UNIVERSITÀ PARTHENOPE
DiST
Dipartimento di Scienze e Tecnologie

Programmazione 3
e
Laboratorio di Programmazione 3

Le basi della programmazione Object Oriented

Proff. Angelo Ciaramella – Emanuel Di Nardo

Componenti fondamentali

- Concetti che sono alla base della conoscenza di Java
  - Classe
  - Oggetto
  - Membro
    - Attributo (variabile membro)
    - Metodo (metodo membro)
  - Costruttore
  - Package

Classe e oggetti

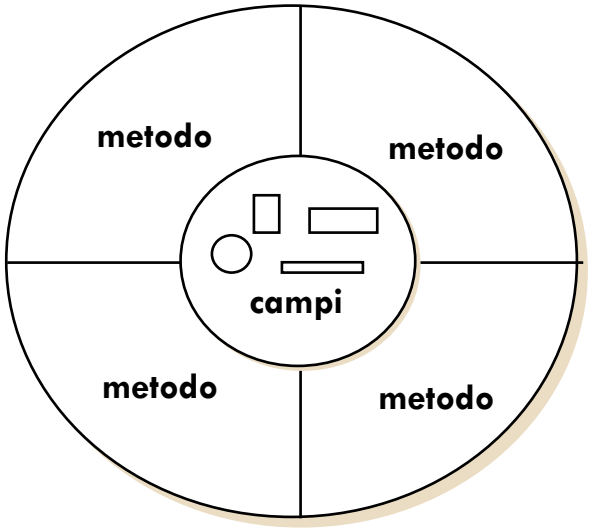
- Oggetti e classi sono concetti fondamentali per la programmazione in Java

Definizione 1
Una **classe** è un'astrazione indicante un insieme di oggetti che condividono le stesse caratteristiche e le stesse funzionalità

Definizione 2
Un **oggetto** è un'istanza (ovvero una creazione fisica) di una classe

Classe

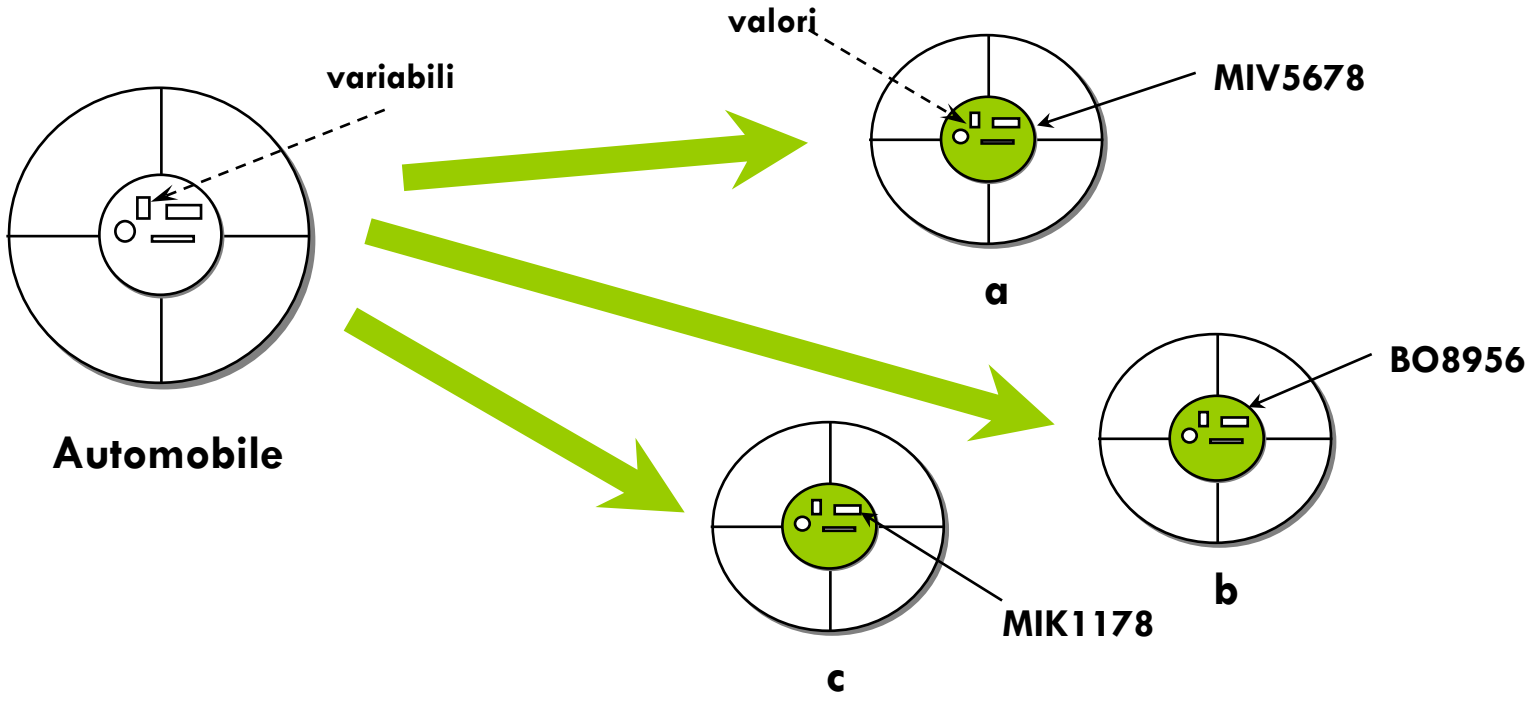
- Una classe ha
  - un nome
  - contiene due tipi di membri
    - campi e metodi



Tipo di dati astratti

Oggetto

- Un oggetto è un'istanza (esemplare) di una classe



Esemplari con identico comportamento e stato diverso

Creare oggetti

- L'operatore `new` permette di creare un oggetto

- Costruzione di un rettangolo tramite la classe `Rectangle` della libreria standard di Java

```
new Rectangle(5,10,20,30)
```

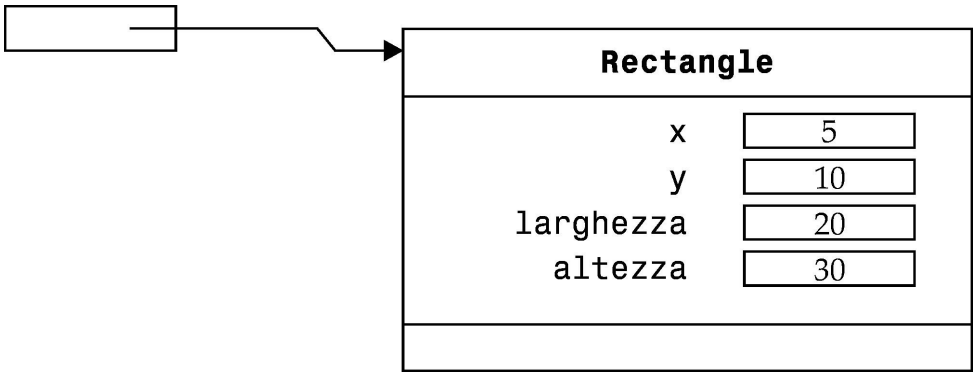

Variabili oggetti

- Per conservare traccia di un oggetto abbiamo bisogno di una `variabile oggetto`

```
Rectangle cerealBox = new Rectangle(5, 10, 20, 30);
```

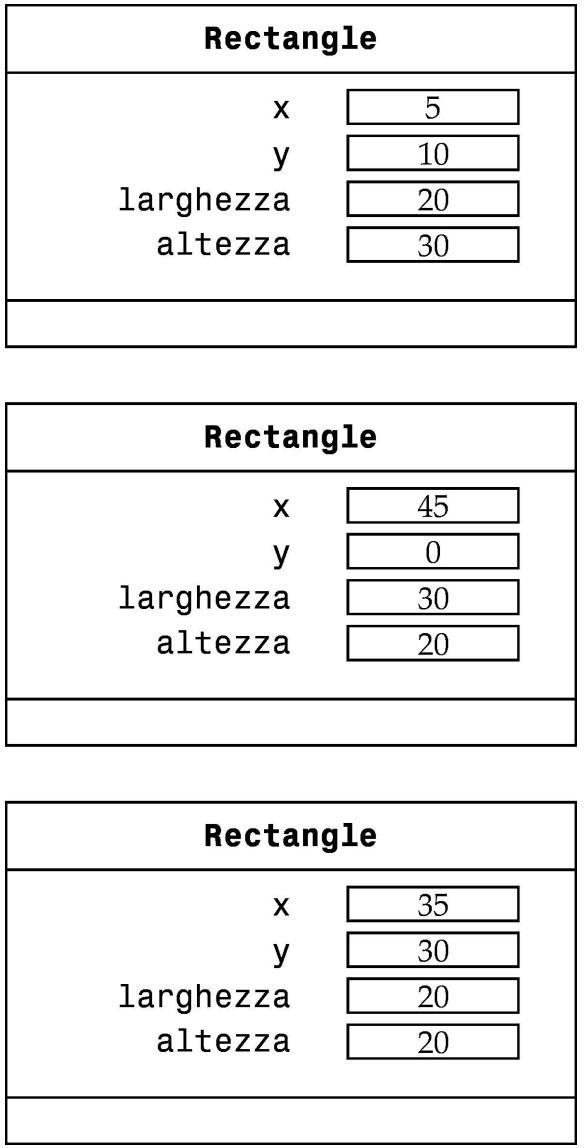
- Regole per la scelta del nome di una variabile
  - I nomi possono essere composti di lettere, cifre e segni di sottolineatura (_), ma non possono iniziare con una cifra
  - Non si possono usare altri simboli, come ad esempio ? o %, e spazi
  - Non si possono usare le parole riservate
  - I nomi delle variabili sono case-sensitive

Variabili oggetto



Variabile oggetto che si riferisce ad un oggetto

Oggetti Rectangle



Importare le classi

- Le classi Java sono raggruppate in pacchetti
  - Se si usa una classe di un pacchetto diverso da `java.lang` (contenente le classi System e String) bisogna importarla

```
import java.awt.Rectangle
```

```
import java.awt.*
```

tutte le classi del pacchetto

MoveTest

```
import java.awt.Rectangle

public class MoveTest {
  public static void main (String[] args) {
    Rectangle cerealBox = new Rectangle(5,10,20,30);

    System.out.println(cerealBox);

    cerealBox.translate(15,25);

    System.out.println(cerealBox);
   }
}
```


Riassumendo …

- Costruzione di oggetti

```
new NomeClasse(parametri)
```

- Definizione di variabile

```
NomeTipo nomeVariabile;
NomeTipo nomeVariabile = espressione;
```

- Importare la classe di un pacchetto

```
import nomePacchetto.NomeClasse;
```


Metodi

- Una classe definisce i metodi che si possono applicare ai suoi oggetti

- Il metodo serve a definire una funzionalità che deve avere il concetto che si sta astraendo con la classe

Greeter

```
public class Greeter {
  public String sayHello() {
    String message = "Hello, World!";

    return message;
   }
}
```

Classe Greeter con un solo metodo

Riassumendo…

```
public class NomeClasse {
  specificatoreDiAccesso tipoRestituito nomeMetodo
  (tipoParametro nomeParametro, ...) {
    corpo del metodo
   }
}
```

Implementazione di un metodo

GreeterTest

```
public class GreeterTest {
  public static void main (String[] args) {
    Greeter worldGreeter = new Greeter();

    System.out.println(worldGreeter.sayHello());

  }
}
```


Aritmetica

```
public class Aritmetica {
  public int Somma(int a, int b) {

    return (a + b);

   }
}
```


AritmeticaTest

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

    Aritmetica oggetto = new Aritmetica();

    System.out.println(oggetto.Somma(5, 6));

   }
}
```


Varargs

- E' possibile creare metodi che dichiarano un numero non definito di parametri di un certo tipo (variable arguments)

```
public class AritmeticaVariable {
  public int Somma(int... interi) {
     corpo del metodo
   }
}
```

```
AritmeticaVariable ogg = new AritmeticaVariable();

ogg.somma();
ogg.somma(1, 2, 3);
...
```


Variabili istanza

- Un oggetto usa variabili istanza (o campi) per memorizzare il suo stato
  - I dati di cui ha bisogno per eseguire i suoi metodi
- Ogni oggetto della classe ha il proprio insieme di variabili istanza
- Generalmente, le variabili istanza si dichiarano private
  - Solo i metodi della stessa classe vi possono accedere
- L'incapsulamento è il processo che nasconde i dati dell'oggetto
  - Solo i metodi possono accedervi

Greeter

```
public class Greeter {
  public String sayHello() {
    String message = "Hello, " + name + "!";

    return message;
   }

 private String name;
}
```


Riassumendo…

```
public class NomeClasse {
   ...

 specificatoreDiAccesso tipoVariabile nomeVariabile;

   ...

}
```


Costruttori

- I costruttori contengono istruzioni per inizializzare gli oggetti
- Il nome del costruttore è sempre uguale al nome della classe
- Generalmente i costruttori sono dichiarati `public`
  - consentire a qualsiasi codice in un programma di costruire nuovi oggetti della classe
- I costruttori non hanno tipi di dati restituiti
- L'operatore `new` invoca il costruttore
- Se proviamo a compilare una classe sprovvista di costruttore il compilatore ne fornisce uno implicitamente

Greeter

```
public class Greeter {
  public Greeter(String aName) {
     name = aName;
   }

  public String sayHello(){
    String message = "Hello, " + name + "!";

    return message;
   }

 private String name;
}
```


Riassumendo…

```
specificatoreDiAccesso class NomeClasse {
  costruttori
  metodi
  campi (variabili istanza)
}
```

- I costruttori e i metodi di una classe costituiscono l'interfaccia pubblica della classe

Esercizio

- Creare il codice completo per la classe Greeter ed una classe di test che visualizzi il nome e il cognome

Riassumendo…

```
specificatoreDiAccesso class NomeClasse {
  specificatoreDiAccesso NomeClasse
  (tipoParametro nomeParametro, ...) {
    corpo del costruttore
   }
}
```


Realizzare una classe

- Implementare una classe `BankAccount` che permette di gestire un conto bancario contenente un saldo che può essere modificato da depositi e prelievi
  - costruttore
  - metodo `deposit` per versare denaro nel conto bancario
  - metodo `withdraw` preleva denaro dal conto bancario
  - metodo `getBalance` ritorna il saldo attuale

- Implementare una classe `BankAccountTest` per il collaudo della classe `BankAccount`
  - Inserimento denaro
  - Prelievo
  - Stampa Saldo

- Domanda: come associare automaticamente un numero di conto in base all'ultimo numero di conto associato

Codice di riferimento

`BankAccount.java, BankAccountTest.java`

Tipi di variabili

- Variabili
  - istanza
    - appartengono ad un oggetto
  - parametro e le variabili locali
    - appartengono ad un metodo

- Le variabili istanza
  - se non viene esplicitamente assegnato loro un valore in un costruttore, vengono inizializzate con un valore predefinito
    - i campi numerici vengono inizializzati a 0; i riferimenti ad oggetti vengono impostati ad un valore ***null***

- Le variabili locali devono essere inizializzate esplicitamente
  - in caso contrario il compilatore segnale un errore

Parametri impliciti ed espliciti

```
momsSavings.deposit(500);

public void deposit(double amount)
{
        double newBalance = balance + amount;
        balance = newBalance;
}
```

`momsSavings` è il parametro implicito (oggetto con cui è stato invocato il metodo)

Parametro esplicito

Parametri impliciti ed espliciti

- Il parametro implicito di un metodo è l'oggetto con il quale il metodo viene invocato

- E' possibile accedere al parametro implicito usando la parola chiave `this`

```
momsSavings.deposit(500);

public void deposit(double amount)
{
        double newBalance = this.balance + amount;
        this.balance = newBalance;
}
```


Overloading

- Overloading
  - Una classe può avere più metodi con lo stesso nome ma con parametri differenti

```
public BankAccount(double initialBalance)
public BankAccount()

public void println(String s)
public void println(double a)
```


Chiamata di costruttori

- Un costruttore può invocare un altro costruttore della stessa classe

```
class BankAccount {

   public BankAccount(double initialBalance) {
      balance = initialBalance;
   }

   public BankAccount() {
      this(0);
   }
}
```


Tipi di metodi

- Un metodo accessore non modifica lo stato del suo parametro implicito (getter)

```
public void getName() {…}
```

- Un metodo modificatore lo può modificare (setter)

```
public void setName(String name) {…}
```

- In Java i numeri e i riferimenti a oggetto sono passati per ??????

Tipi di metodi

- Un metodo accessore non modifica lo stato del suo parametro implicito (getter)

```
public String getName() {…}
```

- Un metodo modificatore lo può modificare (setter)

```
public void setName(String name) {…}
```

- In Java i numeri e i riferimenti a oggetto sono passati per valore (pass-by-value)

Metodi statici

- Un metodo statico (`static`) non ha parametro implicito

```
class Numeric {
   public static boolean approxEqual(double x,
                                     double y) {
     . . .
   }
}

double r = Math.sqrt(2);
if(Numeric.approxEqual(r*r,2))
```


Variabili statiche

- Una variabile statica non appartiene ad alcun oggetto della classe, ma alla classe stessa
  - main

```
public class BankAccount {
    . . .
    private double balance;
    private int accountNumber;
    private static int lastAssignNumber = 0;
}
```

singola inizializzazione

Ciascun oggetto `BankAccount` ha il proprio saldo `balance` e la propria variabile `accountNumber`, però esiste una copia unica della variabile `lastAssignedNumber`

Variabili statiche

- Incremento della variabile statica nel costruttore

```
public class BankAccount {
   public BankAccount() {
      accountNumber = lastAssignNumber++;
    }
  ...
}
```


Documentazione

- In Java esiste una forma standard per i commenti di documentazione

- Se si usa tale forma nelle classi, si può usare `javadoc` per generare una documentazione in formato HTML

- Un commento
  - /* … */ tradizionale
  - // … singola riga
  - /** … */di documentazione

Documentazione

Tag	Descrizione	Si applica a
@see	**Nome di classe collegata**	**Classe, metodo, campo**
@author	**Nome autore**	**Classe**
@version	**Versione**	**Classe**
@param	**Nome e descrizione parametro**	**Metodo**
@return	**Descrizione del valore di ritorno**	**Metodo**
@exception	**Nome e descrizione eccezione**	**Metodo**

Esempio

```
/** Questo è un esempio di commento di
documentazione per una classe "Automobile"
*   @see Ruota
*   @see Carrozzeria
*   @author Angelo
*   @version 1.0
*/
```


Javadoc

- Da una **shell** di comandi

```
javadoc MyClass.java
```

oppure

```
javadoc *.java
```

- produce un file in formato HTML