

IIP (E.T.S. de Enginyeria Informàtica)
Curs 2016-2017

Pràctica 7. Gestió d'un parking
Duració: tres sessions

Professors d'IIP
Departament de Sistemes Informàtics i Computació
Universitat Politècnica de València



Índex

1	Context i planificació	1
2	Plantejament del problema	2
3	La classe Ticket	3
4	La classe Planta	5
5	La classe Parking	8
6	La classe GestorParking	12

1 Context i planificació

Aquesta pràctica resumeix tots els conceptes vistos en l'assignatura i, en particular, els corresponents al “Tema 7. Arrays: definició i aplicacions”. Per resoldre-la pots consultar, entre altres materials, el capítol 9 del llibre “Empezar a programar usando Java” (3^a edició)¹.

Aquesta pràctica té una durada de tres sessions en les quals es desenvoluparan les 8 activitats proposades en aquest butlletí. A la taula que segueix es mostra la planificació per sessions d'aquestes activitats, insistint en el treball no presencial requerit imprescindible per cobrir els objectius en el temps previst.

Planificació de les activitats

	Presencials	No Presencials
Previ	–	Llegir aquest butlletí
Sessió 1	1, 2, 3, 4 i 5	Finalitzar amb èxit la implementació de les classes Ticket i Planta
Sessió 2	6 i 7	Finalitzar amb èxit la implementació de la classe Parking
Sessió 3	8	Finalitzar amb èxit la implementació de la classe GestorParking

¹Capítol 10 de la 2^a edició.

2 Plantejament del problema

Es vol implementar una aplicació que simule un parking intel·ligent en el qual, quan un vehicle accedeix, se li indica on estacionar i, quan el vehicle deixa el parking, se li indica l'import a pagar. A més, l'aplicació oferirà la possibilitat de localitzar el vehicle (és a dir, quina plaça o lloc de parking ocupa i en quina planta) i mostrarà l'ocupació del parking (és a dir, si les diferents places o llocs de parking estan ocupats o buits). Finalment, el parking tindrà un mecanisme per retirar tots els vehicles estacionats al final del dia i calcular l'import corresponent per aquesta acció (és a dir, la suma dels imports de tots els vehicles retirats).



L'aplicació proposada s'ha d'organitzar en diverses classes tipus de dades que instancien els diferents objectes implicats en la mateixa. Aquestes classes són:

- **Instant:** guarda la informació de les hores i els minuts que defineixen una marca o instant de temps; en aquest cas, s'usa per representar l'hora d'arribada d'un vehicle al parking i l'hora de sortida del mateix. Aquesta classe és la que vas implementar en la pràctica 4 en el paquet `iipUtil` i s'importarà en la resta de classes.
- **Ticket:** defineix les dades d'estacionament d'un vehicle en un parking: matrícula, hora d'entrada i localització (número de planta i número de lloc).
- **Planta:** representa una planta d'un parking (amb el número corresponent i el número de llocs lliures) i la seua ocupació mitjançant els tickets associats als vehicles estacionats a la mateixa.
- **Parking:** representa un parking com l'ocupació de les seues plantes (amb un número fixe de llocs per planta) juntament amb la tarifa corresponent (en euros per minut).

A més, és necessària una classe programa `GestorParking` que implemente les diferents funcionalitats que ofereix l'aplicació.

Activitat 1: creació del projecte *BlueJ* pract7

1. Descarrega en el teu directori `$HOME/iip/` el fitxer `pract7.jar` disponible en la carpeta `Recursos/Laboratorio/Práctica 7/Valencia/codi` de la *PoliformaT* d'IIP.
2. Obre'l en *BlueJ*, utilitzant l'opció *Projecte - Obrir No BlueJ...* El projecte `pract7` contindrà (com es mostra a la figura 1) els fitxers de codi `Ticket.java`, `Planta.java`, `Parking.java` i `GestorParking.java`, dos fitxers de text `.txt` (`parkingIIP` i `parkingIIP-Ple`) i els següents fitxers `.class`: `TestTicket`, `TestPlanta` i `TestParking`. Aquests últims són tests preparats per comprovar el correcte funcionament del codi de les classes de la pràctica 7, que vas a desenvolupar.
3. Accedeix al directori `$HOME/asigDSIC/ETSINF/iip/pract7/docValencia` i obre el fitxer `index.html` amb qualsevol navegador. Veuràs que conté la documentació (en el format habitual de Java) de les classes la implementació de les quals has de completar i que, per això, has de consultar.

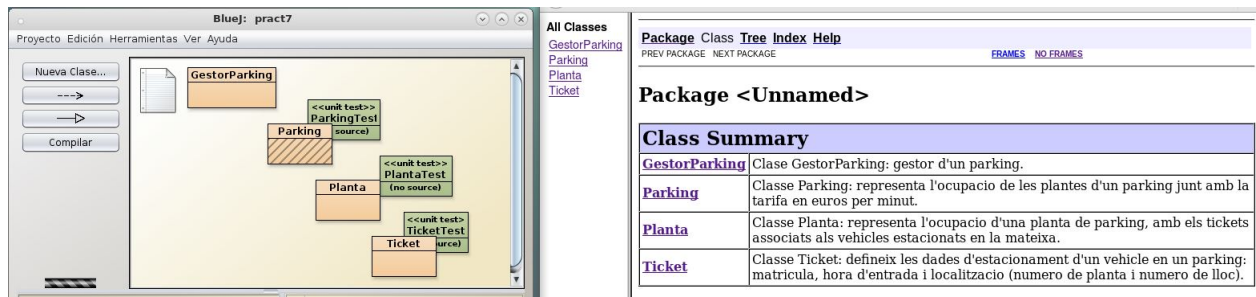


Figura 1: Projecte `pract7` en *BlueJ* i la seua documentació.

3 La classe `Ticket`

La classe `Ticket` defineix les dades d'estacionament d'un vehicle en un parking mitjançant els següents atributs d'instància privats:

- **matricula:** `String` que indica la matrícula del vehicle associat.
- **entrada:** `Instant` de indica l'hora d'entrada del vehicle al parking.
- **planta i lloc:** enters que representen la localització del vehicle al parking, sent inicialment -1 i, un cop estacionat el vehicle associat, el número de planta i el número de lloc, respectivament, que ocupa aquest vehicle.

Aquesta classe consta dels següents mètodes:

- Constructor `Ticket(String, Instant)`: crea un objecte `Ticket` associat a un vehicle, amb matrícula i hora d'arribada donades, que accedeix al parking i, per tant, encara no té localització assignada.
- Un mètode consultor per cada atribut: `getMatricula()`, `getEntrada()`, `getPlanta()` i `getLloc()`.
- Un mètode modificador de la planta i un mètode modificador del lloc: `setPlanta(int)` i `setLloc(int)`, que s'utilitzen per actualitzar la localització del vehicle quan estaciona.
- Un mètode `toString()` (que sobreescriu el d'`Object`) que torna un `String` amb la informació del ticket en el següent format:
 - si el ticket no té localització assignada:
`"Matricula: MATRICULA - Entrada: ENTRADA"`
 - si sí que la té:
`"Matricula: MATRICULA - Entrada: ENTRADA - Planta: PLANTA - Lloc: LLOC"`

Activitat 2: implementació i prova de la classe `Ticket`

- Revisa en la classe `Ticket` la definició dels atributs i els mètodes que et proporcionem.
- Completa el mètode `toString()` seguint la seua especificació.
- Utilitzant el *Code Pad* o l'*Object Bench* de *BlueJ*, crea un objecte `Ticket` d'un vehicle amb matrícula 1234AC que accedeix al parking a les 10:25. Comprova que el resultat és el que es mostra a la figura 2.
- Executa cadascun dels mètodes consultors i comprova que el resultat és correcte.

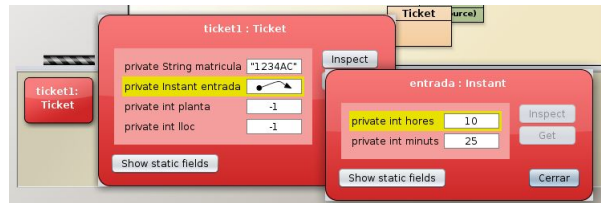


Figura 2: Objecte Ticket d'un vehicle amb matrícula 1234AC que accedeix al parking a les 10:25.

- Executa el mètode `toString()` i comprova que el resultat és el mostrat a la figura 3.

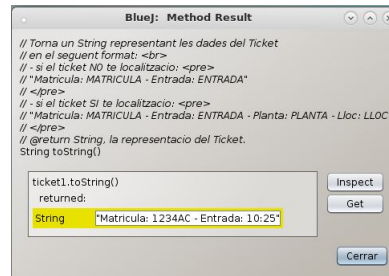


Figura 3: Resultat del mètode `toString()` del Ticket d'un vehicle amb matrícula 1234AC, hora d'entrada 10:25 i sense localització assignada.

- Executa els mètodes `setPlanta(int)` i `setLloc(int)` per tal d'actualitzar la planta i el lloc del Ticket creat amb els valors 2 i 5, respectivament. Comprova que el resultat és el que es mostra a la figura 4.

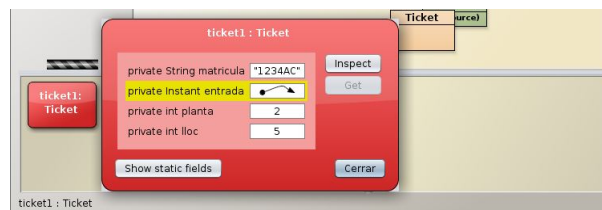


Figura 4: Objecte Ticket d'un vehicle ubicat a la planta 2 i el lloc 5 del parking.

- Executa el mètode `toString()` i comprova que el resultat és el mostrat a la figura 5.

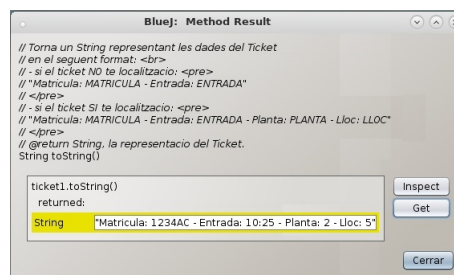


Figura 5: Resultat del mètode `toString()` del Ticket d'un vehicle amb matrícula 1234AC, hora d'entrada 10:25 10:25, ubicat a la planta 2 i el lloc 5 del parking.

Activitat 3: validació de la classe Ticket

1. En general, perquè els tests s'executen correctament és indispensable que:
 - En la definició dels atributs i mètodes de cada classe, utilitzes sempre els noms d'atributs i mètodes proposats en el butlletí de la pràctica i en la documentació dels fitxers `.java` proporcionats, respectant a més les característiques descrites per a cadascun d'ells sobre modificadors i paràmetres.
 - En els mètodes que tornen com a resultat un **String**, seguisques el format i el text indicats al butlletí.
2. Per poder provar el test d'aquesta activitat, és imprescindible la validació de la classe **Instant** (per a això, recorda que a la pràctica 4 et proporcionarem el test **Pract4TestUnit**), ja que aquesta classe s'usa en la classe **Ticket**. Si hi ha un error en la classe **Instant**, probablement provoqui errors en el test de la classe **Ticket** i en qualsevol altre que faci servir **Instant**.
3. Tria l'opció *Test All* del submenú desplegable amb el botó dret sobre la icona del test. S'executarà una bateria de proves sobre els mètodes de la classe que s'està provant, comparant el resultat obtingut amb l'esperat.
4. Igual que en pràctiques anteriors, si els mètodes són correctes, a la finestra *Test Results* de *BlueJ*, apareixeran marcats amb un ✓ (de color verd). Si, per contra, algun mètode no funciona correctament, a la finestra *Test Results*, el test de cada mètode incorrecte apareixerà marcat amb una X. Si es selecciona un d'aquests tests, a la part inferior de la finestra, es mostra un missatge orientatiu sobre la causa de l'error.
5. Per tornar a executar el test, després corregir els errors i compilar de nou la teua classe, si la icona del test apareix ratllat, has de tancar i tornar a obrir el projecte *BlueJ*.

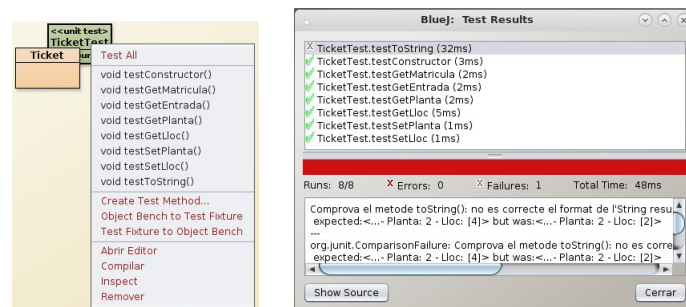


Figura 6: Execució de la *unit test* `TicketTest` i la seua finestra *Test Results* en *BlueJ*.

4 La classe Planta

La classe **Planta** permet representar una planta d'un parking que té associat un número i consta d'una sèrie de llocs que poden estar lliures u ocupats. Es defineix mitjançant els següents atributs d'instància privats:

- **numPlanta**: enter que indica el número de planta.
- **llocs**: array de objectes **Ticket** d'una grandària donada (quan es crea l'objecte en el constructor) que representa l'ocupació dels llocs de la planta. El nombre de cada lloc coincideix amb la seua posició en aquest array, de manera que `llocs[i]` és el **Ticket** associat al vehicle estacionat en el lloc `i` o és `null` si el lloc està lliure, sent $0 \leq i < \text{llocs.length}$.
- **llocsLliures**: enter que indica el número de llocs lliures (inicialment, igual al total de llocs).

Els mètodes d'aquesta classe són els que s'indiquen a continuació:

- Constructor **Planta(int, int)**: donats un número de planta i un número de llocs, crea un objecte **Planta** amb tots els llocs lliures (veure figura 7).

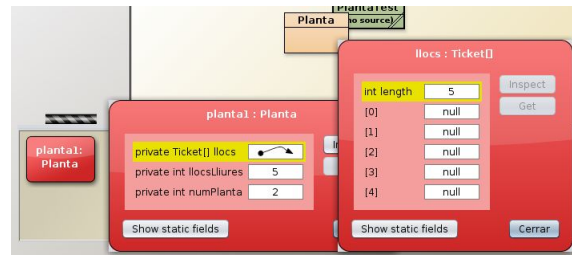


Figura 7: Objecte **Planta** de la planta 2 d'un parking, amb 5 llocs (tots lliures).

- Un mètode consultor del número de planta i un mètode consultor del número de llocs lliures: **getPlanta()** y **getLlocsLliures()**.
- Un mètode **estaPlena()** que torna **true** si no hi ha llocs lliures en la planta o, en cas contrari, torna **false**.
- Un mètode **primerLliure()** que torna el número del primer lloc lliure en la planta (açò és, el de número menor) o -1 si no hi ha llocs lliures.
- Un mètode **estacionar(Ticket)** que, donat un **Ticket** sense localització assignada, simula l'estacionament en la planta del vehicle associat (veure figura 8), com segueix: si hi ha llocs lliures, assigna el **Ticket** al primer lloc lliure (de número menor), i actualitza el número de lloc del **Ticket** i el número de llocs lliures de la planta. Aquest mètode ha de fer ús dels mètodes **estaPlena()** i **primerLliure()**.

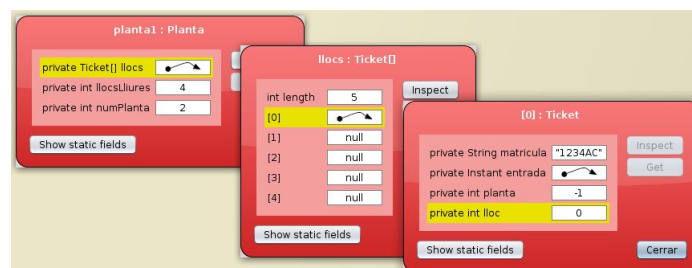
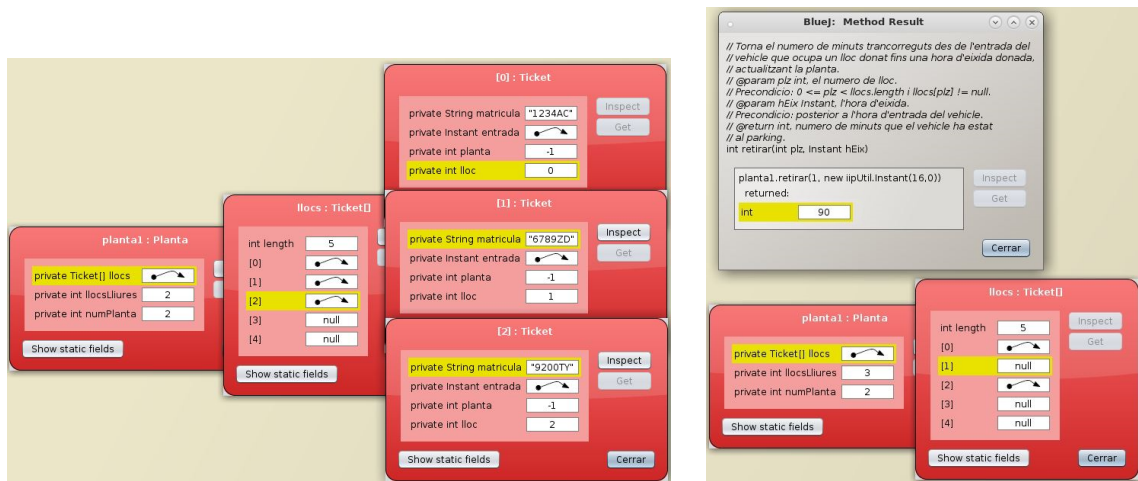


Figura 8: Resultat del mètode **estacionar(Ticket)** en estacionar un vehicle a la planta 2 d'un parking, amb 5 llocs lliures.

- Un mètode **buscarTicket(String)** que comprova si un vehicle, donada la seua matrícula, està en la planta, tornant el **Ticket** associat al vehicle, si es troba, o **null** en cas contrari.
- Un mètode **retirar(int, Instant)** que simula la retirada d'un vehicle de la planta (veure figura 9), donats un número de lloc **plz** vàlid ($0 \leq plz < llocs.length$), ocupat (**llocs[plz] != null**) per aquest vehicle, i un **Instant** que representa una hora d'eixida (posterior a l'hora d'entrada del vehicle); aquest mètode ha de tornar el número de minuts transcorreguts des de l'entrada del vehicle fins l'hora d'eixida donada. La retirada del vehicle consisteix en actualitzar l'array **llocs** i el número de llocs lliures.



(a) Objecte **Planta** de la planta 2 d'un parking, amb 3 llocs ocupats (dels 5 en total).

(b) Resultat de retirar, a les 16:00, el vehicle que ocupa el lloc 1.

Figura 9: Resultat del mètode `retirar(int, Instant)`.

- Un mètode `retirarTots(Instant)` que, donat un `Instant` que representa una hora d'eixida (posterior a l'hora d'entrada de tots els vehicles de la planta), retira tots els vehicles i torna el número total de minuts que els vehicles han estat a la planta fins l'hora d'eixida donada. Aquest mètode ha de fer ús del mètode `retirar(int, Instant)`.
- Un mètode `toString()` (que sobreescriu el d'`Object`) que torna un `String` amb l'ocupació de la planta. Per exemple, si la planta és la 2 i, dels 5 llocs que té, estan ocupats el 0, el 2, el 3 i el 4, l'`String` resultat serà: " 2 X X X X ", com pot veure's en la figura 10. Es mostra, a continuació, aquest `String` però substituint els espais en blanc per - perquè s'aprecie clarament el format utilitzat: "--2---X-----X---X---X-".

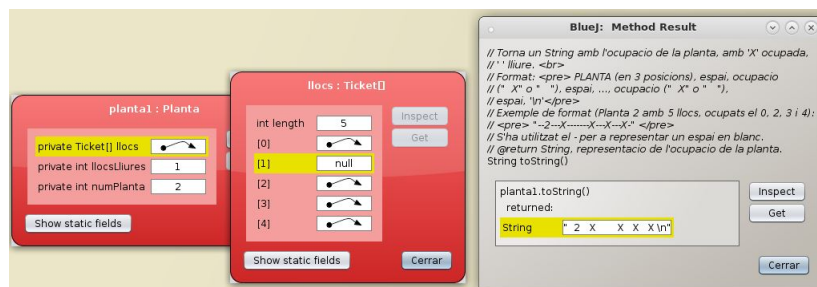


Figura 10: Resultat del mètode `toString()`.

Activitat 4: implementació i prova de la classe **Planta**

Implementa els atributs i mètodes de la classe **Planta**. En acabar, pots provar-la utilitzant el *Code Pad* o l'*Object Bench* de *BlueJ*. En particular, comprova el comportament del constructor i dels mètodes `estacionar(Ticket)`, `buscarTicket(String)`, `retirar(int, Instant)`, `retirarTots(Instant)` i `toString()`, com es mostra a les figures 7 8, 9 i 10.

Activitat 5: validació de la classe **Planta**

1. Recorda el que s'indica a l'Activitat 3 per al correcte funcionament dels tests.

- Per poder provar el test de la classe **Planta**, tria l'opció *Test All* de submenú desplegable amb el botó dret sobre la icona de **TestPlanta**. De la mateixa manera que en la validació de les classes de l'Activitat 3, s'executarà una bateria de proves sobre els mètodes de la classe **Planta**, comparant el resultat obtingut amb l'esperat. Aquest test suposa que la implementació de les classes **Instant** i **Ticket** és correcta. Així doncs, primer has d'aconseguir passar els tests d'aquestes classes.

5 La classe Parking

La classe **Parking** representa un parking com una sèrie de plantes amb llocs lliures i ocupats juntament amb la tarifa corresponent (en euros per minut). Es defineix mitjançant els següents atributs:

- LLOCS_X_PLANTA**: atribut de classe públic constant, amb valor 5, que indica el número de llocs per planta.
- plantes**: atribut d'instància privat, array d'objectes **Planta** d'una grandària donada (quan es crea l'objecte en el constructor) que representa les plantes del parking. El número de cada planta coincideix amb la seua posició en aquest array, de manera que **plantes[i]** és la **Planta** de número *i*, sent $0 \leq i < \text{plantes.length}$.
- tarifa**: atribut d'instància privat, real positiu que representa la tarifa (en euros per minut) del parking.

El comportament dels objectes d'aquesta classe ve donat pels següents mètodes:

- Constructor **Parking(int, double)**: donats el número de plantes i la tarifa, crea un objecte **Parking** per representar un parking buit (veure figura 11).

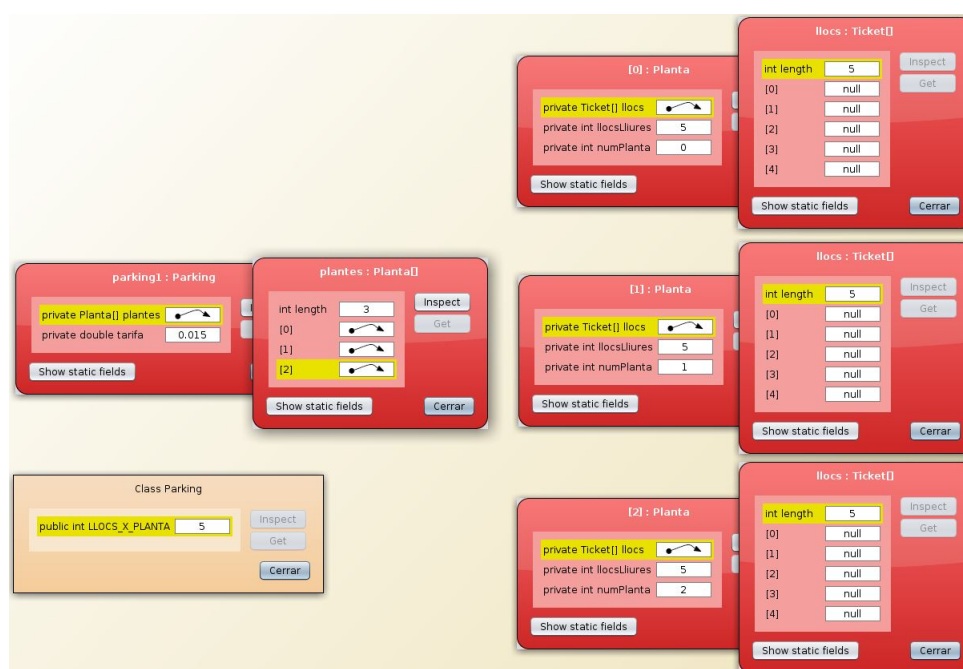
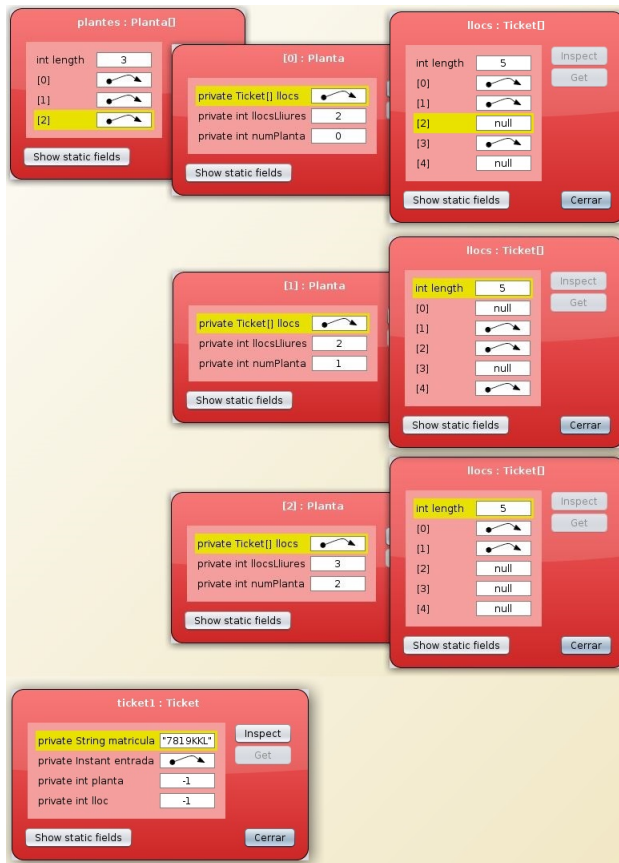
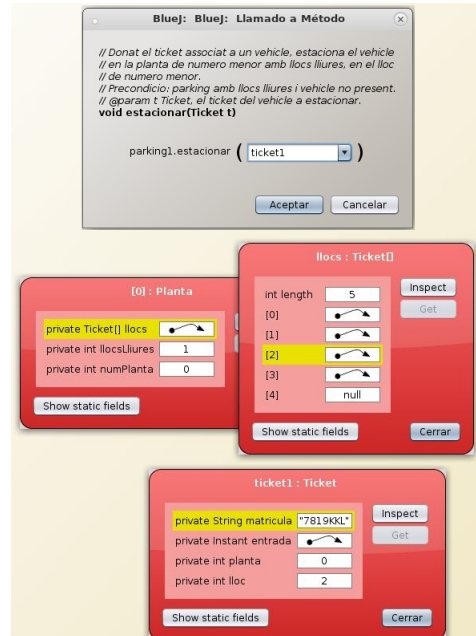


Figura 11: Objecte **Parking** que representa un parking amb 3 plantes, 5 llocs per planta (tots lliures) i amb una tarifa de 0.015€/minut.

- Constructor **Parking(String)**: constructor alternatiu que llegeix d'un fitxer de text, en un format específic, les dades d'ocupació del parking en un moment determinat i inicialitza amb aquestes dades els atributs d'instància de l'objecte **Parking** creat. Si obres amb qualsevol editor de textos el fitxer **parkingIIP.txt** (o **parkingIIP-Ple.txt**) veuràs que a la primera i segona línies apareixen, respectivament, el nombre de plantes i la tarifa del parking; i, en les línies que segueixen, les dades dels vehicles estacionats (nombre de planta, matrícula, hores i minuts d'entrada). Aquest mètode es proporciona completament implementat ja que l'estudi de fitxers és un tema que correspon a l'assignatura PRG del segon quadrimestre. S'utilitza ací per poder fer proves sense necessitat d'introduir manualment les dades del problema en cada execució.
 - Un mètode consultor del nombre de plantes i un mètode consultor de la tarifa: **getNumPlantes()** i **getTarifa()**.
 - Un mètode modificador que permet actualitzar la tarifa: **setTarifa(double)**.
 - Un mètode **estaPle()** que torna **true** si el parking està ple o, en cas contrari, torna **false**. El parking estarà ple si ho estan totes les seues plantes; per això, aquest mètode ha de fer ús del mètode **estaPlena()** de la classe **Planta**.
 - Un mètode **estacionar(Ticket)** que, suposant que al parking hi ha llocs lliures i donat un **Ticket** sense localització assignada, simula l'estacionament del vehicle associat al primer lloc lliure (de número menor) de la planta de número menor amb llocs lliures (veure figura 12); com segueix: una vegada trobada la planta de número menor amb llocs lliures, actualitza amb aquest número el número de planta del **Ticket** i invoca al mètode **estacionar(Ticket)** d'aquesta planta per realitzar l'estacionament.
 - Un mètode **estacionar(Ticket, int)** que, suposant que al parking hi ha llocs lliures i donats un **Ticket** sense localització assignada i un nombre de planta de preferència, simula l'estacionament del vehicle associat a aquesta planta si hi ha llocs lliures o, quan no hi ha cap lloc lliure a la preferència de planta indicada, en la planta més propera amb llocs lliures, segons s'il·lustra en la figura 13, per garantir un lloc en una planta el més propera possible la planta desitjada. Una vegada trobada la planta en la qual estacionar, actualitza amb el seu número el número de planta del **Ticket** i invoca al mètode **estacionar(Ticket)** d'aquesta planta per realitzar l'estacionament.
- Amb més detall, l'estratègia de cerca a implementar, reflectida en la figura 13, quan a la planta de preferència **pref** no hi ha llocs lliures, consisteix en buscar primer a la planta anterior **pref - 1** i, si tampoc n'hi ha, buscar a la planta posterior **pref + 1**. Si tampoc en aquesta planta n'hi ha, es repeteix la cerca per a les plantes **pref - 2** i **pref + 2** i així, successivament, fins a trobar una planta amb llocs lliures.
- Un mètode **buscarTicket(String)** que comprova si un vehicle, donada la seua matrícula, està en el parking (veure figura 14), tornant el **Ticket** associat al vehicle, si es troba, o **null** en cas contrari. Aquest mètode ha de fer ús del mètode **buscarTicket(String)** de **Planta**.
 - Un mètode **retirar(Ticket, Instant)** que simula la retirada d'un vehicle del parking, donats un **Ticket** vàlid (amb localització assignada i el vehicle del qual està al parking) i un **Instant** que representa una hora d'eixida (posterior a l'hora d'entrada del vehicle), tornant l'import a pagar (veure figura 15). Com retirar un vehicle del parking és retirar-lo de la planta on es troba, aquest mètode ha de fer ús del mètode **retirar(int, Instant)** de **Planta**.
 - Un mètode **buidarParking()** que buida el parking, retirant tots els vehicles, suposant que són les 23:59, i tornant l'import total en euros a pagar per tots els vehicles retirats. Com buidar el parking és retirar tots els vehicles de totes les plantes, aquest mètode ha de fer ús del mètode **retirarTots(Instant)** de **Planta**.



(a) Array `plantes` d'un Parking i `Ticket` del vehicle a estacionar.



(b) Resultat d'estacionar en el primer lloc lliure (lloc 2 de la planta 0).

Figura 12: Resultat del mètode `estacionar(Ticket)`.

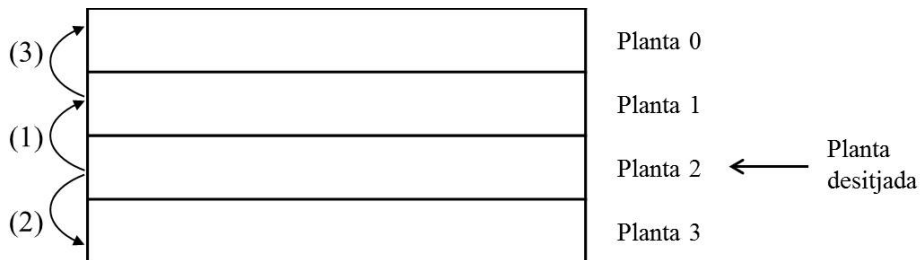


Figura 13: Estratègia de cerca a implementar en el mètode `estacionar(Ticket, int)` per obtenir una planta amb llocs lliures.

- Un mètode `toString()` (que sobreesciu el d'`Object`) que torna un `String` representant l'ocupació del parking amb el següent format: una primera línia per identificar els llocs corresponents a cada planta `i`, a continuació, tantes línies com plantes representat l'ocupació de cada planta (resultat del mètode `toString()` de `Planta`). Per exemple, per a un parking amb 3 plantes i 5 llocs per planta, en el qual estan ocupats: els llocs 0, 1 i 3 a la planta 0; els llocs 1, 2 i 4 a la planta 1; i els llocs 0 i 1 a la planta 2 (situació del Parking de la figura 12(a)), l'`String` resultat serà el que es mostra a la Finestra de Terminal de `BlueJ` de la figura 16.

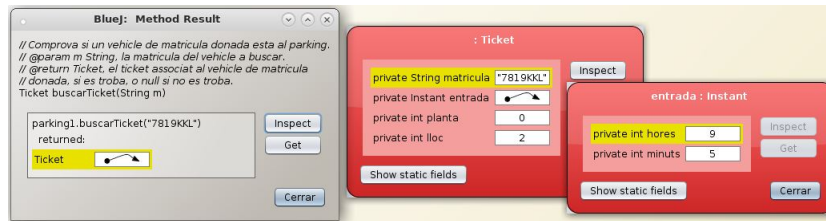


Figura 14: Resultat del mètode `buscarTicket(String)` al Parking de la figura 12(b) per tal de localitzar un vehicle de matrícula 7819KKL.

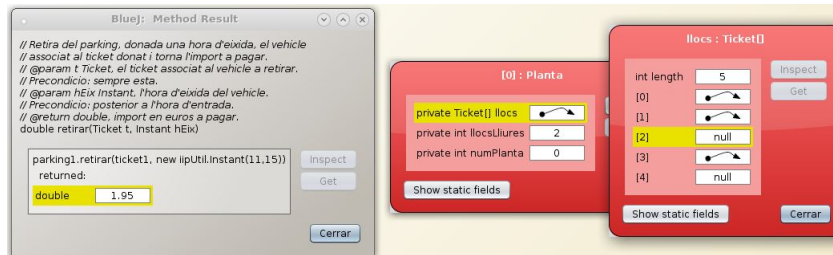


Figura 15: Resultat del mètode `retirar(Ticket, Instant)` al Parking de la figura 12(b) en retirar, a les 11:15, el vehicle de matrícula 7819KKL.

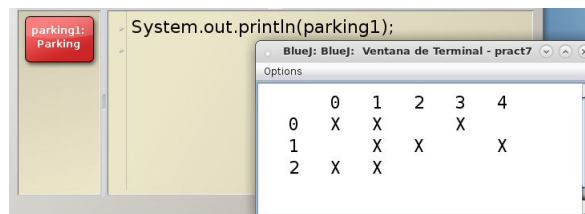


Figura 16: Resultat del mètode `toString()` al Parking de la figura 12(a).

Es mostra, a continuació, aquest `String` però substituint els espais en blanc per - perquè s'aprecie clarament el format utilitzat:

```

-----0---1---2---3---4-
--0---X---X-----X-----
--1-----X---X-----X-
--2---X---X-----

```

Activitat 6: implementació i prova de la classe Parking

Implementa els atributs i mètodes de la classe `Parking`. En acabar, pots provar-la utilitzant el *Code Pad* o l'*Object Bench* de *BlueJ*. En particular, comprova el comportament del constructor i dels mètodes `estacionar(Ticket)`, `estacionar(Ticket, int)` (fent especial atenció al cas en què la planta desitjada no té llocs lliures), `buscarTicket(String)`, `retirar(Ticket, Instant)`, `buidarParking()` i `toString()`.

Pots crear els teus propis fitxers per definir una situació del parking o pots utilitzar els fitxers de text proporcionats (`parkingIIP.txt` i `parkingIIP-Ple.txt`) per crear un parking amb una ocupació concreta (amb el constructor `Parking(String)`) i, així, fer proves més ràpides. Per exemple, la situació de la figura 12(a) és el resultat de crear un `Parking` a partir de les dades del fitxer `parkingIIP-Ple.txt` i anar executant els mètodes `buscarTicket(String)` i `retirar(Ticket, Instant)` per als vehicles amb les següents matrícules: 7819KKL, 8199BBD, 1123DXC, 7264BPR, 67890BD, 54321EF i 98796GH.

Activitat 7: validació de la classe Parking

Valida la classe `Parking` executant el test `TestParking`. 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.

6 La classe GestorParking

La classe `GestorParking` és una aplicació dirigida per menú que ofereix sis opcions diferents: estacionar un vehicle, retirar un vehicle, buscar un vehicle, mostrar l'ocupació, buidar el parking i finalitzar l'aplicació.

Aquesta classe programa, al començament, ha de crear un parking buit, amb un número de plantes i tarifa en euros per minut triats per tu (et recomanem fer servir valors menuts per fer proves ràpides).

El programa ha d'executar un bucle que mostre el menú i demane una opció a l'usuari (et recomanem fer servir un mètode estàtic per a aquesta tasca). Depenent de l'opció escollida ha de realitzar un conjunt diferent d'accions:

1. Estacionar: si el parking no està ple

- Demana la matrícula del vehicle.
- Si no està al parking:
 - Demana una hora d'entrada vàlida.
 - Demana un número de planta vàlid.
 - Crea un ticket per al vehicle i l'estaciona, mostrant les seues dades.
- Si el vehicle està, mostra el missatge “El vehicle ja està al parking”.

Si el parking està ple, mostra el missatge “No queden llocs lliures al parking”.

2. Retirar:

- Demana la matrícula del vehicle.
- Si el vehicle no està, mostra el missatge “El vehicle no està al parking”.
- Si el vehicle està:
 - Demana una hora d'eixida (vàlida i posterior a l'hora d'entrada).
 - Mostra les dades del ticket associat al vehicle.
 - El retira i mostra l'import a pagar.

3. Buscar:

- Demana la matrícula del vehicle.
- Si el vehicle no està, mostra el missatge “El vehicle no està al parking”.
- Si el vehicle està, mostra les dades del ticket associat.

4. Mostrar ocupació: mostra l'ocupació del parking.

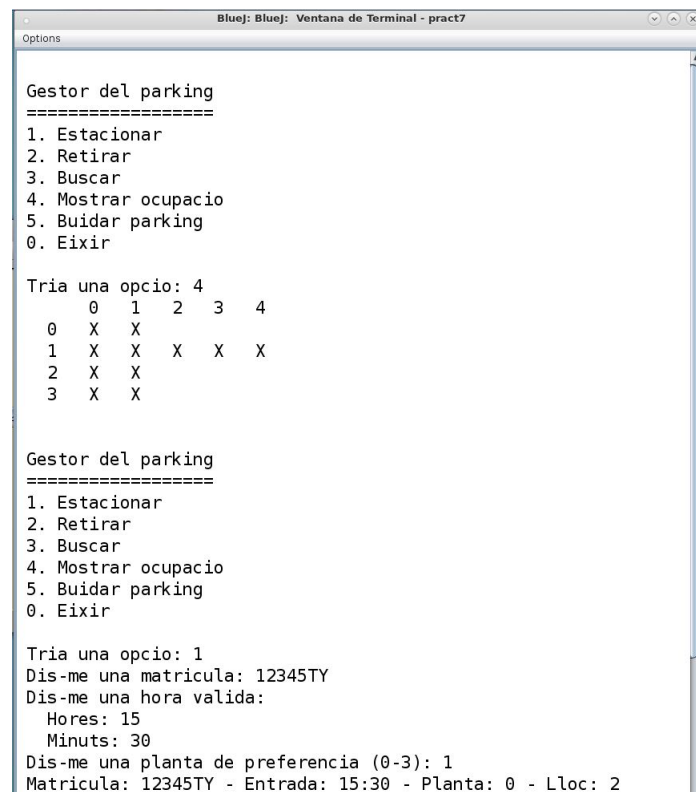
5. Buidar parking: buida el parking i mostra l'import final acumulat.

6. Eixir: finalitza l'execució de l'aplicació mostrant un missatge d'acomiadament.

Per fer més fàcil la lectura des de teclat de l'hora, et recomanem implementar un mètode per llegir una hora correcta (és a dir, hores entre 0 i 23, i minuts entre 0 i 59) i tornar el corresponent objecte `Instant`.

Activitat 8: implementació i prova de la classe GestorParking

Implementa la classe programa amb el mètode `main` i els mètodes auxiliars que necessites (per exemple, `menu(Scanner)` i `llegirInstant(Scanner)`). Implementa primer les opcions per finalitzar l'aplicació i mostrar l'ocupació. Implementa la resta d'opcions en ordre i comprova el comportament de cada opció abans d'implementar la següent opció; pots utilitzar l'opció mostrar l'ocupació per comprovar el comportament eficaç de cada operació. Pots utilitzar els fitxers de text (`parkingIIP.txt` i `parkingIIP-Ple.txt`) per crear un parking amb una ocupació concreta (amb el constructor `Parking(String)`) i fer proves més ràpides en aquesta situació. A la figura 17 es mostra un exemple de l'execució de `GestorParking` amb les dades del fitxer `parkingIIP.txt`.



```
BlueJ: BlueJ: Ventana de Terminal - pract7
Options

Gestor del parking
=====
1. Estacionar
2. Retirar
3. Buscar
4. Mostrar ocupacio
5. Buidar parking
0. Eixir

Tria una opcio: 4
   0  1  2  3  4
0  X  X
1  X  X  X  X  X
2  X  X
3  X  X

Gestor del parking
=====
1. Estacionar
2. Retirar
3. Buscar
4. Mostrar ocupacio
5. Buidar parking
0. Eixir

Tria una opcio: 1
Dis-me una matricula: 12345TY
Dis-me una hora valida:
Hores: 15
Minuts: 30
Dis-me una planta de preferencia (0-3): 1
Matricula: 12345TY - Entrada: 15:30 - Planta: 0 - Lloc: 2
```

Figura 17: Exemple d'execució de `GestorParking`.