PRG (ETS d'Enginyeria Informàtica) - Curs 2015-2016

Pràctica 5. Implementació i ús d'estructures de dades lineals

(3 sessions)

Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València

Índex

1 Context i treball previ

En el context acadèmic, aquesta pràctica correspon a el *"Tema 5. Estructures de dades lineals"*. L'objectiu general d'aquesta pràctica és completar el disseny orientat a objectes d'una aplicació les dades de la qual estan representats en una estructura lineal dinàmica en la qual cadascun dels seus elements individuals conté, al seu torn, una altra estructura lineal. Els objectius més detallats són els següents:

- Treballar amb elements habituals en l'ús de la memòria enllaçada: referències, nodes, etc.
- Implementar i utilitzar una cua enllaçada.
- Implementar i utilitzar una llista o seqüència enllaçada ordenada, posant especial èmfasi en les operacions de recuperació, inserció i eliminació de dades.
- Aplicar els mecanismes de reutilització del programari: definició i ús de paquets i definició i ús d'una classe derivada d'una altra.

Abans de la sessió de laboratori, has de llegir el butlletí de pràctiques tractant de resoldre, en la mesura del possible, els problemes proposats.

2 Plantejament del problema

En la pràctica 4 s'ha completat una aplicació que simula la gestió dels comptes d'un banc. En concret, s'han desenvolupat les classes `LecturaValida`, `Compte`, `Banc` i `GestorBanc`. Recorda que un `Banc` tenia com a atribut un array d'objectes `Compte` i en ell es podia, per exemple, consultar el nombre de comptes, el saldo d'un compte o afegir un nou compte. Llavors no es va plantejar l'esborrat d'un compte que s'hauria d'haver fet mitjançant el desplaçament una posició a l'esquerra de tots els comptes a la dreta del compte a esborrar. Tampoc es va plantejar mantenir el array de comptes ordenat per número de compte, per a açò la inserció també hauria d'haver fet els desplaçaments necessaris, en aquest cas, cap a la dreta.

En aquesta pràctica es proposa realitzar dues modificacions importants sobre l'anterior:

1. Representar els comptes del banc com una *llista o seqüència enllaçada* que permet, de manera eficient, l'esborrat i la inserció *ordenada*. Així doncs, l'array se substituirà per una llista o seqüència de comptes ordenada per número de compte.
2. Afegir a cada compte el registre dels apunts (ingressos o reintegraments) que s'han fet fins al moment sobre ella. En aquest cas, s'utilitzarà una *cua enllaçada* per a mantenir l'ordre en el qual s'han anat realitzant els diferents apunts.

3 Organització de les classes i ús d'un paquet d'usuari

Les activitats que segueixen et permetran tenir organitzades les classes necessàries (algunes incompletes, de moment) per a aquesta pràctica.

Activitat 1: classes de la carpeta `pract5`

- Descarrega en el teu directori `$HOME/DiscoW/prg`, des de `Recursos/Laboratorio/Práctica 5` de `PoliformaT` de `PRG`, el fitxer `pract5.jar`. Per tal d'obrir-lo com un projecte *BlueJ* específic per a aquesta pràctica, usa l'opció *Projecte/Obre No BlueJ...* de *BlueJ*. El projecte `pract5` contindrà els fitxers de codi `CompteAp.java`, `Apunt.java`, `CuaIntEnla.java` i `NodeInt.java`, el fitxer de text `comptesAmbApunts.txt` i els següents fitxers `.class`: `TestApunt`, `TestNodeApunt`, `TestCuaApunts`, `TestCompte`, `TestCompteAp`, `TestNodeCompte` i `TestBanc`. Aquests últims són tests preparats per a comprovar el correcte funcionament del codi de les classes de la pràctica 5 que vas a desenvolupar.
- Afegeix, usant l'opció *Edita/Afegir classe des d'arxiu...*, les classes `Compte.java`, `Banc.java` i `GestorBanc.java` que vas implementar en la pràctica 4.

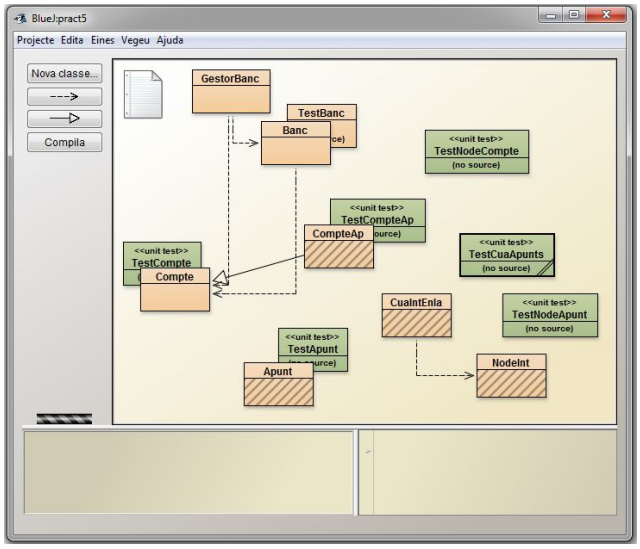
Activitat 2: el paquet `lectura`

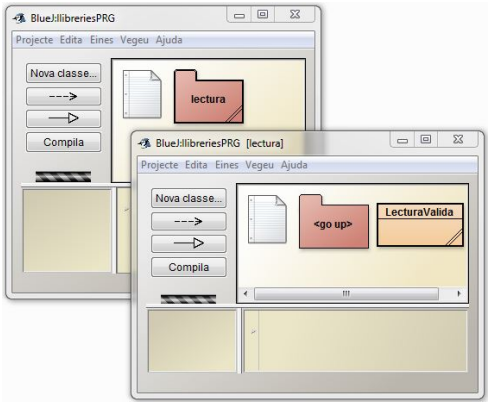
La classe `LecturaValida` que vas desenvolupar en la pràctica 4 és un bon exemple de codi reutilitzable des d'altres classes. La millor forma de fer açò efectiu és:

- Definir un paquet Java; per a açò:
  1. Crea en la teua carpeta `prg` un projecte *BlueJ* anomenat `llibreriesPRG`.
  2. Crea dins d'aquest projecte un paquet de nom `lectura`, usant l'opció *Edita/Nou paquet* de *BlueJ*.
  3. Dins d'aquest paquet, afegeix la teua classe `LecturaValida` i compila-la.
- Per a poder tenir accés a la llibreria `lectura`, afegeix la ubicació del projecte `llibreriesPRG` des de l'opció *Eines/Preferències/Llibreries* de *BlueJ*. Has de reiniciar *BlueJ* perquè els canvis fets a la llibreria de classes tinguen efecte.
- A partir d'aquest moment, ja pots importar en els projectes *BlueJ* que construïsques els elements del paquet `lectura`, seguint per a aquesta importació la notació estàndard Java. En concret, al començament de la classe `GestorBanc` escriu la instrucció d'importació:

```
import lectura.LecturaValida;
```

- Compila les classes `Compte`, `Banc` i `GestorBanc` i comprova que l'accés des de `GestorBanc` als mètodes de la classe `LecturaValida`, que ara està al paquet, és correcte.





4 Classe derivada `CompteAp`

Com s'ha comentat anteriorment, per a mantenir els apunts (ingressos o reintegraments) realitzats en un compte emmagatzemats segons el seu ordre d'arribada, es proposa utilitzar l'estructura de dades lineal *Cua*. Així, cada compte, a més del seu número i el seu saldo, haurà de tenir una cua amb els apunts realitzats en la mateixa. Per a açò, es defineix la classe `CompteAp` com una classe derivada de la classe `Compte`, que afegeix aquesta cua com a atribut. Les classes `Apunt`, `NodeApunt` i `CuaApunts` permeten definir el nou atribut.

Activitat 3: classes `Apunt`, `NodeApunt` i `CuaApunts`

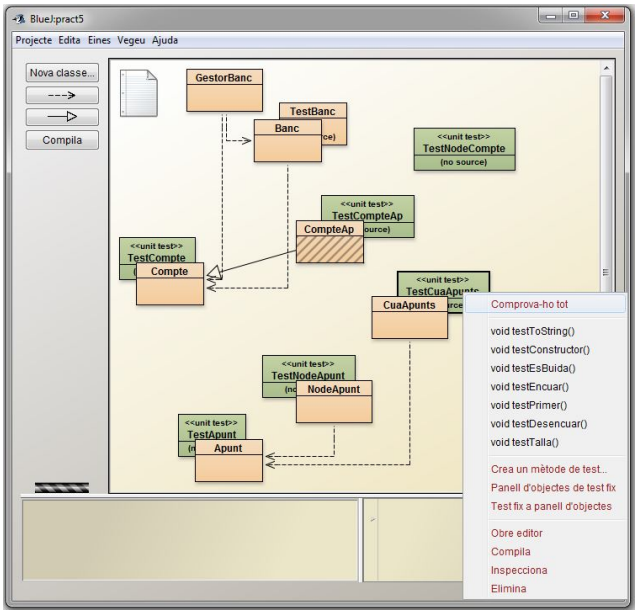
- Revisa la classe `Apunt`. Fixa't que es tracta d'una classe tipus de dades senzilla que permet definir un apunt d'un compte en base a la seua descripció (si es tracta d'un ingrés o d'un reintegrament) i la quantitat (valor real positiu o negatiu) implicada, mitjançant els atributs d'instància privats `descripcio` (`String`) i `quantitat` (`double`), respectivament. Inclou també dos atributs de classe públics i constants[1] (`INGRES` i `REINTEGRAMENT`) de tipus `String` que defineixen els valors possibles de `descripcio`. La funcionalitat de la classe `Apunt` ve donada per un constructor, un consultor per a cada atribut d'instància i un mètode `toString()` que sobreescriu el de la classe `Object`.
- Revisa les classes `NodeInt` i `CuaIntEnla`. Són les vistes en teoria per a representar una cua enllaçada de nombres enters. Canvia'ls el nom per `NodeApunt` i `CuaApunts`, respectivament, i modifica-les per a representar una cua enllaçada d'objectes de tipus `Apunt`, de manera que:
  - `NodeApunt` definisca un node que continga com a dada un objecte `Apunt` i
  - `CuaApunts` definisca una cua d'objectes `Apunt`, en la que cada node és de tipus `NodeApunt`.

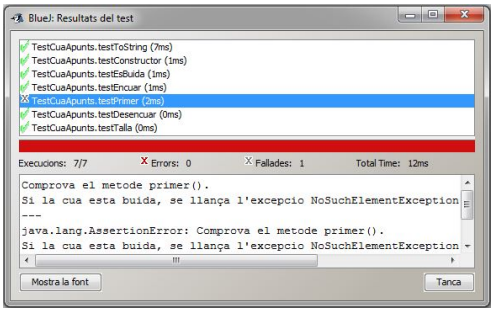
Activitat 4: validació de les classes `Apunt`, `NodeApunt` i `CuaApunts`

- En general, per a que els tests s'executen correctament és indispensable que:
  - En la definició dels atributs i mètodes de les classes, utilitzes sempre els noms d'atributs i mètodes proposats al butlletí de la pràctica i en els arxius `.java` proporcionats en `PoliformaT`, respectant a més les característiques descrites per a cadascun d'ells en quant a modificadors i paràmetres.
  - En els mètodes que tornen com a resultat un `String`, seguisques el format i el text indicats al butlletí.

[1] Aquestes constants hauran d'utilitzar-se sempre que es requerisca en la resta de classes del projecte.

- Per tal de poder provar els tests d'aquesta activitat, recorda que primer has d'aconseguir passar el test de la classe `Apunt`[2], ja que aquesta classe s'usa en la classe `NodeApunt`. Després el test de la classe `NodeApunt` doncs també aquesta, al seu torn, és usada en la classe `CuaApunts`. I, finalment, el test d'aquesta última classe. El motiu és que si hi ha un error en el primer test, probablement provoque errors en els següents.
- Tria l'opció *Comprova-ho tot* (*Test All*) del submenú desplegable amb el botó dret sobre la icona de cada test. S'executarà una bateria de proves sobre els mètodes de la classe que s'està provant, comparant el resultat obtingut amb l'esperat.
- Si els mètodes són correctes, en la finestra *Resultats del test* de BlueJ, apareixeran marcats amb un ✓ (de color verd). Si, pel contrari, algun mètode no funciona correctament, en la finestra *Resultats del test*, el test de cada mètode incorrecte apareixerà marcat amb una `X`. Si es selecciona un d'aquests tests, en la part inferior de la finestra, es mostra un missatge orientatiu sobre la causa de l'error.
- Per a tornar a executar el test, després de corregir els errors i compilar de nou la teua classe, si la icona del test apareix ratllada, has de tancar i tornar a obrir el projecte *BlueJ*.





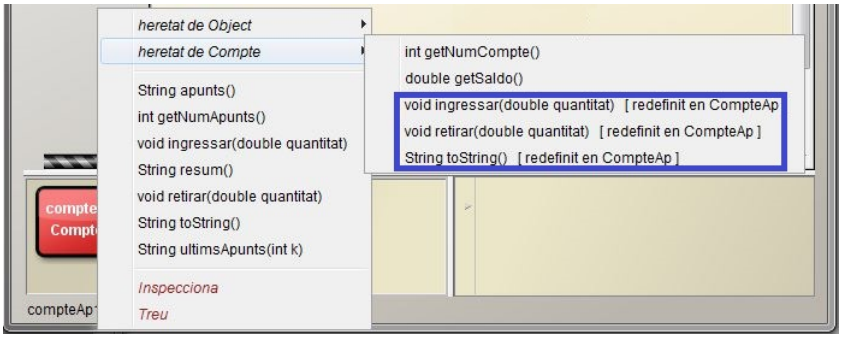
Activitat 5: atributs de la classe `CompteAp`

- Revisa la classe `CompteAp`. Fixa't que en la seua capçalera apareix `extends Compte` indicant que es tracta d'una classe derivada (o subclasse) de la classe `Compte` (classe base o superclasse) i que, per tant, un objecte de tipus `CompteAp` hereta els atributs `saldo` (`double`) i `numCompte` (`int`) definits en `Compte`. Afegeix, a la classe `CompteAp`, un atribut `apunts` de tipus `CuaApunts` per tal de representar com una cua enllaçada els diferents apunts que es van realitzant en el compte.

Activitat 6: mètodes de la classe `CompteAp`

La classe `CompteAp`, a més dels atributs comentats en l'activitat anterior, també hereta els mètodes definits en `Compte`. Aquests mètodes es poden sobreescriure parcialment per incorporar les accions requerides relacionades amb el nou atribut `apunts`.

[2]Passaràs aquest test sense problemes, ja que no has de fer cap modificació en la classe `Apunt` que et proporcionem.



- Completa el mètode constructor per:

  1. Crear un nou compte, donats un número de compte i un saldo inicial, usant el constructor de la superclasse (`super(int, double)`).
  2. Crear la cua d'apunts i encuar un `Apunt` amb la descripció `"Ingres"` i la quantitat del saldo inicial. Així, el compte tindrà com a únic apunt el de l'ingrés d'obertura.

- Completa el mètode `ingressar(double)` perquè, a més de fer l'ingrés (`super.ingressar(double)`), encue un `Apunt` amb descripció `"Ingres"` i la quantitat ingressada.

- Completa el mètode `retirar(double)` perquè, a més de fer el reintegrament (`super.retirar(double)`), encue un `Apunt` amb descripció `"Reintegrament"` i la quantitat retirada (amb signe $-$).

- Completa el mètode `getNumApunts()` perquè torne el número d'apunts total del compte.

- Completa el mètode `ultimsApunts(int)` perquè, donat un enter `k`, $1 \leq$ `k` $\leq$ `getNumApunts()`, torne un `String` amb la informació dels `k` últims apunts realitzats en el compte. El `String` resultat tindrà `k+1` línies. El format de la primera línia serà: `"Num. Apunts: k/X"` indicant que se consulten `k` dels `X` apunts del compte. Les `k` línies següents (una per cada apunt) constaran de la descripció i la quantitat de l'apunt realitzat (separats per `": "`) i el saldo del compte després d'aquest apunt. Per exemple, per a un compte amb 3 apunts en el qual es consulten els 2 últims, el `String` resultant serà el següent:

```
Num. Apunts: 2/3
       Ingres:      45.00   Saldo:     545.00
Reintegrament:     -15.00   Saldo:     530.00
```

  Per a resoldre aquest problema, tenint en compte que l'accés als elements d'una cua es realitza seguint un criteri FIFO (*First In First Out*) i que, òbviament, no es desitja perdre'ls, l'estratègia a seguir podria ser la següent:

  1. desencuar i reencuar `t - k` elements, sent `t` la talla de la cua, i
  2. desencuar i reencuar `k` elements amb els quals anar construint el `String` resultat.

  Fixa't que el saldo després de cada apunt s'ha de calcular a partir de les quantitats de tots els apunts. Per exemple, si els apunts del compte anterior són:

```
       Ingres:     500.00
       Ingres:      45.00
Reintegrament:     -15.00
```

  Inicialment el saldo és `500.00` i es va modificant amb cada apunt, passant a ser `545.00` després de l'ingrés de `45.00` i `530.00` després del reintegrament de `-15.00`.

- Completa el mètode `apunts()` perquè, invocant al mètode anterior, torne un `String` amb la informació de tots els apunts del compte.

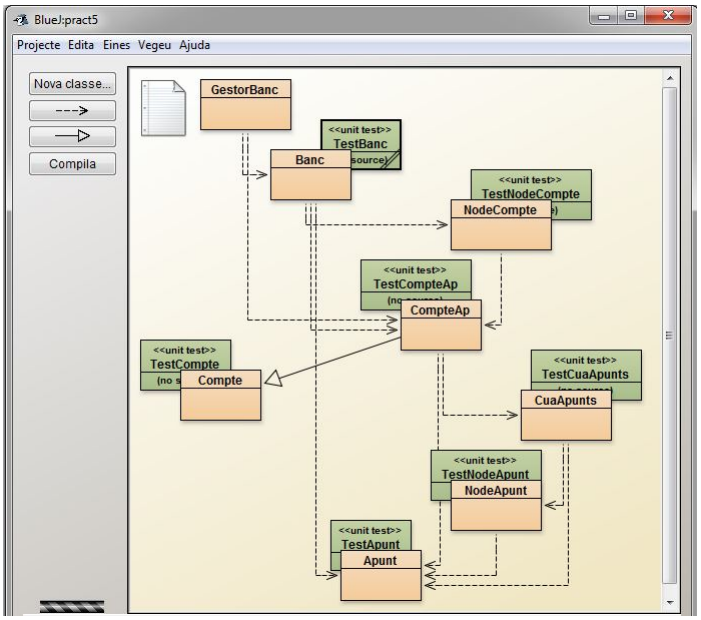
- Completa el mètode `resum()` perquè, invocant al mètode `toString()` de la superclasse, torne un `String` resum de la informació del compte (número i saldo separats per un espai en blanc).
- Completa el mètode `toString()` perquè, utilitzant el `toString()` de la superclasse i el mètode `apunts()`, sobreescriga el de `Compte`.

Activitat 7: validació de la classe `CompteAp`

- Recorda el que s'ha indicat en la Activitat 4 per al correcte funcionament dels tests.
- Per a poder provar el test de la classe `CompteAp`, tria l'opció *Comprova-ho tot* (*Test All*) del submenú desplegable amb el botó dret sobre la icona de `TestCompteAp`. De igual forma que en la validació de les classes de l'Activitat 4, s'executarà una bateria de proves sobre els mètodes de la classe `CompteAp`, comparant el resultat obtingut amb l'esperat. Aquest test suposa que la implementació de les classes `Compte`, `CuaApunts`, `NodeApunt` i `Apunt` és correcta. Així doncs, primer has d'aconseguir passar els tests d'aquestes classes.

5 Implementació enllaçada de la classe `Banc`

Les activitats següents condueixen a la implementació enllaçada d'un `Banc` amb la funcionalitat afegida de poder cancel·lar un compte. Així, el `Banc` tindrà com atribut una llista o seqüència enllaçada de `CompteAp` que se mantindrà en tot moment ordenada de forma creixent per número de compte.

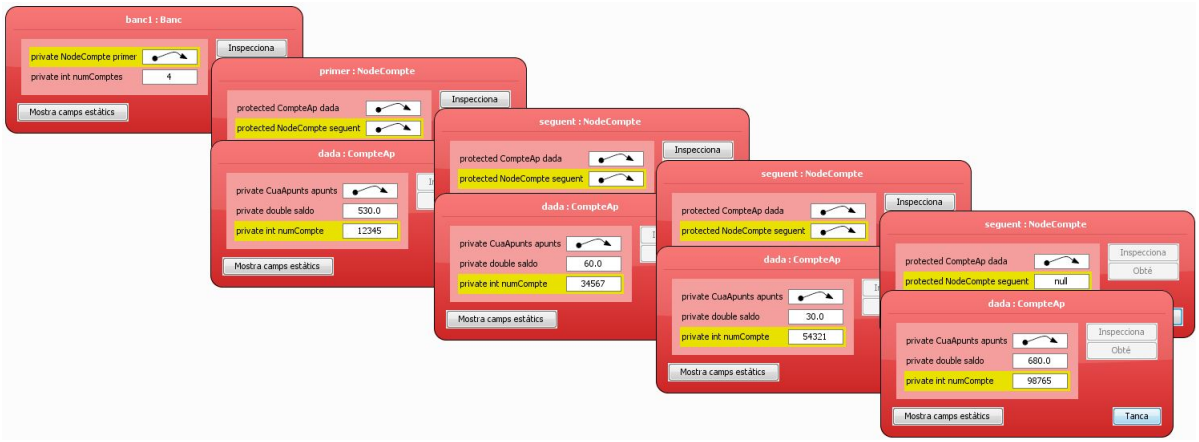


Activitat 8: classe `NodeCompte`

- Crea una classe `NodeCompte`, similar a la classe `NodeApunt`, l'atribut `dada` del qual siga de tipus `CompteAp`, per tal de representar cadascun dels nodes d'una seqüència enllaçada d'objectes `CompteAp`.
- Valida la classe `NodeCompte` executant el test `TestNodeCompte`. Aquest test suposa que la implementació de les classes implementades fins al moment és correcta. Així doncs, primer has d'aconseguir passar els tests d'aquestes classes.

Activitat 9: classe `Banc`

- Modifica la classe `Banc` eliminant els atributs `MAX_COMPTES` i `comptes` i afegint un atribut `primer` de tipus `NodeCompte` per a representar (el primer node d') una seqüència enllaçada de comptes ordenats per número de compte.



- Modifica el mètode constructor per a crear un banc sense comptes.
- Elimina el mètode `duplica()` i modifica el mètode `inserir(CompteAp)` per tal que la inserció es realitze de forma ordenada segons el número de compte en la seqüència enllaçada, en lloc d'en la primera posició lliure de l'array. Nota que ara el paràmetre és de tipus `CompteAp`.
- Modifica el mètode `getCompte(int)` considerant l'ordre segons número de compte dels comptes del banc, de manera que la cerca sense èxit sobre la seqüència enllaçada acabe en trobar un número de compte més gran que el donat o pel fet que tots els números de compte són menors que el donat. El cost d'aquest mètode ha de ser constant en el millor cas i lineal en el pitjor cas.
- Afegeix el mètode `cancelar(int)` per a eliminar del banc, donat un número de compte, el compte amb aquest número. El mètode torna `true` si se pot cancel·lar i si no (el compte no existeix) torna `false`. En aquest mètode s'ha de considerar, com en l'anterior, el fet que els comptes estan ordenats per número de compte. El cost d'aquest mètode també ha de ser constant en el millor cas i lineal en el pitjor cas.
- Modifica el mètode `toString()` per tal que ara el recorregut es faça sobre la seqüència enllaçada en lloc de sobre l'array.

Activitat 10: validació de la classe `Banc`

- Valida la classe `Banc` executant el test `TestBanc`. Aquest test suposa que la implementació de les classes implementades fins al moment és correcta. Així doncs, primer has d'aconseguir passar els tests d'aquestes classes.

Activitat 11: mètodes de la classe `Banc` per a llegir/escriure des de/en un fitxer

A la pràctica 4, vas veure com llegir/escriure comptes bancaris des de/en un fitxer de text o binari d'objectes, realitzant el tractament de les excepcions relacionades. En aquesta activitat, has de realitzar les modificacions oportunes en els mètodes implicats en aquestes operacions, considerant la implementació enllaçada d'un `Banc`.

- Modifica el mètode `carregarFormatObjecte(ObjectInputStream)` per tal que ara el recorregut es faça sobre la seqüència enllaçada en lloc de sobre l'array.
- Fixa't en el format que té el fitxer de text `comptesAmbApunts.txt` i modifica el mètode `carregarFormatText(Scanner)` per tal que puga llegir fitxers amb aquest format. En particular, has d'afegir el codi necessari per a la lectura dels apunts de cada compte. Observa que:

  1. El número total d'apunts d'un compte el pots obtenir de la línia `"Num. Apunts: X/X"`, quedant-te amb el substring a partir del caràcter `'/'` i transformant-lo a un valor de tipus `int` amb el mètode `parseInt(String)` de la classe `Integer`.

2. El saldo inicial que has d'utilitzar per a crear el compte és el del primer apunt. Per exemple, si el primer apunt d'un compte fora `"  Ingres:  500.00  Saldo:  500.00"`, el saldo inicial seria `500.00`. El pots obtenir quedant-te amb el substring comprès des del caràcter `':'` fins el caràcter `'S'`, eliminant els espais en blanc i transformant-lo a un valor de tipus `double` amb el mètode `parseDouble(String)` de la classe `Double`.
3. Una vegada creat el compte, per a obtenir la quantitat de la resta dels apunts, has de fer algo similar al que has fet amb el primer apunt i anar cridant als mètodes `ingressar(double)` o `retirar(double)`, segons corresponga, per a enregistrar els apunts en el compte.

- **NO** has de modificar els mètodes que permeten emmagatzemar la informació d'un banc, `guardarFormatText(PrintWriter)` i `guardarFormatObjecte(ObjectOutputStream)`, en un fitxer de text o en un fitxer binari, respectivament, perquè tal com els vas implementar a la pràctica 4 són correctes.
- Pots validar els mètodes de lectura/escriptura des de/en un fitxer (de text o binari d'objectes), de manera similar a com ho vas fer a la pràctica 4, executant les opcions corresponents de l'aplicació `GestorBanc`.

6 Noves opcions de la classe `GestorBanc`

Per provar la nova funcionalitat de la classe `Banc` afegiràs a la classe `GestorBanc` les opcions *Cancel·lar compte* i *Mostrar els últims k apunts del compte actiu*.

Activitat 12: opció *Cancel·lar compte*

- Afegeix un `case 9` al `switch` del `main` de la classe `GestorBanc` on se demane un número de compte vàlid i, si existeix, se cancel·le aquest compte i si no, se mostre per pantalla un missatge d'error.
- Modifica el mètode `menu(Scanner)` de la classe `GestorBanc` afegint l'opció *9) Cancel·lar compte*. 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.

Activitat 13: opció *Mostrar els últims k apunts del compte actiu*

- Afegeix un `case 10` al `switch` del `main` de la classe `GestorBanc` on se demane un número enter `k` vàlid (açò és, $1 \leq$ `k` $\leq$ número d'apunts del compte actiu) i se mostren per pantalla els últims `k` apunts del compte actiu.
- Modifica el mètode `menu(Scanner)` de la classe `GestorBanc` afegint l'opció *10) Mostrar últims k apunts del compte actiu*. 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.

7 Avaluació

Aquesta pràctica forma part del segon bloc de pràctiques de l'assignatura que serà avaluada en el segon parcial de la mateixa. 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.