Capitolo 9
Interfacce e polimorfismo

Cay S. Horstmann

Concetti di informatica e fondamenti di Java
quarta edizione

Apogeo ©2007

Obiettivi del capitolo

- Conoscere le interfacce
- Saper effettuare conversioni tra riferimenti a classi e a interfacce
- Capire il concetto di polimorfismo
- Utilizzare le interfacce per ridurre l'accoppiamento tra classi

Apogeo ©2007

Utilizzo di interfacce per il riutilizzo del codice

- I tipi interfaccia vengono utilizzati per rendere il codice maggiormente riutilizzabile.
- Cominciamo da una classe `DataSet` per calcolare il valore medio e il valore massimo di un insieme di valori di ingresso
- Se volessimo esaminare conti bancari per trovare il conto con il saldo più elevato, dovremmo modificare la classe

Segue...

Apogeo ©2007

File `DataSet.java`

```
01: /**
02:    Calcola informazioni relative a un insieme di dati.
03: */
04: public class DataSet
05: {
06:    /**
07:       Costruisce un insieme di dati vuoto.
08:    */
09:    public DataSet()
10:    {
11:       sum = 0;
12:       count = 0;
13:       maximum = 0;
14:    }
15:
16:    /**
17:       Aggiunge un valore all'insieme dei dati
18:       @param x un valore
19:    */
```

Segue

Apogeo ©2007

File `DataSet.java`

```
20:    public void add(double x)
21:    {
22:       sum = sum + x;
23:       if (count == 0 || maximum < x) maximum = x;
24:       count++;
25:    }
26:
27:    /**
28:       Restituisce la media dei valori inseriti.
29:       @return la media, o 0 se non ci sono dati
30:    */
31:    public double getAverage()
32:    {
33:       if (count == 0) return 0;
34:       else return sum / count;
35:    }
36:
```

Segue

Apogeo ©2007

File `DataSet.java`

```
37:    /**
38:       Restituisce il valore massimo tra i valori inseriti.
39:       @return il massimo, o 0 se non ci sono dati
40:    */
41:    public double getMaximum()
42:    {
43:       return maximum;
44:    }
45:
46:    private double sum;
47:    private double maximum;
48:    private int count;
49: }
```


Apogeo ©2007

Utilizzo di interfacce per il riutilizzo del codice

```
public class DataSet // modificata per oggetti di tipo BankAccount
{
   . . .
   public void add(BankAccount x)
   {
      sum = sum + x.getBalance();
      if (count == 0 || maximum.getBalance() < x.getBalance())
         maximum = x;
      count++;
   }

   public BankAccount getMaximum()
   {
      return maximum;
   }
   private double sum;
   private BankAccount maximum;
   private int count;
}
```


Apogeo ©2007

Utilizzo di interfacce per il riutilizzo del codice

- Supponete, ora, che volessimo trovare la moneta con il valore più elevato in un insieme di monete: dovremmo modificare nuovamente la classe DataSet.

Segue...

Apogeo ©2007

Utilizzo di interfacce per il riutilizzo del codice

```
public class DataSet // modificata per oggetti di tipo Coin
{
   . . .
   public void add(Coin x)
   {
      sum = sum + x.getValue();
      if (count == 0 || maximum.getValue() < x.getValue())
         maximum = x;
      count++;
   }

   public Coin getMaximum()
   {
      return maximum;
   }
   private double sum;
   private Coin maximum;
   private int count;
}
```


Apogeo ©2007

Utilizzo di interfacce per il riutilizzo del codice

- Il meccanismo fondamentale per l'analisi dei dati è, evidentemente, lo stesso in tutti i casi, ma i dettagli del confronto cambiano.
- Supponete che le diverse classi potessero accordarsi su un unico metodo `getMeasure` che fornisca la misura da usare nell'analisi dei dati
- Potremmo realizzare un'unica classe `DataSet`, riutilizzabile, il cui metodo `add` assomiglierebbe a questo:

```
sum = sum + x.getMeasure();
if (count == 0 || maximum.getMeasure() < x.getMeasure())
   maximum = x;
count++;
```


Apogeo ©2007

Utilizzo di interfacce per il riutilizzo del codice

- Qual è il tipo della variabile `x`? Idealmente, `x` dovrebbe riferirsi a qualsiasi classe che abbia un metodo `getMeasure`
- Per esprimere il concetto di una funzionalità necessaria per una classe, in Java si usa un'*interfaccia*.

```
public interface Measurable
{
   double getMeasure();
}
```

- In Java, un'interfaccia dichiara un insieme di metodi e le loro firme.

Apogeo ©2007

Interfacce e classi

Un'interfaccia è simile a una classe, ma ci sono parecchie differenze importanti:

- Tutti i metodi di un'interfaccia sono *astratti*, cioè hanno un nome, un elenco di parametri, un tipo di valore restituito, ma non hanno un'implementazione.
- Tutti i metodi di un'interfaccia sono automaticamente pubblici.
- Un'interfaccia non ha variabili di esemplare.

Apogeo ©2007

Usare il tipo `Measurable` per dichiarare le variabili `x` e `maximum`

```
public class DataSet
{
   . . .
   public void add(Measurable x)
   {
      sum = sum + x.getMeasure();
      if (count == 0 || maximum.getMeasure() < x.getMeasure())
         maximum = x;
      count++;
   }

   public Measurable getMaximum()
   {
      return maximum;
   }

   private double sum;
   private Measurable maximum;
   private int count;
}
```


Apogeo ©2007

Implementazione di interfaccia

- Per indicare che una classe realizza un'interfaccia si usa la parola chiave `implements`

```
public class BankAccount implements Measurable
{
   public double getMeasure()
   {
      return balance;
   }
   // altri metodi e campi
}
```

- Una classe può realizzare più di una interfaccia
  - la classe deve definire tutti i metodi richiesti da tutte le interfacce che realizza.

Segue..

Apogeo ©2007

Implementazione di interfaccia

- Un altro esempio:

```
public class Coin implements Measurable
{
   public double getMeasure()
   {
      return value;
   }
   . . .
}
```


Apogeo ©2007

Schema UML della classe `DataSet` e delle classi che realizzano l'interfaccia `Measurable`

- Le interfacce possono ridurre l'accoppiamento tra le classi.
- Nella notazioneUML:

1. le interfacce vengono contrassegnate dall'indicazione di stereotipo «`interface`»
2. Una freccia tratteggiata con punta triangolare segnala la relazione di tipo "è un" che esiste tra un'interfaccia e una classe che la realizza.
3. Occorre fare molta attenzione alla punta delle frecce: una linea tratteggiata con la freccia a V aperta indica, invece, una dipendenza (relazione "usa")

- La classe `DataSet` dipende solamente dall'interfaccia `Measurable` e non è accoppiata alle classi `BankAccount` e `Coin`.

Segue..

Schema UML della classe DataSet e delle classi che realizzano l'interfaccia Measurable

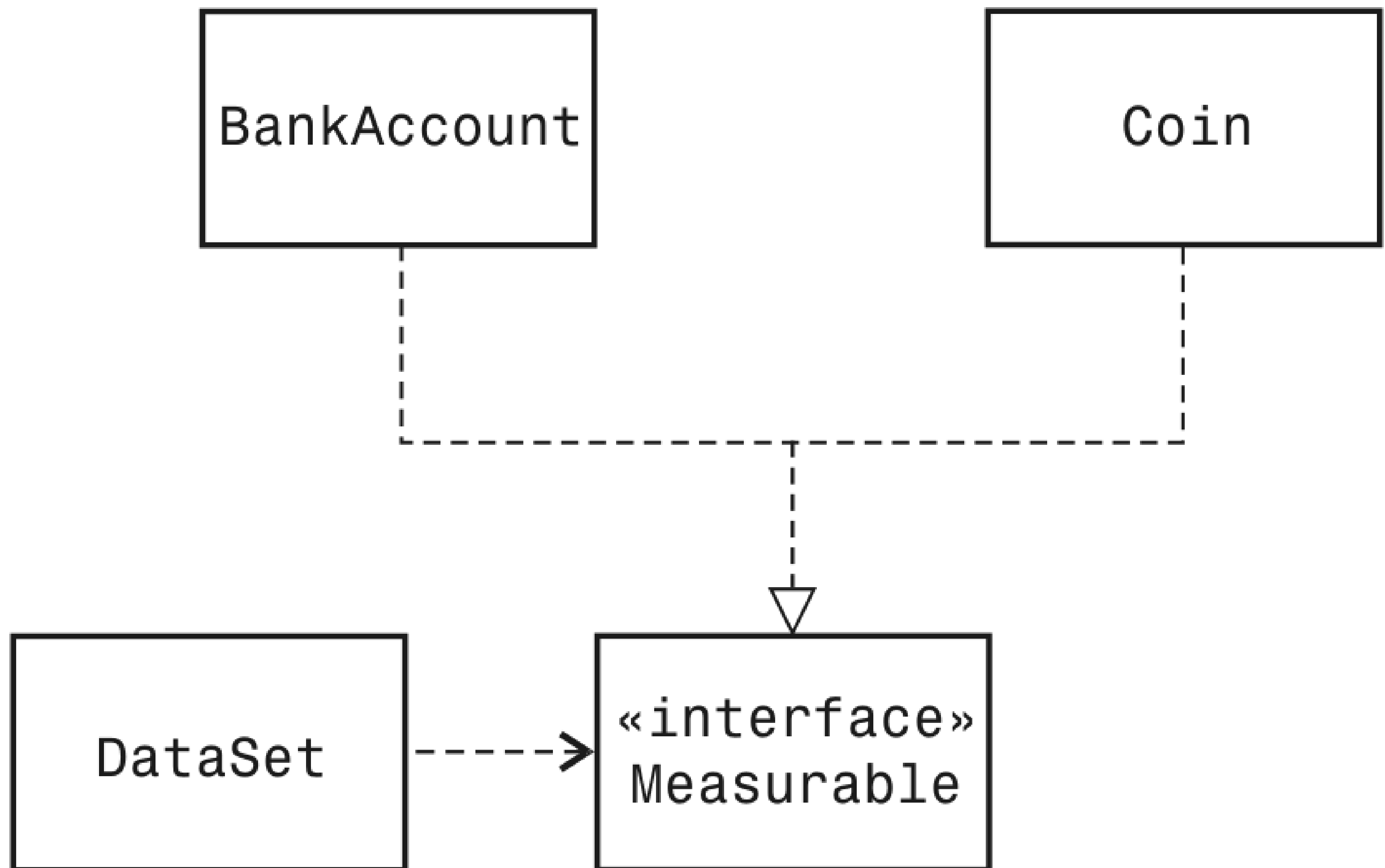

Figura 1

Apogeo ©2007

Sintassi di Java 9.1
Definizione di interfaccia

```
public interface NomeInterfaccia
{
    // firme dei metodi
}
```

Esempio:

```
public interface Measurable
{
    double getMeasure();
}
```

Obiettivo:

Definire un'interfaccia e le firme dei suoi metodi, che sono automaticamente pubblici.

Apogeo ©2007

Sintassi di Java 9.2
Implementazione di interfaccia

```
public class Nomeclasse
    implements NomeInterfaccia,NomeInterfaccia, ...
{
        // metodi
        // variabili di esemplare
}
```

Esempio:

```
public class BankAccount implements Measurable
{
   // altri metodi di BankAccount
   public double getMeasure()
   {
      // realizzazione del metodo
   }
}
```

Obiettivo:

Definire una classe che realizzi i metodi di un' interfaccia.

Apogeo ©2007

File DataSetTester.java

```
01: /**
02:    Questo programma collauda la classe DataSet.
03: */
04: public class DataSetTester
05: {
06:    public static void main(String[] args)
07:    {
08:       DataSet bankData = new DataSet();
09:
10:       bankData.add(new BankAccount(0));
11:       bankData.add(new BankAccount(10000));
12:       bankData.add(new BankAccount(2000));
13:
14:       System.out.println("Average balance: "
15:             + bankData.getAverage());
16:       System.out.println("Expected: 4000");
17:       Measurable max = bankData.getMaximum();
18:       System.out.println("Highest balance: "
19:          + max.getMeasure());
20:       System.out.println("Expected: 10000");
21:
```

Segue…

Apogeo ©2007

File DataSetTester.java

```
22:         DataSet coinData = new DataSet();
23:
24:         coinData.add(new Coin(0.25, "quarter"));
25:         coinData.add(new Coin(0.1, "dime"));
26:         coinData.add(new Coin(0.05, "nickel"));
27:
28:         System.out.println("Average coin value: "
29:               + coinData.getAverage());
30:         System.out.println("Expected: 0.133");
31:         max = coinData.getMaximum();
32:         System.out.println("Highest coin value: "
33:               + max.getMeasure());
34:         System.out.println("Expected: 0.25");
35:      }
36: }
```

Segue…

File DataSetTester.java

Visualizza:

```
Average balance: 4000.0
Expected: 4000
Highest balance: 10000.0
Expected: 10000
Average coin value: 0.13333333333333333
Expected: 0.133
Highest coin value: 0.25
Expected: 0.25
```


Apogeo ©2007

Conversione di tipo fra classi e interfacce

- Potete effettuare conversioni dal tipo di una classe al tipo di un'interfaccia che sia realizzata dalla classe.

```
BankAccount account = new BankAccount(10000);
Measurable x = account; // va bene
```

```
Coin dime = new Coin(0.1, "dime");
Measurable x = dime; // anche questo va bene
```

- Non è però possibile fare conversioni fra tipi non correlati

```
Measurable x = new Rectangle(5, 10, 20, 30); // ERRORE
```

Perché la classe `Rectangle` non realizza l'interfaccia `Measurable`.

Apogeo ©2007

Forzature (cast)

- Il metodo `getMaximum` della classe `DataSet` memorizza l'oggetto con la dimensione maggiore come riferimento di tipo `Measurable`

```
DataSet coinData = new DataSet();
coinData.add(new Coin(0.25, "quarter"));
coinData.add(new Coin(0.1, "dime"));
coinData.add(new Coin(0.05, "nickel"));
Measurable max = coinData.getMaximum(); // la moneta di maggior valore
```

- Ora, cosa potete fare con il riferimento `max`?

```
String name = max.getName(); // ERRORE
```

Segue…

Forzature (cast)

- Per convertire un riferimento a interfaccia in un riferimento a classe serve un *cast*.
- Voi sapete che si riferisce a un oggetto di tipo `Coin`, ma il compilatore non lo sa. Potete usare la notazione di *forzatura* (*cast*)

```
Coin maxCoin = (Coin) max;
String name = maxCoin.getName();
```

- Se vi siete sbagliati e l'oggetto in realtà non è una moneta, il vostro programma lancerà un'eccezione e terminerà.

Segue…

Apogeo ©2007

Forzature (cast)

- Differenze fra tipi numerici e tipi di classe:
  - Quando convertite tipi numerici, c'è una potenziale *perdita di informazioni*, e usate il cast per dire al compilatore che ne siete al corrente.
  - quando convertite tipi di oggetto, *affrontate il rischio* di provocare il lancio di un'eccezione, e dite al compilatore che siete disposti a correre questo rischio.

Apogeo ©2007

Polimorfismo

- Quando più classi realizzano la medesima interfaccia, ciascuna classe realizza i metodi propri dell'interfaccia in modi diversi.
- E' assolutamente lecito, e in effetti molto comune, usare variabili il cui tipo sia un'interfaccia, come `Measurable x;`

```
x = new BankAccount(10000);
x = new Coin(0.1, "dime");
```

- Occorre, però, sempre ricordare che l'oggetto a cui si riferisce `x` non è di tipo `Measurable`. Il tipo dell'oggetto sarà sempre quello di una classe che realizza l'interfaccia `Measurable`.

Segue…

Polimorfismo

- Cosa potete fare con una variabile di tipo interfaccia

```
double m = x.getMeasure();
```

se non conoscete la classe dell'oggetto a cui si riferisce? Potete invocare i metodi dell'interfaccia

Apogeo ©2007

Polimorfismo

- Il principio secondo cui il tipo effettivo di un oggetto determina il metodo da chiamare è detto *polimorfismo.*
- Se `x` si riferisce a un oggetto di tipo `BankAccount`, allora viene invocato il metodo `BankAccount.getMeasure`
- Se `x` si riferisce a un oggetto di tipo `Coin`, viene invocato il metodo `Coin.getMeasure`.
- Il polimorfismo è un principio secondo cui il comportamento di un programma può variare in relazione al tipo effettivo di un oggetto.

Segue…

Apogeo ©2007

Polimorfismo

- *La selezione posticipata (late binding*) si ha quando la scelta del metodo avviene al momento dell'esecuzione del programma.
- Esiste una differenza importante fra polimorfismo e sovraccarico. Il compilatore sceglie un metodo sovraccarico quando traduce il programma, prima che il programma venga eseguito.Questa selezione del metodo é detta *selezione anticipata (early binding).*

Apogeo ©2007

Usare interfacce di smistamento

- Consideriamo queste importanti limitazioni dovute all'utilizzo dell'interfaccia `Measurable`:

1. Potete aggiungere l'interfaccia `Measurable` soltanto a classi che sono sotto il vostro controllo.
2. Potete "misurare" un oggetto in un unico modo. Se volete prendere in esame un insieme di conti bancari di risparmio sia in base al saldo che in base al tasso di interesse, siete bloccati.

- I meccanismi di smistamento e di richiamata---che stiamo per vedere---consentono a una classe di richiamare uno specifico metodo quando necessita di maggiori informazioni.

Usare interfacce di smistamento

- Ripensiamo, quindi, alla classe `DataSet`: un insieme di dati deve poter misurare gli oggetti che vi vengono inseriti.
- Alternativa: un oggetto può effettuare le misurazioni

```
public interface Measurer
{
   double measure(Object anObject);
}
```

- Tutti gli oggetti possono essere convertiti nel tipo `Object`, il “minimo comun denominatore” delle classi in Java

Apogeo ©2007

Usare interfacce di smistamento

- La classe `DataSet` migliorata viene costruita fornendo un oggetto di tipo `Measurer`, cioè un oggetto di una classe che realizzi l'interfaccia `Measurer`, memorizzato nella variabile di esemplare `measurer` e utilizzato per eseguire le misurazioni, in questo modo:

```
public void add(Object x)
{
   sum = sum + measurer.measure(x);
   if (count == 0 || measurer.measure(maximum) < measurer.measure(x))
      maximum = x;
   count++;
}
```

Usare interfacce di smistamento

- Ora siete in grado di definire misuratori per qualsiasi tipo di misurazione.

```
public class RectangleMeasurer implements Measurer
{
   public double measure(Object anObject)
   {
      Rectangle aRectangle = (Rectangle) anObject;
      double area = aRectangle.getWidth() * aRectangle.getHeight();
      return area;
   }
}
```

Usare interfacce di smistamento

- Il parametro di tipo `Object` deve essere convertito in un `Rectangle` con un cast:

```
Rectangle aRectangle = (Rectangle) anObject;
```

- Costruite un oggetto di tipo `RectangleMeasurer` e passatelo al costruttore di `DataSet`:

```
Measurer m = new RectangleMeasurer();
DataSet data = new DataSet(m);
data.add(new Rectangle(5, 10, 20, 30));
data.add(new Rectangle(10, 20, 30, 40));
. . .
```


Apogeo ©2007

Diagramma UML delle classi e delle interfacce

- La classe `Rectangle` non è più accoppiata a un'altra classe: per elaborare rettangoli, dovete usare una piccola classe "ausiliaria", `RectangleMeasurer` che esplicita come si misurano gli oggetti

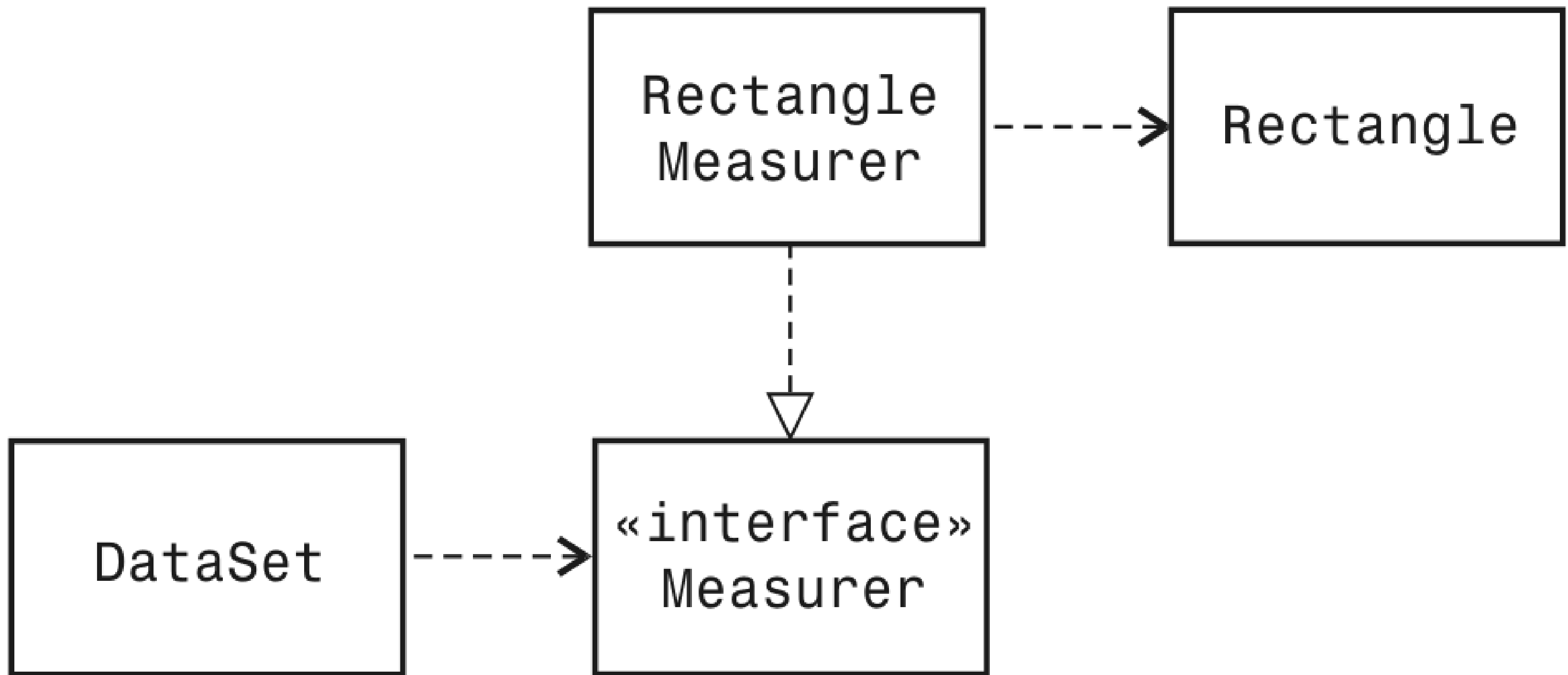

Figura 2:
Schema UML della classe *DataSet* e dell'interfaccia *Measurer*

Apogeo ©2007

File DataSet.java

```
01: /**
02:    Calcola la media di un insieme di valori.
03: */
04: public class DataSet
05: {
06:    /**
07:       Costruisce un insieme vuoto di dati con un misuratore assegnato.
08:       @param aMeasures il misuratore che viene usato
                           per misurare i valori dei dati
09:    */
10:    public DataSet(Measurer aMeasurer)
11:    {
12:       sum = 0;
13:       count = 0;
14:       maximum = null;
15:       measurer = aMeasurer;
16:    }
17:
```

Segue…

File DataSet.java

```
18:     /**
19:        Aggiunge il valore all'insieme dei dati.
20:        @param x il valore da aggiungere
21:     */
22:     public void add(Object x)
23:     {
24:        sum = sum + measurer.measure(x);
25:        if (count == 0
26:              || measurer.measure(maximum) < measurer.measure(x))
27:           maximum = x;
28:        count++;
29:     }
30:
31:     /**
32:        Restituisce la media dei dati inseriti.
33:        @return la media, o 0 se non sono stati inseriti valori
34:     */
```

Segue…

Apogeo ©2007

File DataSet.java

```
35:     public double getAverage()
36:     {
37:        if (count == 0) return 0;
38:        else return sum / count;
39:     }
40:
41:     /**
42:        Restituisce il dato maggiore tra i dati inseriti.
43:        @return il dato maggiore, null se non ci sono valori
44:     */
45:     public Object getMaximum()
46:     {
47:        return maximum;
48:     }
49:
50:     private double sum;
51:     private Object maximum;
52:     private int count;
53:     private Measurer measurer;
54: }
```


Apogeo ©2007

File DataSetTester2.java

```
01: import java.awt.Rectangle;
02: 
03: /**
04:    Questo programma illustra l'utilizzo di un oggetto di tipo Measurer.
05: */
06: public class DataSetTester2
07: {
08:    public static void main(String[] args)
09:    {
10:       Measurer m = new RectangleMeasurer();
11: 
12:       DataSet data = new DataSet(m);
13: 
14:       data.add(new Rectangle(5, 10, 20, 30));
15:       data.add(new Rectangle(10, 20, 30, 40));
16:       data.add(new Rectangle(20, 30, 5, 10));
17: 
```

Segue...

File DataSetTester2.java

```
18:         System.out.println("Average area: " + data.getAverage());
19:         System.out.println("Expected: 625");
20:
21:         Rectangle max = (Rectangle) data.getMaximum();
22:         System.out.println("Maximum area rectangle: " + max);
23:         System.out.println("Expected: java.awt.Rectangle"
                 + "[x=10,y=20,width=30,height=40]");
24:     }
25: }
```


Apogeo ©2007

File Measurer.java

```
01: /**
02:   Descrive una qualsiasi classe i cui esemplari possano
      misurare altri oggetti.
03: */
04: public interface Measurer
05: {
06:    /**
07:       Calcola la misura di un oggetto.
08:       @param anObject lìoggetto da misurare
09:       @return la misura
10:    */
11:    double measure(Object anObject);
12: }
```


Apogeo ©2007

File RectangleMeasurer.java

```
01: import java.awt.Rectangle;
02:
03: /**
04:   Gli oggetti di questa classe misurano rettangoli in base
                                                 alla loro area.
05: */
06: public class RectangleMeasurer implements Measurer
07: {
08:    public double measure(Object anObject)
09:    {
10:       Rectangle aRectangle = (Rectangle) anObject;
11:       double area = aRectangle.getWidth()
                * aRectangle.getHeight();
12:       return area;
13:    }
14: }
15:
```

Segue…

Apogeo ©2007

File RectangleMeasurer.java

Visualizza:

```
Average area: 625
Expected: 625
Maximum area rectangle:java.awt.Rectangle[x=10,y=20,
   width=30,height=40]
Expected: java.awt.Rectangle[x=10,y=20,width=30,height=40]
```


Apogeo ©2007