Java interface

Concetto di Java interface

- Un'interfaccia dichiara un insieme di metodi elencandone le firme (con tipo del valore restituito) ma non fornisce alcuna implementazione
- Es.: un'interfaccia Measurable è

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

- Può contenere anche la dichiarazione di variabili ma vengono considerate **`final`** e **`static`** (costanti di interfaccia)

Concetto di Java interface

- Un’interfaccia esprime una caratteristica che può essere comune a concetti diversi, non è un concetto in sè
  - Ad es. Measurable è una caratteristica di ogni cosa che può essere misurata (BankAccount, Rectangle, Coin, Double,etc.)
  - un'implementazione unica di getMeasure() significativa non è possibile (la definizione non è unica)

Concetto di Java interface

- Un'interfaccia può essere implementata da una o più classi
- Le classi che implementano un'interfaccia sono obbligate a fornire il codice per i metodi indicati nell'interfaccia
- I metodi di un'interfaccia sono **`public`** per default (non serve specificatore d'accesso)
- Un'interfaccia quindi esprime un requisito sull'interfaccia pubblica delle classi che la implementano

Implementazione di un'interfaccia

- Si usa la parola chiave **implements** nella dichiarazione della classe
- 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
}
```

Altro esempio….

- Allo stesso modo possiamo far sì che anche la classe Coin implementi Measurable

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

Implementazione multipla

- Un concetto può avere più caratteristiche
  - una classe può implementare una o più interfacce

```
public class BankAccount implements
                              Measurable,Serializable,Cloneable {
    …..
}
```

  - ma una classe può estendere una sola superclasse

```
public class CheckingAccount extends BankAccount {
    …..
}
```

Variabile del tipo di un'interfaccia

- Una variabile del tipo di un'interfaccia può contenere il riferimento ad oggetti delle classi che implementano l'interfaccia

```
Measurable x = new Coin(…);
Measurable y = new BankAccount(…);
```

Polimorfismo

- I metodi di un'interfaccia sono solitamente polimorfici
  - getMeasure() ha una definizione differente per BankAccount e Coin

- Consentono l'implementazione di soluzioni generali
  - Ad es. possiamo scrivere un unico pezzo di codice che calcola alcune statistiche sulle misurazioni di una collezione di oggetti Measurable
  - Può essere utilizzato con qualsiasi implementazione di Measurable (ad es. insieme di BankAccount, insieme di Coin, ….)

La classe DataSet

```
/**
 Serve a computare la media delle
    misurazioni su un insieme di
    oggetti, l'oggetto con misura
    massima e quello con misura
    minima
*/

public class DataSet {
   /**
       Default: insieme vuoto
 */
   public DataSet() {
      sum = 0;
      count = 0;
      minimum = null;
      maximum= null;
   }
```

```
// Restituisce la media delle
     misurazioni

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

   /**Restituisce un oggetto
     Measurable di misura massima
 */
   public Measurable getMaximum()
   {
      return maximum;
   }
```

La classe DataSet

```
 //   Restituisce un oggetto Measurable di misura minima
   public Measurable getMinimum() {  return minimum; }
//  Aggiunge un oggetto Measurable e aggiorna i dati
   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;
}
```

La classe DataSetTest

```
/**
   Questo programma collauda la classe DataSet per i conti correnti
*/
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());
```

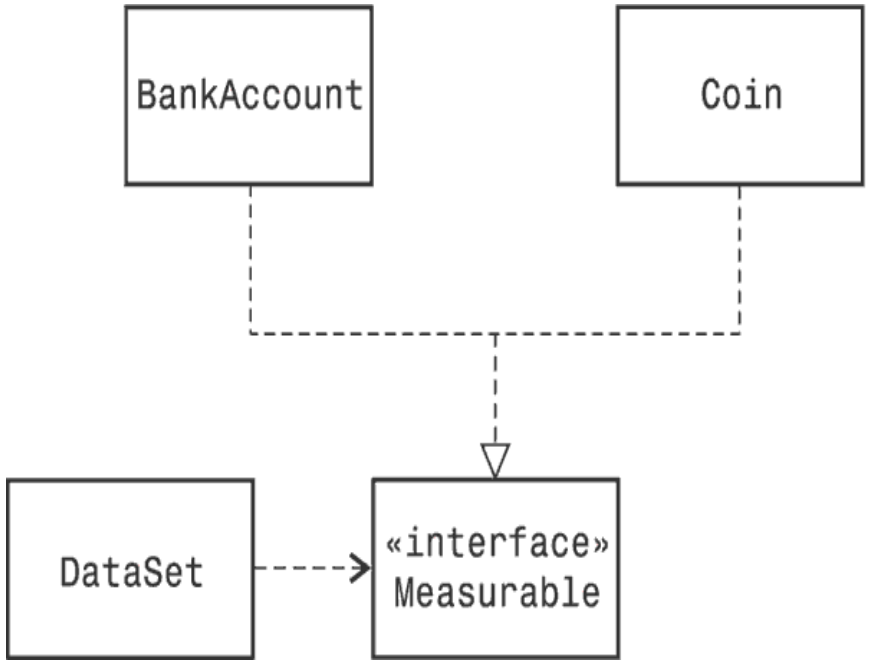
La classe DataSetTest

```
        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());
    }
}
```

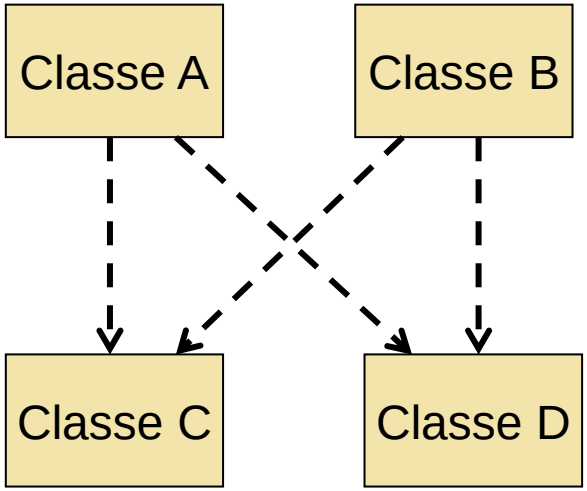
Separazione tra parti di un progetto

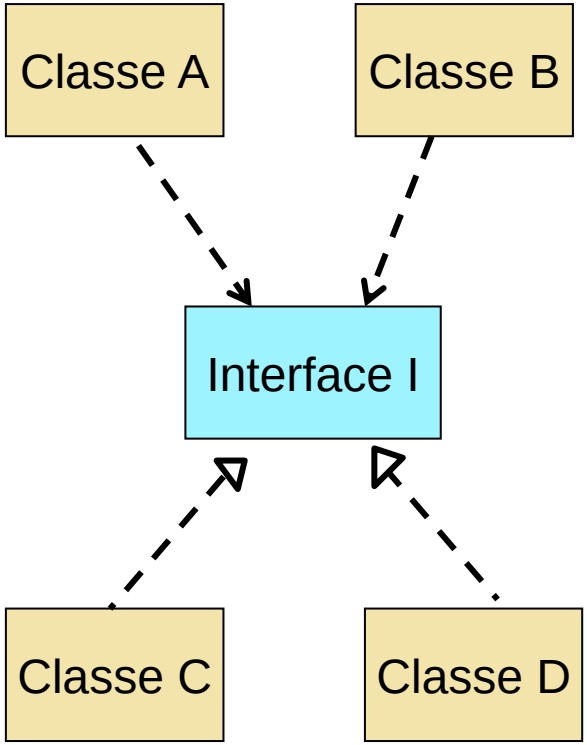
- Un'interfaccia divide un progetto in due
  - Parte che usa la caratteristica definita
  - Parte che implementa la caratteristica

- Le due parti possono essere sviluppate autonomamente e in momenti differenti
  - Serve solo conoscere l'interfaccia



Riduzione livello di accoppiamento





Superclassi vs Interfacce

- entrambi definiscono dei “supertipi”
  - consentono di definire metodi polimorfici
- un’interfaccia non è una classe:
  - non ha uno stato, né un comportamento
  - è un elenco di metodi da implementare
- una superclasse è una classe:
  - ha uno stato e un comportamento che sono ereditati dalla sottoclasse

Differenze tra classi e interfacce

- Tutti i metodi di un'interfaccia sono *astratti*, cioè non hanno un'implementazione
- Un'interfaccia non ha variabili di istanza (può solo contenere la definizione di costanti)
- Esistono variabili del tipo di un'interfaccia ma non esistono istanze di un'interfaccia

Estensione di Interfacce

- il meccanismo delle estensioni si può applicare anche alle interfacce
  - effetto incrementale sulla richiesta di metodi da implementare
- un'interfaccia può estendere più di una interfaccia, così come una classe può implementare più interfacce
- Es.

```
public interface DataIO extends
                    DataInput, DataOutput {……}
```

(DataInput e DataOutput sono interfacce)

Confronto Classi Astratte - Interfacce

- Le classi astratte e le interfacce sono simili:
  - catturano astrazioni che possono raggruppare aspetti comuni di concetti differenti
- Le classi in generale forniscono un'implementazione
  - anche le classi astratte solitamente implementano alcuni metodi

Confronto Classi Astratte - Interfacce

- L'ereditarietà è usata per raggruppare concetti strettamente correlati
  - un'estensione può essere vista come un raffinamento del concetto espresso dalla superclasse (e quindi si può estendere una sola classe)

- Le interfacce possono essere usate per enfatizzare un aspetto comune di concetti eterogenei
  - Ad es., l'uso di Measurable, Measurer, etc.
  - Una classe può implementare più interfacce, ciascuna corrispondente ad un aspetto del concetto espresso dalla classe

Riepilogando….

- Un'interfaccia indica solo dei metodi da implementare
  - consente di separare le parti di un progetto
  - può facilmente essere integrata in un progetto sviluppato indipendentemente
  - è consentito implementare più interfacce con la stessa classe

- Una classe astratta fornisce più struttura
  - definisce alcune implementazioni di default
  - permette di definire delle variabili di istanza/statiche/final

- Non è errato usare entrambe in un progetto:
  - l'interfaccia definisce un supertipo per un aspetto comune a diversi concetti (e consente di scrivere codice comune per utilizzare concetti eterogenei, ad es. DataSet)
  - una classe astratta fornisce una base comune all'implementazione di concetti specifici correlati (una classe può estendere una sola superclasse)

L'interfaccia Comparable

- La classe String implementa l'interfaccia Comparable, appartenente alla libreria standard di Java

```
public interface Comparable <T>{
    int compareTo(T other);
}
```

Tipo di un'interfaccia

- Un'interfaccia definisce un supertipo per
  - tutte le interfacce che la estendono e
  - tutte le classi che la implementano

- Valgono regole analoghe alle regole di conversione tra i tipi definititi dalle classi

Conversione fra tipi

- E` sempre possibile convertire dal tipo di una classe al tipo di un'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 effettuare una conversione 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 dal tipo di un'interfaccia al tipo di una classe che la implementa 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 restituito da getMaximum
// è già di tipo Measurable
```

- Anche se max si riferisce ad un oggetto che in origine è di tipo BankAccount, non e` 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 e`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;
… }
```

Conversione tra tipi

- In generale:
  - Conversioni da un tipo (classe o interfaccia) a un supertipo (classe o interfaccia) sono automatiche
  - Conversioni da un supertipo A ad un tipo B richiedono un casting per la compilazione e hanno successo a runtime solo se l'oggetto è un'istanza di una classe nella discendenza di B
  - Conversioni tra tipi che non sono gerarchicamente collegati (uno nella discendenza dell'altro) non sono possibili (errore in compilazione anche in presenza di casting)