Ejercicio de Programación Orientada a Objetos
Curso 2016/2017
Concursos

1. Funcionalidad básica.

Queremos desarrollar una aplicación para la gestión de concursos de programación.

Un **concurso** se caracteriza por las siguientes propiedades:

- Nombre que identifica al concurso.
- Número de problemas.
- Conjunto de equipos participantes. Cada equipo se identifica mediante una cadena.
- Tiempo del concurso (en minutos).
- Soluciones: lista de cadenas que representan la solución esperada para los problemas (se describe más adelante).
- Envíos: lista que contiene los envíos de los equipos participantes como respuestas a los problemas (se describe más adelante).
- Iniciado: propiedad booleana que indica si el concurso ha sido iniciado.

En la construcción de un concurso se establece: el nombre, número de problemas y el tiempo del concurso. El tiempo de concurso y el número de problemas no se puede modificar una vez establecido. En cambio, el nombre del concurso sí puede ser modificado.

Inicialmente un concurso no contiene ningún equipo participante. Las operaciones para la gestión de los equipos son las siguientes:

- Añadir equipos. Esta operación acepta una secuencia de nombres de equipos para registrarlos en el concurso (argumento tamaño variable).
- Eliminar un equipo, estableciendo su nombre como parámetro. Retorna un valor booleano indicando si ha podido eliminarlo.

Igualmente, un concurso recién creado no tiene soluciones para los problemas. Las operaciones para la gestión de las soluciones son las siguientes:

- Establecer la solución para un problema. Ten en cuenta que los problemas se identifican mediante un número entero positivo, siendo el primer problema el número 1. Además, la solución a un problema se expresa con una cadena.
- Consultar la solución de un problema.

Dado que las soluciones se establecen una vez creado el concurso, decimos que un concurso está *preparado* si se han establecido soluciones para todos sus problemas. Esta propiedad puede ser consultada.

Un concurso se puede *iniciar* si está preparado. En tal caso, se establece a verdadero la propiedad *iniciado*, se inicializa a vacío la colección de envíos y almacena en un *atributo de implementación* (por ejemplo, *fin*) la marca de tiempo que representa el fin del concurso. Para el cálculo de la marca de tiempo, puedes tener en cuenta que el *instante actual* se obtiene con la llamada `System.currentTimeMillis()`. El instante final será la suma del instante actual más el tiempo del concurso expresado en milisegundos (1 minuto son 60.000 milisegundo). La operación finaliza retornando un valor booleano indicando si se ha podido iniciar el concurso.

En relación con el inicio del concurso, se añaden dos nuevas **propiedades calculadas**:

- En marcha: propiedad booleana que indica si el concurso ha sido iniciado y todavía no ha finalizado su tiempo, es decir, el instante actual es inferior a *fin*.
- Ha finalizado: propiedad booleana que indica si un concurso que ha sido iniciado, ha terminado su tiempo.

La funcionalidad más importante de un concurso es la gestión de *envíos*. Un **envío** representa la información sobre la respuesta de un equipo a un problema del concurso. Las propiedades de un envío son las siguientes:

- Nombre del equipo.
- Número de problema. Recuerda que los problemas se indexan desde el número 1.
- Respuesta al problema (cadena).
- Evaluación. Indica el resultado del envío: aceptado o rechazado (enumerado).

Todas las propiedades se establecen en el constructor y una vez establecidas no pueden cambiar.

El concurso ofrece una operación para **registrar los envíos**. La operación recibe como parámetros el nombre del equipo, el número de problema y la respuesta. Para la implementación de esta operación aplica el concepto **método plantilla**. Los pasos que realiza esta operación son los siguientes:

- Los requisitos generales para aceptar un envío son: 1) los parámetros son correctos (el equipo está en la colección de equipos, el índice del problema es válido, y la respuesta no es nula ni la cadena vacía), 2) se produce dentro del tiempo del concurso (está *en marcha*), y 3) el equipo no ha hecho previamente un envío *aceptado* para ese problema. En caso de no cumplir estos requisitos, retorna un valor nulo indicando que el envío no ha sido registrado.
- Comprueba si el envío *cumple las condiciones del concurso*. Las condiciones de registro de los envíos dependen de los tipos de concursos que se describen más adelante. En caso de no cumplir *las condiciones del concurso*, retorna un valor nulo indicando que el envío no ha sido registrado.
- Si se cumplen las condiciones generales y específicas del concurso, evalúa la respuesta al problema. Para ello comprueba si la respuesta coincide con la solución. En tal caso, el envío es aceptado, en caso contrario sería incorrecto, y por tanto, rechazado. A continuación, se construye un envío y se registra en el concurso. Finalmente, retorna el envío. Nótese que en el concurso se registran tanto los envíos aceptados como los rechazados.

Todos los envíos a un concurso deben poder se consultados (como cualquier otra propiedad).

Se definen los siguientes **tipos de concursos**:

- **Concurso limitado**. Este concurso se caracteriza por limitar los envíos que puede realizar cada equipo para un mismo problema (propiedad *número de intentos*). El número de intentos es una propiedad que no puede variar una vez establecida. Por tanto, la *condición* para que un equipo pueda registrar un envío para un problema es que no haya superado el número de envíos registrados para ese problema. Por ejemplo, si se crea un concurso con 3 problemas y limitado por 2 intentos, ningún equipo podrá registrar más de dos envíos para cada problema (como máximo 6 envíos en todo el concurso).
- **Concurso competitivo**. Este concurso se caracteriza por tener como *condición* para el registro de un envío que corresponda a un problema no resuelto, esto es, problemas para los que ningún equipo haya registrado un envío que haya sido evaluado como *aceptado*. Además, permite consultar si un problema determinado ha sido resuelto, esto es, evaluado como aceptado.

  Nota: Se establece como requisito que esta clase mantenga una estructura de datos **eficiente** para consultar si un problema ha sido resuelto. Además, esta estructura de datos debe actualizarse cuando se registre un nuevo envío aceptado.
- **Concurso secuencial**. Es un *concurso limitado* que **añade** como *condición* para el registro de un envío que los problemas se resuelvan de forma secuencial por cada equipo participante. La secuencialidad significa que, en un concurso secuencial recién creado hay que resolver primero el problema número 1. Una vez resuelto se podrá enviar el problema número 2 y así sucesivamente. Esta información se mantiene en una estructura de datos que se denomina *estado del concurso* (**mapa**) que asocia el nombre de cada equipo con su *problema actual*. Se permite consultar el número de *problema actual* (problema que se está resolviendo) de un equipo concreto y el *estado del concurso*. Por último, nótese que el *estado del concurso* es una estructura de datos que debe ser inicializada al iniciar el concurso y debe ser actualizada cuando se registra un envío aceptado.

2. Clasificación.

Queremos introducir en el concurso la funcionalidad relativa a la clasificación de los equipos de acuerdo a la puntuación obtenida en el concurso.

En primer lugar, declara un tipo de datos para que registre la puntuación de cada equipo (`PuntuacionEquipo`). La puntuación de un equipo tiene dos propiedades: el nombre del equipo (no puede ser modificada una vez establecida) y los puntos. Los puntos inicialmente comenzarán en 0, y podrán ser incrementados o decrementados.

El concurso debe mantener un mapa (*puntuaciones*) que asocie cada equipo con su puntuación (tipo de datos anterior). Este mapa debe ser inicializado en la operación *iniciar*.

La clasificación de un concurso es representada como una lista de `PuntuacionEquipo`. La clasificación se calcula bajo demanda ordenando los *valores* del mapa *puntuaciones* (método `sort` de la clase `java.util.Collections`). Por este motivo, la clase `PuntuacionEquipo` debe implementar un **orden natural** que ordene las puntuaciones de mayor a menor número de puntos, y en caso de empate, por orden alfabético de los nombres de los equipos.

Las puntuaciones almacenadas en el mapa *puntuaciones* deben ser actualizadas cuando vaya a registrarse un envío. Si el problema es aceptado, se suman 3 puntos. En cambio, si es rechazado, se resta 1 punto.

3. Copia de concursos.

Implementa el método `clone` en las clases que representan los concursos de acuerdo a la semántica de los tipos de datos y siguiendo las recomendaciones de la asignatura.

Debe tenerse en cuenta lo siguiente:

- Es necesario evitar los casos de *aliasing* que no sean correctos, en especial, los relativos a las colecciones.
- La copia de un concurso debe quedar en estado “no iniciado”, y por tanto, las estructuras de datos como los envíos y las puntuaciones deberán quedar vacíos.
- Siempre interesa redefinir el método `clone` en las clases descendientes, aunque solo sea para aplicar la regla covariante.

4. Programa

- Declara y construye un concurso limitado con nombre “Sesión 1”, 2 problemas, 5 minutos de tiempo y límite de 1 envío por problema. Establece como soluciones a los problemas los valores “23” para el primero y “15” para el segundo.
- Declara y construye un concurso competitivo con nombre “Sesión 2”, 3 problemas y 15 minutos de tiempo. Las soluciones a los problemas son “AACTG”, “null” y “[13, 98]”.
- Declara y construye un concurso secuencial con nombre “Sesión 3”, 3 problemas, 30 minutos y límite de 2 envíos por problema. Las soluciones a los problemas son “null”, “[0, 3]” y “AAA”.
- Crea una lista de concursos y añade los tres concursos creados en los pasos anteriores.
- Recorre los concursos y realiza lo siguiente:
    - Muestra el nombre del concurso.
    - Añade los equipos “Equipo 1”, “Equipo 2” y “Equipo 3”.
    - Inicia el concurso.
    - Recorre los equipos para que cada uno realice el envío “null” a todos los problemas.
    - Muestra la clasificación del concurso en el siguiente formato:

```
Clasificación:
      - Equipo 1 -> 2
      - Equipo 2 -> -2
      - Equipo 3 -> -2
```

    - Si el concurso es secuencial, consulta el estado del concurso y muestra la información por la consola en el siguiente formato:

```
Estado del concurso:
       - Equipo 1 -> problema actual: 2
       - Equipo 2 -> problema actual: 2
       - Equipo 3 -> problema actual: 2
```