Creiamo una progetto contenente le seguenti classi:

- Carta
- Mazzo
- Gioco

Una carta è un'entità avente due proprietà che la caratterizzano: Seme e Valore.
Li rappresentiamo come String in modo che possano essere generali.

Un mazzo è un insieme di carte, e ne rappresenta la natura: un mazzo da ramino avrà certi valori di seme e valore, uno da briscola ne avrà altri.

Class Carta
In Carta scriveremo un metodo (`confronta`) per confrontare una carta data con un'altra, verificando se seme e valore della carta sono uguali a quelli della carta passata come parametro e restituendo un booleano.
Scriveremo anche un metodo `toString` per fare in modo che una carta possa essere stampata in output.

```
package it.unitn.lingprogmod1.esercitazione2;
public class Carta {
    String valore;
    String seme;
    public Carta(String valore, String seme) {
        this.valore = valore;
        this.seme = seme;
    }
    @Override
    public String toString() {
        return "Carta{" +
                "valore=" + valore +
                ", seme=" + seme +
                '}';
    }

    /**
     * metodo per confrontare questa carta con un altra data
     * @param c la carta con cui effettuare il confronto
     * @return vero se le due carte hanno lo stesso seme e lo stesso valore
     */
    boolean confronta(Carta c) {
        // nota: nel confronto tra String si potrebbe usare == ma è più
corretto usare invece .equals()
        // scopriremo perché in una prossima lezione
        return (this.seme.equals(c.seme)) &&
(this.valore.equals(c.valore));
    }
}
```

Nota: il tipo di commento usato per il metodo *confronta* (che inizia con /**) permette di generare automaticamente la documentazione del programma (javadoc) – ne parleremo in una prossima lezione.

Class Mazzo
Il mazzo è un insieme di carte, quindi ci conviene usare una Collection per poter fruire dei metodi che ci mette a disposizione e non doverci occupare della numerosità dell'insieme (con un array avremmo dovuto calcolare la dimensione del mazzo prima di crearlo). Il nostro mazzo sarà quindi sottoclasse di LinkedList (ma potrebbe essere anche di ArrayList). Se usassimo dei Set invece avremmo qualche complicazione che discuteremo una prossima volta.

Caratterizziamo il mazzo con la tipologia di carte usate (da ramino piuttosto che da briscola, dandone valori e semi.

Per il costruttore standard (vuoto) abbiamo due opzioni, che discutiamo dopo aver visto il codice.
Creiamo un secondo costruttore che costruisca un Mazzo a partire da un altro mazzo dato, prendendone solo le prime n carte. I suoi parametri saranno il numero n ed il Mazzo da cui copiarle.

Aggiungiamo:
- un metodo `mescola` (che farà uso delle API di Collections).
- un metodo `trovaPrimaDoppia` che esplora il mazzo cercando una doppia tra le carte
- un metodo `main` per testare del mazzo.

```
package it.unitn.lingprogmod1.esercitazione2;
import java.util.Collections;
import java.util.LinkedList;

    public class Mazzo extends LinkedList<Carta> {
        String[] valore = {"A", "2", "3", "4", "5", "6", "7", "8", "9",
"10", "J", "Q", "K"};
        String[] seme = {"C", "Q", "F", "P"};

        public Mazzo() { // opzione A
            for (int i = 1; i <= 2; i++) {// ripeto due volte il ciclo di
creazione
                for (String v : valore) {
                    for (String s: seme) {
                        Carta c = new Carta(v, s);
                        this.add(c);
                    }
                }
            }
        }
        /*
        public Mazzo() { // opzione B
            for (String v : valore) {
                for (String s: seme) {
                    Carta c = new Carta(v, s);
                    this.add(c);
                    this.add(c);
                }
            }
        }
        */
```

```
        /**
         * Costruttore che, dato un mazzo, ne costruisce un secondo
costituito dalle prime n carte
         * @param n numero di carte che compongono in mazzo creato
         * @param m mazzo da cui copiare le prime n carte
         */
        public Mazzo(int n, Mazzo m) {
            for (int i=0;i<n;i++) {
                this.add(m.get(i));
            }
        }

        void mescola() {
            Collections.shuffle(this);
        }

        @Override
        public String toString() {
            String s="Mazzo {\n";
            for (Carta c: this) {
                s = s + c.toString()+"\n";
            }
            s=s+'}';
            return s;
        }

        /**
         * Metodo che controlla se nel mazzo ci sono due carte uguali
         * @return la (prima) carta duppia trovata, o null se non vi sono
carte doppie
         */
        public Carta trovaPrimaDoppia(){
            int n=this.size();
            for (int i=0; i<n-1;i++) {
                Carta c1=this.get(i);
                for (int j=i+1; j<n; j++) {
                    Carta c2=this.get(j);
                    if (c1.confronta(c2)) {
                        // stampa di controllo, da levare dopo il debugging
                        System.out.println("found at "+i+" "+j+" :
"+c1.toString());
                        return c1;
                    }
                }
            }
            return null;
        }

        /**
         * main di test
         * @param a
         */
        public static void main(String a[]){
            Mazzo m=new Mazzo();
            m.mescola();
            m.trovaPrimaDoppia();
        }
    }
```

Nota sui due metodi alternativi di scrivere il costruttore: nella 1 due cicli interni (sui semi e sui valori) ci permettono di aggiungere una carta per tipo; un ciclo esterno ripetuto due volte fa si che mettiamo assieme due carte per ciascuna coppia seme-valore.
In alternativa (opzione 2) la stessa carta viene invece aggiunta due volte all'interno del doppio ciclo su semi e valori.
Nel caso uno, in heap avremo 108 carte, e nel mazzo 108 identificatori (uno per ogni carta creata in heap).
Nel caso due, in heap avremo solo 54 carte, e nel mazzo 108 identificatori (due per ogni carta creata in heap).
Ai fini di quel che vogliamo fare le due soluzioni sono equivalenti. Se invece volessimo avere un metodo "segnaCarta" che emula il fatto che una carta possa essere scarabocchiata, sarebbe necessario usare il primo approccio, perché con in secondo segnando il 2 di cuori scarabocchierei TUTTI i due di cuori, e non solo quello sul cui identificatore sto agendo!

Classe Gioco
Gioco contiene il main program: crea un mazzo, ne estrae un mazzetto composto di n carte (con n chiesto all'utente) e controlla se nel mazzetto vi sono doppie. Infine comunica al giocatore l'esito.

```
package it.unitn.lingprogmod1.esercitazione2;

import java.util.Scanner;

public class Gioco {
    public static void main(String a[]) {
        new Gioco();
    }
    public Gioco(){
        String msg;
        int n=0;
        Mazzo m = new Mazzo();
        m.mescola();
        Scanner scanner=new Scanner(System.in);
        boolean error=true;
        do {
            error=true;
            System.out.println("dammi un numero tra 1 e "+m.size());
            String valoreLetto=scanner.next();
            try {
                n=Integer.parseInt(valoreLetto);
                if (n>0 && n<= m.size()) {
                    error = false;
                } else {
                    System.out.println("ERRORE: Numero non accettabile");
                }
            } catch (NumberFormatException ex) {
                System.out.println("ERRORE: Non hai dato un numero
intero");
            }
        } while (error);
        Mazzo mazzetto = new Mazzo(10, m);
        // stampa il mazzetto estratto
        System.out.println(mazzetto);
        Carta c = mazzetto.trovaPrimaDoppia();
        if (c == null) {
            msg="Spiacente, hai perso";
```

```
        } else {
            msg="Hai vinto con " + c;
        }
        System.out.println(msg);
    }
}
```