Java

Giacomo Cabri

24 Ottobre 2003

Seminario di
Principi di Sistemi Operativi

La programmazione ad oggetti

- **Nata per superare le limitazioni della programmazione modulare**
- **Più adatta allo sviluppo di sistemi software complessi**
- **Fornisce come astrazioni base:**
  - **la possibilità di definire classi**
  - **di mettere in relazione le classi tramite ereditarietà**
  - **di istanziare oggetti a partire dalla classe**

Il concetto di classe

- **Una classe descrive le proprietà di un insieme di "oggetti" aventi:**
  - **la medesima struttura interna (attributi)**
  - **lo stesso protocollo di accesso (insieme di metodi, o interfaccia)**
  - **lo stesso comportamento**

Il concetto di classe (2)

La classe riunisce le proprietà di:

- **componente software**
  - **può essere dotato di suoi propri dati e operazioni**
- **tipi di dato astratto (ADT)**
  - **funge da "stampo" per creare nuove istanze di oggetti**
- **modulo e centro di servizi**
  - **riunisce dati e relative operazioni (servizi)**
  - **fornisce meccanismi di protezione**

Caratteristiche della programma OO

- **Incapsulamento**
- **Ereditarietà**
- **Polimorfismo**

- **Introspezione**

Storia di Java

- **Java (in origine OAK) nasce nei laboratori della SUN, per opera di James Gosling, come linguaggio per Personal Digital Assistant**
- **Sintatticamente deriva dal C++**
  - **ma con importanti differenze semantiche**

Caratteristiche

- **Imperativo (come il C, il C++, il Pascal, a differenza del Lisp o del Prolog)**
- **Fortemente tipizzato (a differenza del C)**
- **Ad oggetti (simile al C++)**
- **Interpretato (richiede un interprete per essere eseguito)**
- **Indipendente dalla piattaforma (portabile)**

Gestione della memoria

- **Per la gestione della memoria, Java segue la filosofia Smalltalk**
  - **allocazione esplicita (parola chiave new)**
  - **deallocazione gestita dal garbage collector**

Ambiente di sviluppo

- **JDK (a linea di comando)**
  - **Compilatore (javac)**
    - **Produce codice intermedio: bytecode: file .class**
  - **Interprete JVM (java)**
  - **Debugger (jdb)**
  - **Visualizzatore di applet (appletviewer)**
- **Ambienti integrati**
  - **Borland Jbuilder**
  - **Microsoft J++**
  - **IBM VisualAge**
  - **…**

Librerie

- **I/O**
- **Interfaccia grafica (AWT e Swing)**
- **Networking (socket, RMI)**
- **Multi-programmazione (thread)**
- **Database (sql)**
- **…**

Classi e Istanze

- **Tipi Primitivi (semantica per valore)**
  - **byte, short, int, long, float, double, char (16 bit-Unicode), boolean**
- **Tipi Riferimento**
  - **Oggetto istanza di una classe (puntatori del C)**

Tipi primitivi

- **byte** **8 bit**
- **short** **16 bit**
- **int** **32 bit**
- **long** **64 bit**
- **float** **singola precisione (IEEE 754)**
- **double** **doppia precisione (IEEE 754)**
- **char** **carattere (16 bit Unicode)**
- **boolean** **true o false**

Classe

- **Classe**
  - **Insieme di variabili (stato) e metodi (interfaccia)**
  - **Tipo di dato astratto, template che definisce i metodi (operazioni) e le variabili comuni a tutti gli oggetti di un certo tipo**
- **Classi Wrapper**
  - **Classi "contenitore" per uniformare i tipi di dati primitivi rispetto alle classi: Byte, Short, Integer, …**
  - **Esempio: inserire un intero in un array di oggetti**

Esempio di classe e oggetto

```
class Veicolo
{
  private int VelocitaMassima;
  private int NumeroPosti;

  public Veicolo(int VM, int NP)  /* costruttore */
  {
     VelocitaMassima = VM;
     NumeroPosti = NP;
  }
}
```

Esempio di classe e oggetto (2)

```
public class Esempio1
{
  public static void main(String args[])
  {
    Veicolo MiaMacchina;
    /* variabile con semantica per riferimento
       un oggetto di classe Veicolo
       viene instanziato e inizializzato */
    MiaMacchina = new Veicolo(150, 5);
    System.out.println("Creato un oggetto di classe
     Veicolo");
  }
}
```


Risultato di esecuzione

- **Compilazione, esecuzione e risultato (Unix)**

```
polaris$ javac Esempio1.java
polaris$ java Esempio1
Creato un oggetto di classe Veicolo
polaris$
```

- **Ma anche (Prompt di DOS di Windows)**

```
C:\JavaProg> javac Esempio1.java
C:\JavaProg> java Esempio1
Creato un oggetto di classe Veicolo
C:\JavaProg>
```

Il main e gli argomenti

- **Il main è il primo metodo che viene eseguito**
- **Deve obbligatoriamente avere questa signature**

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

- **`args` è un array di stringhe, che contiene tutti i parametri passati sulla linea di comando**

Esempio di main

```
public static void main(String args[]){
      if (args.length == 0)
      System.out.println("Occorrono argomenti");
      else
      for(int i=0; i<args.length; i++)
      System.out.println("arg" + (i+1) +
                                 ": " + args[i]);
      }
```

Costruttori

- **Metodi con lo stesso nome della classe**
- **Più di uno se con parametri diversi**
- **Non hanno valore di ritorno**
- **Inizializzano gli oggetti creati**
- **Vengono chiamati automaticamente quando si istanzia un nuovo oggetto**

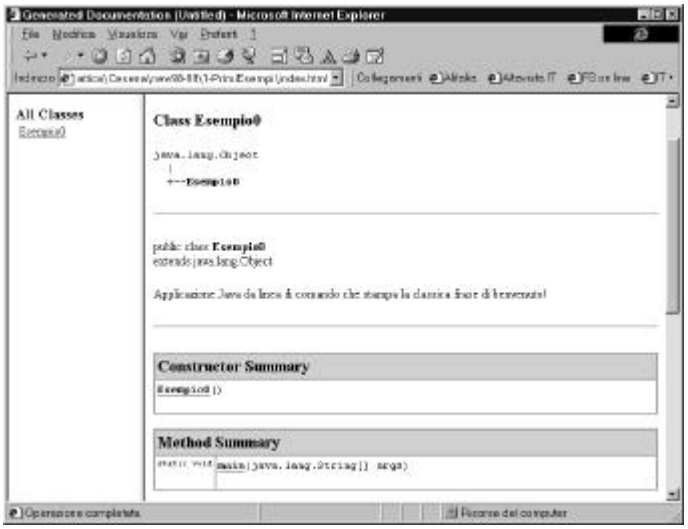
Esempi di costruttori

```
public Veicolo(int VM, int NP)  /* costruttore */
  {
     VelocitaMassima = VM;
     NumeroPosti = NP;
  }
public Veicolo()  /* costruttore con valori di
    default */
  {
     VelocitaMassima = 50;
     NumeroPosti = 4;
  }
```

Commenti

- **Tre tipi di commenti**
  - **Su più righe, delimitati da ‘/*’ e ‘*/’ (alla C)**
  - **Su una riga, iniziati da ‘//’ (alla C++)**
  - **Su più righe, usati per la documentazione, delimitati da ‘/**’ e */**
- **Gli ultimi possono essere usati per creare documentazione HTML tramite il programma javadoc (fornito con il JDK)**

Esempio di documentazione



Package

- **Insieme di più classi**
- **Riferimento alla directory di un file system**
  - **Il Package**
    - **it.unimo.dsi.agentgroup.application1**
  - **Identifica la sottodirectory**
    - **it\unimo\dsi\agentgroup\application1**

Package 2

- **Definizione della classe**

```
package myPackage;
public class myClass { … }
```

- **Riferimento alla classe**

```
myPackage.myClass myVar= new myPackage.myClass();
```

Oppure

```
import myPackage.*;
myClass myVar= new myClass();
```

Il file corrispondente alla classe: myClass.class viene cercato in:
{CLASSPATH}\myPackage\myClass.class

Attenzione!

- **La maggior parte dei problemi in compilazione/esecuzione è relativa all'uso dei package**

Incapsulamento

- **In una classe possono essere dichiarati:**
  - **variabili di qualsiasi tipo (primitivo o riferimento)**
  - **metodi (funzioni)**
  - **altre classi, chiamate inner class**
- **I metodi devono essere definiti all'interno della classe che li dichiara (a differenza del C++)**

Visibilità

- **Le variabili, e metodi possono avere uno dei seguenti attributi di visibilità**
  - **private: non visibile all'esterno**
  - **public: visibile da tutti**
  - **protected: visibile solo dalle classi figlie**
  - **(friendly): visibile dalle classi discendenti dello stesso package**
- **Le classi solo public o (friendly)**

Esempio di incapsulamento

```
class Veicolo
{
  private int VelocitaMassima;  // variabili private
  private int NumeroPosti;      // semantica per
    valore

  public Veicolo(int VM, int NP)  // costruttore
  {
    VelocitaMassima = VM;
    NumeroPosti = NP;
  }
```

Esempio di incapsulamento (2)

```
 public int getVelocitaMax()  // metodi pubblici
  {
    return VelocitaMassima;
  }

  public int getNumeroPosti()
  {
    return NumeroPosti;
  }
}
```


Esempio di incapsulamento (3)

```
public class Esempio2
{
public static void main(String args[])
  {
    int Intero;
    Veicolo MiaMacchina;

    MiaMacchina = new Veicolo(150, 5);
    System.out.print("La mia macchina ha ");
    System.out.print(MiaMacchina.getNumeroPosti()+"
    posti");
    System.out.print(" e raggiunge la velocita' di
    ");
    System.out.println(MiaMacchina.getVelocitaMax() +
    " km/h.");
  }}
```

Esempio di incapsulamento (4)

- **Se avessi scritto**

```
Intero = MiaMacchina.NumeroPosti;
```

il compilatore mi avrebbe dato un errore

Variabili e metodi di classe

- **Parola chiave static**
- **public static void main(String[ ] args)**

Ereditarietà

- In Java una classe può derivare da un'altra classe
  - Ereditarietà singola
  - Parola chiave: extends
- La classe figlia eredita tutti i metodi e le variabili *visibili* che non vengono ridefiniti
- Per quelli ridefiniti, accessibilità tramite
  - super
- Grazie alla ereditarietà si possono riutilizzare componenti già definiti, specializzandoli

La gerarchia

- Viene definita una classe radice che raggruppa al suo interno le definizioni comuni ad ogni oggetto e dalla quale hanno origine tutte le classi
  - classe Object

This e super

- `this` rappresenta l'istanza corrente
- Viene usato:
  - in un costruttore, per richiamare un altro costruttore della stessa classe
  - per indicare un campo dati in caso di omonimia con un parametro o con una variabile locale
- `super` rappresenta la classe padre
- Viene usato:
  - in un costruttore, per richiamare il costruttore della classe padre
  - per indicare una variabile o un metodo della classe padre in caso di omonimia

Esempio di uso di this

```
public Veicolo()
/* costruttore con valori di default */
  {
     this(50, 4);
  }


public void setVal(int val)
  {
    this.val = val;
  }
```

Esempio di ereditarietà

```
class Veicolo /* extends Object */
{
  private int NumeroPosti;
  public Veicolo(int NP) // costruttore
  {  NumeroPosti = NP;  }
  public int getNumeroPosti()
  {  return NumeroPosti; }
}
```


Esempio di ereditarietà (2)

```
class VeicoloTerrestre extends Veicolo
{
  private int NumeroRuote;

  public VeicoloTerrestre(int NP, int NR) /*
    costruttore */
  {
    super(NP); /* chiama il costruttore del padre */
    NumeroRuote = NR;
  }

  public int getNumeroRuote()
  {    return NumeroRuote;  }
}
```

Esempio di ereditarietà (3)

```
class VeicoloMarino extends Veicolo
{
  private long Stazza;

  public VeicoloMarino(int NP, long S)
  {
    super(NP);
    Stazza = S;
  }
  public long getStazza()
  {
    return Stazza;
  }
}
```


Esempio di ereditarietà (4)

```
public class Esempio3
{

 public static void main(String args[])
 {
  VeicoloTerrestre MiaMacchina;
  VeicoloMarino MiaNave;

  MiaMacchina = new VeicoloTerrestre(5, 4);
  System.out.print("La mia macchina ha ");
  System.out.print(MiaMacchina.getNumeroPosti() + "
    posti e ");
  System.out.println(MiaMacchina.getNumeroRuote() + "
    ruote");
```

Esempio di ereditarietà (5)

```
  MiaNave = new VeicoloMarino(10, 10);
  System.out.print("La mia nave ha ");
  System.out.print(MiaNave.getNumeroPosti() + " posti
    e ");
  System.out.println("una stazza di " +
    MiaNave.getStazza());
 }
}
```


Risultato dell'esecuzione

```
sirio$ java Esempio3
La mia macchina ha 5 posti e 4 ruote
La mia nave ha 10 posti e una stazza di 10
sirio$
```

Classi Astratte

- **Classe che non implementa tutti i metodi che definisce**
- **Parola chiave** abstract

Interfaccia

- **Insieme di definizione di metodi (manca l'implementazione)**
- **Parola chiave interface**
- **Ereditarietà multipla per le interfacce**
  - **Non ci sono conflitti**
  - **Parola chiave implements**

Interfaccia vs. classe

- **In C++ è possibile l'ereditarietà multipla**
  - **una classe può derivare da più classi distinte**
- **In Java non esiste l'ereditarietà multipla**
  - **una classe può derivare da una sola classe**

MA

- **In Java una classe deriva al più da un'altra classe ma può implementare zero o più interfacce**
- **Parola chiave: implements**

Esempio

```
abstract class Veichle
{
 private int NumeroPosti;
 public int getNumeroPosti();
}

interface avviabile
{
  void avvia();
  void ferma();
}
```

Esempio (2)

```
class Veicolo extends Veichle implements avviabile
{
  public Veicolo(int NP) // costruttore
  {  super(NP);  }

  public int getNumeroPosti()
  {  return NumeroPosti; }

  public void avvia() // metodi derivati
    dall'interfaccia
  {  /* avvia il motore del veicolo */ }

  public void ferma()
  {  /* ferma il motore del veicolo */ }
}
```


Esempio (3)

```
class Timer implements avviabile
{
  private int status;
  public Timer();
  { status = 0; }
  public int getStatus()
  { return status; }

  public void avvia() // metodi derivati
    dall'interfaccia
  {  /* avvia il timer interno */ }

  public void ferma()
  {   /* ferma il timer interno */ }
}
```

Ereditarietà singola

- **L'eredità singola è meno potente?**
- **Sì**

MA

- **Riduce la complessità in fase di compilazione**
- **Evita equivoci sull'eredità di variabili/metodi**
- **Quello che interessa è il comportamento, non l'implementazione o lo stato**

Polimorfismo

- **Il polimorfismo di Java si basa su:**
  - **Binding dinamico**
  - **Operation dispatching dinamico**
  - **Overloading e overriding**
  - **Possibilità di assegnare ad una variabile di un tipo/classe un'istanza di una qualsiasi classe discendente**

Polimorfismo (2)

- **A tempo di esecuzione verranno invocati i metodi della classe di cui l'oggetto è effettiva istanza**
- **A differenza di uno switch-case del C, non è necessario conoscere in anticipo tutti i tipi diversi da gestire**
- **NOTA:**
  - **il binding dinamico viene ottenuto in C++ solo tramite l'uso dei puntatori**
  - **anche in Java si usano puntatori, ma è il sistema che li gestisce in modo trasparente al programmatore**

Esempio di polimorfismo

```
class Veicolo // extends Object
{
  private int NumeroPosti;

  public Veicolo(int NP)  // costruttore
  {
    NumeroPosti = NP;
  }

  public int getNumeroPosti()
  { return NumeroPosti; }

  public String toString()
  { return "Veicolo con " + NumeroPosti + "posti"; };
}
```

Esempio di polimorfismo (2)

```
class Topolino extends Veicolo
{
  public Topolino(int NP) // costruttore
  { super(NP); }

  public String toString() //ridefinisce il metodo
  {
    return "Sono una Topolino e ho " +
    getNumeroPosti() +
" posti";
  }
}
```


Esempio di polimorfismo (3)

```
class SeicentoFamiliare extends Veicolo
{
  public SeicentoFamiliare(int NP) // costruttore
  { super(NP); }

  public String toString() //ridefinisce il metodo
  {
    return "Sono una SeicentoFamiliare e ho " +
    getNumeroPosti() + " posti";
  }
}
```

Esempio di polimorfismo (4)

```
public class Esempio
{
  public static void main(String args[])
  {
    Veicolo V;  // istanza della classe padre
    Topolino A1 = new Topolino(4);
    SeicentoFamiliare A2 = new SeicentoFamiliare(6);

    V = A1;
    System.out.println(V.toString());
    V = A2;
    System.out.println(V.toString());
  }
}
```


Risultato di esecuzione

```
sirio$ java Esempio4
Sono una Topolino e ho 4 posti
Sono una SeicentoFamiliare e ho 6 posti
sirio$
```

Array in Java

- In Java gli array sono degli oggetti ✍ vanno istanziati
- A differenza del C, è possibile definire il numero degli elementi a runtime
- Dichiarazione di un array di stringhe:
  - `String sa[];`
- Creazione dell'istanza:
  - `sa = new String[5];`
- Uso:
  - `sa[2] = "pippo";`
  - `System.out.println(sa[4]);`

Esempio di array e wrapper

```
import java.util.Date;

class Veicolo // extends Object
{
  // come prima
}

public class Esempio5
{
  public static void main(String args[])
  {
    Object pool = new Object[3]; // array di oggetti
```

Esempio di array e wrapper (2)

```
    Veicolo V = new Veicolo(4);
    Integer I = new Integer(10);
    Date D = new Date();

    pool[0] = I;
    pool[1] = D;
    pool[2] = V;
    for (int i = 0; i < pool.length ; i++)
      System.out.println(pool[i].toString());
  }
}
```


Risultato di esecuzione

```
sirio:~/java$ java Esempio5
10
Wed Jun 04 14:04:30 GMT+03:30 1997
Veicolo con 4 posti
sirio:~/java$
```

La gestione degli errori

- **In Java la gestione degli errori avviene attraverso *eccezioni***
- **Eccezione: evento che capita durante l'esecuzione e sconvolge il normale flusso di esecuzione**
- **Esistono classi per rappresentare le eccezioni**

Sollevare un'eccezione

- **Ogni metodo può sollevare una o più** eccezioni
  - **Costrutto** throws**: indica quali eccezioni può sollevare**
    - `nomeMetodo(…) throws tipoEccezione`
  - **Costrutto** throw**: solleva un'eccezione**
    - `throw new tipoEccezione()`

Catturare un'eccezione

- **Un'eccezione sollevata viene catturata e gestita**
  - **Costrutto try-catch**

```
try
{
... // blocco di istruzioni
}
catch (tipo_di_eccezione_da_catturare
   variabile_eccezione)
{
... // istruzioni da eseguire in caso di
   eccezione
}
```


Introspezione

- **L'introspezione Java si basa sulla** Reflection **API**
  - **Metaclassi che descrivono i concetti generali di:**
    - **Classi, metodi, etc.**
  - **Operazioni che permettono di accedere ai dati significativi per le metaclassi:**
    - **getClass(), getName()**

Esempio di reflection

```
import java.lang.reflect.*;

class SampleName {

  public static void main(String[] args) {
    Integer i = new Integer(5);
    printName(i);
  }

  static void printName(Object o) {
    Class c = o.getClass();
    String s = c.getName();
    System.out.println(s);
  }
}
```


Risultato di esecuzione

```
sirio$ java SampleName
java.lang.Integer
sirio$
```

Somiglianze tra Java e C++

- **Paradigma a oggetti**
  - **incapsulamento**
  - **ereditarietà**
  - **polimorfismo**
- **Sintassi molto simile (ad es. for, while, switch)**

Differenze tra Java e C++

Java	C++
Semplifica un linguaggio (il C++)	estende un linguaggio (il C)
Non esistono puntatori (solo semantica per riferimento)	uso dei puntatori
Gestione della memoria a carico del sistema (garbage collector)	gestione della memoria a carico del programmatore
Ereditarietà singola per le classi	ereditarietà multipla per le classi
Strettamente tipizzato	tipizzato in modo lasco
Solo class	struct, union e class
Portabilità del codice compilato	velocità di esecuzione

Risorse

- **Documentazione del JDK**
- **Sorgenti delle librerie**
- **URL**
  - **http://java.sun.com/docs/books/tutorial**
  - **http://developer.javasoft.com**
  - **http://www.mokabyte.it**
- **Libri**
  - **Java 1.2 Guida completa, Apogeo**
  - **Java 1.2 Tutto & oltre, Apogeo**
  - **Thinking in Java, B. Eckel, http://www.BruceEckel.com**