Laberint Bàsic

En aquesta sessió i les dos següents desenvoluparem un projecte complet. Es tracta d'un joc conversacional (un joc textual) en el qual un aventurer es desplaça per un laberint i recull o deixa els objectes (*items*) que va trobant per les sales del laberint. El joc té un menú d'opcions per mitjà del qual l'usuari/a dona ordres a l'aventurer.

- Les tres sessions del projecte són:
- Laberint bàsic (aquesta).
- Laberint estès. A més dels items de la primera sessió, el laberint també pot contenir elements secrets.
- Laberint amb entrada/sortida. S'afegeixen opcions per llegir i escriure en fitxers.

Objectius

Els objectius d'aquesta sessió són:

- Traduir un diagrama de classes UML a codi Java correctament, utilitzant també el javadoc.
- Crear un programa complet a partir de les interaccions entre objectes de diverses classes.

Material que s'adjunta

El material que s'adjunta per fer aquesta sessió és el següent:

- Aquest enunciat, amb el diagrama de classes UML del codi d'aquesta sessió.
- Un projecte NetBeans "LaberintBasic_plantilla" incomplet. Les classes i mètodes que es donen fets són: **Controller**, **UserInterface** i **IOoperations**; també es donen els mètodes "toString()" de les classes **Room** i **Item**, i el mètode "inventoryToString()" de la classe **Adventurer**. I la classe **LaberintException**.
- El *javadoc* complet de la sessió (per obrir-lo, busca i clica index.html).
- Un exemple d'execució: "ExempleExecucio.txt", perquè pugueu comprovar el funcionament del vostre programa.

Punt de partida

Un dels jocs mítics de l'alba de la informàtica, de mitjans dels anys 70, és l'anomenat *Colossal Cave Adventure*[1], un joc que es podria considerar com el precursor de les aventures conversacionals. En aquest joc, mitjançant una interfície totalment en mode text, el jugador dona ordres al seu aventurer mitjançant comandes, que li permeten observar el seu entorn, desplaçar-se per les sales que conformen el laberint, i realitzar accions amb diferents elements dins de cada sala. Les aventures conversacionals van evolucionar cap a diversos formats, com les aventures gràfiques o els MUDs (*Multi-User Dungeons*). Aquests darrers es poden considerar els rebesavis dels jocs de rol online, com el World of Warcraft. Actualment, l'avenç de la tecnologia ha relegat al desús aquesta mena d'interfícies només de text (ja fa més de 30 anys de tot plegat!), però si teniu curiositat, encara podeu jugar amb elles a:

http://www.web-adventures.org

Com la majoria de jocs, aquest és un escenari en el qual és molt senzill aplicar l'orientació a objectes, ja que els elements que interactuen són fàcilment identificables (objectes), tenen unes propietats molt concretes (atributs), i reben ordres directes (mètodes). Per tant, en aquest exercici completareu el motor bàsic del que seria una petita aventura conversacional.

En el joc que aneu a implementar, l'objectiu és simplement vagar per l'entorn i anar recopilant *items*, que tenen un cert valor i una certa càrrega (*encumbrance*). El diagrama estàtic UML proposat per al joc és el que mostra

[1] http://en.wikipedia.org/wiki/Colossal_Cave_Adventure

la Figura 1.

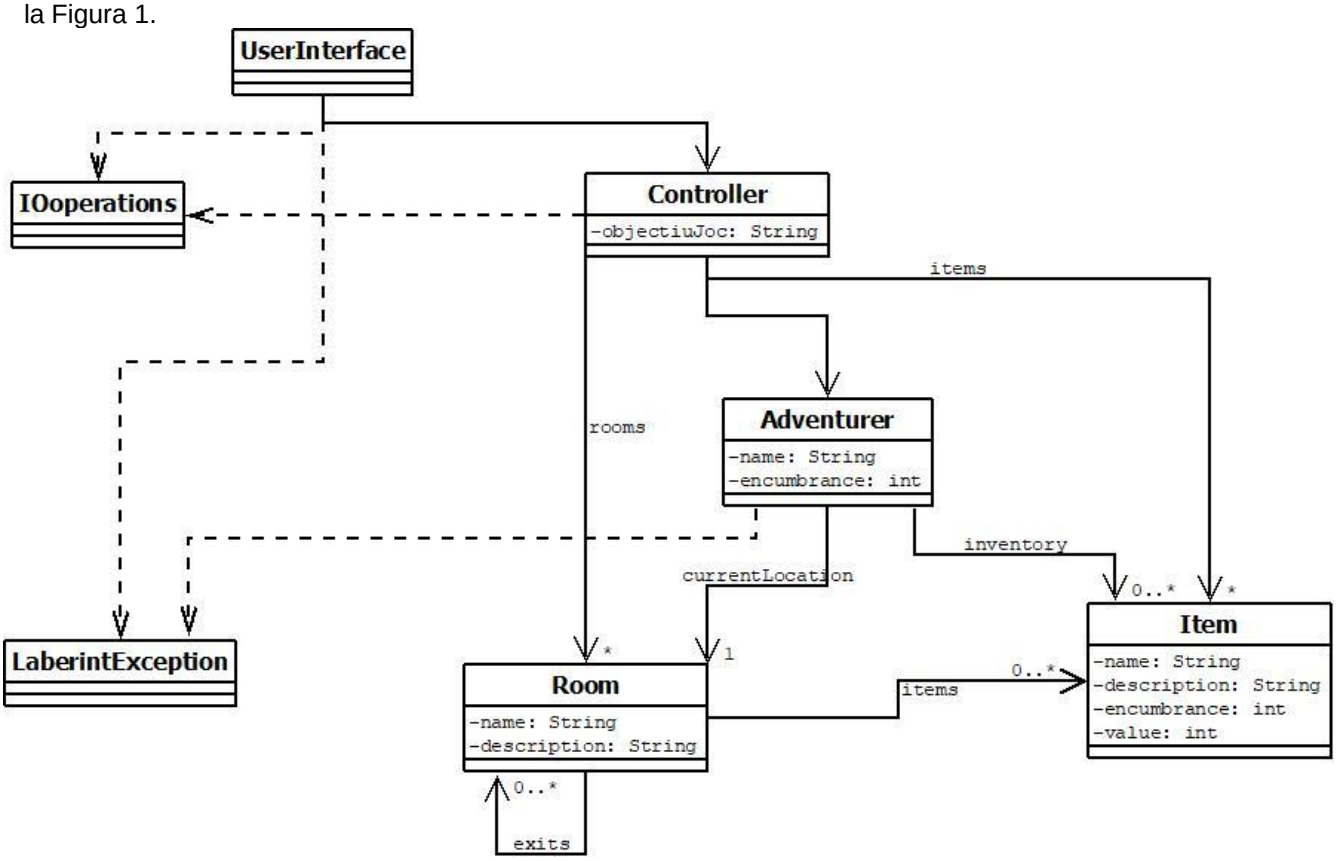

Figura 1. Diagrama de classes del programa

Associació "exits"

L'associació "exits" connecta una sala donada amb totes les sales a les quals es pot accedir des d'aquesta sala. Des de la sala origen, cada sortida o accés s'identifica amb un nom. Aquesta associació s'ha d'implementar amb un Map<String,Room> on la clau és una String (el nom de la sortida o accés), i Room és la sala destí a la qual s'accedeix per aquesta sortida.

En el diagrama es pot veure que les entitats principals de l'aplicació són l'aventurer, les sales que composen el laberint per on es mou (connectades entre si), i els *items* que, o bé estan en alguna sala, o bé els transporta l'aventurer.

Cal tenir present que les sales no conformen una graella amb una topologia predefinida, i que les connexions no cal que siguin bidireccionals. Pot haver-hi sortides en un únic sentit.

Tasques a realitzar

Aquesta sessió es divideix en un seguit d'apartats diferenciats, que convé resoldre en l'ordre d'exposició.

Apartat 1: Funcionament del programa

Copieu el projecte NetBeans "LaberintBasic_plantilla" i la carpeta "javadoc" al disc local de l'ordinador. Obriu el projecte amb el NetBeans i canvieu-li el nom: "LaberintBasic_<Nom><Cognom>. Primer de tot, estudieu les classes **UserInterface, Controller** i **IOoperations,** que s'us donen acabades, per tenir una idea aproximada de quin és el funcionament del programa.

L'aplicació acceptarà les ordres llistades a la taula següent. Cada ordre es pot expressar de diverses maneres: l'ordre completa, només la inicial, o l'ordre en altres idiomes (vegeu les constants definides a la classe **UserInteface**).

Comanda	**Acció**
Mou *nomSortida*	L'aventurer es desplaça cap a la sortida indicada.
Agafa *nomItem*	L'aventurer agafa l'item indicat i el guarda al seu inventari.

Deixa *nomItem*	L'aventurer deixa l'item indicat a la sala actual.
Inventari	Llista els items que poseeix l'aventurer en aquell moment.
Ajuda	Mostra les ordres disponibles.
Fi	Acaba la partida.

Apartat 2: Generació inicial de les classes

Inicialment, com que el projecte que se us dona està incomplet, apareixen molts errors a NetBeans. Falta completar les classes **Adventurer**, **Room** i **Item**. La vostra primera tasca és generar el codi indispensable perquè desapareguin tots els errors. Per fer-ho caldrà:

1. Definiu tots els atributs explícits de les classes incompletes, d'acord amb el diagrama de classes (*private*...).
2. Definiu totes les constants descrites dins el *javadoc* (*public static final*...). Fixeu-vos que en el *javadoc* no apareix el modificador *final*, però vosaltres li heu de posar.
3. Definiu tots els atributs corresponents a la implementació de les associasions del diagrama de classes (recordeu que la relació "exits" de **Room** s'implementa amb un **HashMap**).
4. Definiu els constructors. Mireu el *javadoc*.
5. Definiu la capçalera de tots els mètodes de totes les classes, de moment amb codi buit. Mireu el *javadoc*. En el cas de mètodes que retornen algun valor, podeu posar temporalment que retornen 0 o *null,* perquè no hi hagi un error de compilació.

L'objectiu final d'aquest apartat és que NetBeans no mostri cap error de compilació.

Apartat 3: Desplaçament entre sales

Comenceu codificant tots els mètodes que us faci falta per poder provar només el desplaçament de l'aventurer entre sales (comanda "mou nomSortida"). Per això, us caldrà implementar, com a mínim, els mètodes de la classe **Adventurer** que es llisten tot seguit.

- Room **getCurrentLocation**()
- int **move**(String exitName)
- String **getName**()

Per fer que aquests funcionin, caldrà implementar també alguns dels mètodes de la classe **Room** (és tasca vostra veure quins).

Per tal de que l'aventurer es pugui moure per les sales cal crear el laberint. Això ho fa el mètode "createDungeon()", de la classe **Controller**, i per tant ja està fet, però reviseu el codi d'aquest mètode per veure com es fa.

En concret, la connexió entre sales requereix el mètode "connect()" de la classe **Room** (aquest l'heu d'implementar vosaltres). El mètode "connect()" s'invoca sobre una sala (la sala origen de la connexió), i rep com a paràmetres la sala destí, i el nom de l'accés (o sortida) que porta de la sala origen a la sala destí.

Quan s'ordena que un aventurer es mogui, pot donar-se el cas que la sortida indicada no existeixi. En aquest cas, el programa no ha de fer cap acció amb l'aventurer i ha de comunicar l'error a l'usuari/a. Existeixen diferents maneres de que una classe "avisi" al programa principal (la interfície d'usuari). Aquí usarem la més senzilla i típica, que és mitjançant excepcions de la classe LaberintException que portin com a missatge una explicació de l'error:

```
throw new LaberintException("Descripció de l'error");
```

Estudieu amb deteniment el codi del mètode "start()" de la classe **UserInterface** on es captura l'excepció i es mostra el missatge pertinent, per avisar a l'usuari/a.

Apartat 4: Agafar ítems

Afegiu ara la possibilitat d'agafar ítems i veure l'inventari actual de l'aventurer. Primer completeu la classe **Item**, i a continuació feu el següent mètode de la classe **Adventurer**:

- int **pickUp**(String itemName)

Fixeu-vos que l'inventari de l'aventurer té una capacitat limitada. La càrrega acumulada (*encumbrance*) que l'aventurer porta a sobre *no* ha de superar **MAX_ENCUMBRANCE**. Això vol dir que si la càrrega actual sumada a la carrega de l'item que anem a agafar superen el límit, aleshores no podem agafar aquest item.

De la mateixa manera que passava amb "move()", en aquest mètode cal controlar el resultat de l'acció mitjançant una LaberintException, amb el missatge d'error pertinent.

Apartat 5: Deixar ítems

Finalment, afegiu la possibilitat de deixar ítems de l'inventari a la sala actual. Per fer-ho, la classe Adventurer té el mètode:

- int **drop**(String itemName)

En aquest cas, cal llençar una LaberintException amb el missatge pertinent si l'item no existeix a l'inventari.

Apartat 6: Organització de les classes

Ara que el programa d'aquesta sessió està acabat, observeu la funció que fa cada una de les classes en el projecte. Les classes **Adventurer**, **Room** i **Item** gestionen l'estat (atributs) i el comportament (mètodes) dels element propis del joc. D'alguna manera, aquestes classes descriuen tot allò que es propi de l'aplicació (el domini o *model* de l'aplicació): l'aventurer, les sales que conformen el laberint, i els items que es troben a les sales, o que l'aventurer porta en el seu inventari.

Les classes **UserInterface**, **Controller** i **IOoperations** són classes genèriques que trobem sovint en una aplicació ben organitzada.

La classe **IOoperations** encapsula les operacions d'entrada sortida de manera que sigui fàcil fer canvis sense haver de modificar tot el programa. A la sessió "LaberintAmbES", per exemple, afegirem mètodes per a llegir i escriure en un fitxer.

La classe **UserInterface** és l'encarregada de gestionar tot el que és interacció amb l'usuari/a (menú principal, lectura de les comandes que l'usuari/a introdueix, missatges d'error i presentació d'informació per pantalla). El motiu és que, si en un futur volguessim canviar la interficie textual actual per una interficie gràfica, per exemple, només hauriem de modificar aquesta classe.

El mètode "main()" i el bucle principal de l'aplicació (el mètode "start()", en el nostre cas) es troben habitualment a la classe **UserInterface**. El bucle principal implementa el menú de comandes de l'aplicació. Per cada comanda del menú (com ara "Mou nomSortida"), la classe **UserInterface** invoca un mètode de la classe **Controller** ("controller.move()"), i gestiona les excepcions. Els mètodes d'inicialització, com ara "createDungeon()" també s'invoquen des del menú principal, i s'implementen al controlador.

La classe **Controller** té, com a mínim, un mètode per a cada una de les comandes del menú (i per cada una de les ordres d'inicialització). El controlador és qui "mou els fils" (invoca els mètodes) de les classes del domini (**Adventurer**, **Room** i **Item**), de tal manera que cada comanda s'executi de la manera adequada.

A vegades, com en el cas de "move()", el mètode del controlador és molt simple (el codi consisteix únicament en invocar "adventurer.move()"), però tot i així, l'existència de la classe **Controller** és primordial perquè

l'aplicació estigui ben organitzada.

Exercicis addicionals (opcional)

Modifiqueu el mètode “createDungeon()”, a la classe **Controller**, per crear el laberint que més us agradi a vosaltres. Per exemple, podrieu fer una part del mapa del joc original (http://www.spitenet.com/cave), que es mostra a continuació: