Programació de readline amb capacitats d'edició (Model/View/Controller)

Introducció a les pràctiques

Les pràctiques de l'assignatura proposen la programació de mini-aplicacions que il·lustrin patrons de disseny de software. Encara que s'han tractat de buscar exemples que puguin resultar interessants, no deixen de ser l'excusa per programar aquells patrons, tècniques, metodologies que s'han mostrat útils al llarg de l'història del desenvolupament del software i que per tant tot programador hauria de conèixer.

Els enunciats no són ni auto-continguts ni estrictament dirigistes. El lema no seria cenyeixi's vostè a l'enunciat sinó prengui's vostè l'enunciat com a punt de partida i investigui. Creiem que aquest plantejament és molt més instructiu. Qualsevol problema de disseny i programació de software te punts subtils davant dels quals el programador ha de parar-se a pensar cóm resoldre'ls. De poc serviria que aquests punts es donguéssin resolts d'entrada. Solament quan un s'enfronta a aspectes de disseny o programació no trivials és quan un progressa com a programador o enginyer de software. Es amb aquesta filosofia que estan plantejades les pràctiques.

El paper dels professors consisteix en proporcionar els coneixement tècnics necessaris per a la seva correcta resolució i en assessorar o donar pistes de resolució en els punts subtils que s'acaben de mencionar. Els criteris d'avaluació tindran sobre tot en compte l'interès i iniciativa per l'assignatura i la resolució de les pràctiques i exercicis proposats.

Model/View/Controller

En aquesta primera pràctica es vol programar el patró *Model/View/Controller* d'importància cabdal en qualsevol programa amb interfície gràfica.

Aquest patró es popularitza amb l'aparició del llenguatge *Smalltalk* als anys 80. Es considera important separar el codi de presentació (*View*) que pot presentar-se de varies formes, textual, gràfic, etc; del codi que representa l'estat del programa (*Model*). El codi d'entrada (*Controller*) modifica l'estat (*Model*) però no modifica directament la presentació (*View*) sinó que s'informa adequadament a la presentació de que hi han hagut canvis en el model per tal de que s'actualitzi.

Es considera unànimement que aquesta separació de codis simplifica enormement el manteniment de l'aplicació, d'aquí la importància que te en la pràctica. Avui en dia qualsevol programa amb interfície de presentació: un processador de text, un visor de pdf, etc, es programa amb el patró *MVC*.

Exemple

Per tal de programar el patró *MVC* amb un exemple senzill es proposa programar una classe `EditableBufferedReader` que anul·li el mètode `readline` amb les següents capacitats d'edició:

- **right, left**: caràcter dreta, caràcter esquerra amb les fletxes.
- **home, end**: principi, final de línia.
- **ins**: commuta mode inserció/sobre-escriptura.
- **del, bksp**: esborra caràcter actual o caràcter a l'esquerra.

Al haver de detectar aquestes tecles la consola s'ha de passar de mode `cooked`, el mode per defecte on els caràcters es llegeixen després de prémer `cr`, es a dir línia a línia, a mode `raw`, on els caràcters es llegeixen immediatament després de ser premuda la seva tecla. Malauradament no existeix en les apis standard de Java mètodes que permetin fàcilment aquestes conversions.

Llavors, cóm es podria programar un editor de text de consola en Java? Una forma senzilla de fer-ho és invocant un programa de consola amb els paràmetres adequats a l'estil `fork/exec` de unix. Aquest programa és `stty -echo raw` per passar de mode `cooked` a mode `raw`.

L'alumne haurà d'investigar les apis per fer `fork/exec` en Java.

Una altre qüestió és la lectura de les tecles de cursor anteriors. Aquestes tecles es llegeixen i s'escriuen com a seqüències d'escape. Una referència pot ser el manual de seqüències d'escape d'*xterm* o be man console_codes. S'haurán d'estudiar aquests documents o de semblants per saber quines seqüències d'escape corresponen a cada tecla tant per lectura com per escriptura.

Els següents apartats no segueixen un ordre estricte. S'han de llegir tots per tenir una visió de conjunt. Hi han apartats obligatoris i opcionals. La primera versió de la pràctica consisteix en les classes `TestReadLine`, `EditableBufferedReader` i `Line` sense *MVC*. El parser de seqüències d'escape a `read` es programa llegint caràcter-a-caràcter sense fer ús de la classe `Scanner`. `Line` encapsula l'estat i és independent de la presentació.

Es demana:

1. Programeu la classe `EditableBufferedReader` que estengui `BufferedReader` amb els mètodes:

   - `setRaw`: passa la consola de mode `cooked` a mode `raw`.
   - `unsetRaw`: passa la consola de mode `raw` a mode `cooked`.
   - `read`: llegeix el següent caràcter o la següent tecla de cursor.
   - `readLine`: llegeix la línia amb possibilitat d'editar-la.

   Fixeu-vos que `read` llegeix caràcters simples o símbols— les tecles d'edició— com a sencers. Caldrà que els sencers retornats com a símbols no es solapin amb els sencers retornats com a caràcters simples.

   El reconeixement de símbols s'ha de fer de dues formes: la primera versió utilitzant tècniques de *parsing* semblants a les vistes a classe i després fent servir la classe `Scanner` i *expressions regulars*. S'ha de comparar la complexitat de les dues implementacions i treure'n conclusions.

   La classe `Scanner` te les seves subtileses. En particular no esta massa pensada per fer-la servir com a parser general. Una discussió dels seus problemes es troba a Scanner_bugs. Trobeu una implementació que solventi aquestes subtileses.

2. Programar la classe `Line` que mantingui l'estat de la línia en edició amb els seus corresponents mètodes.

3. Fer servir com a classe principal `TestReadLine` que es proporciona feta:

   ```
   import java.io.*;

   class TestReadLine {
     public static void main(String[] args) {
       BufferedReader in = new EditableBufferedReader(
         new InputStreamReader(System.in));
       String str = null;
       try {
         str = in.readLine();
       } catch (IOException e) { e.printStackTrace(); }
       System.out.println("\nline is: " + str);
     }
   }
   ```

   tan sols caldrà canviar el constructor per `EditableBufferedReader`.

4. Programeu l'exemple sense fer servir el patró mvc. Fixeu-vos com es barregen en `EditableBufferedReader` invocacions a `Line` per mantenir l'estat i invocacions a les seqüències d'escape per actualitzar la presentació. Aquesta barreja de codi fa que el programa sigui poc mantenible.

5. Programeu l'exemple fent servir el patró mvc i Observer/Observable. Programeu tres classes `Line` (Model), `Console` (View) i `EditableBufferedReader` (Controller) més la de test `TestReadLine`.

6. Cóm programaríeu suport d'edició de línies multilínea? La Vista haurà de tenir estat: parells de coordenades inicial, actual, final i tamany de la finestra. Aquestes coordenades s'expressaran en files i columnes, començant per u. Aquest apartat te una certa complexitat i es proposa com a part opcional.

7. Programeu un suport bàsic de ratolí. Per exemple un clic del botó esquerre hauria de posicionar el cursor de la línia—opcional—

8. Programeu un suport de formulari bàsic—també opcional—. Per exemple, una calculadora bàsica de sencers, l'esquelet de la qual podria ser quelcom semblant a:

   ```
   // define widgets

   Form f = new Form().add(new Widget[] {
       new Field("first", "first operand: "),
       new Field("second", "second operand: "),
       new Field("operator", "operator: "),
       new Label("result", "result: ")
   });

   // event loop
   while (f.read()) { // true: form ready, false: EOF
       // Read first operand
       ...

       // Read operator
       ...

       // Read second operand
       ...

       // Evaluate result
       result = calcula(operator, first, second);

       // Write result
       ...
   }

   // close form
   f.close();
   ```

   Fixeu-vos que hi ha una part declarativa de definició de `Widgets`:

   - Tres classes: `Form`, formulari i contenidor de `Widgets`, `Field`, camp amb un nom—p.e. `operator`—, una etiqueta imprimible—p.e. `second operand:`— i una línia de text d'entrada, i `Label`, igual que `Field`, però amb una línia de text de sols sortida—p.e. per mostrar missatges d'error o el resultat—

   i una part operativa de processat de les dades del formulari:

   - Lectura del camps, càlcul i escriptura del resultat

   Teniu un `.jar` executable d'aquest exemple aquí

9. Fixeu-vos ara el senzill que seria programar un editor de text bàsic de consola. Aquesta part és opcional. Deixeu-la per el final de l'assignatura si teniu temps i interès.

10. L'API `Observer/Observable` ha estat desaprovada a `Java9`. Reprogrameu el patró `MVC` amb l'API `PropertyChangeListener/PropertyChangeSupport`, més flexible, de *Java beans*.

Francesc Oller