Laboratori de Java

Estructura de la Informació / Disseny d'Estructures de Dades

Annex: exemples d'interrelacions de TADs

Índex de continguts

1. Introducció

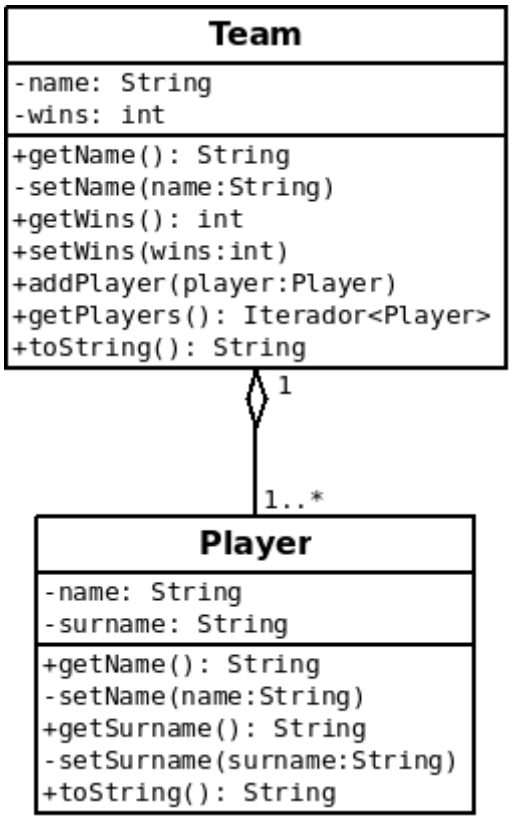
El present document té com a objectiu mostrar un seguit d'exemples d'interrelacions de TADs. No es pretén entrar en la presa de decisions prèvia a l'elecció de TADs, sinó que en base a un escenari predeterminat, es volen exposar des d'una vessant pràctica algunes relacions que es poden establir entre TADs, usant la biblioteca de TADs com a element de suport per a construir les noves estructures.

2. Primer escenari

Es vol gestionar un conjunt d'equips de bàsquet, on cada equip està format per un conjunt de jugadors.

2.1. Relació entre classes

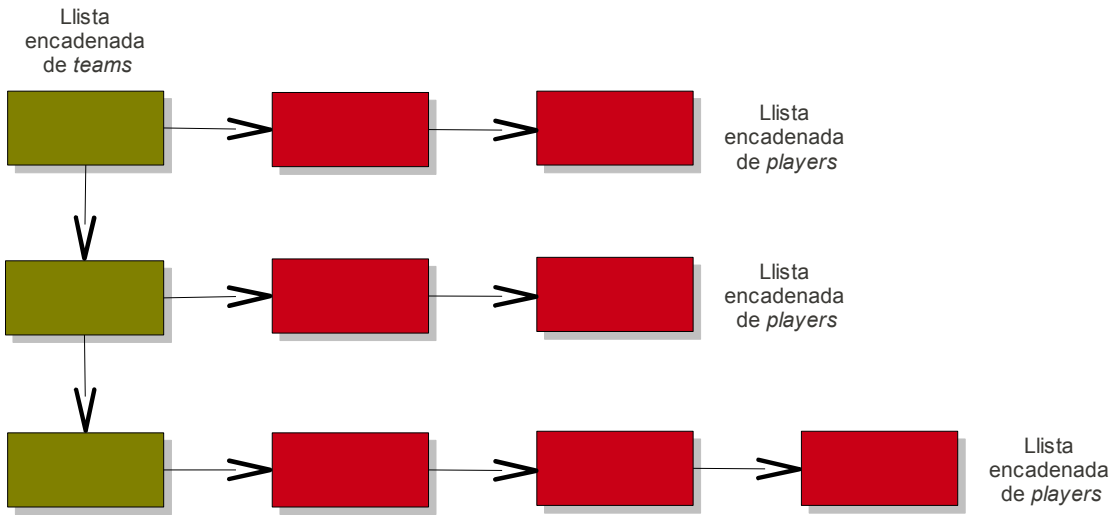
Es disposa d'una classe `Team` per a implementar un equip de bàsquet, i una classe `Player` per a instanciar els jugadors que formen l'equip.



Donada la relació de composició entre les classes, `Team` contindrà com a atribut un TAD que contindrà un conjunt de tipus `Player`.

2.2. TADs

El primer escenari planteja una *llista encadenada* d'equips (objectes de tipus `Team`), cadascun dels quals està composat per una *llista encadenada* de jugadors (objectes de tipus `Player`).



2.3. Implementació

Els passos a seguir per a implementar aquest escenari són:

1. Crear la classe `Player`, segons el diagrama UML anterior:

```
public class Player {
	private String name;
	private String surname;

	public Player(String name, String surname) {
		setName(name);
		setSurname(surname);
	}

	public String getName() {
		return name;
	}

	private void setName(String name) {
		this.name = name;
	}

	public String getSurname() {
		return surname;
	}
```

```
	private void setSurname(String surname) {
		this.surname = surname;
	}

	public String toString() {
		return getSurname()+", "+getName();
	}
}
```

2. Crear la classe `Team`, segons el diagrama UML anterior:

```
import uoc.ei.tads.Iterador;
import uoc.ei.tads.LlistaEncadenada;

public class Team {

	private String name;
	private int wins;
	private LlistaEncadenada<Player> players=new LlistaEncadenada<Player>();

	public Team(String name) {
		setName(name);
		setWins(0);
	}

	public String getName() {
		return name;
	}

	private void setName(String name) {
		this.name = name;
	}

	public int getWins() {
		return wins;
	}

	public void setWins(int wins) {
		this.wins = wins;
	}

	public void addPlayer(Player player) {
		players.afegirAlFinal(player);
	}

	public Iterador<Player> getPlayers(){
		return players.elements();
	}

	public String toString() {
		return getName()+", ["+getWins()+"]";
	}
}
```

Com es pot veure, `Team` conté un atribut intern de tipus `LlistaEncadenada`, en el qual es desaran els objectes de tipus `Player` que conté l'equip:

```
private LlistaEncadenada<Player> players=new LlistaEncadenada<Player>();
```

Les referències `<Player>` que s'indiquen per `LlistaEncadenada` defineixen quin tipus d'objectes contindrà el TAD `LlistaEncadenada`.

Per tal d'inserir una instància `Player` dins de la `LlistaEncadenada` de `Team`, s'utilitza el mètode `addPlayer(Player p)`, el qual realitza una crida al mètode `afegirAlFinal(Player)` propi de `LlistaEncadenada`.

3. Crear la classe executable, que permetrà la validació de l'escenari plantejat:

```
import uoc.ei.tads.Iterador;
import uoc.ei.tads.LlistaEncadenada;

public class Example1 {

	public static void main(String args[]){

	/* Inici inicialització */
		/* Es creen els equips */
		Team chi=new Team("Chicago Bulls");
		Team ind=new Team("Indiana Pacers");
		Team mil=new Team("Milwaukee Bucks");
		Team cle=new Team("Cleveland Cavaliers");
		Team det=new Team("Detroit Pistons");

		/* S'assignen els equips a la Central Division */
		LlistaEncadenada<Team> centralDivision = new
                                                LlistaEncadenada<Team>();
		centralDivision.afegirAlFinal(chi);
		centralDivision.afegirAlFinal(mil);
		centralDivision.afegirAlFinal(cle);
		centralDivision.afegirAlFinal(ind);
		centralDivision.afegirAlFinal(det);

		/* Es creen els jugadors */
		Player chi05=new Player("Carlos", "Boozer");
		Player chi01=new Player("Derrick", "Rose");
		Player chi13=new Player("Joakim", "Noah");
		Player chi09=new Player("Luol", "Deng");
		Player chi03=new Player("Omek", "Asik");
		Player cle02=new Player("Kyrie", "Irving");
		Player cle17=new Player("Anderson", "Varejao");
		Player cle04=new Player("Antawn", "Jamison");
		Player cle01=new Player("Daniel", "Gibson");
		Player cle33=new Player("Alonzo", "Gee");
		Player ind33=new Player("Danny", "Granger");
		Player ind02=new Player("Darren", "Collison");
		Player ind21=new Player("David", "West");
		Player ind17=new Player("Lou", "Amundson");
		Player ind55=new Player("Roy", "Hibbert");
```

```
        Player mil06=new Player("Andrew", "Bogut");
        Player mil00=new Player("Drew", "Gooden");
        Player mil07=new Player("Ersan", "Ilyasova");
        Player mil03=new Player("Brandon", "Jennings");
        Player mil05=new Player("Stephen", "Jackson");
        Player det08=new Player("Ben", "Gordon");
        Player det22=new Player("Tayshaun", "Prince");
        Player det07=new Player("Brandon", "Knight");
        Player det10=new Player("Greg", "Monroe");
        Player det03=new Player("Rodney", "Stuckey");

        /* S'assignen els jugadors als respectius equips */
        chi.addPlayer(chi05);
        chi.addPlayer(chi01);
        chi.addPlayer(chi13);
        chi.addPlayer(chi09);
        chi.addPlayer(chi03);
        cle.addPlayer(cle02);
        cle.addPlayer(cle17);
        cle.addPlayer(cle04);
        cle.addPlayer(cle01);
        cle.addPlayer(cle33);
        ind.addPlayer(ind33);
        ind.addPlayer(ind02);
        ind.addPlayer(ind21);
        ind.addPlayer(ind17);
        ind.addPlayer(ind55);
        mil.addPlayer(mil06);
        mil.addPlayer(mil00);
        mil.addPlayer(mil07);
        mil.addPlayer(mil03);
        mil.addPlayer(mil05);
        det.addPlayer(det08);
        det.addPlayer(det22);
        det.addPlayer(det07);
        det.addPlayer(det10);
        det.addPlayer(det03);

        /* S'inicialitzen els partits guanyats dels equips */
        chi.setWins(33);
        mil.setWins(15);
        cle.setWins(14);
        ind.setWins(23);
        det.setWins(13);

    /* Fi inicialització */
    /* Inici recorreguts */

        /* Es mostra per pantalla els equips de la Central Division */
        System.out.println("\nEquips de la Divisió Central:\n");
        Iterador<Team> teams=centralDivision.elements();
        while (teams.hiHaSeguent()) {
              Team teamAux=teams.seguent();
              System.out.println(teamAux.toString());
        }
```

```
                /* Es mostra per pantalla els jugadors definits de la Central
                       Division */
                System.out.println("\nJugadors de la Divisió Central:\n");
                teams=centralDivision.elements();
                while (teams.hiHaSeguent()) {
                       Team teamAux=teams.seguent();
                       Iterador<Player> players=teamAux.getPlayers();
                       while (players.hiHaSeguent()) {
                              Player playerAux=players.seguent();
                              System.out.println(playerAux.toString());
                       }
                }

                /* Es mostra per pantalla els jugadors definits de la Central
                       Division, agrupats per equip */
                System.out.println("\nJugadors per equip de la Divisió Central:\n");
                teams=centralDivision.elements();
                while (teams.hiHaSeguent()) {
                       Team teamAux=teams.seguent();
                       System.out.println("\n"+teamAux.getName());
                       System.out.println("------------------------");
                       Iterador<Player> players=teamAux.getPlayers();
                       while (players.hiHaSeguent()) {
                              Player playerAux=players.seguent();
                              System.out.println("\t"+playerAux.toString());
                       }
                }
        /* Fi recorreguts */
        }
}
```

Si ens fixem en el codi, el conjunt de `Team` s'inclouen dins de `LlistaEncadenada<Team>`, de forma que aquest TAD en concret únicament podrà contenir objectes de tipus `Team`.

La sortida que genera el primer bucle és:

```
Equips de la Divisió Central:

Chicago Bulls, [33]
Milwaukee Bucks, [15]
Cleveland Cavaliers, [14]
Indiana Pacers, [23]
Detroit Pistons, [13]
```

La sortida del segon bucle és:

```
Jugadors de la Divisió Central:

Boozer, Carlos
Rose, Derrick
Noah, Joakim
Deng, Luol
Asik, Omek
Bogut, Andrew
Gooden, Drew
Ilyasova, Ersan
```

```
Jennings, Brandon
Jackson, Stephen
Irving, Kyrie
Varejao, Anderson
Jamison, Antawn
Gibson, Daniel
Gee, Alonzo
Granger, Danny
Collison, Darren
West, David
Amundson, Lou
Hibbert, Roy
Gordon, Ben
Prince, Tayshaun
Knight, Brandon
Monroe, Greg
Stuckey, Rodney
```

La del tercer bucle:

```
Jugadors per equip de la Divisió Central:

Chicago Bulls
------------------------
       Boozer, Carlos
       Rose, Derrick
       Noah, Joakim
       Deng, Luol
       Asik, Omek

Milwaukee Bucks
------------------------
       Bogut, Andrew
       Gooden, Drew
       Ilyasova, Ersan
       Jennings, Brandon
       Jackson, Stephen

Cleveland Cavaliers
------------------------
       Irving, Kyrie
       Varejao, Anderson
       Jamison, Antawn
       Gibson, Daniel
       Gee, Alonzo

Indiana Pacers
------------------------
       Granger, Danny
       Collison, Darren
       West, David
       Amundson, Lou
       Hibbert, Roy

Detroit Pistons
------------------------
       Gordon, Ben
       Prince, Tayshaun
       Knight, Brandon
       Monroe, Greg
       Stuckey, Rodney
```

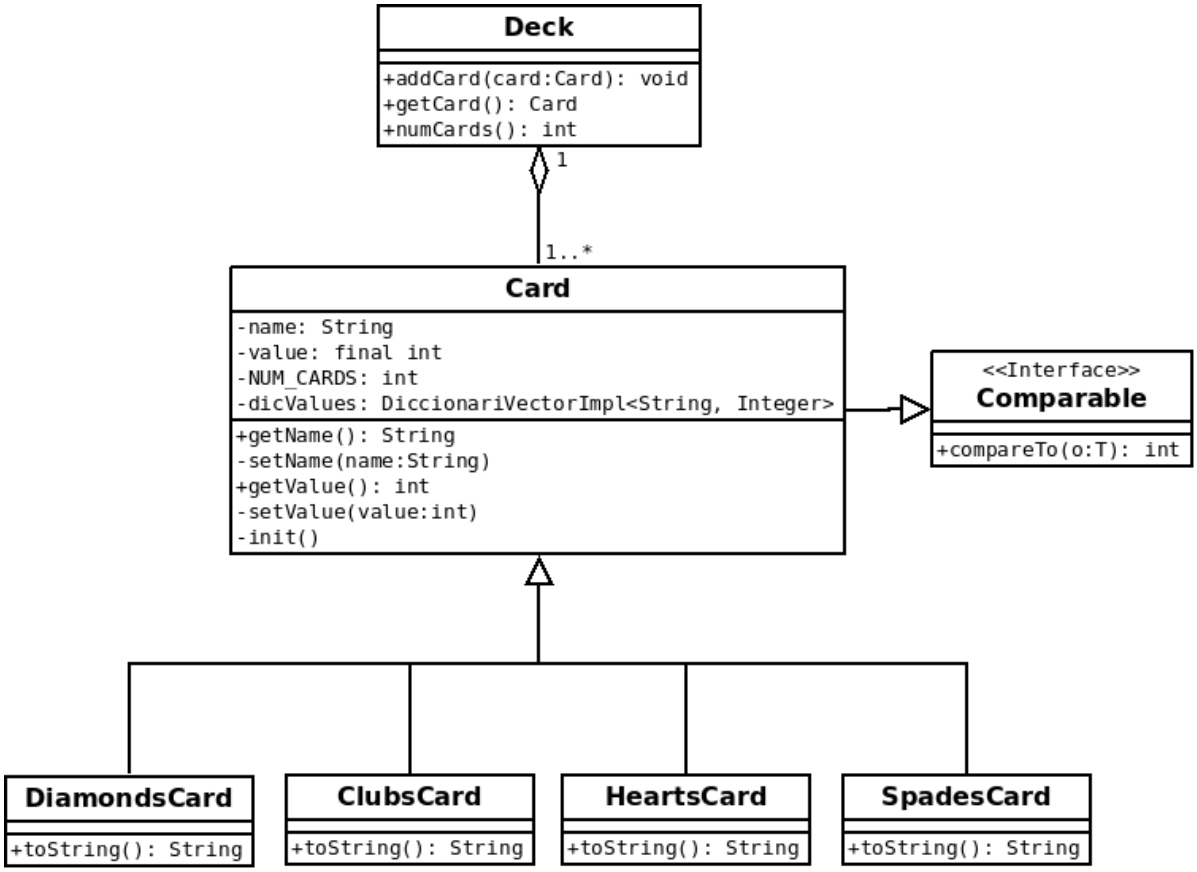
3. Segon escenari

Es vol simular les piles de cartes que es produeixen quan s'està, per exemple, ordenant una baralla manualment, agafant una carta cada cop i inserint-la a la pila que li correspon.

3.1. Relació entre classes

Es disposa d'una classe `Card`, de la qual hereten les filles `DiamondsCard`, `ClubsCard`, `HeartsCard` i `SpadesCard`. Totes les cartes s'inclouran a la classe `Deck`, de pòquer. La relació de composició entre les classes implica que `Deck` té un atribut TAD amb el conjunt de les `Card`.

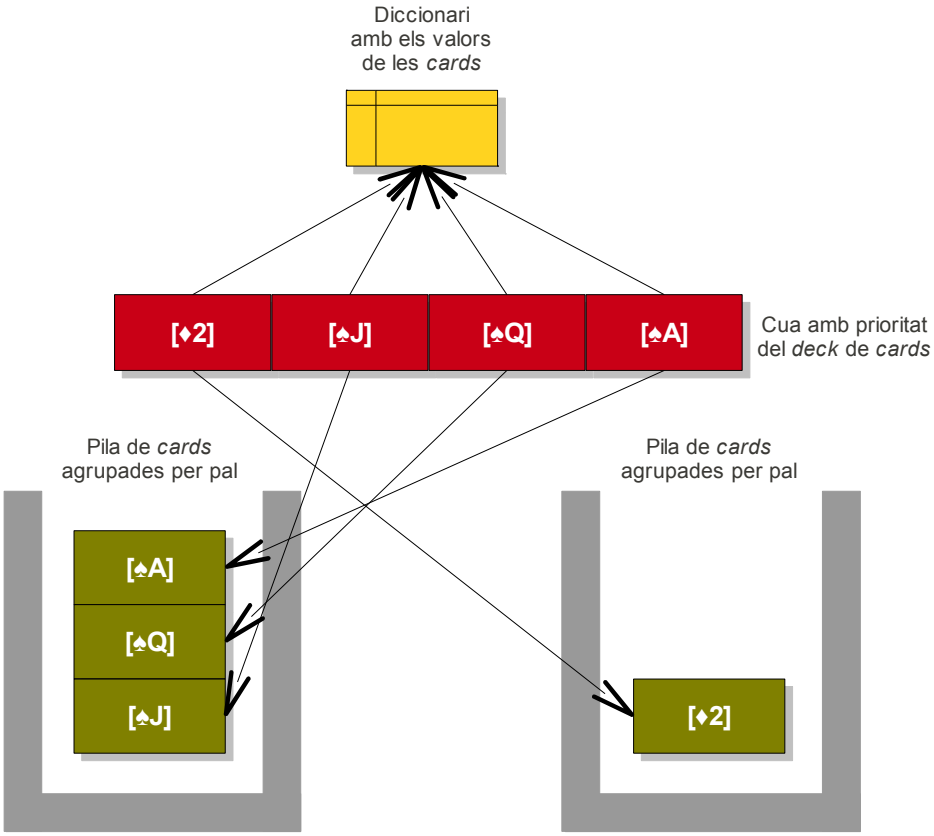
A més, `Card` implementa la *interfície* `Comparable`: l'objectiu no és cap altre que permetre la comparació entre dues instàncies `Card` i poder establir així una ordenació determinada entre elles.



3.2. TADs

El segon escenari utilitza el TAD *cua amb prioritat* per contenir les `Card`. La prioritat de la cua ens permetrà tenir les cartes ordenades ascendentment segons el seu valor. Tot seguit, s'utilitzaran quatre piles per tal d'agrupar les cartes segons el seu pal; encuar i desencuar les cartes de les piles provocarà que les obtinguem en ordre invers.

Per establir la relació entre el nom d'una `Card` i el seu valor, s'ha utilitzat un TAD de tipus *diccionari*.



3.3. Implementació

Els passos a seguir per implementar el segon escenari són:

1. Crear la classe `Card`, segons el diagrama UML anterior:

```
import uoc.ei.tads.DiccionariVectorImpl;

public class Card implements Comparable<Card> {

	private static final int NUM_CARDS=13;
	private int value;
	private String name;
	private static DiccionariVectorImpl<String, Integer> dicValues
		= new DiccionariVectorImpl<String, Integer>(NUM_CARDS);

	private void init() {
		dicValues.afegir(new String("2"), new Integer(1));
		dicValues.afegir(new String("3"), new Integer(2));
		dicValues.afegir(new String("4"), new Integer(3));
		dicValues.afegir(new String("5"), new Integer(4));
		dicValues.afegir(new String("6"), new Integer(5));
		dicValues.afegir(new String("7"), new Integer(6));
		dicValues.afegir(new String("8"), new Integer(7));
		dicValues.afegir(new String("9"), new Integer(8));
		dicValues.afegir(new String("10"), new Integer(9));
		dicValues.afegir(new String("J"), new Integer(10));
		dicValues.afegir(new String("Q"), new Integer(11));
		dicValues.afegir(new String("K"), new Integer(12));
		dicValues.afegir(new String("A"), new Integer(13));
	}

	public Card(String nameCard) {
		init();
		setName(nameCard);
		setValue(dicValues.consultar(nameCard));
	}

	public int getValue() {
		return value;
	}

	private void setValue(int value) {
		this.value = value;
	}

	public String getName() {
		return name;
	}

	private void setName(String name) {
		this.name = name;
	}

	public int compareTo(Card card) {
```

```
                return this.getValue()-card.getValue();
        }
}
```

2. Crear la classe filla `ClubsCard`:

```
public class ClubsCard extends Card {

        public ClubsCard(String nameCard){
                super(nameCard);
        }

        public String toString(){
                return "[♣"+this.getName()+"]";
        }
}
```

3. Crear la classe filla `DiamondsCard`:

```
public class DiamondsCard extends Card {

        public DiamondsCard(String nameCard){
                super(nameCard);
        }

        public String toString(){
                return "[♦"+this.getName()+"]";
        }
}
```

4. Crear la classe filla `SpadesCard`:

```
public class SpadesCard extends Card {

        public SpadesCard(String nameCard){
                super(nameCard);
        }

        public String toString(){
                return "[♠"+this.getName()+"]";
        }
}
```

5. Crear la classe filla `HeartsCard`:

```
public class HeartsCard extends Card {

        public HeartsCard(String nameCard){
```

```
                super(nameCard);
        }

        public String toString(){
                return "[♥"+this.getName()+"]";
        }
}
```

6. Crear la classe `Deck`, segons el diagrama UML anterior:

```
import uoc.ei.tads.Cua;
import uoc.ei.tads.CuaAmbPrioritat;

public class Deck {

        private Cua<Card> cards=new CuaAmbPrioritat<Card>();

        public void addCard(Card card) {
                cards.encuar(card);
        }

        public Card getCard() {
                return cards.desencuar();
        }

        public int numCards() {
                return cards.nombreElems();
        }
}
```

En aquest cas, `Deck` conté un atribut intern de tipus `CuaAmbPrioritat`, en el qual es desaran les `Card` que conté la baralla de cartes:

```
private Cua<Card> cards=new CuaAmbPrioritat<Card>();
```

7. Crear la classe executable, que permetrà la validació de l'escenari plantejat:

```
import uoc.ei.tads.Pila;
import uoc.ei.tads.PilaVectorImpl;

public class Example2 {

        public static void main(String args[]){

        /* Inici inicialització */

                /* Es crea la baralla de cartes de pòquer */
                Deck deck = new Deck();

                /* Es creen les cartes */
                Card h2=new HeartsCard("2");
                Card h3=new HeartsCard("3");
```

```
        Card hJ=new HeartsCard("J");
        Card c4=new ClubsCard("4");
        Card c6=new ClubsCard("6");
        Card sJ=new SpadesCard("J");
        Card sQ=new SpadesCard("Q");
        Card sK=new SpadesCard("K");
        Card d2=new DiamondsCard("2");
        Card dK=new DiamondsCard("K");
        Card dA=new DiamondsCard("A");

        /* S'afegeixen les cartes a la baralla */
        deck.addCard(h2);
        deck.addCard(sJ);
        deck.addCard(d2);
        deck.addCard(dK);
        deck.addCard(sK);
        deck.addCard(sQ);
        deck.addCard(h3);
        deck.addCard(c6);
        deck.addCard(hJ);
        deck.addCard(c4);
        deck.addCard(dA);

        /* Es creen quatre piles de cartes, una per cadascun dels pals
         * de la baralla */
        Pila<ClubsCard> clubsCards=new PilaVectorImpl<ClubsCard>();
        Pila<DiamondsCard> diamondsCards=new PilaVectorImpl<DiamondsCard>();
        Pila<SpadesCard> spadesCards=new PilaVectorImpl<SpadesCard>();
        Pila<HeartsCard> heartsCards=new PilaVectorImpl<HeartsCard>();

    /* Fi inicialització */
    /* Inici recorreguts */

        /* Es mostra per pantalla cadascuna de les cartes de la baralla,
         * de forma ordenada. A més, en funció de la naturalesa de cada
         * carta, s'insereix a la pila corresponent del mateix pal
         */
        System.out.print("Total cartes :\n\t");
        while (deck.numCards()>0) {
              Card aux=deck.getCard();
              System.out.print(aux.toString()+" ");
              if (aux instanceof ClubsCard) {
                    clubsCards.empilar((ClubsCard)aux);
              } else if (aux instanceof DiamondsCard) {
                    diamondsCards.empilar((DiamondsCard)aux);
              } else if (aux instanceof SpadesCard) {
                    spadesCards.empilar((SpadesCard)aux);
              } else if (aux instanceof HeartsCard) {
                    heartsCards.empilar((HeartsCard)aux);
              }
        }

        /* Es desapilen les cartes diamond, i es mostren per pantalla*/
        System.out.print("\n\nDiamons desempilades :\n\t");
        while (!diamondsCards.estaBuit()) {
```

```
                System.out.print(diamondsCards.desempilar()+" ");
            }

            /* Es desapilen les cartes spades, i es mostren per pantalla*/
            System.out.print("\n\nSpades desempilades :\n\t");
            while (!spadesCards.estaBuit()) {
                System.out.print(spadesCards.desempilar()+" ");
            }

            /* Es desapilen les cartes hearts, i es mostren per pantalla*/
            System.out.print("\n\nHearts desempilades :\n\t");
            while (!heartsCards.estaBuit()) {
                System.out.print(heartsCards.desempilar()+" ");
            }

            /* Es desapilen les cartes clubs, i es mostren per pantalla*/
            System.out.print("\n\nClubs desempilades :\n\t");
            while (!clubsCards.estaBuit()) {
                System.out.print(clubsCards.desempilar()+" ");
            }
        /* Fi recorreguts */
        }
}
```

La sortida que genera el primer recorregut és:

```
Total cartes :
      [♥2] [♦2] [♥3] [♣4] [♣6] [♥J] [♠J] [♠Q] [♦K] [♠K] [♦A]
```

Com es pot observar, malgrat s'insereixen les `Card` al `Deck` de forma desordenada, en recórrer la cua amb prioritat es mostren en ordre. L'ordre ascendent o descendent queda definit pel resultat positiu o negatiu que retorna el mètode `compareTo(Card card)` de `Card`.

Els següents quatre recorreguts corresponen a l'acció de desempilar les `Card` de les cues per cadascun dels pals de la baralla:

```
Diamons desempilades :
      [♦A] [♦K] [♦2]

Spades desempilades :
      [♠K] [♠Q] [♠J]

Hearts desempilades :
      [♥J] [♥3] [♥2]

Clubs desempilades :
      [♣6] [♣4]
```