Tecnología de la Programación - Curso 2014/2015
Grado en Ingeniería Informática, Grado en Ingeniería de Computadores
Grado en Ingeniería del Software, Doble grado en Matemáticas-Informática
Examen de junio - Parte práctica - Duración: 2,5 horas.

Instrucciones

- Lee cuidadosamente el examen hasta el final. Algunas decisiones de diseño tomadas para implementar las primeras partes pueden afectar a la complejidad en la implementación de las últimas.
- En el examen deberás partir de la *práctica 5* realizada durante el curso. **En breve se indicará el procedimiento de acceso a dicho código.**
- El código entregado ***debe compilar***.
- En la corrección del examen se valorará el funcionamiento, la claridad del código, el uso conveniente de los medios proporcionados por la Programación Orientada a Objetos (herencia, polimorfismo...) y comentarios. Para la evaluación se tendrá en cuenta tanto lo pedido en el examen como el código del que se parte (implementación de la práctica 5). Romper la encapsulación de las clases (acceso a atributos privados y protegidos desde clases externas, utilización de atributos públicos, etc.) implica suspender el examen.
- Incluye, además del código, una descripción de los pasos y modificaciones que has realizado para implementar lo que se pide en el examen. Este informe de cambios puedes hacerlo en papel o adjuntarlo como un fichero de texto con el resto de los archivos de código fuente. Adjunta este fichero de texto `descripcion.txt` con el resto de los archivos de código fuente. Deja el fichero en "el raíz" del .zip entregado.
- La puntuación de cada apartado está calculada *sobre 10 puntos*, aunque supondrá el 50% de la nota final.

Consejos

Es preferible realizar el examen en el orden siguiente:

1. Implementar cada uno de los apartados. Tras la implementación de cada parte, comprobar que el código compila y funciona y guardar una copia del mismo. De este modo siempre tendremos algo que entregar que sabemos que compila y que funciona (aunque no esté completo).
2. Revisar y corregir los posibles errores o deficiencias que se arrastren de la práctica base, pues ese código también se evalúa.

Instrucciones de entrega

- Para entregar la solución al examen, crea un fichero zip. En él debes incluir todo el proyecto una vez limpiado de archivos intermedios, y con el fichero `alumnos.txt` con tu nombre completo.
- Nombra al fichero `NN_Apellido1Apellido2.zip`, donde `NN` indica el número de grupo (con dos dígitos).
- La estructura del fichero será la misma que la utilizada durante el curso.
- Si optas por escribir la descripción de los pasos y modificaciones realizadas durante el examen en un fichero de texto, inclúyelo en el raíz del archivo comprimido.
- Para entregar el examen, se utilizará el mecanismo de entregas disponible en el laboratorio. En particular, cerca del final del examen se habilitará la unidad `U:` en la que deberás dejar la solución al examen. Si deseas entregarlo antes, indícaselo al profesor para que habilite la unidad.
- Antes de abandonar el laboratorio debes pasar por el puesto del profesor para asegurarte de que lo que se ve en el puesto del profesor es lo que has entregado y firmar en la hoja de entregas.

Tecnología de la Programación - Curso 2014/2015
Grado en Ingeniería Informática, Doble grado en Matemáticas-Informática
Examen de junio - Parte práctica - Duración: 2,5 horas.

Enunciado

We are going to build a standalone application that (1) creates m games of the same type; (2) executes n random moves in total, alternating between the games after each move; (3) collects game statistics while the games are played; and (4) at the end prints these statistics.

It is important that assignment 5 keeps working normally – both in the console and window modes.

[2 puntos] Parte A

Modify the class `Game` to internally maintain a list of games that are currently played, instead of only one as it is now. For each game you should keep all relevant information such as: (1) the game rules; (2) the turn/winner; (3) the undo stack; and (4) the status (finished or not). Every game should be assigned a unique integer identifier.

Modify the `reset` method, and the constructors if needed, such that (1) it deletes all currently available games; and (2) initializes the game list with the new game passed to `reset`. This game must have the identifier 0. This game is set as the currently played game. Note that the observers are common to all games.

[2 puntos] Parte B

Add the following two new methods to class `Game`:

- `int addNewGame(GameRules g)`

  Saves the current game, so it can be restored if needed, and then adds the new game `g` to the list of games together with its corresponding information (turn, undo stack, etc.). Finally sets it as the currently played game. The game should be assigned a new identifier and returned to the caller.

- `void switchGame(int i)`

  Saves the currently played game, and restores the game with identifier `i`. If no game with such identifier exists an appropriate exception should be thrown.

[2 puntos] Parte C

Modify the `GameObserver` interface to include the following two new methods:

- `void onAddNewGame(....)`

  This method should be called when new game is added (in `addnewGame` and `reset`).

- `onSwitchGame(....)`

  This method should be called when switching to a new game (in `switchGame)`.

You should decide which information you need to pass to these methods, depending on what you need for implementing the view the we explain next.

Write a new view class, called `StatView`, that collects statistics on the game in order to print some statistics as indicted next. This class should have a public method "`void showStats()`" that prints, for each game, the number of moves that have been made by each player so far, the player who playes next or the winner if the game is over, and the current board.

[2 puntos] Parte D

Write a class `ExamMain`, with a `main` that receives 3 integer numbers from the command line:

- The first one, call it T, can be a number between 0 and 3, and it indicates which game the use want to run where 0 means `Connect4`, 1 means `Complica`, 2 means `Gravity` with 9×9 board, and 3 means `Reversi`.
- The second one, call it n, is the number of instance of game T that we want to play (in parallel)
- The third one, call it m, is the number of moves we want to make (in total for all games)

The `main` method should create n games of type T, and make a loop for m iteration where in each iteration we make move in the currently played game (if not finished), and switch to the next game (When arriving to the last one we switch to the first).

When the loop terminates, `main` should print the game statistic using the view developed in part C.