



Interfacce e polimorfismo

- Saper dichiarare e usare interfacce
- Capire il concetto di polimorfismo
- Utilizzare le interfacce per ridurre l'accoppiamento tra classi
- Imparare a realizzare classi ausiliarie e classi interne



Esercizio

- Definite una classe **DataSet** che modella un insieme di valori numerici double e che fornisca i metodi per restituire la media, il minimo ed il massimo di tutti i valori inseriti nel **DataSet**



La classe DataSet

```
/** Serve a calcolare la media di
    un insieme di valori numerici,
    il minimo e il massimo
 */
public class DataSet {
    /**
    Costruisce un insieme vuoto
    */
    public DataSet() {
        sum = 0;
        count = 0;
        minimum = 0;
        maximum=0;
    }
}
```

```
/** Aggiunge il valore di un dato
    all'insieme, aggiorna min/max
    @param x : valore di un dato
 */
public void add(double x) {
    sum += x;
    if (count == 0 || minimum > x)
        minimum = x;
    if (count == 0 || maximum < x)
        maximum = x;
    count++;
}
```



La classe DataSet

```
/**
Restituisce la media dei dati
@return la media o 0 se nessun
dato è stato aggiunto
*/
public double getAverage() {
    if (count == 0) return 0;
    else return sum / count;
}

/**
Restituisce il più grande dei dati
@return il massimo o 0 se nessun
dato e` stato aggiunto
*/
public double getMaximum() {
    return maximum;
}
```

```
/**
Restituisce il piu` piccolo dei dati
@return il minimo o 0 se nessun
dato è stato aggiunto
*/
public double getMinimum(){
    return minimum;
}
private double sum;
private double minimum;
private double maximum;
private int count;
}
```



La classe DataSetTest

```
import java.util.Scanner;

public class DataSetTest{
    public static void main(String[ ] args) {
        Scanner input = new Scanner(System.in);
        DataSet ds = new DataSet();
        boolean done = false;
        while (!done){
            String x = input.nextLine();
            if (x.equalsIgnoreCase("fine") )
                done = true;
            else
                ds.add(Double.parseDouble(x));
        }
        System.out.println("la media e`:" + ds.getAverage());
    }
}
```



Scrivere codice riutilizzabile

- Supponiamo ora di voler calcolare la media dei saldi di un insieme di conti bancari
 - Dobbiamo modificare la classe **DataSet** in modo che funzioni con oggetti di tipo **BankAccount**



La classe DataSet per i conti correnti

```
/**
Serve a computare la media dei
saldi di un insieme di conti correnti.
*/
public class DataSet {
/**
Costruisce un insieme vuoto
*/
    public DataSet() {
        sum = 0;
        count = 0;
        minimum = null;
        maximum = null;
    }
```

```
/** Restituisce la media dei saldi dei
conti correnti
*/
    public double getAverage()
    {
        if (count == 0) return 0;
        else return sum / count;
    }

/**Restituisce il conto con il saldo
più grande
*/
    public BankAccount getMaximum()
    {
        return maximum;
    }
```



La classe DataSet per i conti correnti

```
// Restituisce il conto con il saldo più piccolo
public BankAccount getMinimum() { return minimum; }
// Aggiunge un conto corrente
public void add(BankAccount x) {
    sum = sum + x.getBalance();
    if (count == 0 || minimum.getBalance() > x.getBalance())
        minimum = x;
    if (count == 0 || maximum.getBalance() < x.getBalance())
        maximum = x;
    count++;
}
private double sum;
private BankAccount minimum;
private BankAccount maximum;
private int count;
}
```




La classe DataSetTest

```
/**
 * Questo programma collauda la classe DataSet per i conti correnti
 */
public class DataSetTest {
    public static void main(String[ ] args) {
        DataSet bankData = new DataSet();

        bankData.add(new BankAccount(0));
        bankData.add(new BankAccount(10000));
        bankData.add(new BankAccount(2000));

        System.out.println("Saldo medio = "+
                           bankData.getAverage());
        System.out.println("Saldo piu' alto = "+
                           bankData.getMaximun().getBalance());
    }
}
```



Scrivere codice riutilizzabile

- Supponiamo ora di voler calcolare la media dei valori di un insieme di monete
 - Dobbiamo modificare di nuovo la classe **DataSet** in modo che funzioni con oggetti di tipo **Coin**



La classe DataSet per le monete

```
/**  
Serve a computare la media dei  
valori di un insieme di monete  
*/  
public class DataSet {  
  
/** Costruisce un insieme vuoto  
*/  
    public DataSet() {  
        sum = 0;  
        count = 0;  
        minimum = null;  
        maximum = null;  
    }  
}
```

```
/** Restituisce la media dei valori  
delle monete  
*/  
    public double getAverage()  
    {  
        if (count == 0) return 0;  
        else return sum / count;  
    }  
  
/** Restituisce una moneta con il  
valore più grande  
*/  
    public Coin getMaximum()  
    {  
        return maximum;  
    }  
}
```



La classe DataSet per i conti correnti

```
// Restituisce una moneta con il valore più piccolo
    public Coin getMinimum() { return minimum; }

// Aggiunge una moneta
    public void add(Coin x) {
        sum = sum + x.getValue();
        if (count == 0 || minimum.getValue() > x.getValue())
            minimum = x;
        if (count == 0 || maximum.getValue() < x.getValue())
            maximum = x;
        count++;
    }
    private double sum;
    private Coin minimum;
    private Coin maximum;
    private int count;
}
```



La classe DataSetTest

```
/**
Questo programma collauda la classe DataSet per le monete
*/
public class DataSetTest {
    public static void main(String[] args) {
        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"));

        System.out.println("Media dei valori delle monete = "+
            coinData.getAverage());
        System.out.println("Moneta con valore piu` alto = "+
            coinData.getMaximun().getValue());
    }
}
```



Scrivere codice riutilizzabile

- Le classi **DataSet** per
 - i valori numerici
 - i conti correnti
 - le monete
- ... differiscono solo per la misura usata nell'analisi dei dati:
 - **DataSet** per double usa il valore dei dati
 - **DataSet** per oggetti di tipo BankAccount usa il valore dei saldi
 - **DataSet** per oggetti di tipo Coin usa il valore delle monete



Scrivere codice riutilizzabile

- Il meccanismo usato dalle varie classi **DataSet** è lo stesso in tutti i casi: cambiano solo i dettagli
- In tutti i casi gli oggetti che vengono aggiunti al **DataSet** hanno un certo valore, che viene usato per calcolare la media, il minimo ed il massimo
- Tutte le classi in questione potrebbero accordarsi su un unico metodo **getMeasure** che dia per ogni oggetto il valore da considerare per il **DataSet** (ogni classe decide cioè il valore da considerare “misura” dei suoi oggetti)



Scrivere codice riutilizzabile

- Supponiamo che esista un metodo **getMeasure** che fornisce la grandezza da usare nell'analisi dei dati
 - **Esempio:**
 - `x.getMeasure()`** ;
 - se **x** è un **Double**, restituisce il valore di **x**
 - se **x** è un conto, restituisce il saldo del conto
 - se **x** è una moneta, restituisce il valore della moneta
- Possiamo implementare un'unica classe **DataSet** riutilizzabile



Scrivere codice riutilizzabile

- Abbiamo bisogno di una classe che fornisca il metodo **getMeasure**
- Il comportamento di **getMeasure** varia a seconda di ciò che rappresenta realmente l'oggetto (**double**, **conto**, **moneta**,...)
- Non è quindi possibile scrivere un'implementazione unica di **getMeasure** che vada bene per tutti gli oggetti



Le interfacce

- Un' interfaccia dichiara una collezione di metodi elencando le loro firme (con tipo del valore restituito) ma non fornisce alcuna implementazione dei metodi
- Es.: l' interfaccia che dichiara il metodo **getMeasure** è

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



Interfacce

- Questa dichiarazione dichiara semplicemente un “**contratto**”
- Il contratto **Measurable** dice che per essere rispettato da una certa classe c'è bisogno che essa fornisca un metodo di nome **getMeasure**, senza parametri, che restituisca un **double**
- Un' interfaccia **non è una classe**: non si possono creare oggetti di tipo **Measurable!**



Differenze tra classi e interfacce

- Tutti i metodi di un' interfaccia sono *astratti*, cioè non hanno un' implementazione
- Tutti i metodi di un' interfaccia sono automaticamente **public** (non serve lo specificatore d' accesso)
- Un' interfaccia non ha variabili di istanza, può definire solo costanti
- Esistono variabili del tipo di un' interfaccia ma non esistono istanze di un' interfaccia
- Una variabile del tipo di un' interfaccia può contenere istanze delle classi che implementano l' interfaccia



Interfacce: esempio

```
interface I { int m1(int); B m2(String); }
```

```
public class A implements I {  
    public int m1(int i){return i;}  
    public B m2(String s){return new B();}  
    public void m3(){return;}  
}
```

```
public class B implements I {  
    public int m1(int i){return i+1;}  
    public B m2(String s){return this;}  
}
```



La nuova classe DataSet

```
/**
Serve a computare la media di
un insieme di valori
*/
public class DataSet {

/**
Costruisce un insieme vuoto
*/
    public DataSet() {
        sum = 0;
        count = 0;
        minimum = null;
        maximum = null;
    }
}
```

```
// Restituisce la media dei valori
public double getAverage()
{
    if (count == 0) return 0;
    else return sum / count;
}

/**Restituisce un oggetto
Measurable con il valore più
grande
*/
public Measurable getMaximum()
{
    return maximum;
}
```



La nuova classe DataSet

```
// Restituisce un oggetto Measurable con il valore più piccolo
public Measurable getMinimum() { return minimum; }
// Aggiunge un oggetto Measurable
public void add(Measurable x) {
    sum = sum + x.getMeasure();
    if (count == 0 || minimum.getMeasure() > x.getMeasure())
        minimum = x;
    if (count == 0 || maximum.getMeasure() < x.getMeasure())
        maximum = x;
    count++;
}
private double sum;
private Measurable minimum;
private Measurable maximum;
private int count;
}
```



Classi che implementano l'interfaccia

- La nuova classe **DataSet** può essere usata per analizzare oggetti di qualsiasi classe che realizza l'interfaccia **Measurable**
- Una classe **realizza** (*implementa*) un'interfaccia se fornisce l'implementazione di **tutti** i metodi dichiarati nell'interfaccia
 - Corrispondenza con i metodi dell'interfaccia data dalle firme dei metodi
 - Può contenere metodi non dichiarati nell'interfaccia

- **Esempio:**

```
public class BankAccount implements Measurable {  
    public double getMeasure() {  
        return balance;  
    }...  
// tutti gli altri metodi di BankAccount  
}
```




Classi che implementano l' interfaccia

- Allo stesso modo posso scrivere la classe **Coin** che implementa l' interfaccia **Measurable**

```
public class Coin implements Measurable {  
    public double getMeasure() {  
        return value;  
    }  
    ... // tutti gli altri metodi di Coin  
}
```

Syntax

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

Example

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

The methods of an interface are automatically public. —

No implementation is provided. —

Syntax

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

Example

```
public class BankAccount implements Measurable
{
    . . .
    public double getMeasure()
    {
        return balance;
    }
    . . .
}
```

— List all interface types that this class implements.

— This method provides the implementation for the method declared in the interface.

BankAccount
instance variables

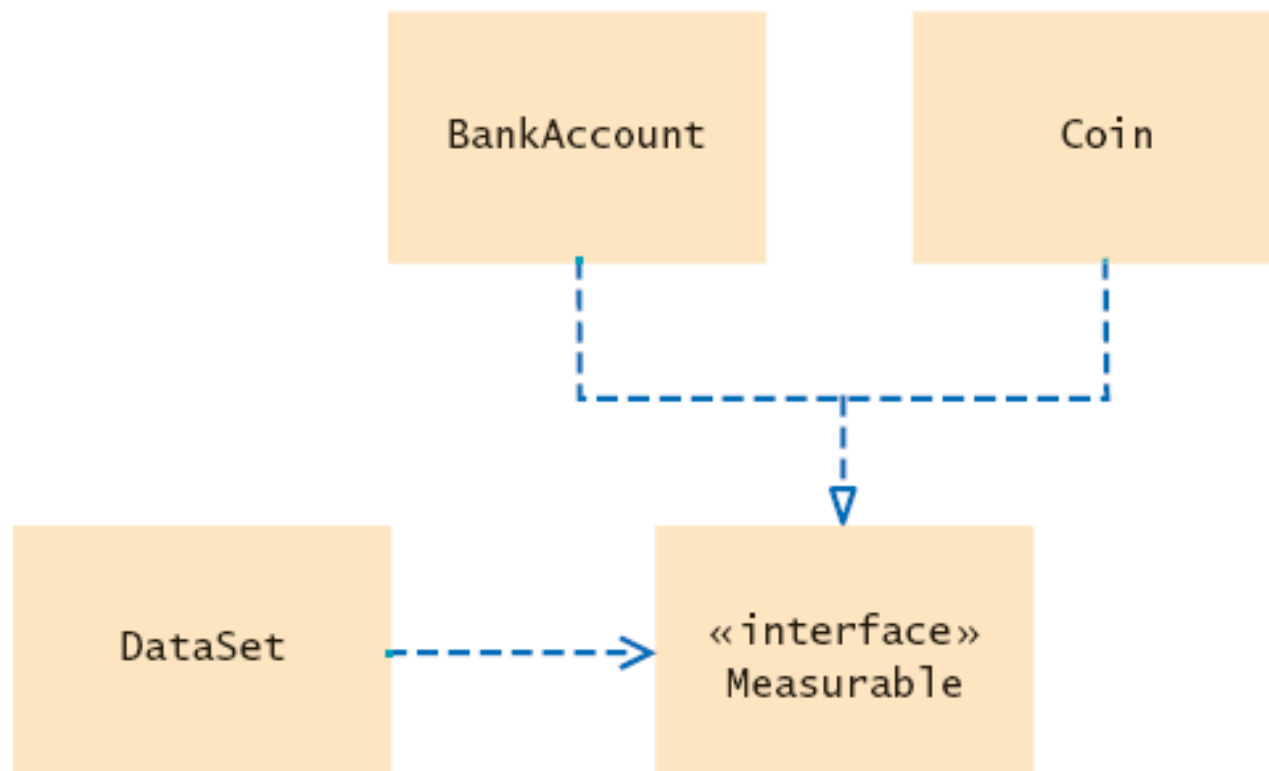
Other
BankAccount methods

Implementare una interfaccia

- In tal modo la classe si impegna al rispetto di un contratto



Schema UML della classe `DataSet` che dipende da `Measurable` e delle classi che implementano l'interfaccia `Measurable`



Le interfacce possono ridurre l'accoppiamento tra classi



Riduzione di accoppiamento

- Come si vede dal grafico la classe **DataSet** non è accoppiata né con **BankAccount** né con **Coin**
- Essa dipende solo dall'interfaccia **Measurable**
- Questo disaccoppiamento rende riutilizzabile la classe **DataSet**
- Ogni classe che implementi l'interfaccia **Measurable** può essere usata con la classe **DataSet** senza che questa ne dipenda



La classe DataSetTest

```
/**
```

```
    Questo programma collauda la classe DataSet per i conti correnti  
    e le monete
```

```
*/
```

```
public class DataSetTest
```

```
{
```

```
    public static void main(String[] args) {
```

```
        DataSet ds = new DataSet();
```

```
        ds.add(new BankAccount(0));
```

```
        ds.add(new BankAccount(10000));
```

```
        ds.add(new BankAccount(2000));
```

```
        System.out.println("Saldo medio = " + ds.getAverage());
```

```
        Measurable max = ds.getMaximum();
```

```
        System.out.println("Saldo piu` alto = " + max.getMeasure());
```



La classe DataSetTest

```
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"));

System.out.println("Valore medio delle monete = " +
    coinData.getAverage());
max = coinData.getMaximum();
System.out.println("Valore max delle monete = " +
    max.getMeasure());
}
```



Interfacce

- Una classe può implementare anche più di una interfaccia
- In questo caso basta elencare, separate da virgola, tutte le interfacce che implementa dopo la parola riservata **implements**
- I metodi della classe che corrispondono a quelli dell'interfaccia implementata devono obbligatoriamente essere dichiarati **public**
- Un'interfaccia va definita in un file .java che si chiama con lo stesso nome dell'interfaccia (esattamente come per le classi pubbliche)



Conversione fra tipi

- E' possibile convertire dal tipo di una classe al tipo dell'interfaccia implementata dalla classe
- Esempi:

```
ds.add(new BankAccount(100));
```

- il tipo **BankAccount** dell'argomento è convertito nel tipo **Measurable** del parametro del metodo **add**

```
BankAccount b = new BankAccount(100);  
Coin c = new Coin(0.1, "dime");  
Measurable x = b; /* x si riferisce ad un oggetto  
                   di tipo BankAccount */  
x = c; /* ora x si riferisce ad un oggetto di tipo  
       Coin */
```

- Possiamo assegnare ad una variabile di tipo **Measurable** un oggetto di una qualsiasi classe che implementa **Measurable**



Conversione fra tipi

- Ovviamente non è possibile convertire dal tipo di una classe al tipo di un'interfaccia che NON è implementata da quella classe

- **Esempio:**

```
Measurable r = new Rectangle(1,2,5,3);
```

```
// errore: Rectangle non implementa Measurable
```



Conversione fra tipi

- Per convertire un tipo interfaccia in un tipo classe occorre un casting

- Esempio:

```
BankAccount b = new BankAccount(100);
```

```
Measurable x = b;
```

```
BankAccount account = (BankAccount) x;
```



Conversione fra tipi

- Consideriamo la seguente istruzione:
`Measurable max = bankData.getMaximum();`
`// l'oggetto di tipo BankAccount restituito da`
`// getMaximum è convertito nel tipo Measurable`
- Anche se `max` si riferisce ad un oggetto che in origine è di tipo `BankAccount`, non è possibile invocare il metodo `deposit` per `max`
 - **Esempio:**
`max.deposit(35); // ERRORE`
- Per poter invocare i metodi di `BankAccount` che non sono contenuti nell'interfaccia `Measurable` si deve effettuare il cast dell'oggetto al tipo `BankAccount`
 - **Esempio:**
`BankAccount acc = (BankAccount) max;`
`acc.deposit(35); //OK`



Conversione fra tipi

- E' possibile effettuare il casting di un oggetto ad un certo tipo solo se l'oggetto in origine era di quel tipo
 - Esempio:

```
BankAccount b = new BankAccount(100);  
Measurable x = b;  
Coin c = (Coin) x; /* errore che provoca  
un'eccezione: il tipo originale  
dell'oggetto a cui si riferisce x  
non è Coin ma BankAccount */
```



Conversione fra tipi

- L'operatore **instanceof** permette di verificare se un oggetto appartiene ad un determinato tipo
- Al fine di evitare il lancio di un'eccezione, prima di effettuare un cast di un oggetto ad un certo tipo classe possiamo verificare se l'oggetto appartiene effettivamente a quel tipo classe

- **Esempio:**

```
if (x instanceof Coin ) {  
    Coin c = (Coin) x;  
    ... }
```



Un metodo di ordinamento generico

```
public static void sort(ArrayList v) {  
    int k;  
    int n = v.size();  
    k = 0;  
    while (k != n-1) {  
        int j = getSmallest(v, k);  
        exchange(v, k, j);  
        k++;  
    }  
}
```



Il metodo getSmallest

```
public static int getSmallest(ArrayList v, int k) {
    if (v==null || v.size()==k)
        return -1;
    int i;
    int small = k;
    i = k+1;
    while (i != v.size()) {
        String current = (String) v.get(i);
        String smallest = (String) v.get(small);
        if (current.compareTo(smallest)<0)
            small = i;
        i++;
    }
    return small;
}
```

// funziona solo su String



Modifichiamo il metodo getSmallest

```
public static int getSmallest(ArrayList v, int k) {  
    // funziona su qualunque implem. di Comparable  
    if (v==null || v.size()==k)  
        return -1;  
    int i;  
    int small = k;  
    i = k+1;  
    while (i != v.size()) {  
        Comparable current = (Comparable) v.get(i);  
        Comparable smallest = (Comparable) v.get(small);  
        if (current.compareTo(smallest) < 0)  
            small = i;  
        i++;  
    }  
    return small;  
}
```



Implementazione di Comparable

- Per usare il metodo `sort` basta implementare l'interfaccia `Comparable`, ad esempio, per gli interi

```
class MyInteger implements Comparable{
    public MyInteger(int val) {this. val = val;}
    public int getVal() {return val;}
    public int compareTo(Object o) {
        MyInteger mi = (MyInteger) o;
        if (this.val < mi.getVal()) return -1;
        else if (this.val > mi.getVal()) return 1;
        else return 0;
    }
    private int val;
}
```



Alcune interfacce standard

- L'interfaccia **Comparable** contiene un metodo astratto chiamato **compareTo**, usato per confrontare oggetti
 - La classe **String** implementa l'interfaccia **Comparable** che consente di confrontare stringhe in ordine alfabetico mediante il metodo **compareTo** specificato ad hoc
 - **int compareTo(Object obj)**
- L'interfaccia **Iterator** indica i metodi da implementare per gestire una collezione di oggetti
 - Caso per caso si deve decidere l'ordine con cui gli oggetti della collezione devono essere restituiti dai metodi
 - **boolean hasNext()**
 - **Object next()**
 - **void remove()**