Polimorfismo

Classi: BankAccount e CheckingAccount

```
public class BankAccount
   {  public double getBalance() {…}
      public void deposit(double d) {…}
      public void withdraw(double d) {…}
      private double balance;
   }

public class CheckingAccount extends BankAccount{
      public void deposit(double d) {…}
      public void withdraw(double d) {…}
      public void deductFees() {…}
      private int transactionCount;
   }
```

Polimorfismo

- L'invocazione

```
x.deposit(100);
```

  può chiamare metodi diversi a seconda del tipo reale dell'oggetto x
- Il metodo deposit(..) viene detto polimorfico (multiforme)
- Il polimorfismo in Java è realizzato attraverso:
  - Ereditarietà -- Overriding
  - Uso di interfacce (prossime lezioni)
- Altro caso di polimorfismo in senso lato
  - Overloading --- metodi sono distinti dai parametri espliciti

Polimorfismo vs Overloading

- Entrambi invocano metodi distinti con lo stesso nome, ma...
  - Con l'overloading scelta del metodo appropriato avviene in fase di compilazione, esaminando il tipo dei parametri
    - early binding, effettuato dal compilatore
  - Con il polimorfismo avviene in fase di esecuzione
    - late binding, effettuato dalla JVM

Polimorfismo: riutilizzo codice

- Consideriamo un metodo **`transfer`**:

```
public void transfer(BankAccount other,
      double amount)
   {
      withdraw(amount);
      other.deposit(amount);
   }
```

- Possiamo usarlo con parametri di un qualsiasi tipo di **`BankAccount`**
  - ad es. un **`SavingsAccount`** o un **`CheckingAccount`**
- Stesso metodo **`transfer`** per tipi differenti

Polimorfismo

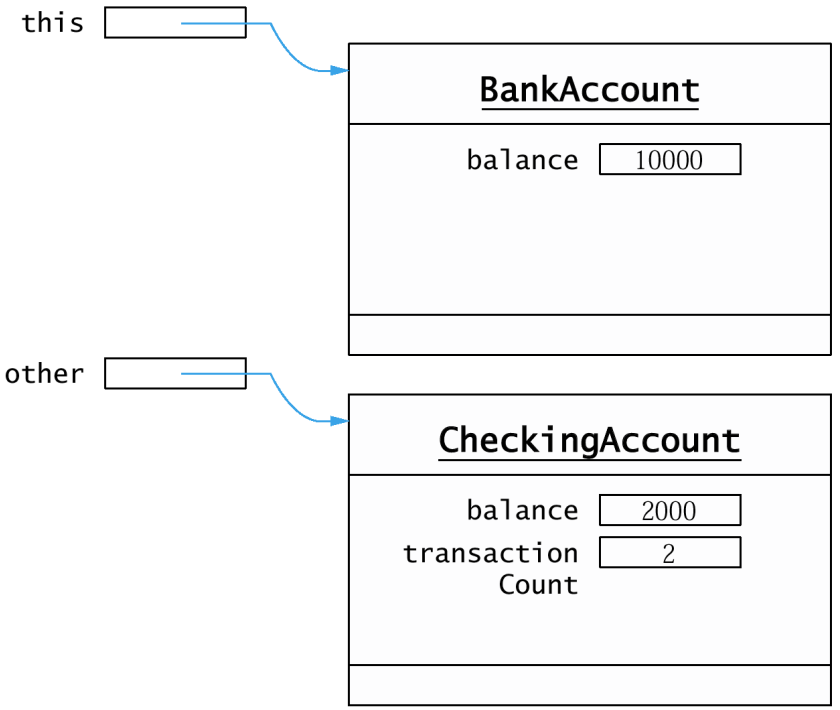
- E' lecito passare un riferimento di tipo **`CheckingAccount`** a un metodo che si aspetta un riferimento di tipo **`BankAccount`**

```
BankAccount momsAccount = . . .;
CheckingAccount harrysChecking = . . .;
momsAccount.transfer(harrysChecking, 1000);
```

- Il compilatore copia il riferimento all'oggetto harrisChecking (di tipo CheckingAccount) nella variabile **`other`** del tipo della superclasse BankAccount

Polimorfismo

- a tempo di compilazione non è possibile stabilire il tipo effettivo della variabile other di transfer
  - non è possibile stabilire ad esempio che si riferisce a un oggetto di tipo **`CheckingAccount`**
- l'unica informazione è che other è di tipo **`BankAccount`**



Polimorfismo

- il metodo transfer invoca il metodo **`deposit`**.
  - Quale?
- la decisione avviene a runtime (late binding)
  - dallo spazio dell'oggetto si segue il link al codice da eseguire
- su un oggetto di tipo **`CheckingAccount`** viene invocato **`deposit`** di **`CheckingAccount`**
  - vale il tipo effettivo dell'oggetto non il tipo della variabile

File BankAccount.java

```
/**
   Un conto bancario ha un saldo che può essere modificato
   con versamenti e prelievi.
*/
public class BankAccount
{
   /**
      Costruisce un conto bancario con saldo zero.
   */
   public BankAccount()
   {
      balance = 0;
   }

   /**
      Costruisce un conto bancario con un saldo assegnato.
      @param initialBalance il saldo iniziale
   */
   public BankAccount(double initialBalance)
   {
      balance = initialBalance;
   }
```

```
/**
   Versa denaro nel conto bancario.
   @param amount la somma da versare
*/
public void deposit(double amount)
{
   balance += amount;
}

/**
   Preleva denaro dal conto bancario.
   @param amount la somma da prelevare
*/
public void withdraw(double amount)
{
   balance -= amount;
}

/**
   Restituisce il valore del saldo del conto bancario.
   @return il saldo attuale
```

```
*/
   public double getBalance()
   {
      return balance;
   }

   /**
      Trasferisce denaro dal conto ad un altro conto.
      @param amount la somma da trasferire
      @param other l'altro conto
   */
   public void transfer(double amount, BankAccount other)
   {
      withdraw(amount);
      other.deposit(amount);
   }

   private double balance;
}
```

File CheckingAccount.java

```
/**
   Un conto corrente che addebita commissioni per ogni
transazione.
*/
public class CheckingAccount extends BankAccount
{
   /**
      Costruisce un conto corrente con un saldo assegnato.
      @param initialBalance il saldo iniziale
   */
   public CheckingAccount(double initialBalance)
   {
      // chiama il costruttore della superclasse
      super(initialBalance);

      // inizializza il conteggio delle transazioni
      transactionCount = 0;
   }
```

```
//metodo sovrascritto
public void deposit(double amount){
      transactionCount++;
      // ora aggiungi amount al saldo
      super.deposit(amount);
 }

//metodo sovrascritto
 public void withdraw(double amount){
      transactionCount++;
      // ora sottrai amount dal saldo
      super.withdraw(amount);
 }
```

```
//metodo nuovo
public void deductFees(){
      if (transactionCount > FREE_TRANSACTIONS){
         double fees = TRANSACTION_FEE *
               (transactionCount - FREE_TRANSACTIONS);
         super.withdraw(fees);
      }
      transactionCount = 0;
   }


  private int transactionCount;

   private static final int FREE_TRANSACTIONS = 3;
   private static final double TRANSACTION_FEE = 2.0;
}
```

File SavingsAccount.java

```
/**
   Un conto bancario che matura interessi ad un tasso
fisso.
*/
public class SavingsAccount extends BankAccount{
   /**
      Costruisce un conto bancario con un tasso di
      interesse assegnato.
      @param rate il tasso di interesse
   */
   public SavingsAccount(double rate){

      interestRate = rate;
   }
```

```
   /**
      Aggiunge al saldo del conto gli interessi maturati.
   */
      public void addInterest()
      {
            double interest = getBalance()
                  * interestRate / 100;
            deposit(interest);
      }

      private double interestRate;
}
```

File AccountTest.java

```
/**
   Questo programma collauda la classe BankAccount
   e le sue sottoclassi.
*/
public class AccountTest{
   public static void main(String[] args){
      BankAccount momsSavings
            = new SavingsAccount(0.5);

      BankAccount harrysChecking
            = new CheckingAccount(100);

      momsSavings.deposit(10000);
      momsSavings.transfer(2000, harrysChecking);
      harrysChecking.withdraw(1500);
      harrysChecking.withdraw(80);
```

```
    momsSavings.transfer(1000, harrysChecking);
     harrysChecking.withdraw(400);

     // simulazione della fine del mese
     ((SavingsAccount) momsSavings).addInterest();
     ((CheckingAccount) harrysChecking).deductFees();

     System.out.println("Mom's savings balance = $"
           + momsSavings.getBalance());

     System.out.println("Harry's checking balance = $"
           + harrysChecking.getBalance());
   }
}
```