```java
package classePunto;
// import java.math.*;
// libreria messa sotto commentato perché per la
// radice quadrata non era necessario dichiararla
import java.util.*;

public class ClassePunto {

    public static class Punto {
        // Dichiarazione degli Attributi
        private char nome;
        private double x, y;
        /**
         *  Gli attributi devono essere private per proteggerli
         *  da manipolazione esterna e vanno gestiti tramite
         *  metodi public (incapsulamento)
         */

        // metodo costruttore
        /**
         * Il costruttore è un metodo particolare che restituisce
```

```
22          * un oggetto del tipo della classe che stiamo definendo,
23          * per questo non si indica il tipo restituito.
24          * Può prendere come input dei parametri che vanno a
25          * caratterizzare l'ggetto. Nel nostro caso:
26          * @param nome
27          * @param x
28          * @param y
29          * Nel nostro esempio riempiremo gli attributi dell'oggetto
30          * in creazione con i valori passati.
31          * Per richiamare il costruttore fuori dalla definizione
  della
32          * classe si usa il comando new Classe(), per richiamarlo
33          * internamente alla definizione della classe usiamo this().
34          */
35         public Punto(char nome, double x, double y) {
36             super();
37             /**
38              * Non è necessario richiamare il costruttore di object,
39              * da cui stiamo ereditando implicitamente, ma il
  costruttore
40              * del genitore, nel caso, si richiama con il comando
```

```
   "super()"
41                 */
42                setNome(nome);
43                setX(x);
44                setY(y);
45                /**
46                 * Internamente alla classe potrei manipolare gli
   attributi ma
47                 * posso anche usare i metodi definiti (in questo caso i
   setter
48                 * dei miei attributi privati
49                 */
50            }
51
52            public Punto (char nome) {
53                /**
54                 * setNome(nome);
55                 * setX(0);
56                 * setY(0);
57                 */
58                this(nome, 0, 0);
```

```
59              /**
60               * Tramite this(), a cui passiamo gli opportuni
  parametri,
61               * possiamo chiamare i costruttori della nostra classe.
62               * In questo caso va a semplificare le azioni nel primo
63               * commento perché, di fatto, sono le stesse istruzioni
  del
64               * primo costruttore ma con x ed y a 0.
65               */
66          }
67
68          // altri metodi
69
70          /**
71           * Il metodo muovi è pubblico, prende in input le nuove
  coordinate
72           * @param x
73           * @param y
74           * @return distanza dal punto precedente corrispondente
  all'ipotenusa
75           * del triangolo ideale formato da questa e dalla differenza
```

```
   tra le
 76          * coordinate x ed y dei due punti
 77          */
 78
 79         public double muovi (double x, double y) {
 80             // sposta il punto e restituisce la distanza di cui si è
   mosso
 81             double cateto1 = this.x - x;
 82             double cateto2 = this.y - y;
 83             this.x = x;
 84             this.y = y;
 85             return Math.sqrt((cateto1*cateto1)+(cateto2*cateto2));
 86         }
 87
 88         /**
 89          * Muove un punto in una direzione orizzontale (x) o
   verticale (y)
 90          * @param coord -  Quale coordinata voglio variare
 91          * @param val - di quanto voglio variare la coordinata
 92          * @return la nuova coordinata
 93          */
```

```
 94
 95          public double muovi (char coord, double val) {
 96              if (coord == 'x') {
 97                  x = x + val;
 98                  return x;
 99              } else if (coord == 'y') {
100                  y = y + val;
101                  return y;
102              } else {
103                  return 0.0;
104              }
105          }
106
107          // Metodi getter e setter
108
109          /**
110           * Questi metodi servono alla lettura o scrittura
111           * degli attributi private.
112           * In questo caso x ed y accettano tutti i numeri reali
113           * (double) quindi non serve un controllo ma i setter,
114           * le funzioni di scrittura, possono fare un controllo
```

```
115          * preventivo alla modifica per verificare se il valore
116          * che si vuole scrivere è valido (Es. Sesso, in una classe
117          * persona è 'm' o 'f' non esiste un sesso 'z').
118          */
119
120         public char getNome() {
121             return nome;
122         }
123
124         public void setNome(char nome) {
125             this.nome = nome;
126         }
127
128         public double getX() {
129             return x;
130         }
131
132         public void setX(double x) {
133             this.x = x;
134         }
135
```

```
136         public double getY() {
137             return y;
138         }
139 
140         public void setY(double y) {
141             this.y = y;
142         }
143     }
144 
145     public static void main(String[] args) {
146         /**
147          * Il main è il metodo da cui parte un programma.
148          * Tutto ciò che avviene parte da qui, poi dal main
149          * si possono richiamare le varie parti del codice
150          * da utilizzare tramite gli oggetti ed i loro metodi.
151          */
152 
153         //dichiarazione delle variabili
154 
155         Double x0, y0, x1, y1;
156         // punto di partenza (x0, y0) e di destinazione (x1, y1)
```

```
157         Character nome; // nome del punto
158         Scanner sc; // Scanner per l'I/O
159         Punto p; // variabile p per un oggetto di tipo Punto
160 
161         sc = new Scanner(System.in); //Creo lo scanner
162 
163         System.out.println("Inserire il nome del punto:");
164         nome = Character.valueOf(sc.next().charAt(0));
165         // sc.next() legge una Stringa, noi preleviamo il
166         // primo carattere, alla posizione 0
167 
168         System.out.println("Inserire il valore della coordinata x di
    partenza:");
169         x0 = Double.valueOf(sc.nextDouble());
170         /**
171          * Lo scanner non ha una funzione specifica per leggere un
    singolo
172          * carattere quindi leggiamo una stringa, una sequenza di
    caratteri,
173          * e ne prendiamo il primo: charAt(0) -> Carattere alla
    posizione 0
```

```
174          */
175
176         System.out.println("Inserire il valore della coordinata y di
    partenza:");
177         y0 = Double.valueOf(sc.nextDouble());
178
179         System.out.println("Inserire il valore della coordinata x di
    destinazione:");
180         x1 = Double.valueOf(sc.nextDouble());
181
182         System.out.println("Inserire il valore della coordinata y di
    destinazione:");
183         y1 = Double.valueOf(sc.nextDouble());
184
185         /**
186          * Vediamo qui sopra l'uso di oggetti Wrapper di tipo
    Character, per char, e Double
187          * per double. I Wrapper incapsulano i tipi primitivi ed
    offrono altre funzionalità.
188          * Nel nostro specifico caso sarebbe stato più comodo usare
    i tipi primitivi ma
```

```
189          * ne abbiamo approfittato per vedere la creazione e la
    lettura dei valori dei Wrapper.
190          */
191
192         sc.close(); // Chiudo lo scanner per non sprecare risorse
193
194         p = new Punto(nome.charValue(), x0.doubleValue(),
    y0.doubleValue());
195         // New richiama il metodo costruttore e crea un oggetto
    dalla classe
196
197         System.out.println("La distanza percorsa dal punto " +
    p.getNome() + " è pari a: " + p.muovi(x1.doubleValue(),
    y1.doubleValue()));
198     }
199
200}
201
202/**
203 * Quando lanciamo il programma i valori decimali vanno inseriti con
    ','
```

```
204  * perché il programma è "localizzato" in Italia (lo recupera da Eclipse o
205  * dal Sistema Operativo) mentre nel codice dobbiamo usare il '.' perché
206  * è standard e non "localizzato" ed è più usato a livello internazionale
207  */
208
```