Vettori e Array

Collezione di oggetti: Array List

- La classe `Purse` non tiene traccia dei singoli oggetti di tipo **Coin**, ma memorizza solo il valore totale
- Possiamo memorizzare una collezione di oggetti mediante la classe ArrayList (pacchetto `java.util`)

```
ArrayList coins = new ArrayList();
coins.add(new Coin(0.1, "dime"));
. . .
```

- Il metodo `size()` restituisce il numero di elementi della collezione

Aggiungere un elemento

- Per aggiungere l'elemento alla fine della collezione si usa il metodo `add(obj)`:

```
ArrayList coins = new ArrayList ();
coins.add(new Coin(0.1, "dime"));
coins.add(new Coin(0.25, "quarter");
```

- Dopo l'inserimento, la dimensione della collezione aumenta di uno

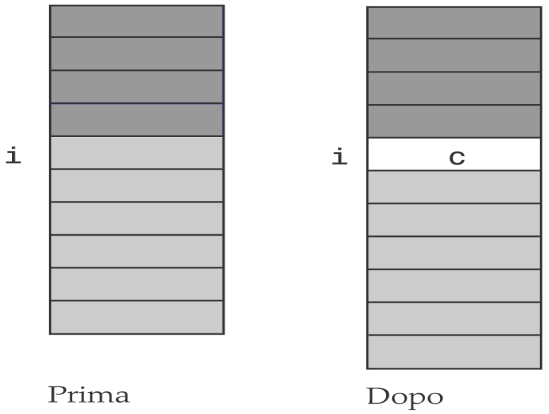
Aggiungere un elemento

- Per aggiungere l'elemento in una certa posizione, facendo slittare in avanti gli altri, si usa il metodo `add(i,obj)`:

```
ArrayList coins = new ArrayList ();
coins.add(new Coin(0.1, "dime"));
coins.add(new Coin(0.25, "quarter");

Coin aNickel = new Coin(0.05, "nickel");
coins.add(1, aNickel);
//quarter ora è il terzo oggetto della lista
```

Aggiungere un elemento alla i-esima posizione fa slittare gli oggetti seguenti di una posizione



Accedere agli elementi

- Bisogna usare il metodo `get(indice)`

```
coins.get(2);
```

- Un ArrayList gestisce oggetti di tipo Object
  - Possiamo passare qualsiasi oggetto al metodo add

```
ArrayList coins = new ArrayList();
coins.add(new Rectangle(5, 10, 20, 30));
//nessun problema
```

Accedere agli elementi

- Per utilizzare metodi dell'oggetto inserito occorre fare il cast, altrimenti si possono solo usare i metodi di Object

```
Rectangle aCoin = (Rectangle) coins.get(i);
aCoin.translate(x,y);
```

- Il cast ha successo solo se si usa il tipo corretto per l'oggetto considerato

```
Coin aCoin = (Coin) coins.get(i);
//ERRORE
//un Rectangle non può essere convertito in un Coin!
```

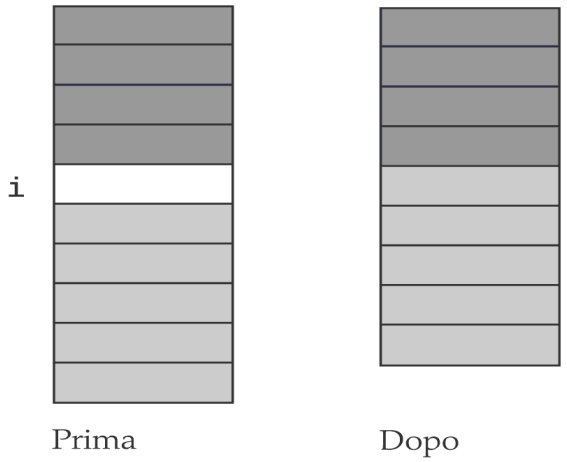
Rimuovere un elemento

- Per rimuovere un elemento da una collezione si usa il metodo `remove(indice)`
  - Restituisce l'oggetto rimosso
  - Gli elementi che seguono slittano di una posizione all'indietro

```
ArrayList coins = new ArrayList ();
coins.add(new Coin(0.1, "dime"));
coins.add(new Coin(0.25, "quarter");
Coin aNickel = new Coin(0.05, "nickel");
coins.add(1, aNickel);

coins.remove(0);
//il vettore ora ha due elementi:
//                    quarter e nickel
```

Eliminare l'elemento alla i-esima posizione
fa slittare gli oggetti seguenti di una posizione



Esaminare in sequenza gli elementi

- Calcolare il valore totale delle monete nel vettore

```
double total = 0;
for (int i = 0; i < coins.size(); i++)
{
       Coin c = (Coin) coins.get(i);
       total = total + c.getValue();
}
```

Modificare un elemento

- Si usa il metodo `set(indice, obj)`
  - Restituisce l'oggetto rimpiazzato

```
ArrayList coins = new ArrayList();
coins.add(new Coin(0.1, "dime"));
coins.add(new Coin(0.25, “quarter”));
Coin aNickel = new Coin(0.05, “nickel”);
coins.set(0, aNickel);

//la posizione 0 viene sovrascritta
```

Nuova classe Purse

```
import java.util.ArrayList;

public class Purse{

   public Purse(){
      coins = new ArrayList();
   }

   public void add(Coin aCoin){
      coins.add(aCoin);
   }

   public double getTotal(){
      double total = 0;
      for (int i = 0; i < coins.size(); i++){
         Coin aCoin = (Coin)coins.get(i);
         total = total + aCoin.getValue();
      }
      return total;
   }

   private ArrayList coins;
}
```

Range degli indici per ArrayList

- Gli indici ammissibili per i metodi che fanno riferimento ad oggetti memorizzati (get, remove, set,..) sono:

  0,1,……, size()-1

- Gli indici ammissibili per i metodi che inseriscono nuove posizioni (add) sono:

  0,1,……, size()

- Se si specifica un indice fuori da questi domini viene generata a run time l'eccezione:

  IndexOutOfBoundsException (java.lang)

Ricerca Lineare

```
public class Purse
{
   public boolean find(Coin aCoin)
   {
      for (int i = 0; i < coins.size(); i++)
      {
         Coin c =(Coin) coins.get(i);
         if (c.equals(aCoin))
              return true; //trovato
       }
       return false; //non trovato
    }
  ...
}
```

Contare elementi di un certo tipo

```
public class Purse
{
   public int count(Coin aCoin)
   {
      int matches = 0;
      for (int i = 0; i < coins.size();
   i++)
      {
         Coin c =(Coin) coins.get(i);
         if (c.equals(aCoin)) matches++;
                                  //found a
   match
      }
      return matches;
    }
      ...
}
```

Trovare il massimo

```
public class Purse
{
  public Coin getMaximum()
  {
     //inizializza il max al primo valore
     Coin max =(Coin) coins.get(0);
     for (int i = 1; i <coins.size(); i++)
     {
        Coin c =(Coin) coins.get(i);
        if (c.getValue() > max.getValue()) max =c;
     }
     return max;
  }
  ...
}
```

Trovare il minimo

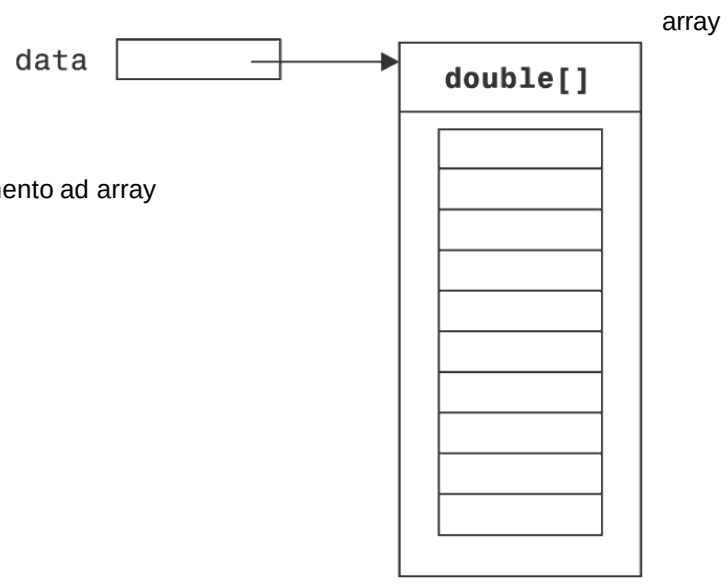
```
public class Purse
{
  public Coin getMinimum()
  {
     //inizializza il min al primo valore
     Coin min =(Coin) coins.get(0);
     for (int i = 1; i <coins.size(); i++)
     {
         Coin c =(Coin) coins.get(i);
         if (c.getValue() < min.getValue()) min =c;
     }
     return min;
  }
  ...
}
```

Memorizzare numeri in vettori

- ArrayList memorizza oggetti
- Per i numeri (dati primitivi) si utilizzano classi wrapper (involucro)
  - `Double d = new Double(123.45);`
  - `double x = d.doubleValue();`
- Aggiungiamo numeri a un vettore:
  - ```
    ArrayList data = new ArrayList();
    Double d = new Double(123.45);
    data.add(d);
     //recuperiamo il numero
    x = ((Double) data.get(i)).doubleValue();
    ```
    ```

Array

- Array = sequenza di lunghezza prefissata di valori dello stesso tipo (classe o tipo primitivo)
  - Ogni posizione è individuata da un indice
  - La prima posizione ha indice 0
  - Deve essere creato con `new`
    - I valori sono inizializzati a 0 (per int o double), false (per boolean) o null (per oggetti)
  - E' un oggetto
  - Accesso attraverso variabili di riferimento



Array vs. ArrayList

- Differenze:
  - Un array ha un campo chiamato `length`, piuttosto che un metodo chiamato `size()`.
  - Array possono contenere anche dati primitivi oltre che oggetti
  - Elementi di un array sono tutti dello stesso tipo e questo tipo deve essere dichiarato.
  - Array non posso crescere in grandezza

Vantaggi e svantaggi

- Vantaggi degli array:
  - Possono memorizzare dati primitivi senza ricorrere a “wrapper”
    - sono più veloci.
  - Permettono di fare il type checking al tempo di compilazione perché occorre dichiarare il tipo

Vantaggi e svantaggi

- Svantaggi degli array:
  - La grandezza dell'array deve essere dichiarata all'inizio e non può cambiare.
  - Non si possono usare i metodi che invece sono disponibili per gli `ArrayList`

Dichiarare un array

- Si scrive il tipo di dati e poi delle parentesi chiuse:
  - `int[ ] unSaccoDiNumeri;`
  - `String[ ] vincitori;`
  - `Employee[ ] personale;`
- Finalmente capiamo che significa
  `public static void main(String[] args)`
  - `args` è un array di stringhe (gli argomenti della linea di comando)

```
Java MyProgram –d file.txt
  args[0] = "-d"
  args[1]= "file.txt"
```

Creare un'istanza di un array

- Per creare un istanza di un array si usa new seguito dal tipo e quindi dalla grandezza in parentesi quadre:

```
int[] unSaccoDiNumeri;
unSaccoDiNumeri = new int[10000];
//un array di 10000 int
```

Usare gli array

- Bisogna indicare l'indice tra parentesi quadre

```
int[] unSaccoDiNumeri;
unSaccoDiNumeri = new int[10000];
for (int i = 0;i < unSaccoDiNumeri.length;i++) {
        unSaccoDiNumeri[i] = i;
   }

System.out.println(unSaccoDiNumeri[0]);
```

- Range degli indici di a: 0,1,……,a.length-1
- Se si usa un indice fuori dal range, viene sollevata a run time l'eccezione:

  ArrayIndexOutOfBoundsException (java.lang)

Esempio:

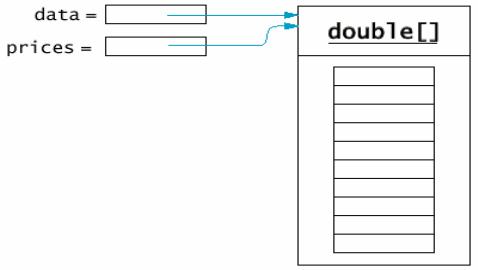
- Stampiamo gli argomenti della linea di comando

```
class PrintArgs {
   public static void main(String[] args) {
     for (int i = 0; i < args.length; i++) {
       System.out.println (args[i]);
     }
   }
 }
```

Copiare Array

- Una variabile array memorizza un riferimento all'array
- Copiando la variabile otteniamo un secondo riferimento allo stesso array

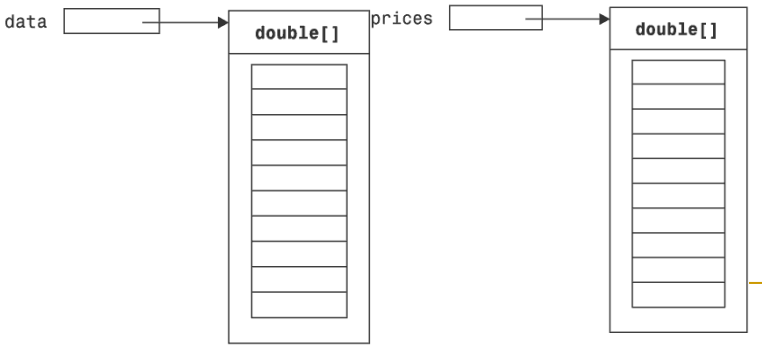
```
double[] data = new double[10];
  // riempi array . . .
  double[] prices = data;
```



Copiare Array

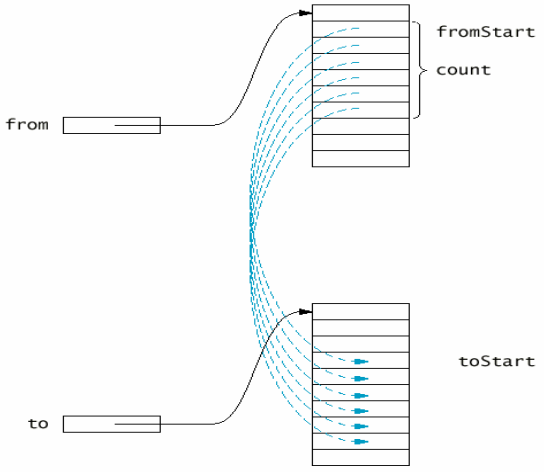
Per fare una vera copia deve creare un nuovo array e copiare tutti gli elementi

```
double[] prices = new double[data.length];
for (int i=0; i < data.length; i++)
    prices[i] = data[i];
```



Copiare elementi da un array all'altro

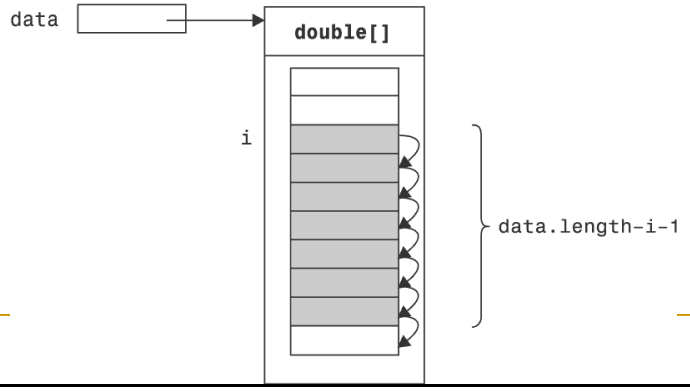
```
System.arraycopy(from, fromStart, to, toStart, count);
```



Uso di System.arraycopy

- Posso aggiungere un elemento in posizione i
  - Sposto di una posizione in avanti tutti gli elementi a partire da i

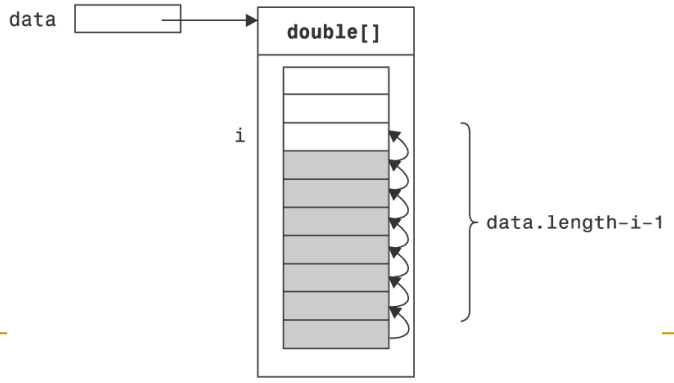
```
System.arraycopy(data, i, data, i+1, data.length-i-1);
data[i]=x;
```



Uso di System.arraycopy

- Posso eliminare un elemento in posizione i
  - Sposto di una posizione in indietro tutti gli elementi a partire da i+1

```
System.arraycopy(data, i+1, data, i, data.length-i-1);
```



Array riempiti solo in parte

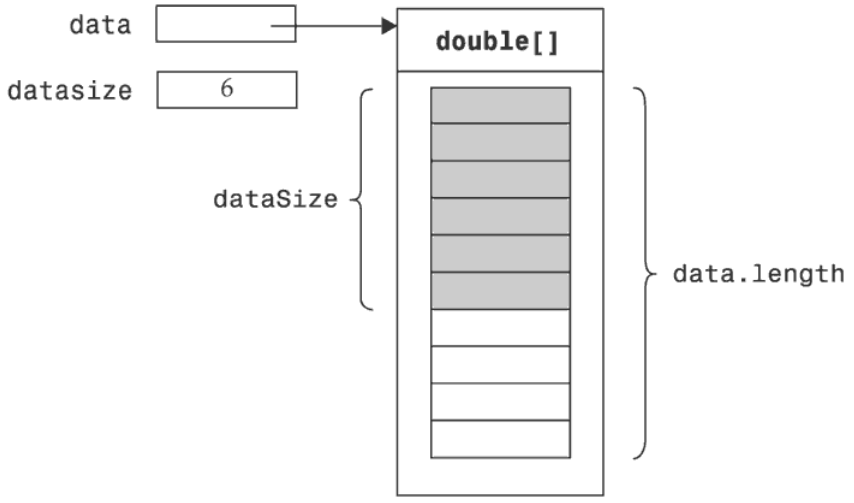
- Il max numero di elementi nell'array è prefissato
- Se non riempiamo tutto l'array dobbiamo tenere traccia del numero di elementi

```
final int DATA_LENGTH = 100;
    double[] data = new double[DATA_LENGTH];
    int dataSize = 0; //variabile complementare
//data.length è la capacità dell'array
//dataSize è la dimensione reale
```

- Se inseriamo elementi dobbiamo incrementare la dimensione

```
Data[dataSize] = x;
dataSize++;
```

Array riempiti solo in parte



Array riempiti solo in parte

- In un ciclo, fermarsi a `dataSize` e non a `data.length`

```
for (int i = 0; i < dataSize; i++)
   sum = sum + data[i];
```

- Non riempire l'array oltre i suoi limiti

```
if (dataSize >= data.length)
   System.out.println("Mi dispiace, l'array è pieno");
```

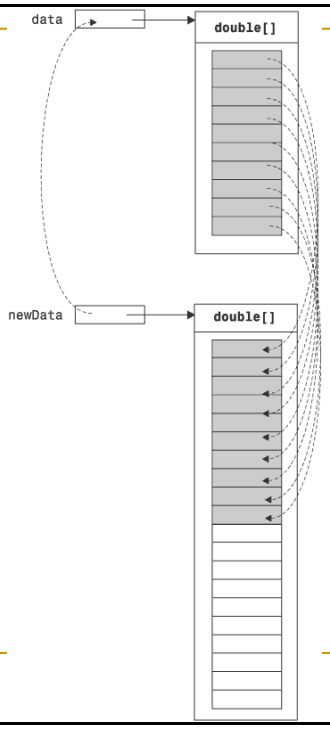
- Oppure creare un nuovo array più grande, copiare gli elementi e assegnare il nuovo array alla variabile vecchia

```
double[] newData = new double[2 * data.length];
System.arraycopy(data, 0, newData, 0, data.length);
data = newData;
```

Esempio

```
class StringArray {
   private String[] stringhe;
   private int stringheSize;

   public StringArray () {
     stringhe = new String[100];
     stringheSize = 0;
   }
   public void addString (String s) {
     stringhe[stringheSize] = s;
     stringheSize++;
   }
   public void print () {
     for (int i = 0; i < stringheSize; i++) {
       System.out.println(stringhe[i]);
     }
   }
}
```



Far crescere un array

Classe ExtendibleTable

```
/** Questa classe mantiene un      */

public class ExtendibleTable{

  public ExtendibleTable(){
    final int DATA_LENGTH = 100;
    data = new double[DATA_LENGTH];
    dataSize = 0;
  }

  public double get(int i) {
    if (i < 0 || i >= dataSize) return -1;
    return data[i];
  }
  public void set(int i, double x) {
    if (i < 0 || i >= dataSize) return -1;
    data[i] = x;
  }
```

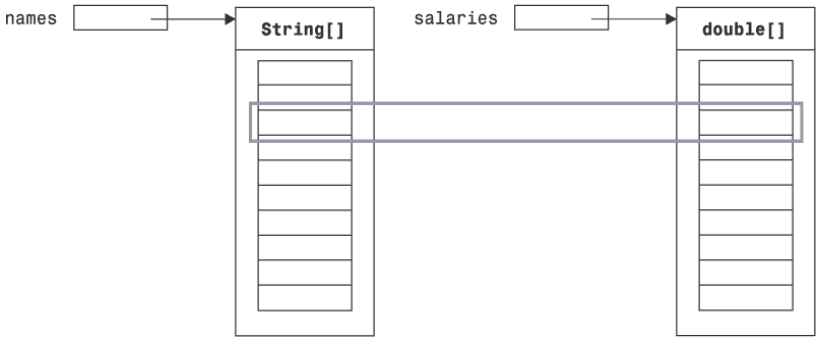
```
public void add(double x){
    if (dataSize >= data.length){
  // crea un nuovo array grande il doppio
      double[ ] newD = new double[2 * data.length];
  // copia tutti gli elementi da data a newData
      System.arrayCopy(data, 0, newD, 0,
                                    data.length);
  // abbandona il vecchio array e inserisci in data
  // un riferimento al nuovo array
      data = newD;
    }
    data[dataSize] = x;
    dataSize++;
  }

  private double[ ] data;
  private int dataSize;
}
```

Array paralleli

- Non utilizzate array paralleli

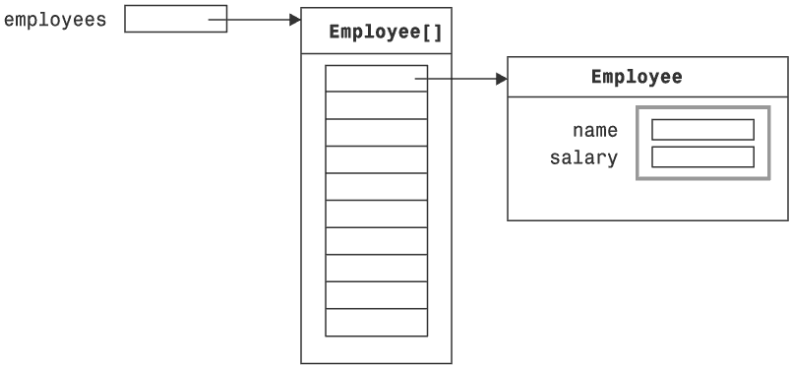
```
String[] names;
double[] salaries;
```



Array di oggetti

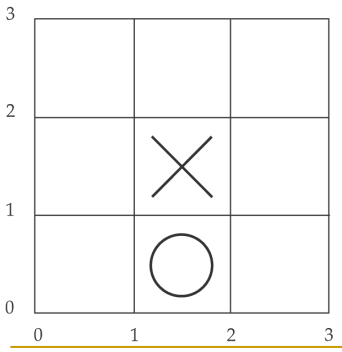
- Riorganizzate i dati in array di oggetti

```
Employee[] staff;
```



Array a due dimensioni

- Tabella con righe e colonne
- Esempio: la scacchiera del gioco Tris




```
char[][] board = new char[3][3];
//array di 3 righe e 3 colonne

board[i][j] = 'x';
// accedi all'elemento della riga
//     i e colonna j
```

Classe Tris

```
/**
   Una scacchiera 3x3 per il gioco Tris.
*/
public class Tris{
   /**
      Costruisce una scacchiera vuota.
   */
   public Tris(){
      board = new char[ROWS][COLUMNS];
      // riempi di spazi
      for (int i = 0; i < ROWS; i++)
         for (int j = 0; j < COLUMNS; j++)
            board[i][j] = ' ';
   }
```

```
/**
      Crea una rappresentazione della scacchiera
      in una stringa, come ad esempio
      |x   o|
      |  x  |
      |    o|
      @return la stringa rappresentativa
*/
   public String toString()
   {
      String r = "";
      for (int i = 0; i < ROWS; i++)
      {
         r = r + "|";
         for (int j = 0; j < COLUMNS; j++)
            r = r + board[i][j];
         r = r + "|\n";
      }
      return r;
   }
```

```
     /**
        Imposta un settore della scacchiera.
        Il settore deve essere libero.
        @param i l'indice di riga
        @param j l'indice di colonna
        @param player il giocatore ('x' o 'o')
     */
     public void set(int i, int j, char player)
     {
        if (board[i][j] != ' ')
           throw new IllegalArgumentException(
                 "Position occupied");
        board[i][j] = player;
     }

   private char[][] board;
   private static final int ROWS = 3;
   private static final int COLUMNS = 3;
  }
```

File Test.java

```
import java.io.*;

/**
   Questo programma collauda la classe Tris
   chiedendo all'utente di selezionare posizioni sulla
   scacchiera e visualizzando il risultato.
*/
public class Test
{
   public static void main(String[] args)
   {
      char player = 'x';
      Tris game = new Tris();
      InputStreamReader r = new ImputStreamReader(System.in);
      BufferedReader b = new BufferedReader(r);
```

```
while (true) {
   System.out.println(game);//invoca game.toString()

   System.out.println("Inserisci riga per " + player);
   String input = b.readLine();

   if (input.equals("")) return;

   System.out.println("Inserisci colonna per " + player);
   input = b.readLine();

   int row = Integer.parseInt(input);
   int column = Integer.parseInt(input);

   game.set(row, column, player);
   if (player == 'x') player = 'o';
   else player = 'x';
}
}
}
```