	FUNDAMENTOS DE PROGRAMACIÓN **EXAMEN ESCRITO DEL PRIMER CUATRIMESTRE. 22 de enero de 2016**	**Curso: 2015/16** **Versión: 1.0.0**

Ejercicio 1

(2 puntos)

PilotoF1 y PilotoF1Puntuado

Para realizar este ejercicio suponga definidas las siguientes interfaces:

```
public interface PilotoF1 {
        // Propiedades
        String getNombre();
        String getEscuderia();
        String getPais();
}

public interface PilotoF1Puntuado extends PilotoF1 {
        // Propiedades
        List<String> getCircuitos();
        List<Integer> getPuntos();
        Integer getPuntuacionTotal();
        // Operaciones
        void puntuar(String circuito, Integer puntos);
        Integer obtenerPuntuacion(String circuito);
}
```

PilotoF1 describe al tipo piloto de Fórmula 1 a través de su **nombre**, la **escudería** en la que compite y su **país** de origen. La clase que lo implementa cuenta con un único constructor al que se le pasan el nombre, la escudería y el país, en este mismo orden.

PilotoF1Puntuado representa a un piloto de Fórmula 1 con las puntuaciones obtenidas en los distintos circuitos a lo largo de un mundial. Tiene las siguientes propiedades:

- **Circuitos:** lista de circuitos en los que ha puntuado el piloto.
- **Puntos:** lista con las puntuaciones obtenidas por el piloto. La posición i-ésima de esta lista almacena la puntuación obtenida por el piloto en el circuito i-ésimo.
- **PuntuacionTotal**: suma de todas las puntuaciones.

Se pide que escriba el siguiente código de la clase **PilotoF1PuntuadoImpl**:

a) **(0.25 puntos)** Cabecera y constructor de la clase, al que se le pasan el **nombre**, la **escudería** y el **país** como parámetros. Inicialmente se considera que el piloto no ha puntuado en ningún circuito.

b) **(0.25 puntos)** Propiedad **PuntuacionTotal**, que se calcula como la suma de las puntuaciones obtenidas por el piloto en todos los circuitos.

c) **(0.5 puntos)** Operación **puntuar(circuito, puntos).** Si el piloto ya tiene una puntuación para el circuito, esta operación la sustituye por los nuevos **puntos**. En caso contrario, la añade como una puntuación nueva. Los **puntos** que obtiene un piloto en un **circuito** es un entero entre 0 y 25, ambos valores incluidos. En caso de intentar puntuar a un piloto con un valor no permitido se debe elevar la excepción **ExcepcionPilotoF1PuntuadoOperacionNoPermitida**. No es necesario escribir el código de dicha excepción.

d) **(0.5 puntos)** Operación **obtenerPuntuacion(circuito)**. Devuelve la puntuación obtenida por el piloto en el **circuito**. En el caso de que el piloto no tenga ninguna puntuación para el **circuito** debe devolver cero.

e) **(0.5 puntos)** Método main que realice las siguientes acciones:
   - Crear un piloto puntuado con los siguientes datos:
     - Nombre: “Lewis Hamilton”
     - Escudería: “Mercedes”
     - País: “UK”
     - Circuitos: [“Mónaco”,”Shanghai”]
     - Puntos: [25, 18]
   - Mostrar por pantalla la puntuación total.
   - Mostrar por pantalla la puntuación obtenida en el circuito de “Mónaco”.

Ejercicio 2

(3 puntos)

CarreraF1

Para realizar este ejercicio suponga definida la siguiente interfaz:

```
public interface CarreraF1 extends Comparable<CarreraF1> {
        // Propiedades
        String getCircuito();
        String getPais();
        LocalDate getFecha();
        List<PilotoF1> getParrillaSalida();
        List<PilotoF1> getClasificacion();
        // Operaciones
        Integer obtenerPuntuacionPiloto(PilotoF1 piloto);
        Integer lineaDeSalida(PilotoF1 piloto);
        List<PilotoF1> primerosParrillaSalida(Integer n);
        List<PilotoF1> primerosClasificacion(Integer n);
        Set<PilotoF1> parrillaYClasificacion(Integer n);
}
```

CarreraF1 describe al tipo carrera de Fórmula 1 a través de las propiedades **circuito** en el que se celebra la carrera, **país** en el que se celebra y **fecha** en la que se celebra. Además, cuenta con otras dos propiedades que representan a la **parrilla de salida** (lista de pilotos ordenada por la posición en la que salen los pilotos) y la **clasificación** (lista de pilotos ordenada por la posición en la que finalizan los pilotos la carrera).

Se pide que escriba el siguiente código de la clase **CarreraF1Impl**:

a) **(0.5 puntos)** Los métodos **equals**, **hashCode** y **compareTo** teniendo en cuenta que dos carreras son iguales si lo son las fechas en las que se celebran, y que el orden natural es también por la fecha en la que se celebran las carreras.

b) **(0.5 puntos)** Operación **obtenerPuntuacionPiloto(piloto)** que devuelve la puntuación que ha obtenido el **piloto** en función de la posición que ocupa en la **clasificación** de la siguiente forma:

Puesto	Puntos	Puesto	Puntos
1º	25 puntos	6º	8 puntos
2º	18 puntos	7º	6 puntos
3º	15 puntos	8º	4 puntos
4º	12 puntos	9º	2 puntos
5º	10 puntos	10º	1 punto

Si el **piloto** se clasifica a partir de la posición undécima obtiene cero puntos. Si el **piloto** no está en la lista de clasificados se elevará la excepción **ExcepcionCarreraF1OperaciónNoPermitida**. No tiene que escribir el código de la excepción.

c) **(0.5 puntos)** Operación **lineaDeSalida(piloto)**, que devuelve la línea que ocupa el **piloto** en la parrilla de salida, teniendo que cuenta que cada línea tiene dos pilotos. Por ejemplo, un piloto que ocupa la posición 11º en la parrilla, saldrá en la 6ª línea de salida. Si el **piloto** no está en la parrilla de salida se elevará la excepción **ExcepcionCarreraF1OperaciónNoPermitida**. No tiene que escribir el código de la excepción.

d) **(0.5 puntos)** Operaciones **primerosParrillaSalida(n)** y **primerosClasificacion(n)**, que devuelven la lista de pilotos que ocupan las primeras **n** posiciones de la parrilla de salida y de la clasificación, respectivamente. Si **n** tiene un valor mayor que el número de pilotos que hay en la parrilla de salida o en la clasificación, devolverán todos los pilotos de la parrilla de salida o de la clasificación, respectivamente.

e) **(1 punto)** Operación **parrillaYClasificacion(n)**, que devuelve el conjunto de pilotos que salieron entre las **n** primeras posiciones de la parrilla de salida y han quedado entre los **n** primeros puestos de la clasificación, es decir, que están entre los **n** primeros de la parrilla y de la clasificación.

Ejercicio 3

(5 puntos)

MundialF1

Para realizar este ejercicio suponga definida la siguiente interfaz:

```
public interface MundialF1 {
        // Propiedades
        Integer getAño();
        SortedSet<CarreraF1> getCalendario();
        Set<PilotoF1Puntuado> getPilotos();
        // Operaciones
        List<CarreraF1> posterioresA(CarreraF1 c);
        void puntuarCarrera(CarreraF1 c);
        Integer carrerasGanadas(PilotoF1 p);
        String masCarrerasGanadas();
        PilotoF1 ganaEnCasa();
        Boolean existeGanador(String pais, Integer n);
}
```

MundialF1 describe al tipo Mundial de Fórmula 1 a través de las propiedades **calendario** (conjunto ordenado de carreras), el **año** en el que se celebra y las puntuaciones obtenidas por los **pilotos** que participan en el mundial.

Se pide que escriba el siguiente código de la clase **MundialF1Impl**:

a) **(0.5 puntos)** Operación **posterioresA(carrera)**, que devuelve una lista con las carreras que se celebran en una fecha posterior a la **carrera.**
b) **(0.75 puntos)** Operación **puntuarCarrera(carrera)**, que puntúe a todos los pilotos de la **carrera** en función del puesto en el que terminaron.
c) **(0.75 puntos)** Operación **carrerasGanadas (piloto)**, que devuelve el número de carreras que ha ganado el **piloto**. En el caso de que el piloto no se haya clasificado en ninguna carrera del calendario esta operación devuelve cero.
d) **(1 punto)** Operación **masCarrerasGanadas()**, que devuelve el nombre del piloto que mayor número de carreras ha ganado en el mundial.
e) **(1 punto)** Operación **ganaEnCasa()**, que devuelve el primer piloto del calendario que gana en casa, es decir, que el país del circuito es el mismo que el país de origen del piloto. En caso de no existir ningún piloto que haya ganado en casa la operación debe elevar la excepción **NoSuchElementException**.
f) **(1 punto)** Operación **existeGanador(String pais, Integer n)**, que devuelve true si existe al menos un piloto del **país** dado que ha ganado **n** o más carreras, y false en caso contrario.

Anexo

```
public interface Collection<E> extends Iterable<E> {
   boolean add(E e);
   boolean addAll(Collection<? extends E> c);
   void clear();
   boolean contains(Object o);
   boolean containsAll(Collection<?> c);
   boolean isEmpty();
   boolean remove(Object o);
   boolean removeAll(Collection<?> c);
   boolean retainAll(Collection<?> c);
   int size();
}
public interface SortedSet<E> extends Set<E> {
   SortedSet<E> headSet(E toElement);
   SortedSet<E> tailSet(E fromElement);
   SortedSet<E> subSet(E fromElement, E toElement);
   E first();
   E last();
}
```

```
public interface List<E> extends Collection<E> {
   void add(int index, E element);
   boolean addAll(int index, Collection<? extends E> c);
   E get(int index);
   int indexOf(Object o);
   int lastIndexOf(Object o);
   E remove(int index);
   E set(int index, E element);
   List<E> subList(int fromIndex, int toIndex);
}

public class String {
   char charAt (int index) {..}
   boolean contains (CharSequence s) {..}
   int indexOf (char c) {..}
   boolean isEmpty() {..}
   int length() {..}
   Boolean startsWith(String s) {..}
}
```