PlayaException`.

c. *(0.75 ptos.)* Una redefinición del método `void setNumPersonas(int)` que, además de asignar el número de personas, cambie el estado en función del aforo máximo. Los rangos de los diferentes estados deben actualizarse simplemente multiplicando por el porcentaje, de forma que, por ejemplo, el rango de estado MODERADO pasa a ser [25*porcentaje/100, 75*porcentaje/100).

d. *(0.25 ptos.)* Una redefinición del método `toString` en el que se incluya el porcentaje de restricción sobre el aforo máximo. Un ejemplo será el siguiente:

```
[(La carihuela, Torremolinos, 36.607, -4.504, 200, 0 - Nivel de aforo: VACIO), 50%]
```

Obsérvese que toda la información está encerrada entre corchetes.

Como parte del enunciado del examen se puede encontrar la clase `PruebaPlayas.java`, la cual, situada en el paquete por defecto del proyecto, se puede utilizar para comprobar el correcto funcionamiento de las clases requeridas. La salida esperada como salida de dicho programa de prueba está incluida como un comentario al final del fichero `PruebaPlayas.java`.