Lezione 9

Usare le Classi ed i Metodi

Riferimenti

Per le slide seguenti si è fatto riferimento alle seguenti fonti:

- Libro Deitel-Deitel. Fondamenti di programmazione Java
- M. Tarquini, A. Ligi. Java mattone dopo mattone. Edizione free sul web
  - http://www.java-net.it/jmonline/index.html
- Tutorial sun:
  - http://java.sun.com/docs/books/tutorial/index.html
- Documentazione della API Java:
  - http://java.sun.com/javase/6/docs/api/

Si applica a tutte queste slide se non diversamente specificato

Classi

Dichiarare una classe

- Ogni dichiarazione di classe inizia con la parola riservata class (in genere preceduta dal modificatore di accesso public (o private)

```
public class <Nomeclasse>
{
…
}
```

- Legenda
  - In inclinato le parti opzionali
  - <nome della classe (iniziale maiuscola), gli angolari non si scrivono nel programma>

```
public class CiaoMondo
{
   …
}
```

Dichiarare una classe

- Ogni classe deve essere memorizzata in un file con lo stesso nome della classe ed estensione .java.
- Se sono presenti più classi in un progetto esse vanno compilate separatamente.
- Se per esempio abbiamo una serie di file .java da compilare all'interno di una cartella (se li dobbiamo compilare tutti) possiamo digitare:

  javac *.java

- Dichiarare più di una classe public all'interno di un unico file provoca un errore di compilazione (a meno che non si tratti di *inner class* che non faremo)

Classi, attributi e metodi

All'interno di una classe si possono dichiarare i suoi attributi e metodi

```
public class <Nomeclasse>
{
  attributi
  metodi
}
```

Metodi

Dichiarare un metodo

- Dichiarazione di un metodo:

```
public tipo_ritorno <nome_metodo> ([tipo param])
{
   …
}
```

- La parola riservata public si chiama *modificatore di accesso*. Si può anche utilizzare *private* o *protected*.
  - La dichiarazione di un metodo inizia con la parola riservata public se vogliamo che il metodo sia accessibile da qualsiasi altra classe.

```
public void saluta ()
{
      …
}
```

Dichiarare un metodo

```
public void saluta ()
{
        …
}
```

- La parola riservata void specifica che il metodo non ritorna alcun valore al *metodo chiamante* ma semplicemente svolge un'azione.
  - In alternativa si può specificare il tipo di dato ritornato
  - Tipi primitivi: boolean, char, byte, short, int, long, float, double
- Per convenzione il nome di un metodo inizia sempre con la lettera minuscola, come le variabili, ma sono seguiti da parentesi.
- La prima parola nel nome di un metodo dovrebbe essere un verbo

Dichiarare una classe con un metodo

- Anche il corpo di un metodo è delimitato da parentesi graffe.
- Il corpo di un metodo contiene le istruzioni che ne implementano il compito.

```
public void saluta ()
{
    System.out.println("Ciao, mondo");
}
```

Dichiarare una classe con un metodo

```
public class CiaoMondo
{
  public void saluta()
  {
   System.out.println("Ciao, mondo");
  }
}
```

- La classe CiaoMondo non può essere usata così com'è.
- Perché?
  - Non possiede un metodo main, che è il metodo speciale che permette alla virtual machine di eseguire l'applicazione (la JVM invoca automaticamente solo il metodo main)

Dichiarare una classe con un metodo

- Ogni classe che contiene un metodo main è un'applicazione Java.
- La JVM usa main per iniziare l'esecuzione -> la classe CiaoMondo non è un'applicazione
- La classe CiaoMondo deve essere quindi usata all'interno di una applicazione
- Se istanziamo la classe con BlueJ non otterremo nessun risultato da essa (primoMetodo non viene eseguito)

Esempio di applicazione

L'intestazione del metodo main è sempre la stessa:

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

- La parola static dichiara un metodo statico e permette alla JVM di localizzare, invocare ed eseguire l'applicazione.
- I metodi statici possono essere chiamati senza richiedere la creazione di un oggetto nella classe in cui sono chiamati.
- Lo scopo di questa applicazione è invocare il metodo primoMetodo della classe CiaoMondo per fargli scrivere "Ciao, Mondo!".
- NON POSSIAMO
- chiamare un metodo dichiarato in un'altra classe fino a quando non abbiamo creato (istanziato) un oggetto di quella classe

Esempio di applicazione

L'intestazione del metodo main è sempre la stessa:

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

- Il metodo main non ritorna mai alcun valore quindi è sempre dichiarato void
- Talvolta è utile passare dei parametri all'applicazione. Questi sono sempre delle stringhe e quindi per il metodo main si dichiara il parametro String arg[] che è un vettore arg di tipo stringa
- In ogni applicazione c'è un solo main

Esempio di applicazione/2

- Proveremo adesso a riempire il corpo del metodo main
- All'interno del main di solito si instanziano le classi da cui inizia il comportamento del programma e si invocano i loro metodi

```
public static void main(String args[])
    {
        CiaoMondo ciao = new CiaoMondo();
        ciao.saluta();
    }
```

- Provare in Bluej la nuova versione del programma (CiaoMondo2)

CiaoMondo

```
public class CiaoMondo
{
   public void saluta ()
   {
      System.out.println("Ciao, mondo");
   }
   public static void main(String args[])
   {
      CiaoMondo ciao = new CiaoMondo();
      ciao.saluta();
   }
}
```

Esempio di applicazione/3

- La comprensione del corpo del metodo main richiede concetti non ancora affrontati.
- In particolare la istanziazione delle classi e la invocazione dei metodi
- Prima di affrontare questi concetti è conveniente studiarne degli altri.
- Si passerà quindi alla dichiarazione delle variabili.

Attributi

Gli attributi della classe

- Si è già visto che la classe può contenere attributi e metodi:

```
public class CiaoMondo
{
   attributi
   metodi
}
```

- Una variabile (o meglio attributo) si dichiara così:

```
private tipo var_name;
```

Gli attributi

```
private tipo var_name;
```

- E' buona norma dichiarare le variabili TUTTE private (e non public)
- Il tipo della variabile può essere un tipo primitivo o una classe (dichiarata nel programma o appartenente alle librerie del linguaggio)
- Il nome inizia, per convenzione, con un carattere minuscolo

Tipi primitivi

Tipo	Descrizione	Dimensione
int	Intero con intervallo -2147483648…2147483647	4 byte
byte	Descrive un singolo byte -128…127	1 byte
short	intero corto -32768…32767	2 byte
long	intero lungo 10	8 byte
double	tipo in virgola mobile a doppia precisione 10^{308}	8 byte
float	tipo in virgola mobile a singola precisione 10^{38}	4 byte
char	rappresenta i caratteri codificati secondo lo schema unicode	2 byte
boolean	tipo per i due valori logici true e false	1 bit

Esempi di dichiarazione di variabili

```
private int marcia;
private char inizialeNome;
private boolean studenteAvvisato;
```

Tipi non primitivi

- Il linguaggio Java è corredato di librerie che definiscono tipi di dati molto utili:
  - String, Integer, Array, List, Object
- Inoltre una volta definite delle classi nel proprio programma, si possono dichiarare variabili del tipo delle proprie classi

```
List unaLista;
Object unOggettoGenerico;
String cognome;
```

Le variabili reference (di riferenza)

- Java fa una netta distinzione tra Classi e tipi primitivi. Una delle maggiori differenze è che un oggetto non è allocato dal linguaggio al momento della dichiarazione. Per chiarire questo punto, consideriamo la seguente dichiarazione:

```
int counter;
```

- Questa dichiarazione crea una variabile intera chiamata counter ed alloca subito quattro byte per lo "storage" del dato. Con le classi lo scenario cambia e la dichiarazione

```
String s;
```

- crea una variabile che referenzia l'oggetto, ma non crea l'oggetto String. Una variabile di referenza, è quindi una variabile speciale che tiene traccia di istanze di tipi non primitivi. Questo tipo di variabili hanno l'unica capacità di tracciare oggetti del tipo compatibile: ad esempio una referenza ad un oggetto di tipo Stack non può tracciare oggetti di diverso tipo.

Le variabili reference (riferimento)

- Oltre che per gli oggetti, Java utilizza lo stesso meccanismo per gli array, che non sono allocati al momento della dichiarazione, ma viene semplicemente creata una variabile per referenziare l'entità. Un array può essere dichiarato utilizzando la sintassi:

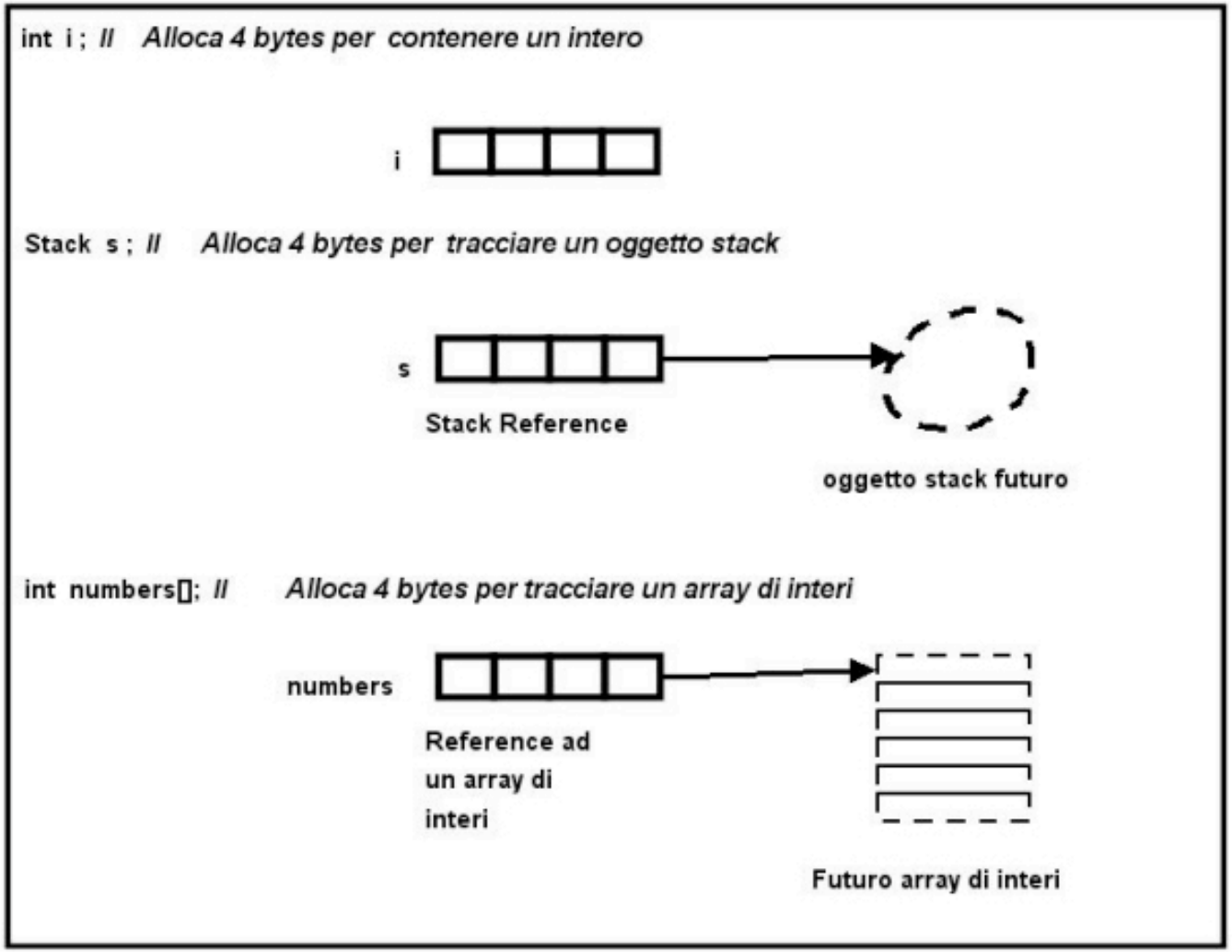
```
int numbers[];
```

- Una dichiarazione così fatta crea una variabile che tiene traccia di un array di interi di dimensione arbitraria.

Variabili reference

- La variabile reference allora non ***è*** l'oggetto dichiarato ma ***punta*** ad esso (contiene l'indirizzo in cui esso è memorizzato).
- In pratica le variabili reference sono quindi concettualmente simili ai puntatori in C e C++.
- Attenzione:
  - Le variabili reference non consentono le conversioni intero/indirizzo e le operazioni matematiche che si possono fare sui puntatori in C/C++

Variabili primitive e reference



L'oggetto null

- Il linguaggio Java prevede un valore speciale per le variabili reference che non referenzia nessuna istanza di un oggetto. Il valore speciale null rappresenta un oggetto inesistente, e viene assegnato di default ad ogni variabile reference.

```
String s = null;
```

- Quando ad una variabile reference viene assegnato il valore null, l'oggetto referenziato verrà rilasciato e, se non utilizzato verrà dato in pasto alla garbage collection che si occuperà di rilasciare la memoria allocata per la entità.
- Altro uso che può essere fatto dell'oggetto null riguarda le operazioni di comparazione come visibile nell'esempio seguente. L'istruzione if controlla se l'oggetto array sia stato instanziato o no.

```
int numbers[];
if (numbers == null)
{
      ……..
}
```

Istanziare gli oggetti

Istanziare gli oggetti

- Creata la variabile refence, siamo pronti a creare una istanza di un nuovo oggetto o di un array.
- L'operatore new istanzia un oggetto allocando la memoria necessaria e tornando la locazione in memoria della entità creata. Questa locazione può quindi essere memorizzata nella variabile reference di tipo appropriato ed utilizzata per accedere all'oggetto quando necessario.
- La sintassi è la seguente:

```
new tipo_classe();
```

- Le parentesi sono necessarie ed hanno un significato particolare che sveleremo presto.
- Tipo_classe è una classe appartenente alle API di Java oppure definita dal programmatore.
- Esempio:

```
Stack s = new  Stack();
```

Istanziare oggetti

L'istruzione:

```
Stack s = new  Stack();
```

E' equivalente a:

```
Stack s;
s = new Stack();
```

Ed alla seguente:

```
Stack s = null;
s = new Stack();
```

Istanziare oggetti

- Un array si istanzia in modo simile:

```
int my_array[] = new int[20];
```

- o, analogamente al caso precedente:

```
int my_array[] = null;
my_array = new int[20];
```

Le stringhe

- Le stringhe sono oggetti che possono essere inizializzati usando una notazione con doppi apici senza l'utilizzo dell'operatore new:

```
String prima = "Ciao";
String seconda = "mondo";
```

- Possono essere concatenate usando l'operatore di addizione:

```
String terza = prima + seconda;
```

- Hanno un membro che ritorna la lunghezza della stringa rappresentata:

```
int lunghezza = prima.lenght();
```

La spiegazione dell'operatore punto avverrà nella prossima slide

L'operatore punto "."

- Questo operatore è utilizzato per accedere ai membri di un oggetto tramite la variabile reference.
- Riprendendo l'esempio fatto con le stringhe:

```
int lunghezza = prima.lenght();
```

- L'operatore punto è adoperato per invocare (eseguire) il metodo *lenght* dell'oggetto *prima*.
- Ovviamente è possibile invocare *lenght* perché la classe String di cui *prima* è una instanza contiene la dichiarazione di questo metodo

Esempio di applicazione completo

- A questo punto è possible comprendere gli ultimi dettagli della classe CiaoMondo. In particolare il main elencava:

```
public static void main(String args[])
    {
        CiaoMondo ciao = new CiaoMondo();
        ciao.primoMetodo();
    }
```

- Come è ora noto la prima istruzione:
  - Dichiara la variabile(oggetto) *ciao* di tipo *CiaoMondo*
  - inizializza l'oggetto (istanzia la classe cioè crea l'oggetto *ciao* di tipo *CiaoMondo*)

Esempio di applicazione completo

```
public static void main(String args[])
    {
        CiaoMondo ciao = new CiaoMondo();
        ciao.saluta();
    }
```

- La seconda istruzione:
  - Invoca (esegue) il metodo saluta dell'oggetto *ciao*

Esempio di applicazione completo

- Compilare ed eseguire la classe CiaoMondo con BlueJ:

```
public class CiaoMondo
{
    public void saluta ()
    {
        System.out.println("Ciao, mondo");
    }
    public static void main(String args[])
    {
        CiaoMondo ciao = new CiaoMondo();
        ciao.saluta();
    }
}
```

L'oggetto System

- Un'altra delle classi predefinite in Java è la classe System. Questa classe ha una serie di metodi statici e rappresenta il sistema su cui la applicazione Java sta girando.
- Due metodi statici di questa classe sono System.out e System.err che rappresentano rispettivamente lo standard output e lo standard error dell'interprete java. Usando il loro metodo statico println(), una applicazione Java è in grado di inviare un output sullo standard output o sullo standard error.

```
System.out.println("Scrivo sullo standard output");
System.err.println("Scrivo sullo standard error");
```

- Il metodo statico `System.exit(int number)` causa la terminazione della applicazione Java.

Variabili nell'applicazione esempio

- Per esemplificare l'uso delle variabili nella applicazione esempio si consideri il seguente programma:

```
public class CiaoMondo3
{
    String saluto = "Ciao, mondo";

    public void saluta ()
    {
        System.out.println(saluto);
    }
    public static void main(String args[])
    {
        CiaoMondo3 ciao = new CiaoMondo3();
        ciao.saluta();
    }
}
```

Metodi: parametri e variabili locali

Un metodo con un parametro

- A volte sono necessarie informazioni aggiuntive che permettono ad una classe (attraverso i suoi metodi) di portare a termine una operazione.
- Queste informazioni si chiamano parametri.
- Un metodo può richiedere più di un parametro
  - i valori (ad esempio numerici) che si danno ai parametri nella chiamata di un metodo si chiamano argomenti.
- Ogni parametro deve sempre specificare un tipo ed un identificatore.
- Il numero di argomenti nella chiamata deve corrispondere sempre al numero di parametri nella lista dei par. nella dichiarazione del metodo **e al loro ordine**

Dichiarazione di metodo con parametri

```
public tipo_ritorno <nome_metodo> ([tipo param])
{
   …
}
```

Parametri dei metodi nell'esempio

- L'esempio seguente utilizza il passaggio di parametri al metodo *saluta* per definire il testo da visualizzare

```
public class CiaoMondo4
{
    public void saluta (String s)
    {
        System.out.println(s);
    }
    public static void main(String args[])
    {
        CiaoMondo4 ciao = new CiaoMondo4();
        ciao.saluta("ciao, mondo");
    }
}
```

Variabili nei metodi

- Le variabili dichiarate all'interno del corpo di un metodo sono chiamate variabili locali e sono valide solo all'interno del metodo.
- Di solito una classe contiene uno o più metodi che manipolano gli attributi di un suo specifico oggetto.

Variabili nei metodi

- Quando ogni oggetto instanziato da una classe mantiene la propria copia di un attributo questa si chiama variabile istanza.
- La maggior parte delle dichiarazioni di variabile di istanza sono precedute dalla parole chiave private.
- private come public è un modificatore di accesso, le variabili dichiarate private possono essere "accedute" e quindi modificate solo dai metodi della classe in cui sono dichiarate.
- In genere le variabili di istanza vengono dichiarate private mentre i metodi public.

Variabili nei metodi

- Quanto detto precedentemente richiama il concetto di incapsulamento:
  - esso prevede l'occultamento dei dati di un oggetto fornendo i metodi per accedervi.
- Le variabili istanza vengono usate dagli oggetti per memorizzare il loro stato, cioè i dati di cui necessita per eseguire i suoi metodi.

Costruttori

I costruttori

- Per inizializzare gli oggetti si usano metodi particolari che si chiamano costruttori , essi contengono le istruzioni per la inizializzazione.
- Un costruttore ha sempre il nome uguale a quello della classe.
- I costruttori ed i metodi pubblici di una classe costituiscono la sua interfaccia.

I costruttori

```
public <NomeClasse> (tipoParam nomeParam
[, tipoParam nomeParam])
{
  …
}
```

Legenda:
- In inclinato le parti opzionali
- [contiene parti ripetibili n volte]

- I costruttori
  - hanno lo stesso nome della classe
  - Non hanno tipo di ritorno
  - Possono avere uno o più parametri
- In una classe ci possono essere uno o più costruttori (con parametri diversi)

I costruttori

- Esempio di costruttore per la classe CiaoMondo5:

```
public class CiaoMondo5
{
  public CiaoMondo5()
     {
        saluta("ciao mondo!");
     }
...
}
```

Applicazione esempio con costruttore

- Compilare ed eseguire

```
public class CiaoMondo5
{
    String s;
    public CiaoMondo5()
    {
        s="ciao mondo!";
        saluta(s);
    }
    public void saluta (String s)
    {
        System.out.println(s);
    }
    public static void main(String args[])
    {
        CiaoMondo5 ciao = new CiaoMondo5();
    }
}
```

Le dichiarazione delle classi

Dopo aver studiato i costruttori si può estendere la sintassi per la dichiarazione delle classi che si è già studiata:

```
public class <Nomeclasse>
{
  attributi
  costruttori
  metodi
}
```

Altro esempio con costruttori

```
public class CiaoMondo6
{
    String s;
    public CiaoMondo6()
    {
        s="ciao mondo!";
        saluta(s);
    }
    public CiaoMondo6(int vers)
    {
        s="ciao mondo!";
        saluta(s+" Versione "+vers);
    }

    public void saluta (String s)
    {
        System.out.println(s);
    }
    public static void main(String args[])
    {
        CiaoMondo6 ciao = new CiaoMondo6();
        CiaoMondo6 ciao1 = new CiaoMondo6(6);
    }
}
```

Costanti

Costanti

- Le costanti numeriche sono valori che non vengono modificati ed hanno un significato ben preciso all'interno dell'elaborazione.
- In Java le costanti sono identificate dalla parola chiave final; dopo un'assegnazione di valori ad una costante questa non può essere più modificata.
- Per un metodo:

```
final nomeTipo nomeVariabile = valore;
```

- per una classe:

```
specificatoreDiAccesso static final
  nomeTipo nomeVariabile  = espressione;
```