Programmazione II

A.A. 2022-23

Prof. Maria Tortorella

Interfacce e polimorfismo

- Packages
- Interfacce
- Polimorfismo

Packages

- Package: Un insieme di classi correlate
- Per mettere una classe in un package

```
package packageName;
```

- Esiste un default package senza nome

Packages

- Esempio, una classe `Financial` può essere inserita in un package `com.horstmann.bigjava`
- La linea iniziale di `Financial.java`:

```
package com.horstmann.bigjava;

public class Financial
{
   . . .
}
```

Nota sintattica

package *packageName*;

Esempio:

package com.horstmann.bigjava;

Packages

Package	Purpose	Sample Class
java.lang	Language Support	Math
java.util	Utilities	Random
java.io	Input and Output	PrintStream
Java.awt	Abstract Windowing Toolkit	Color
Java.applet	Applets	Applet
Java.net	Networking	Socket
Java.sql	Database Access	ResultSet
Java.swing	Swing user interface	JButton
Org.omg.COBRA	Common Object Request Broker Architecture	IntHolder

Importare un package

È necessario usare il nome completo di una classe

```
java.util.Scanner in = new java.util.Scanner(System.in);
```

- Fastidioso e soggetto ad errori
- Non è obbligatorio
  - Import consente di usare la classe direttamente

```
import java.util.Scanner;
. . .
Scanner in = new Scanner(System.in)
```

- Oppure, E' possibile importare tutte le classi in un package

```
import java.util.*;
```

- NON è necessario importare `java.lang`

Package e directory

- I package aiutano a prevenire conflitti sui nomi delle classi

```
java.util.Timer vs. javax.swing.Timer
```

- I nome devono essere NON ambigui

- Una prassi per definire il nome di un packace: usare l'indirizzo e-mail di un utente, al contrario
  - reversed domain name

  `edu.sjsu.cs.walters`
  per le classi di Bertha Walters
  (walters@cs.sjsu.edu)

Package e directory

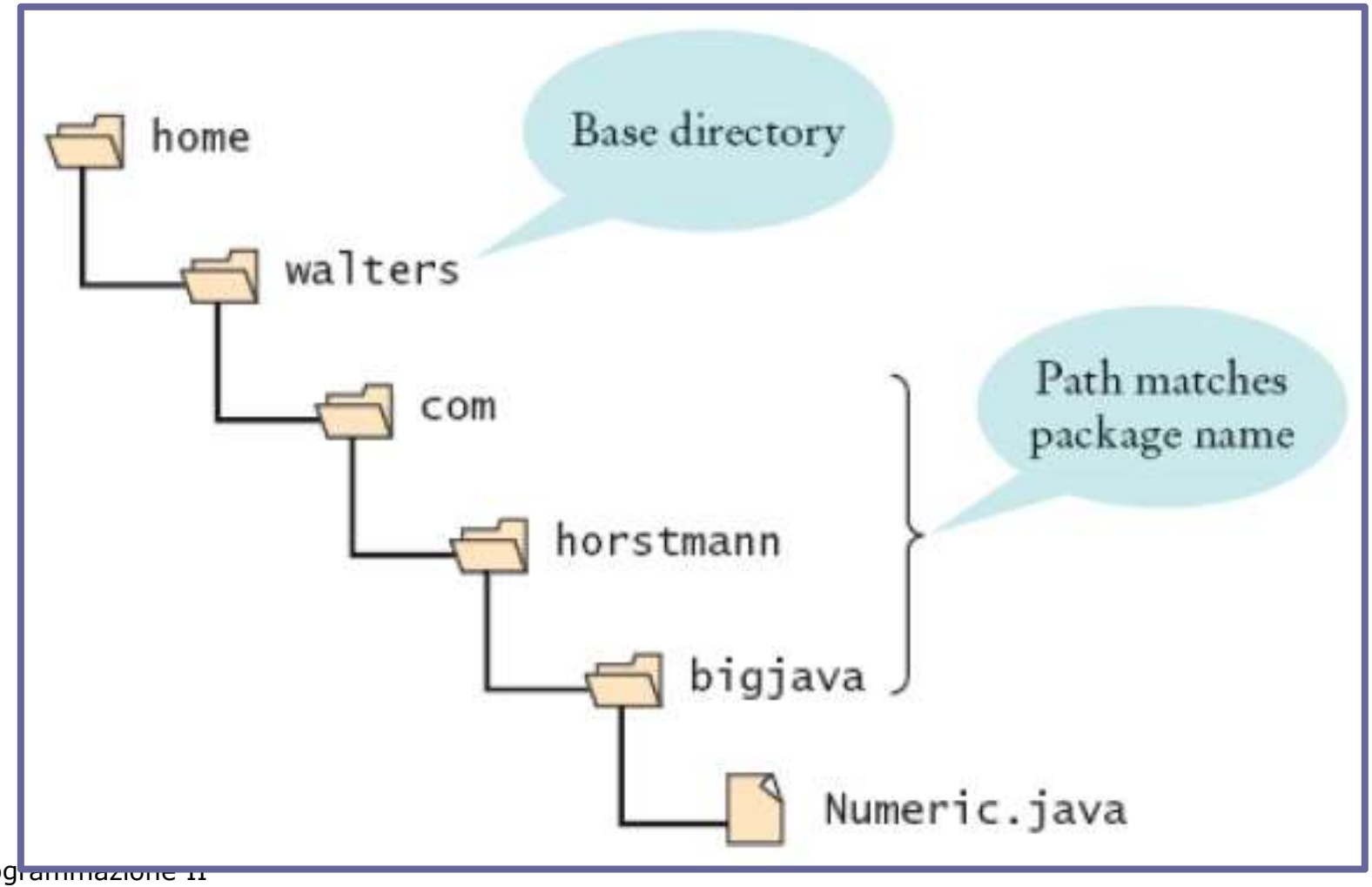
- Path name e package name coincidono

```
com/horstmann/bigjava/Financial.java
```

- Il path name parte dalla classpath
  - Classpath: definisce le directory di base da cui partono le package directories

```
export CLASSPATH=/home/walters;.
set CLASSPATH=c:\home\walters;.
```

Package e directory



Interfacce

- L'utilizzo di *interface types* per migliorare la riusabilità del codice
- Esempio: `DataSet` per la ricerca di media e massimo in un insieme di valori numerici

Interfacce e riuso – la classe DataSet

Problema: ricercare media e massimo in un insieme di double

```
/** Calculate information regarding a set of data */

public class DataSet {
    public DataSet(){
        sum = 0;
        count = 0;
        maximun = 0;
    }

/** add a value to a set of data */

    public void add(double x){
        sum = sum + x;
        if (count == 0 || maximum < x)
            maximum = x;
        count++;
    }
```

Interfacce e riuso - la classe DataSet

```
/** returns the average of the added data */

   public double getAverage(){
      if (count == 0) return 0;
      else return sum / count;
   }

/** returns the maximum value among the added values */

   public double getMaximum(){
      return maximum;
   }

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

File DataSetTester.java

```
01: /**  This program tests the DataSet class. */
02:
03: public class DataSetTester {
04:
05:    public static void main(String[] args){
06:       DataSet data = new DataSet();
07:       data.add(4.5);
08:       data.add(7.9);
09:       data.add(10.3);
10:
11:       System.out.println("Average double numbers = "
12:             + data.getAverage());
13:       double max = data.getMaximum();
14:       System.out.println("Highest double number = "
15:          + max);
16:    }
17:}
```

Output

```
Average double numbers = 7.57
Highest double number = 10.3
```

Interfacce e riuso

- E se volessimo ora ricercare media e massimo del saldo in un insieme di oggetti `BankAccount`?

```
public class DataSet {// Modified for BankAccount objects
public DataSet(){
       sum = 0;
       count = 0;
       maximun = null;
   }
   public void add(BankAccount x){
       sum = sum + x.getBalance();
       if (count==0 || maximum.getBalance()<x.getBalance())
          maximum = x;
       count++;
   }
   public double getAverage(){
       if (count == 0) return 0;
       else return sum / count;
   }
   public BankAccount getMaximum() {
       return maximum;
   }
   private double sum;
   private BankAccount maximum;
   private int count;
}
```

File DataSetTester.java

```
01: /**  This program tests the DataSet class. */
02:
03: public class DataSetTester {
04:
05:    public static void main(String[] args){
06:       DataSet bankData = new DataSet();
07:
08:       bankData.add(new BankAccount(0));
09:       bankData.add(new BankAccount(10000));
10:       bankData.add(new BankAccount(2000));
11:
12:       System.out.println("Average balance = "
13:             + bankData.getAverage());
14:       BankAccount max = bankData.getMaximum();
15:       System.out.println("Highest balance = "
16:          + max.getbalance());
17:    }
18:}
```

Output

```
Average balance = 4000
Highest balance = 10000
```

Interfacce e riuso

- Supponiamo ora di voler ricercare la moneta con il valore massimo in un insieme di monete.
- Ancora una volta sarà necessario modificare `DataSet`

```
public class DataSet {// Modified for Coin objects
public DataSet(){
       sum = 0;
       count = 0;
       maximun = null;
   }
   public void add(Coin x){
       sum = sum + x.getValue();
       if (count == 0 || maximum.getValue() < x.getValue())
          maximum = x;
       count++;
   }
   public double getAverage(){
       if (count == 0) return 0;
       else return sum / count;
   }
   public Coin getMaximum(){
       return maximum;
   }
   private double sum;
   private Coin maximum;
   private int count;
}
```

Interfacce e riuso

- Lo schema di analisi dei dati per la ricerca del massimo è lo stesso: cambia il meccanismo di "misura" dei singoli valori
- È possibile generalizzare
- Bisogna fare in modo che tutte le classi forniscano uno stesso metodo `getMeasure` che effettua la misura da utilizzare sia in fase di analisi che di confronto
- In questo modo, la stessa classe `DataSet` può lavorare con valori di tipi diversi:

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

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

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

Interfacce e riuso

- Quale deve essere il tipo di x?
  `x` dovrebbe far riferimento ad una qualunque istanza di una classe che possiede un metodo `getMeasure`
- In Java, le interfacce consentono di specificare un nuovo tipo, interface type, che dichiara un insieme di operazioni

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

- La dichiarazione di una interfaccia lista tutti i metodi (signatures) richiesti da un particolare interface type

Interfacce vs. Classi

- Un interface type è simile ad una classe, con alcune differenze fondamentali:
  - Tutti i metodi di una interfaccia sono astratti, ossia non è fornita alcuna implementazione
  - Tutti i metodi di una interfaccia sono automaticamente pubblici
  - Non è possibile creare istanze da una interfaccia

dataset generico per "Measureable Objects"

```
public class DataSet {
public DataSet(){
       sum = 0;
       count = 0;
       maximun = null;
   }
   public void add(Measurable x){
       sum = sum + x.getMeasure();
       if (count==0 || maximum.getMeasure()<x.getMeasure())
          maximum = x;
       count++;
   }
   public double getAverage(){
       if (count == 0) return 0;
       else return sum / count;
   }
   public Measurable getMaximum(){
       return maximum;
   }
   private double sum;
   private Measurable maximum;
   private int count;
}
```

Implementare una interfaccia

- Il metodo `getMeasure` deve essere implementato nelle classi che implementano l'interface
- Si utilizza la parola chiave `implements` per indicare che una classe implementa una data interface

```
public class BankAccount implements Measurable {
   public double getMeasure() {
      return balance;
   }
   // tutti i methods and fields della classe BankAccount
}
```

- Ogni classe può implementare anche più di una interface
  - Una classe deve necessariamente implementare tutti i metodi richiesti dalla interface che implementa

Implementare una interfaccia

- Ancora un esempio:

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

- Come un contratto
- Se una classe implementa (`implements`) un'interface, essa si impegna ad implementare tutti i metodi che essa dichiara

Le interfacce definiscono un tipo

- È possibile dichiarare variabili di riferimento con l'interface type: **`Measurable x;`**
- Non è possibile creare oggetti dell'interface type
- È possibile assegnare a variabili dell'interface type oggetti di una classe che la implementa
- Per cui, se BankAccount implementa Measurable, è possibile:

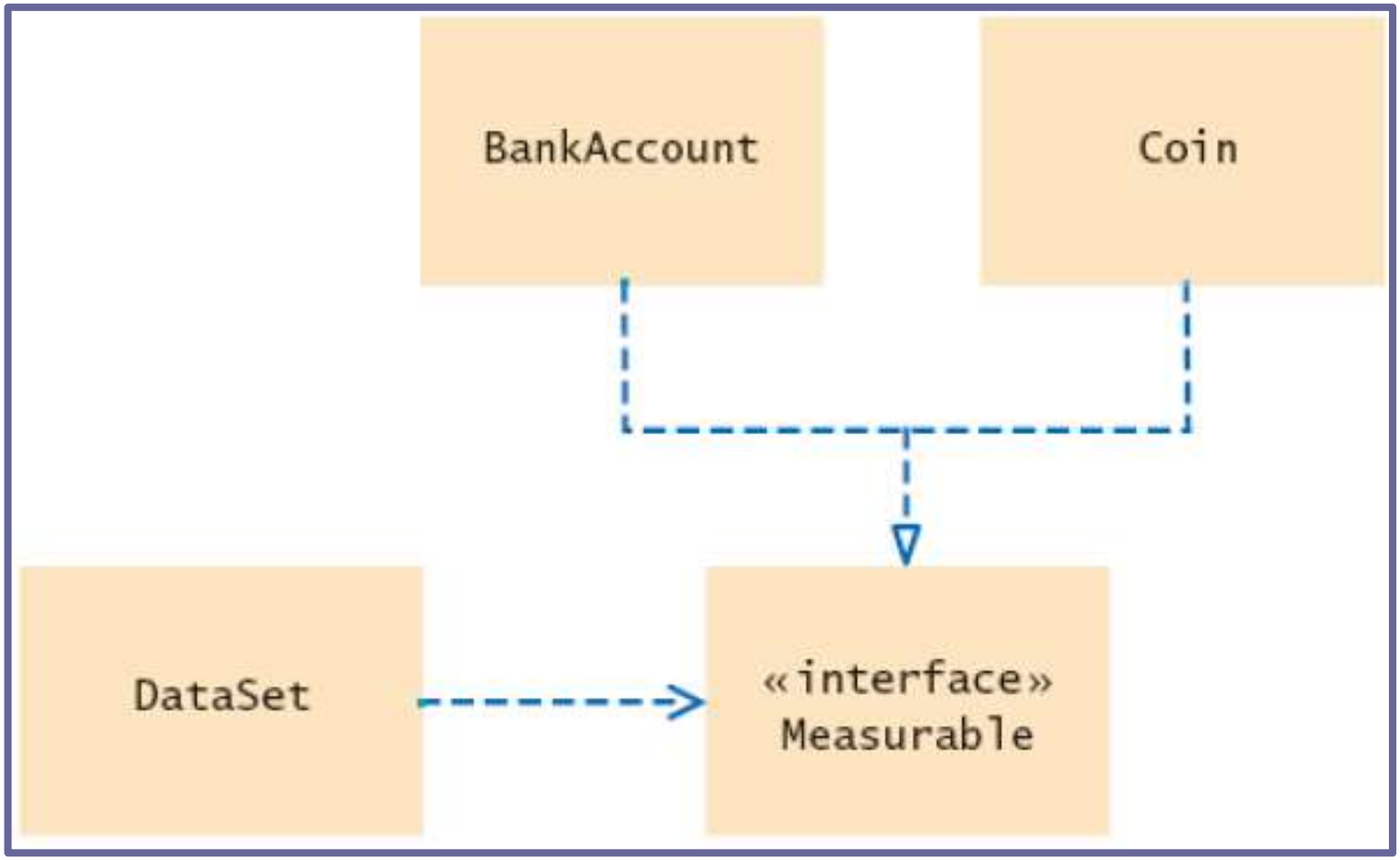
```
Measurable x = new BankAccount(10000);
```

- La variabile x può referenziare istanze delle classi che implementano l'interfaccia

UML Diagram

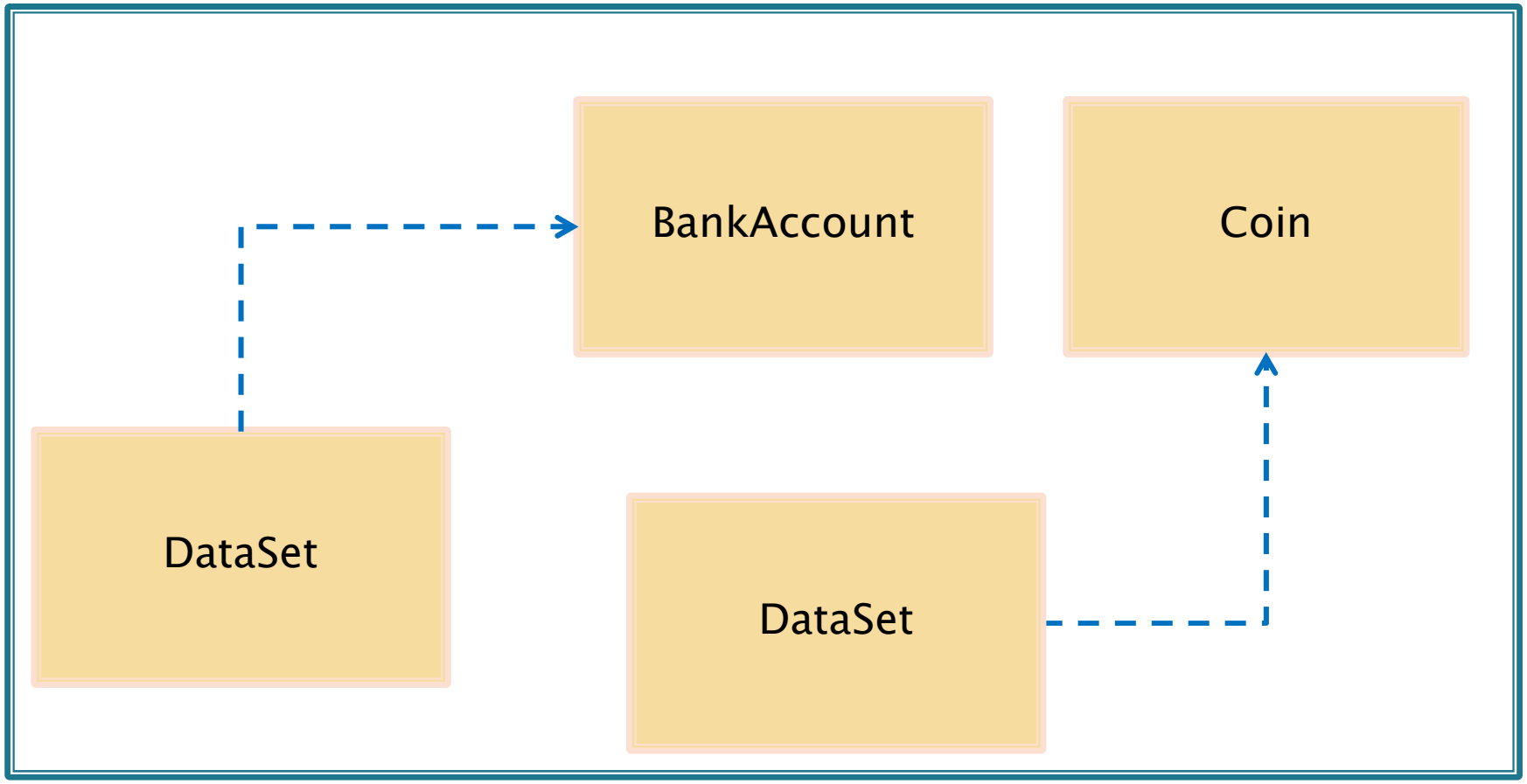
- Le interface aiutano a ridurre il coupling fra le classi
- UML :
  - Le interfacce sono etichettate con lo steriotipo «interface»
  - Un arco tratteggiato con un triangolo denota la relazione "is-a" fra una classe ed una interfaccia
  - Un arco tratteggiato con un arco a V aperta denota la relazione d'uso o di dipendenza

UML Diagram



Nota che DataSet è *decoupled* da BankAccount e Coin

- Nell'implementazione precedente:



Nota sintattica

```
public interface InterfaceName
{
   // method signatures
}
```

Esempio:

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

Nota sintattica

```
public class ClassName
    implements InterfaceName, InterfaceName, ...
{
        // methods
        // instance variables
}
```

Esempio:

```
public class BankAccount implements Measurable
{
   // Other BankAccount methods
   public double getMeasure()
   {
      // Method implementation
   }
}
```

File DataSetTester.java

```
01: /**
02:    This program tests the DataSet class.
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:       Measurable max = bankData.getMaximum();
17:       System.out.println("Highest balance = "
18:          + max.getMeasure());
```

File DataSetTester.java

```
19:
20:         DataSet coinData = new DataSet();
21:
22:         coinData.add(new Coin(0.25, "quarter"));
23:         coinData.add(new Coin(0.1, "dime"));
24:         coinData.add(new Coin(0.05, "nickel"));
25:
26:         System.out.println("Average coin value = "
27:               + coinData.getAverage());
28:         max = coinData.getMaximum();
29:         System.out.println("Highest coin value = "
30:               + max.getMeasure());
31:     }
32: }
```


File DataSetTester.java

Output:

```
Average balance = 4000.0
Highest balance = 10000.0
Average coin value = 0.13333333333333333
Highest coin value = 0.25
```

Conversioni fra classi e interfacce

- Il riferimento di ogni istanza di una classe può essere assegnata ad una variabile dell'interface type corrispondente

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

```
Coin dime = new Coin(0.1, "dime");
Measurable x = dime; // Also OK
```

Conversioni fra classi e interfacce

- Ovviamente la assegnazione non è possibile con classi che non implementano l'interfaccia

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

Casts

- Aggiungere oggetti `Coin` a `DataSet`

```
DataSet coinData = new DataSet();
coinData.add(new Coin(0.25, "quarter"));
coinData.add(new Coin(0.1, "dime"));
. . .
Measurable max = coinData.getMaximum();
                                  // Get the largest coin
```

- Cosa posso fare con l'oggetto max ? Non è un `Coin` e non gli possono essere inviati messaggi con metodo non dichiarati nell'interface type Measurable

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

Casts

- È necessario fare un cast per ottenere il valore della classe di interesse
- Noi sappiano che si tratta di un coin, ma dobbiamo dirlo al compilatore:

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

Polymorphism

- Una interface variable referenzia un oggetto di una classe che implementa l'interfaccia

```
Measurable x;

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

L'oggetto referenziato NON è di tipo `Measurable`; il tipo reale dell'oggetto è una qualche classe che implementa l'interfaccia Measurable

Polymorphism

- Quando si invoca un metodo di una interfaccia:

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

- Qual è il metodo (codice) effettivamente eseguito?

Dipende dal tipo dell'oggetto effettivamente referenziato.

- Se `x` referenzia un bank account, viene invocato il metodo `getMeasure` di `BankAccount`
- se x referenzia un coin, …
- Polymorphism (forme diverse):
  - Il comportamento in risposta ad un messaggio può variare a seconda dall'oggetto effettivamente coinvolto

Binding e Polymorphism

- Il termine **Binding** fa riferimento al meccanismo che determina il collegamento tra l'invocazione di un metodo ed il metodo che deve essere realmente eseguito
- Due tipi di binding:
  - Early binding, definito quando il programma viene compilato
  - Late binding, definito quando il programma viene eseguito

Binding e Polymorphism

- Se un metodo è chiamato considerando una interface variable, il compilatore non può sapere a quale oggetto essa farà riferimento
  - Il metodo con cui eseguire il binding dipende dall'oggetto a cui la interface variable fa riferimento
  - Il binding viene ritardato
  - late binding - l'accoppiamento tra invocazione e entry point, ovvero il metodo da eseguire, è risolto a runtime
- Se un metodo è chiamato considerando una class variable, il compilatore sa a quale oggetto essa fa riferimento ed il metodo che è stato invocato
  - early binding – accoppiamento tra invocazione e metodo da eseguire risolto a tempo di compilazione

Esercizio

- Ricercare, commentare e fornire una implementazione per alcune delle interface definite nella libreria Java

Esercizio

- Creare un insieme di figure geometriche di vario tipo: quadrato, rettangolo, triangolo, cerchio. Determinare la figura geometrica con il perimetro maggiore e quella con l'area maggiore

Esercizio

- Un museo ha organizzato il proprio archivio in tre file, come segue:
  Libri (Autore//Titolo//anno//collocazione)
  Stampe
  (Autore//Titolo//tecnica//anno//collocazione)
  Opere Multimediali
  (Autore//Titolo//supporto//anno//collocazione)

- Scrivere un programma che consenta la ricerca per:
  autore
  titolo
  anno
  una qualsiasi combinazione di tali fattori