PRG (E.T.S. d'Enginyeria Informàtica) - Curs 2015-2016

Pràctica 4. Tractament d'excepcions i fitxers

(3 sessions)

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Índex

1 Context i treball previ

En el marc acadèmic, aquesta pràctica correspon al "*Tema 3. Elements de la POO: herència i tractament d'excepcions*" i al "*Tema 4. E/S: fitxers i fluxes*". L'objectiu principal que es pretén aconseguir amb ella és que reforces i fiques en pràctica els conceptes introduïts en les classes de teoria sobre el tractament d'excepcions i la gestió de l'E/S mitjançant fitxers i fluxes. En particular, en finalitzar aquesta pràctica has de ser capaç de:

- Llançar, propagar i capturar excepcions local i remotament.
- Llegir/escriure des d'/en un fitxer de text.

- Llegir/escriure des d'/en un fitxer binari d'objectes.

- Tractar les excepcions relacionades amb l'E/S.

Per això, durant les tres sessions de pràctiques, es treballarà amb una aplicació que gestiona comptes bancaris, experimentant amb capturar i llançar excepcions, i afegint la possibilitat de realitzar entrada i sortida de dades amb fitxers.

Perquè aprofites al màxim les sessions de pràctiques, t'aconsellem que, abans de la primera sessió realitzes una lectura comprensiva de les seccions 2 a 5 d'aquest butlletí i intentes resoldre les activitats 1 i 2. I abans de la sessió 2 completes la lectura del butlletí.

2 Plantejament del problema

Com ja hem avançat, en aquesta pràctica es treballarà amb una aplicació que simula la gestió dels comptes d'un banc. L'aplicació, inicialment, consta de les següents classes:

- La classe d'utilitats `LecturaValida` que permet realitzar la lectura validada de dades de tipus primitius des de l'entrada estàndard.

- La classe `Compte` que permet representar un compte bancari mitjançant els atributs `saldo` (`double` que indica el saldo disponible al mateix) i `numCompte` (`int` que representa el número de compte). D'un compte es poden consultar les seues dades, obtenir una `String` de les mateixes, ingressar i retirar una quantitat de diners.

- La classe `Banc` que permet representar un número màxim de comptes (`MAX_COMPTES`) mitjançant els atributs `comptes` (un array d'objectes `Compte`) i `numComptes` (`int` que indica el número actual de comptes al banc i també el primer índex lliure de l'array `comptes`). D'un banc es pot consultar quants comptes té, comprovar si existeix algun compte amb un número donat, afegir un nou compte i obtenir una `String` amb la informació de tots els seus comptes.

- La classe `GestorBanc` que permet gestionar els ingressos i disposicions de diners dels comptes d'un banc. De tots ells, en un moment determinat de l'execució, només pot treballar amb un, el compte actiu. Així, aquesta classe permet crear o buscar un compte bancari, que passa a ser el compte actiu; fer ingressos i disposicions de diners del compte actiu, consultar les seues dades i les de tots els comptes del banc.

Per començar

- Crea un projecte *BlueJ* `pract4` específic per a aquesta pràctica. Afegeix a `pract4` els fitxers de codi `LecturaValida.java`, `Compte.java`, `Banc.java` i `GestorBanc.java`, i els fitxers de text `comptes.txt` i `badinput.txt`, disponibles per a la seua descàrrega en `Recursos/Laboratorio/Práctica 4` de `PoliformaT` de `PRG`.

3 Detecció d'errors lògics i en temps d'execució

Una vegada resolts els errors de compilació, és possible que en el nostre codi tinguem errors d'execució que provoquen un mal funcionament del mateix. Es distingeix entre els anomenats errors en temps d'execució i errors lògics. Els primers provoquen la detenció de l'execució; els segons, els més difícils de descobrir, consisteixen en que els resultats obtinguts, o els processos realitzats, pel programa o una part d'aquest no són correctes encara que el programa pot semblar que funciona correctament.

Activitat 1: detecció d'errors lògics a la classe `Compte`

- Revisa la classe `Compte` i fixa't en les precondicions del constructor i dels mètodes `ingressar(double)` i `retirar(double)`.
- Crea, al *banc d'objectes* (*Object Bench*) de *BlueJ*, un objecte `Compte` amb un número de compte de 3 dígits i un saldo negatiu. Pots fer-ho?
- Invoca al mètode `ingressar(double)` amb una quantitat negativa. És possible?
- Invoca al mètode `retirar(double)` amb una quantitat superior al saldo del compte. Què ocorre?

Són errors lògics que donen lloc a resultats no desitjats. Per evitar-los, hem d'assegurar que els arguments compleixen les precondicions. A la classe `LecturaValida` disposes d'alguns mètodes per realitzar la lectura, des de l'entrada estàndard, de dades vàlides de tipus `int` i `double`.

Activitat 2: detecció d'errors en temps d'execució a la classe `LecturaValida`

- Fixa't en el mètode `llegirDoublePos(Scanner,String)` que permet realitzar la lectura d'un valor de tipus `double` ≥ 0, amb un bucle `do-while` que es repeteix mentre el valor llegit siga < 0. El segon argument de tipus `String` és el missatge de petició del valor.
- Prova aquest mètode a la *zona de codi* (*Code Pad*) de *BlueJ*. Per això, executa les instruccions següents:

```
import java.util.*;
Scanner t = new Scanner(System.in).useLocale(Locale.US);
double realPos = LecturaValida.llegirDoublePos(t, "Valor: ");
```

- Des de la *finestra del terminal* de *BlueJ*, introdueix valors reals negatius. Observa que l'execució del mètode no acaba fins que no introdueixes un valor positiu o zero.
- Torna a executar-lo. Què passa si no introdueixes un valor de tipus `double`, per exemple, si introdueixes el caràcter `'k'`?

L'execució es deté i es mostra un missatge indicant què ha passat i en quina instrucció del codi. Es tracta d'un error en temps d'execució o *excepció*. En les seccions que segueixen veurem com gestionar aquests tipus d'errors.

4 Tractament d'excepcions predefinides

Com ja saps, quan es produeix una fallada o una anomalia en l'execució d'un programa Java, ocorre una *excepció*. El codi que va causar aquesta fallada immediatament deixa d'executar-se i Java intenta gestionar l'excepció, buscant el codi que pot resoldre-la (en el mètode on es va produir la fallada o fins a trobar un mètode que la resolga). Si no pot trobar aquest codi, el programa es deté i es mostra un missatge descrivint què va fallar i on va ocórrer el problema.

En Java es pot distingir entre excepcions *checked* o comprovades, el tractament (captura o propagació) de les quals és obligatori, i excepcions *unchecked* o no comprovades, en les que el seu tractament no és obligatori però sí és possible. El tractament d'excepcions és una forma de que el programa es puga recuperar de la fallada o, al menys, acabar d'una manera acceptable.

En aquesta secció et proposem una sèrie d'activitats que et guiaran en el tractament d'algunes excepcions predefinides en Java, mitjançant la seua captura utilitzant un bloc `try-catch-finally`.

Activitat 3: anàlisi del mètode `llegirInt(Scanner,String)`

- Fixa't ara en el mètode `llegirInt(Scanner,String)` de la classe `LecturaValida` que permet realitzar la lectura d'un valor de tipus `int`. La lectura es realitza amb el mètode `nextInt()` de la classe `Scanner`, que pot llançar l'excepció `InputMismatchException` si el valor introduït per l'usuari no és un enter.

- Consulta la documentació del mètode `nextInt()` (i les excepcions que pot generar) en l'API de Java de la classe `Scanner`:
  `http://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html`

- Consulta també la documentació de la classe `InputMismatchException`, comprovant que és una excepció *unchecked* (derivada de `RuntimeException`) i que la seua situació en la jerarquia de classes coincideix amb la que es mostra a continuació.

```
java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--java.lang.RuntimeException
            |
            +--java.util.NoSuchElementException
                |
                +--java.util.InputMismatchException
```

- El mètode `llegirInt(Scanner,String)` captura (`catch`) aquest tipus d'excepció, mostrant un missatge d'error per tal d'indicar a l'usuari quina acció correctiva és necessària.

- A la *zona de codi* de *BlueJ*, executa el mètode `llegirInt(Scanner,String)` i, des de la *finestra del terminal*, introdueix un valor no enter, per exemple, un valor real. Observa el missatge mostrat i que l'execució no acaba fins que no introdueixes un valor enter.

- Fixa't ara en la clàusula `finally` del mètode `llegirInt(Scanner,String)`. Fins i tot quan es produeix un error en un mètode, pot haver instruccions que es requereixen abans que el mètode o programa acabe. La clàusula `finally` s'executa si totes les instruccions del bloc `try` s'executen (i cap bloc `catch`) o si es produeix una excepció i un dels blocs `catch` s'executa. En el mètode `llegirInt(Scanner,String)`, per a qualsevol possible lectura, sempre s'executa la instrucció `tec.nextLine()` de la clàusula `finally`, permetent descartar el salt de línia que s'emmagatzema en el buffer d'entrada quan l'usuari prem la tecla *Enter* o el token incorrecte que fa que es llance l'excepció `InputMismatchException`, evitant que el mètode entre en un bucle infinit.

Activitat 4: tractament d'excepcions en `llegirDoublePos(Scanner,String)`

- Completa el mètode `llegirDoublePos(Scanner,String)` de la classe `LecturaValida` per tal que capture l'excepció `InputMismatchException` si el valor introduït per l'usuari no és un `double`, de manera similar al mètode `llegirInt(Scanner,String)`, mostrant un missatge d'error apropiat en lloc d'avortar l'execució.

El que acabes de fer és afegir un controlador d'excepcions per a detectar una excepció a nivell local, és a dir, en el mateix mètode on es produeix la fallada.

Activitat 5: tractament d'excepcions en `llegirInt(Scanner,String,int,int)`

- Completa el mètode `llegirInt(Scanner,String,int,int)` de la classe `LecturaValida` perquè capture l'excepció `InputMismatchException` si el valor introduït per l'usuari no és un `int`, de manera similar al mètode `llegirInt(Scanner,String)`, mostrant un missatge d'error apropiat en lloc d'avortar l'execució.

- Aquest mètode, a més, ha de controlar que el valor introduït està en el rang `[lInferior,lSuperior]`. Hi ha dues formes de realitzar aquest control: la primera consisteix en afegir la condició apropiada a la guarda del bucle, com en el cas del mètode `llegirDoublePos(Scanner,String)`, i la segona en llançar una excepció. Optem per aquesta última. Afegeix una instrucció condicional tal que si el valor introduït no està en el rang anterior, llance l'excepció `IllegalArgumentException`, usant una instrucció `throw`, amb un missatge que indique que el valor llegit no està en aquest rang.

- Ara, afegeix una clàusula `catch` per a capturar localment aquesta excepció, de forma similar a la captura de l'excepció `InputMismatchException`, mostrant el missatge de l'excepció mitjançant el mètode `getMessage()` (heretat de la classe `Throwable`).

El que acabes de fer és un bloc multi-catch. Recorda que sempre que apareix una clàusula `try`, ha d'existir almenys una clàusula `catch` o `finally`. Nota que, per a una única clàusula `try`, poden existir tantes `catch` com siguen necessaries per tractar les excepcions que es puguen produir al bloc de codi del `try`. Quan al bloc `try` es llança una excepció, els blocs `catch` s'examinen en ordre, i el primer que s'executa és aquell que el seu tipus siga compatible amb el de l'excepció llançada. Així doncs, l'ordre dels blocs `catch` és important. L'excepció més específica ha d'aparèixer abans que qualsevol excepció d'una classe antecessora. Encara que, en aquest cas, l'ordre no és rellevant, en una activitat posterior relacionada amb el tractament de fitxers veuràs un exemple en què sí que ho és.

Recorda també que a partir de la versió 7.0 de Java, un únic bloc `catch` pot utilitzar-se per tal de capturar més d'un tipus d'excepció, separant-les per una barra vertical (|), sempre que no hi haja cap que siga subclasse d'una altra.

Activitat 6: tractament d'excepcions en la classe `GestorBanc`

- Executa la classe `GestorBanc` i, mitjançant el menú, tria l'opció 2 per tal d'ingressar diners (tot i que no s'ha creat cap compte, és a dir, no hi ha cap compte actiu). Fixa't que l'execució s'avorta: no hi ha un compte actiu per fer l'ingrés.

- Fes un cop d'ull al codi de la classe `GestorBanc`. S'utilitza una variable `compte` de tipus `Compte` per representar el compte actiu, que inicialment és `null`.

- En el `case 2` de la instrucció `switch`, observa que al tractar d'ingressar diners en el compte, ja que la variable `compte` és `null` inicialment, es llança l'excepció `NullPointerException`.

- Completa el `case 2` de la instrucció `switch` per a capturar aquest tipus d'excepció, mostrant el missatge "*ERROR: No hi ha cap compte actiu! Primer busca'l o crea un nou compte*".

- Completa de la mateixa manera les opcions 3 i 5 de la instrucció `switch`.

Activitat 7: prova de la classe `GestorBanc`

- Executa la classe `GestorBanc` i prova les diferents opcions amb dades que conduirien a situacions errònies i comprova que el tractament d'errors realitzat evita el seu mal funcionament.

Activitat 8: propagació d'excepcions i el seu tractament remot

- En el mètode `retirar(double)` de la classe `Compte`, afegeix codi per tal de comprovar si la quantitat que se li passa com a paràmetre és superior al saldo del compte. Si ho és, aleshores llança (`throw`) l'excepció `IllegalArgumentException` amb el missatge "*No es poden retirar més diners dels que hi ha en el compte!*".

- A continuació, aquesta excepció es capturarà en el mètode `main` de la classe `GestorBanc`. Modifica el codi del `case 3` de la instrucció `switch`, eliminant la instrucció condicional i afegint una clàusula `catch` per a capturar aquest tipus d'excepció. Tot el que has de fer és mostrar un missatge d'error juntament amb el missatge de l'excepció.

- Torna a executar el programa. Crea un compte i intenta retirar una quantitat més gran que el saldo disponible. Ara tampoc hauria de ser possible fer-ho.

Fixa't que hem capturat l'excepció en el mètode `main` de la classe `GestorBanc`, en lloc d'en el mètode on passa el problema. Quan es detecta una excepció d'aquesta manera, és a dir, en un mètode diferent d'aquell en què va ocórrer el problema, es diu que la captura es realitza de manera remota. Les excepcions poden ser capturades tant localment com de forma remota segons on tinga més sentit tractar de resoldre el problema.

5 Llegir/escriure des d'/en un fitxer de text

En aquesta secció, veurem com llegir/escriure comptes bancaris des d'/en un fitxer de text, realitzant el tractament de les excepcions relacionades. Per això implementaràs dos mètodes en la classe `Banc` i ampliaràs els casos del mètode `main` de la classe `GestorBanc`.

Activitat 9: lectura dels comptes guardats en un fitxer de text

- Els comptes bancaris estan guardats en un fitxer amb un format determinat. El format indica l'ordre i disposició de la informació en el fitxer. Quan es processen fitxers, hem de ser capaços de llegir fitxers vàlids que segueixen les especificacions de format (i rebutjar els fitxers no vàlids). En el nostre cas, cada línia del fitxer té dos elements d'informació, un número de compte de tipus `int` seguit d'un saldo de tipus `double` (per exemple, vegeu `comptes.txt`, que s'ha descarregat anteriorment). Els fitxers d'entrada poden tenir qualsevol nombre de línies. Es pot utilitzar el mètode `hasNext()` de `Scanner` per llegir mentre quede text en el fitxer per llegir. Per a cada línia, hauràs de separar el número de compte i el saldo. Hi ha dues maneres de fer-ho:

  Primera aproximació La manera més fàcil de llegir informació d'un fitxer requereix que els valors estiguen separats per espais en blanc, tabulacions o salts de línia. El fitxer `comptes.txt` està formatejat d'aquesta manera pel que es pot utilitzar `Scanner` per llegir cada fragment d'informació per separat (és a dir, usant `nextInt()` i `nextDouble()`). Aquest enfocament funciona bé sempre que es vaja amb compte amb el processament del salt de línia, que marca el final de cada línia.

  Segona aproximació Si els valors en un fitxer formatejat estan separats per algun altre delimitador, com una coma, és millor llegir cada línia completa del fitxer per separat (usant `nextLine()`) i després dividir la línia en els fragments apropiats. Hi ha diverses maneres de fer això, però et suggerim que utilitzes el mètode `split` de la classe `String`. Després d'obtenir les cadenes de caràcters amb els números, es poden utilitzar els mètodes `Integer.parseInt(String)` i `Double.parseDouble(String)` per convertir-los a format numèric.

- Afegeix a la classe `Banc` el mètode `public void carregarFormatText(Scanner f)`. Mentre hi haja alguna cosa que llegir del `Scanner f`, aquest mètode llegeix el número de compte (`int`) i el saldo (`double`) d'una línia del fitxer seguint una de les dues aproximacions anteriors. Amb aquestes dades, crea un objecte de tipus `Compte` i l'afegeix al banc (`this`) usant el mètode `inserir(Compte)`.

- Afegeix un `case 7` al `switch` del `main` de la classe `GestorBanc` on es demane a l'usuari el nom del fitxer en el qual estan guardats els comptes. Amb aquest nom crearà un objecte `File`, a partir del qual es crearà un objecte de tipus `Scanner` per fer la lectura (aquest objecte s'haurà d'instanciar especificant `Locale.US` per a que la lectura del nombres reals siga correcta, donat el format d'aquests en el fitxer). En crear aquest objecte `Scanner`, s'intenta localitzar un fitxer al directori on s'executa l'aplicació. Això pot produir una excepció de tipus `FileNotFoundException`, entre altres raons, perquè no existeix el fitxer o no es tenen els permisos d'accés apropiats. Has de capturar l'excepció, informant a

l'usuari que no s'ha localitzat el fitxer i sortir del `case` sense fer cap acció. Si el fitxer s'ha localitzat amb èxit, s'invoca al mètode `carregarFormatText(Scanner)` aplicant-lo sobre el banc que es defineix al `main`. Recorda que, a la clàusula `finally` del `try`, cal comprovar si el `Scanner` s'ha creat i, si es així, tancar-lo.

- Modifica el mètode `menu(Scanner)` de la classe `GestorBanc` afegint l'opció *7) Carregar banc des de fitxer*. Tingues en compte que has afegit una opció i que en la crida a `llegirInt(Scanner,String,int,int)` s'ha d'incrementar el rang de valors permès.

- Per provar la lectura des de fitxer de text, executa el `main` de l'aplicació, tria l'opció de carregar el fitxer `comptes.txt` i mostra tots els comptes del banc (opció 6 del menú) per tal de comprovar que s'han llegit correctament.

- Torna a executar l'opció de carregar el fitxer `comptes.txt` i mostra tots els comptes. Què ocorre? S'han tornat a carregar els mateixos comptes obtenint comptes duplicats. Intenta arreglar-ho modificant el mètode `carregarFormatText(Scanner)`, de manera que es comprove prèviament si el compte existeix a l'array `comptes`, usant el mètode `getCompte(int)`.

- Executa de nou l'opció de carregar el fitxer però ara amb el fitxer `badinput.txt`. Veuràs que es deté l'execució en llegir la línia que conté un número de compte que no és un `int`. Per tal que l'execució no es detinga si el valor llegit no és un `int` o un `double`, captura l'excepció `InputMismatchException` en el mètode `carregarFormatText(Scanner)`. El tipus d'aquesta excepció dependrà de l'aproximació escollida per a implementar el mètode: si vares escollir la primera aproximació, llavors hauràs de capturar l'excepció `InputMismatchException`, però si vares escollir la segona, usant el mètode `split`, llavors s'haurà de capturar l'excepció `NumberFormatException`.

- A més, has d'assegurar que, tant si es produeix una excepció com si no es produeix, es completa la lectura del fitxer (és a dir, el bucle de lectura acaba). Per tal de garantir-ho pots usar una clàusula `finally` on hauràs d'incloure la instrucció necessària per a que l'objecte `Scanner` avance a la següent línia del fitxer, si la hi ha, en qualsevol situació.

- Executa una altra vegada l'opció de carregar el fitxer amb el fitxer `badinput.txt` i mostra tots els comptes. Què ocorre? S'han carregat comptes no vàlids. Modifica el mètode `carregarFormatText(Scanner)` de manera que si la línia que es llegeix no és vàlida, açò és, si el número de compte no té 5 dígits o si el saldo és negatiu, no es carregue aquest compte en el banc.

Activitat 10: escriptura dels comptes en un fitxer de text usant el mètode `toString()` de la classe `Compte`

- Afegeix a la classe `Banc` el mètode `public void guardarFormatText(PrintWriter f)`. Aquest mètode ha d'escriure per a cada compte del banc una línia, en el fitxer representat per l'objecte `PrintWriter f` donat, amb el seu número de compte i el seu saldo, en aquest ordre, i separats per un espai en blanc. Pots usar el mètode `println(String)` de la classe `PrintWriter` i el mètode `toString()` de la classe `Compte`.

- Afegeix un `case 8` al `switch` del `main` de la classe `GestorBanc` on es demane a l'usuari el nom del fitxer en el que es guardaran els comptes. Amb aquest nom crearà un objecte `File`, a partir del que es crearà un objecte de tipus `PrintWriter` per realitzar l'escriptura. En crear aquest objecte, s'intenta crear un fitxer al directori on s'executa l'aplicació i es pot produir una excepció de tipus `FileNotFoundException`, entre altres raons, per falta d'espai en disc o per la no possessió de permisos d'escriptura. Has de capturar l'excepció, informant a l'usuari que no s'ha creat el fitxer. Si s'ha creat amb èxit, invoca el mètode `guardarFormatText(PrintWriter)` aplicat sobre el banc que es defineix en el `main`. Recorda que, en la clàusula `finally` del `try`, cal comprovar si el `PrintWriter` s'ha creat i, en cas afirmatiu, tancar-lo.

- Modifica el mètode `menu(Scanner)` de la classe `GestorBanc` afegint l'opció *8) Guardar banc en fitxer*. Tingues en compte que has afegit una nova opció i que en la crida a `llegirInt(Scanner,String,int,int)` s'ha d'incrementar el rang de valors permès.

- Per tal de provar l'escriptura en fitxer de text, executa el `main` de l'aplicació, crea un parell de comptes nous i guarda els comptes del banc en un fitxer. Obre el fitxer des del terminal i comprova que per cada compte conté una línia amb el número de compte i saldo que has introduït prèviament. A més, comprova que els valors numèrics estan separats per un espai en blanc.

Activitat 11: escriptura dels comptes en un fitxer de text usant el mètode `toString()` de la classe `Banc`

En aquesta activitat vas a modificar el mètode `guardarFormatText(PrintWriter f)` de la classe `Banc` per obtenir una implementació més senzilla, aprofitant el fet que el mètode `toString()` de la classe `Banc` utilitza el `toString()` de la classe `Compte`, tornant un `String` amb tantes línies com comptes té el banc, cada línia amb un número de compte i un saldo separats per un espai en blanc. Justament el format del fitxer `comptes.txt`!

- Modifica[1] el mètode `guardarFormatText(PrintWriter f)` per que s'escriguen d'una sola vegada tots els comptes del banc, utilitzant el mètode `print(String)` de la classe `PrintWriter` i el mètode `toString()` de la classe `Banc`.

- Executa l'aplicació, crea alguns comptes i tria l'opció de guardar en un fitxer. Des del terminal, obre el fitxer que s'ha creat i comprova que conté la mateixa informació que abans i amb el mateix format.

- Executa de nou l'aplicació, però aquesta vegada no creïs cap compte abans de guardar els comptes. Comprova el contingut del fitxer creat. Què s'ha guardat? Si s'intenten carregar els comptes d'aquest fitxer es produirà un error de format. Això és perquè conté la frase que retorna `toString()` quan un banc no té cap compte ("*No hi ha comptes al banc*"). Per solucionar-ho serà suficient considerar que només s'ha de guardar alguna cosa al fitxer si la quantitat de comptes (`numComptes`) és més gran que zero.

- Amb quantes línies de codi has implementat el mètode?

[1] Comenta primer les instruccions del cos del mètode implementat en la activitat 10.

6 Llegir/escriure des d'/en un fitxer binari d'objectes

A la secció anterior hem vist lo fàcil que resulta la implementació del mètode `guardarFormatText(PrintWriter)` usant el mateix format en els fitxers a guardar que en els `String` resultat dels mètodes `toString()`. Aquests mètodes es fan servir per retornar informació continguda en els objectes en un format especificat pel dissenyador. El llenguatge Java proporciona una manera d'automatitzar aquest procés fent servir una classe especial anomenada `Serializable`. Aquesta classe permet obtenir la informació d'un objecte en un format estàndard en Java sempre que tot el seu contingut, inclosos els objectes referenciats, utilitzen el mateix format. Per obtenir aquesta utilitat del llenguatge, s'ha d'afegir `implements Serializable` després del nom de la classe en la seua definició.

En aquesta secció farem servir aquest format estàndard per recuperar/guardar des d'/en un fitxer la informació d'un banc (i per tant, la informació de tots els seus comptes, als que referencia l'array `comptes` de la classe `Banc`. Per això implementaràs dos nous mètodes en la classe `Banc` i modificaràs les opcions 7 (carregar) i 8 (guardar) del `switch` del `main` de la classe `GestorBanc`.

Activitat 12: escriptura d'un objecte `Banc` en un fitxer binari d'objectes

- Afegeix `implements Serializable` en les capçaleres de les classes `Banc` i `Compte`, i la instrucció `import java.io.Serializable` al començament de les mateixes.
- Afegeix el mètode `public void guardarFormatObjecte(ObjectOutputStream f)` a la classe `Banc`. Aquest mètode ha d'escriure l'objecte `Banc` (el mateix objecte invocador del mètode) en el fitxer representat pel fluxe `f`, usant el mètode `writeObject(Object)`. Aquest mètode propaga l'excepció `IOException`; afegeix la clàusula necessària en la capçalera de `guardarFormatObjecte(ObjectOutputStream)` per tal que també propague aquesta excepció.
- Modifica[2] el `case 8` del `switch` de la classe `GestorBanc` perquè després de llegir el nom d'un fitxer i crear objectes de les classes `FileOutputStream` i `ObjectOutputStream`, invoque al mètode `guardarFormatObjecte(ObjectOutputStream)`. Has de tractar les següents excepcions, informant a l'usuari de l'error ocorregut en el procés d'escriptura:
  - `FileNotFoundException` que es llança si l'objecte `FileOutputStream` no localitza el fitxer que se li passa com a paràmetre.
  - `IOException` que pot llançar-se per l'objecte `ObjectOutputStream` o ser propagada pel mètode `guardarFormatObjecte(ObjectOutputStream)`.
- Recorda comprovar si el `ObjectOutputStream` s'ha creat i, si és així, tancar-lo en la clàusula `finally` del `try`. En aquest cas, aquest tancament requereix tractar dins de la mateixa clàusula una `IOException` propagada pel mètode `close()`.
- Executa el `main` de l'aplicació, tria l'opció de carregar el fitxer `comptes.txt` i, a continuació, tria l'opció 8 per a guardar els comptes, en aquest cas, en un fitxer binari d'objectes de nom `comptes.obj`. Ara tens dos fitxers amb la mateixa informació però en diferent

[2]Comenta primer les instruccions de l'escriptura amb fitxers de text implementades en la activitat 10.

format. El fitxer de text és accessible mitjançant qualsevol editor de text mentre que el fitxer binari requereix d'instruccions Java per a la seua lectura.

Activitat 13: lectura d'un objecte `Banc` des d'un fitxer binari d'objectes

- Afegeix el mètode `public void carregarFormatObjecte(ObjectInputStream f)` a la classe `Banc`. Aquest mètode ha de llegir un objecte `Banc` emmagatzemat en un fitxer binari d'objectes, usant el mètode `readObject()`. Assigna l'objecte llegit a una variable `b` que referencie un objecte `Banc`.

- Compila la classe `Banc`. A què es deu l'error que es produeix? Soluciona el problema forçant la conversió del tipus de l'objecte que retorna el mètode `readObject()` al tipus `Compte`.

- La variable `b` referencia a un objecte `Banc` amb tots els seus comptes, però es tracta d'un objecte nou, distint al `Banc this`. Recorre l'array `comptes` de `b` i, usant el mètode `inserir(Compte)`, afegeix els comptes de `b` al banc `this`. Per a cada compte comprova si ja existeix en `this` usant el mètode `getCompte(int)` passant-li com a paràmetre el número del compte de `b`.

- La instrucció `readObject()` pot llançar excepcions de tipus `ClassNotFoundException` i `IOException`, que es propagaran per ser tractades a la classe `GestorBanc`.

- Modifica[3] el `case 7` del `switch` de la classe `GestorBanc` perquè després de llegir el nom d'un fitxer i crear objectes de les classes `FileInputStream` i `ObjectInputStream`, invoque al mètode `carregarFormatObjecte(ObjectInputStream)`. En aquest cas, has de tractar les tres excepcions que segueixen, informant a l'usuari de l'error ocorregut en el procés de lectura del fitxer:

  - `FileNotFoundException` que es llança si l'objecte `FileInputStream` no localitza el fitxer que se li passa com a paràmetre.
  - `IOException` que pot llançar-se per l'objecte `ObjectInputStream` o ser propagada pel mètode `carregarFormatObjecte(ObjectInputStream)`.
  - `ClassNotFoundException` que també pot ser propagada pel mètode anterior si no es pot determinar la classe de l'objecte que s'intenta llegir.

- En quin ordre has de capturar les excepcions anteriors? Recorda que l'excepció més específica s'ha de citar abans que qualsevol excepció d'una classe antecessora. Per assegurar-te que realitzes la seua captura en l'ordre adequat, consulta en l'API de Java la documentació de les classes `ClassNotFoundException`, `FileNotFoundException` i `IOException`.

  La seua situació en la jerarquia de classes de Java és la que es mostra a l'inici de la pàgina següent.

[3] Comenta primer les instruccions de la lectura amb fitxers de text implementades en la activitat 9.

```
java.lang.Object
 |
 +--java.lang.Throwable
     |
     +--java.lang.Exception
         |
         +--java.lang.ClassNotFoundException
         |
         +--java.io.IOException
            |
            +--java.io.FileNotFoundException
```

- Recorda comprovar si el `ObjectInputStream` s'ha creat i, si és així, tancar-lo en la clàusula `finally` del `try`. També en aquest cas, aquest tancament requereix tractar dins d'aquesta clàusula una `IOException` llançada pel mètode `close()`.
- Per provar la lectura des de fitxer binari d'objectes, executa el `main` de l'aplicació, tria l'opció de carregar el fitxer `comptes.obj` i mostra tots els comptes del banc (opció 6 del menú) per comprovar que s'han llegit correctament.

7 Ampliació: Tractament d'excepcions definides per l'usuari

Fins ara hem estat usant les excepcions que proporciona Java, però també podem crear les nostres pròpies excepcions. A manera d'exemple, anem a canviar el tractament dels errors lògics corresponents a les següents situacions:

- En la creació d'un nou compte i la seua inserció en el banc, els casos en els quals el número de compte és incorrecte (no és un nombre de 5 dígits), o el saldo és incorrecte (no és un nombre major o igual a zero), o el nombre del compte a inserir coincideix amb el nombre d'algun compte ja existent en el banc.
- En l'operació d'ingressar en compte, el cas en el qual la quantitat no siga un valor estrictament positiu.
- En l'operació de retirar de compte, el cas en el qual la quantitat no siga un valor estrictament positiu o siga un valor superior al saldo actual del compte.

En totes aquestes situacions, es llançarà una excepció de nom `BancException` definida per l'usuari, la qual haurà de ser propagada i capturada remotament per a donar-li el tractament adequat. En les següents activitats (des de la 14 fins a la 18), es descriu pas a pas una forma de resoldre el problema plantejat.

Activitat 14: declarar la classe `BancException` com a subclasse de `Exception`

- Afegeix al projecte una nova classe, de nom `BancException`, que herete de la classe `Exception` (usant la paraula reservada `extends` en la capçalera de la classe).

- En aquesta nova classe, declara un constructor que reba un argument de classe `String`. En aquest constructor simplement s'ha d'invocar al constructor de la seua superclasse (usant la paraula reservada `super`), passant-li com a argument la mateixa `String` que rep. Com es veurà en les següents activitats, el valor de l'argument del constructor serà una descripció del motiu de l'excepció (número de compte incorrecte, saldo incorrecte, compte ja existent, etc.).

- Com podràs apreciar, el cos de la classe `BancException` és bastant trivial, perquè la classe `Exception` de Java, gràcies a l'herència, evita que s'haja de tornar a escriure el codi.

Activitat 15: modificar la classe `LecturaValida` i la classe `GestorBanc`

Atès que ara la validesa del rang dels valors enters o reals llegits es gestionarà d'una altra forma (mitjançant excepcions de la classe `BancException`), no es requereix que els mètodes de la classe `LecturaValida` s'encarreguen d'açò. Per tant, en els mètodes de `LecturaValida` solament es necessitarà capturar l'excepció `InputMismatchException` per a tractar els casos en els quals l'usuari introduïsca valors que no corresponguen a nombres enters o reals.

- En la classe `LecturaValida`:
  - Cal conservar el mètode `llegirInt(Scanner,String)`, que permet llegir un nombre enter ben format.
  - S'ha d'implementar un mètode `llegirDouble(Scanner,String)` que permeta llegir un nombre real ben format. Es pot implementar quasi immediatament, partint del codi del mètode `llegirDoublePos(Scanner,String)` i eliminant la verificació que el valor llegit haja de ser positiu.
  - Per altra banda, no s'usaran els mètodes `llegirInt(Scanner,String,int,int)` i `llegirDoublePos(Scanner,String)`. Es recomana deixar el seu codi comentat.

- En la classe `GestorBanc`:
  - Substitueix les invocacions al mètode `llegirDoublePos(Scanner,String)` per invocacions al mètode `llegirDouble(Scanner,String)`; i les invocacions al mètode `llegirInt(Scanner,String,int,int)` per invocacions a `llegirInt(Scanner,String)`.

Activitat 16: modificar els mètodes `ingressar` i `retirar` de la classe `Compte`

- En el mètode `ingressar(double)` de la classe `Compte`, s'ha de gestionar el cas en el qual el mètode reba com a argument un valor no positiu. Modifica la implementació del mètode perquè en tal cas:
  - No s'efectue l'ingrés.
  - Es llance (mitjançant `throw`) una excepció de la classe `BancException` amb el missatge "*La quantitat ha de ser un valor positiu!*".
  - A més, ja que `BancException` és una excepció *checked*, has d'afegir una clàusula `throws` a la capçalera del mètode, com segueix:

    ```
    public void ingressar(double quantitat) throws BancException
    ```

indicant al compilador que l'excepció podria ser llançada i que el mètode `ingressar` no tractarà de solucionar el problema. En el seu lloc, es propaga l'excepció al mètode que crida a `ingressar`.

- En el mètode `retirar(double)` de la classe `Compte`, s'ha de gestionar tant el cas en el qual el mètode reba com a argument un valor no positiu com el cas en el qual reba un valor positiu superior al saldo en el compte. En tots dos casos, cal modificar la implementació del mètode perquè:
  - No s'efectue la disposició d'efectiu.
  - Es llance una excepció de la classe `BancException` amb el missatge "*La quantitat ha de ser un valor positiu!*" o amb el missatge "*No es pot retirar més diners del que hi ha en el compte!*", segons quin siga la causa.
  - Es propague (mitjançant `throws` en la capçalera del mètode) aquesta excepció quan es produïsca.

Activitat 17: modificar el mètode `inserir` de la classe `Banc`

- En el mètode `inserir` de la classe `Banc`, es presenten tres situacions que, si no es tracten adequadament, podrien portar a la inclusió de comptes amb informació errònia en el banc. S'ha de modificar el mètode per a detectar-les, i llançar i propagar excepcions de la classe `BancException` amb els missatges descriptius que corresponguen:
  - Si el número de compte és incorrecte (no és un nombre de 5 dígits), s'ha de llançar i propagar una excepció `BancException` amb el missatge "*Compte no vàlid:* `numCompte` *no té 5 dígits*", sent `numCompte` el valor incorrecte del número de compte.
  - Si el saldo és incorrecte (no és un nombre major o igual a zero), s'ha de llançar i propagar una excepció `BancException` amb el missatge "*Compte no vàlid:* `saldo` *és un saldo negatiu*", sent `saldo` el valor incorrecte del saldo.
  - Si el nombre del compte a inserir coincideix amb el nombre d'algun compte ja existent en el banc, s'ha de llançar i propagar una excepció `BancException` amb el missatge "*El compte:* `numCompte` *ja existeix. Compte no modificat*", sent `numCompte` el valor del número de compte ja existent.

- Solament si no es presenta cap de les tres situacions anteriors, el compte rebut com a argument del mètode `inserir` es considerarà vàlida, i s'executaran les instruccions corresponents a la seua inclusió en el banc.

Activitat 18: modificar els mètodes que invoquen a mètodes que propaguen l'excepció `BancException`

- En la classe `Banc`, els mètodes `carregarFormatText` i `carregarFormatObjecte` invoquen al mètode `inserir`. Has de modificar aquests mètodes perquè capturen l'excepció `BancException` que el mètode `inserir` pot propagar. El tractament de l'excepció es limitarà a mostrar en consola el missatge descriptiu de la mateixa (el missatge passat com a argument en instanciar l'objecte excepció).

- En la classe `GestorBanc`, has de modificar el codi en totes les invocacions als mètodes que poden propagar `BancException` (és a dir, els mètodes `ingressar` i `retirar` de la classe `Compte` i el mètode `inserir` de la classe `Banc`). En tots els casos, la modificació serà la mateixa: captura de l'excepció i tractament de la mateixa consistent a mostrar en consola el seu missatge descriptiu.

- Finalment, torna a executar el programa, provant totes les situacions que generen l'excepció `BancException`. Verifica que, en tots els casos, l'excepció és capturada, es mostra el corresponent missatge, i l'execució del programa pot continuar sense més incidències.

8 Ampliació: Crear i llegir objecte des de fitxer binari

Fins ara, has implementat les operacions de lectura de comptes bancaris de manera que els comptes existents en el fitxer (tant si era de text com si era binari) s'afegien al banc ja existent en l'aplicació.

Anem a considerar que es pot disposar, en un fitxer binari, d'un objecte de la classe `Banc` i que aquest objecte es vol carregar en la nostra aplicació però de manera que, després de l'operació, solament estiguen els comptes bancaris llegits del fitxer. És a dir, no volem afegir comptes al banc preexistent, sinó reemplaçar aquest pel banc llegit del fitxer. Per a açò, es plantegen les següents activitats.

Activitat 19: implementar en `Banc` un mètode que llija un objecte de fitxer

- Afegeix, en la classe `Banc`, un mètode la capçalera del qual siga:

  ```
  public static Banc carregaBancNou(ObjectInputStream f)
  ```

  El mètode rep un flux binari d'entrada. Has de llegir del mateix un objecte de la classe `Banc`. Aquest objecte és el que el mètode retornarà. Es poden presentar excepcions de tipus `IOException` i `ClassNotFoundException`. Fes que aquestes excepcions siguen propagades pel mètode.

Activitat 20: modificacions en `GestorBanc` per a usar la nova funcionalitat

- Afegeix un `case` addicional, en el `switch` de la classe `GestorBanc`. En el mateix, escriu el codi necessari per a sol·licitar a l'usuari el nom del fitxer binari a llegir, i invocar el mètode `carregaBancNou`. Ací hauràs de tractar adequadament les excepcions que aquest mètode pot propagar.

- Modifica el mètode `menu` de la classe `GestorBanc` perquè mostre una opció més, la corresponent a la nova funcionalitat, i admeta seleccionar-la.

- Torna a executar el programa. Primer, crea alguns comptes nous. Després, selecciona la nova opció, i carrega el contingut d'algun banc, que tingues guardat en algun fitxer binari. A continuació, mostra tots els comptes. Veuràs que solament apareixen els comptes llegits del fitxer i que, per tant, s'ha perdut la informació dels comptes nous que es van crear abans.

- Modifica l'últim `case` perquè, abans de carregar els comptes d'un nou banc, pregunte a l'usuari si vol guardar els comptes del banc actual i, en cas afirmatiu, sol·licite un nom de fitxer i guarde eixa informació.
- Torna a executar el programa, repetint la situació anterior, i comprovant que es pot guardar el banc actual abans de carregar el banc nou.

9 Avaluació

Aquesta pràctica forma part del segon bloc de pràctiques de l'assignatura que serà avaluat en el segon parcial d'aquesta. El valor d'aquest bloc és d'un 60% respecte al total de les pràctiques. El valor percentual de les pràctiques en l'assignatura és d'un 20% de la nota final.