Università degli Studi di Palermo

Corso di Laurea
Informatica

La Programmazione Orientata agli Oggetti

Ing. Massimo Cossentino

Ingegneria del Software

a.a. 2012/2013

Testo di riferimento

- Testi di riferimento e approfondimento
  - Bruce Eckel. Thinking in Java 3rd edition
    - Scaricabile gratuitamente da: http://www.mindview.net/Books/TIJ/
  - Oracle. Learning the Java Language.
    - http://docs.oracle.com/javase/tutorial/java/index.html

Le slide relative alla programmazione object-oriented sono ispirate dai due testi sopra descritti e da:

Harvey M. Deitel, Paul J. Deitel. **Programmazione Java Fondamenti**. Ed. Pearson-Prentice Hall.

Come studiare l'OOP

- Dalle slide
- Dagli appunti presi a lezione
- Consultando i testi consigliati per approfondimenti e/o chiarire quanto non sia chiaro dagli appunti

- Esercizi di uso del linguaggio Java non sono necessari (ma possono aiutare la comprensione)
  - Non verranno proposti in questo corso perché fuori dagli obiettivi posti

Cosa è Java

- E' un linguaggio di programmazione che si adatta bene alla realizzazione di sistemi funzionanti in internet
- E' orientato agli oggetti
- E' un linguaggio per piattaforme differenti; i programmi possono essere progettati per funzionare allo stesso modo su Windows, Mac, Linux, Solaris, etc.
- Viene utilizzato anche su telefoni di ultima generazione e altri dispositivi di vario tipo

Cosa è Java

- Programmazione nel web:
  - Java può essere eseguito facilmente sulle pagine web
  - I browser possono prelevare un programma java incluso in una pagina web ed eseguirlo localmente sul sistema dell'utente
  - Questi programmi si chiamano APPLET; appaiono come immagini e possono essere interattivi ed usati per creare giochi, grafici, effetti multimediali etc.

Java è orientato agli oggetti

- La programmazione orientata agli oggetti (OOP) è un modo per concettualizzare un programma per computer come insieme di oggetti che interagiscono tra loro.
- Un oggetto contiene le informazioni e i metodi per accedervi
- Java eredita i suoi concetti da c++ e prende spunto anche da altri linguaggi object oriented
- Somiglianze con c++ molto forti, ma:
  - In java non ci sono i puntatori o aritmetica dei puntatori
  - Le stringhe e gli array sono oggetti all'interno di java
  - In Java (quasi) tutto è un oggetto

Compilazione di programmi Java

- Java è stato creato per essere eseguito su più dispositivi e più piattaforme -> funziona nello stesso modo in qualsiasi ambiente.
- I Programmi Java sono compilati in codice macchina per una *macchina virtuale*. Questo codice macchina è chiamato *bytecode* e la macchina virtuale lo interpreta convertendolo nel codice macchina specifico del processore.
- La macchina virtuale è più comunemente nota come interprete Java ed ogni ambiente che supporta Java deve avere un interprete progettato appositamente per il proprio sistema operativo e per il processore.

Cosa sono oggetti e classi

ICAR CNR

Concetti fondamentali : gli oggetti

- Ovunque guardiamo vediamo oggetti: persone, animali, piante, automobili, computer, etc.
- Le persone pensano in termini di oggetti (o di categorie di oggetti)
- I programmi per computer (ed i programmi Java) possono essere visti come oggetti, costituiti a loro volta da altri oggetti software interconnessi.
- Gli oggetti (nel mondo reale) si dividono in due categorie: animati ed inanimati.

Concetti fondamentali : gli oggetti

- Gli oggetti animati sono vivi, si muovono, quello inanimati sembrano non far nulla
- Tutti gli oggetti hanno qualcosa in comune:
- Possiedono *attributi* (come dimensione, forma, peso, colore) e tutti mostrano un *comportamento* (es. la palla rimbalza, il bambino piange, la radio suona…).
- Al livello software è importante studiare i tipi di attributi e i comportamenti che gli oggetti software mostrano
- Gli uomini imparano a conoscere gli oggetti studiando/ osservando i loro attributi ed il loro comportamento.

Cosa è un oggetto?

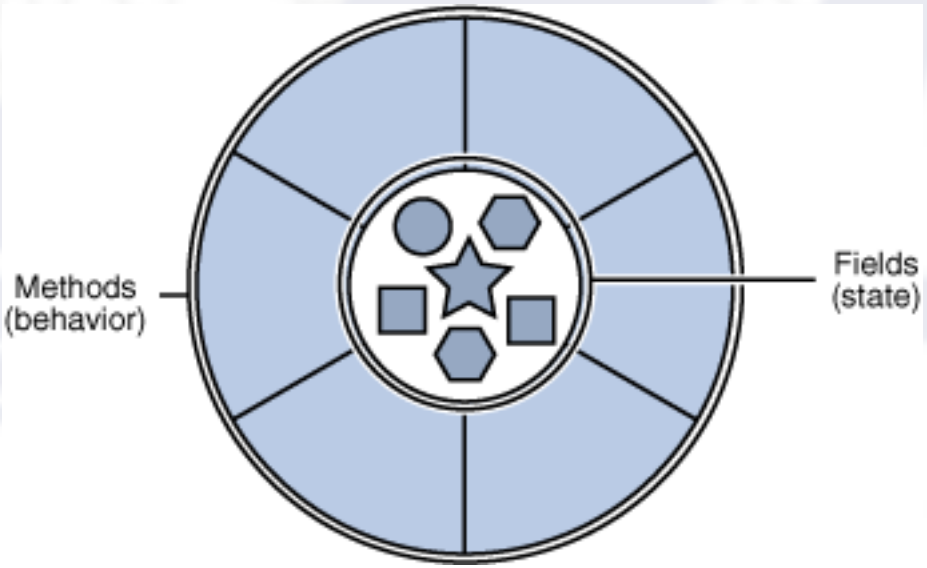
- Gli oggetti del mondo reale condividono due caratteristiche:
  - uno stato
  - Un comportamento
- Un cane ad esempio:
  - Lo stato è identificato dai valori dati a: nome, colore, razza, …
  - Il comportamento potrebbe essere uno dei seguenti: abbaia, scodinzola, odora, …
- Per una bicicletta:
  - Stato, valori dati a: marcia ingranata, cadenza della pedalata, velocità attuale
  - Comportamento: cambia marcia, cambia cadenza di pedalata, frena

Alcune di queste slide sono ispirate dal tutorial sun: http://java.sun.com/docs/books/tutorial/java/index.html

Cosa è un oggetto?

In altre parole in ciascun oggetto che ci circonda possiamo trovare:

- Degli *attributi* a cui assegnare dei valori (che determinano lo stato)
- Dei comportamenti, descritti da *metodi*.



Cosa è un oggetto?

Gli oggetti software sono concettualmente simili agli oggetti del mondo reale: anch'essi hanno uno stato e un comportamento.

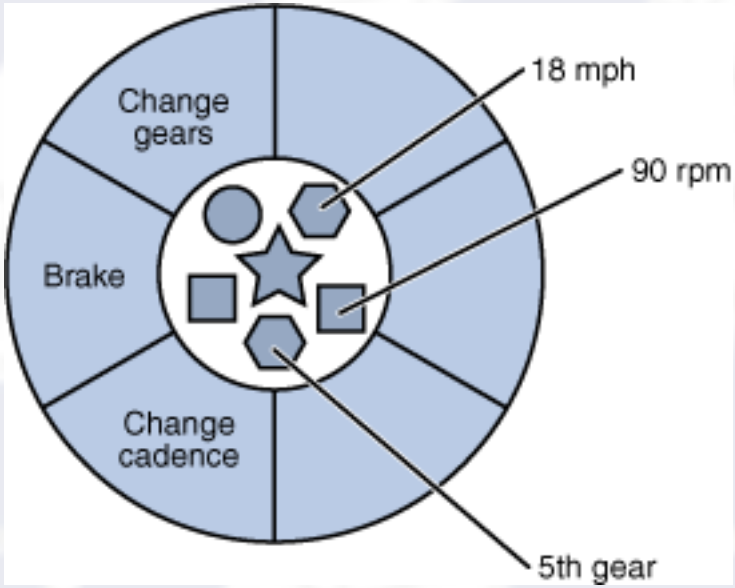
Un oggetto memorizza il suo stato in variabili dette anche attributi (*attribute*) o campi (field)

Il comportamento viene esposto tramite funzioni o metodi (*method*).

I metodi agiscono sullo stato dell'oggetto a cui appartengono e sono il meccanismo di comunicazione principale tra oggetti (invocazione di metodi)

Cosa è un oggetto?

- Esempio: bicicletta



Cosa è un oggetto?

Perché non mettere tutto il codice in un unico contenitore (oggetto, funzione, unità, modulo, …)?

- Modularità: il codice di un oggetto può essere scritto e manutenuto indipendentemente dal resto del codice
- Information-hiding: limitando l'interazione con i soli metodi di un oggetto i dettagli della sua struttura interna sono nascosti al mondo esterno
- Riuso: se un oggetto esiste già, può essere riusato in altre parti del programma o in un altro programma.
- Componibilità e debug: se un oggetto crea problemi può essere rimosso dalla applicazione e al suo posto se ne può mettere un altro che svolge le stesse funzioni (anche nel mondo reale funziona così: se il motore di un auto si rompe, si cambia il solo motore non l'intera auto)

Information hiding

- Un buono stile di programmazione prescrive l'utilizzo dell'incapsulamento dei dati (*data encapsulation*)
  - Lo stato dell'oggetto è interno (o privato) e può essere manipolato solo tramite un metodo
- Esempio: automobile
  - Attributi (che descrivono lo stato):motore_acceso, velocità, luci_accese
  - Metodi: accendi, accelera, frena, accendiLuci

Cosa è una classe?

- Nel mondo reale si trovano spesso parecchi oggetti dello stesso tipo.
- Per esempio esistono molte biciclette, tutte dello produttore e modello.
  - Ogni bicicletta contiene gli stessi componenti e nasce dallo stesso progetto.
- In termini object-oriented si dice che una certa bicicletta (la mia ad esempio) è una istanza della classe (*instance of class*) di oggetti di nome Bicicletta.
- La classe Bicicletta è il progetto da cui la mia bicicletta è stata creata.

Ecco un esempio di codice java (che verrà studiato in seguito):

```
class Bicycle {
        int cadenza = 0;
        int velocita = 0;
        int marcia = 1;
        void cambiaCadenza (int nuovoValore) {
             cadenza = nuovoValore;
        }
        void cambiaMarcia(int nuovoValore) {
             marcia = nuovoValore;
        }
        void accelera (int incremento) {
             velocita = velocita + incremento;
        }
        void applicaFreni(int decremento) {
             velocita = velocita - decremento;
        }
        void stampaStato() {
             System.out.println("cadenza:"+cadenza+" velocita:"+velocita+"
               marcia:"+marcia);
        }
}
```

La progettazione object-oriented

- Oggetti diversi possono avere attributi simili e mostrare comportamenti simili.
- La progettazione orientata agli oggetti modella il software in termini simili a quelli che gli uomini usano per descrivere gli oggetti del mondo reale.
  - Viene sfruttato il concetto di classe -> gli oggetti di una stessa classe hanno le stesse caratteristiche.
  - Il concetto di ereditarietà (anche multipla) con cui è possibile derivare nuove classi "assorbendo" le caratteristiche di classi già esistenti ed aggiungendone di nuove
    - Es: un oggetto della classe 'spider' ha le caratteristiche della classe più generale 'automobile' ma in più ha il tettuccio apribile

Concetti fondamentali : gli oggetti

- La progettazione orientata agli oggetti modella i componenti software proprio come vengono descritti gli oggetti del mondo reale, usando i loro attributi ed il loro comportamento.
- Gli oggetti nel mondo reale interagiscono tra di loro comunicando -> La OOP modella anche la comunicazione tra gli oggetti tramite messaggi.
- La OOP incapsula gli attributi e le funzionalità degli oggetti -> gli oggetti hanno la proprietà di nascondere le informazioni:
  - sebbene essi possano comunicare tra loro attraverso specifiche interfacce, non possono sapere come altri oggetti sono implementati.
  - I dettagli dell'implementazione sono nascosti all'interno degli oggetti stessi.

Concetti fondamentali : gli oggetti

- Nascondere le informazioni (Information Hiding) è cruciale nell'ambito della progettazione software.

- Nella programmazione procedurale (linguaggi tipo C, Pascal) la programmazione tende ad essere orientata all'azione-> l'unità di programmazione è la funzione , gruppi di azioni che svolgono qualche compito comune vengono trasformate in funzioni e queste ultime e loro volta raggruppate a formare programmi.
- In Java l'unità di programmazione è la classe da cui gli oggetti vengono istanziati (creati); le classi Java contengono metodi (che implementano delle funzionalità) e campi (che implementano attributi).

Concetti fondamentali : gli oggetti

- Creare un programma in Java significa creare delle classi, ogni classe contiene campi ed un insieme di metodi per manipolare questi campi per fornire determinati servizi ai clienti.
- Le classi vengono usate come mattoni per costruire nuove classi.
- Un software ad oggetti viene scritto come aggregati di classi, ogni 'aggregato' può essere riusato per creare altri programmi.
- Parola d'ordine RIUSO.

I primi comandi Java

Un semplice Programma in Java

```
// programma che scrive una riga di testo
public class Welcome {


  public static void main(String args[])
  {
    System.out.println(”Welcome to java programming");
  } //fine del metodo main
} //fine della classe Welcome
```

Un semplice Programma in Java

- I commenti:

  // commento a riga singola

  /* commento
  a riga multipla, può essere distribuito su più righe*/

  /** commento Javadoc */

  javadoc è una utility che legge i commenti in Javadoc e li utilizza per preparare automaticamente la documentazione di un programma in HTML.

  Dimenticare di chiudere i commenti può generare errori di sintassi che vengono rilevati in fase di compilazione

Un semplice Programma in Java

```
// programma che scrive una riga di testo
public class Welcome {


  public static void main(String args[])
  {
    System.out.println(”Welcome to java programming");
  } //fine del metodo main
} //fine della classe Welcome
```

Un semplice Programma in Java

public class Welcome {

- Rappresenta l'inizio di una dichiarazione di classe per la classe Welcome.
- Ogni programma Java consiste di almeno una dichiarazione di classe definita dal programmatore.
- La parola chiave class introduce una dichiarazione di classe ed è immediatamente seguita dal nome della classe.
- Per convenzione i nomi delle classi iniziano per lettera maiuscola (Java è sensibile alle maiuscole e minuscole)

Un semplice Programma in Java

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

- è il punto di partenza di ogni applicazione Java.
- il metodo main fa partire l'esecuzione di un'applicazione Java.
- Le parentesi dopo main indicano che main è un blocco di codice del programma detto metodo.
- Le dichiarazioni di classe Java contengono uno o più metodi, in un programma uno di questi metodi deve essere chiamato main e deve essere definito come mostrato (altrimenti la JVM non sarebbe in grado di eseguire l'applicazione)
- I metodi sono in grado di eseguire delle attività e ritornare informazioni -> la parola void indica che questo metodo non ritorna nessuna informazione

Un semplice Programma in Java

System.out.println(”Welcome to java programming");

- Comunica al computer che deve eseguire un’azione , ovvero visualizzare una stringa di caratteri contenuti tra doppie virgolette.
- Vengono chiamate stringhe di caratteri o messaggi di testo e gli spazi non vengono ignorati.
- System.out è un oggetto speciale chiamato output standard che permette alle applicazioni Java di visualizzare sequenze di caratteri nella finestra di comando.
- Il metodo System.out.println visualizza una riga di testo nella finestra di comando ed il testo tra virgolette è l’argomento del metodo.

Un semplice Programma in Java

```
// programma che scrive una riga di testo
public class Welcome {


  public static void main(String args[])
  {
    System.out.println(”Welcome to java programming");
  } //fine del metodo main
} //fine della classe Welcome
```

Errori Tipici

- Dimenticare di chiudere una parentesi -> errore di sintassi.
- Dimenticare un punto e virgola –> errore di sintassi.
- Quando il compilatore segnala errori di sintassi, l'errore potrebbe non trovarsi nella riga indicata ma in quelle sopra.
- *bad command or filename* oppure *javac:command not found* oppure *javac is not recognized…* -> non avete installato correttamente la JVM oppure non avete settato correttamente il Path.
- *public ClassName must be defined in a file called ClassName* -> non c'è corrispndenza tra il nome della classe che volete compilare e quello del file.
- *Exception in thread main java.lang.NoClassDefFoundError* -> la variabile di ambiente potrebbe non essere stata settata correttamente

Tipi e Variabili

Tipi e Variabili

- Java (come altri programmi) non manipola solo stringhe e numeri ma anche dati più complessi (es.conti bancari, informazioni impiegati etc.)
- In Java si definiscono le classi per definire il comportamento di questi oggetti.

Tipi e Variabili

- In Java ogni valore è di un determinato tipo.
- I valori vengono memorizzati in attributi (della classe)
- Il tipo indica cosa si può fare con i valori
  - p.es. con oggetti di tipo System.out si può invocare il metodo println
  - oppure con due numeri di tipo intero (Integer o *int*) si può fare la somma.
- L'attributo della classe è caratterizzato da un tipo, un nome ed un contenuto.

Tipi e Variabili

- Sintassi Java per la **definizione di attributo (variabile)**:

  nomeTipo nomeVariabile = valore;

  nomeTipo nomeVariabile;

  Esempio:

  String greeting = “Welcome in Java”;

  Obiettivo:

  definire una nuova variabile di tipo nomeTipo e fornire eventualmente un valore iniziale

Tipi e Variabili

- Le variabili possono essere utilizzate al posto degli oggetti che memorizzano.

  per esempio:

  ```
  String greeting = “Hello, World”;
  PrintStream printer = System.out;

  printer.println(greeting) ;
  ```

  al posto di

  ```
  System.out.println(“Hello, World”);
  ```

Nomi degli attributi

- Quando si dichiara una variabile si deve deciderne il tipo e che nome darle.
- Il tipo dipende dall'utilizzo che se ne deve fare.
- Il nome si chiama identificatore.
- Gli identificatori di variabili, metodi e classi sono composti di
  - lettere, cifre, caratteri dollaro e segni di sottolineatura
  - non possono cominciare con una cifra.
  - Non si possono usare altri simboli come per esempio ! ? %.
- gli spazi non sono ammessi all'interno degli identificatori.
- Le parole riservate non si possono usare.
- anche gli identificatori sono **sensibili alle maiuscole e minuscole**.

Nomi degli attributi

- Per convenzione i nomi degli attributi dovrebbero iniziare con lettera minuscola.
- Si possono usare alcune lettere maiuscole all'interno, p.es.:

```
int luckyNumber = 13;
```

- Per modificare il valore di una variabile si usa l'operatore si assegnazione (=).

```
luckyNumber = 12;
```

Assegnazione

- Sintassi di assegnazione:

  nomeVariabile = valore;

esempio:

luckyNumber =13;

Tipi e Variabili

- L'operatore (=) indica un'azione atta a sostituire il nome memorizzato in una variabile -> non è sinonimo di uguaglianza.
- E' errato usare una variabile a cui non è mai stato assegnato un valore, es:

```
int luckyNumber;
System.out.println(luckyNumber);
```

Oggetti, classi e metodi

- Gli oggetti sono entità di un programma che si possono manipolare invocando metodi.
- Gli oggetti sono istanze di classi.
- Un metodo è una sequenza di istruzioni che accede ai dati di un oggetto.
- Una classe specifica i metodi che possono essere applicati ai suoi dati.
- Esempio:
  - il metodo System.out appartiene alla classe PrintStream che ha due metodi println e print.
  - l'oggetto "Hello, World" appartiene alla classe String che mette a disposizione diversi metodi da applicare agli oggetti di tipo String .
  - il metodo length conta il numero di caratteri

```
String greeting ="Hello, World";
int n = greeting.length();
```

Oggetti, classi e metodi

- I metodi di una classe costituiscono la sua interfaccia pubblica.
- L'interfaccia pubblica di una classe specifica cosa si può fare con i suoi oggetti mentre l'implementazione nascosta specifica come si svolgono le azioni
  - L'implementazione o realizzazione privata descrive i dati interni agli oggetti e le istruzioni per le esecuzioni dei propri metodi.
- La realizzazione privata non è nota ai programmatori che non conoscono "l'interno" di un oggetto.

Parametri dei metodi

- Alcuni metodi hanno bisogno di valori di ingresso che specificano il tipo di elaborazione che deve essere svolta.

```
System.out.println("Hello, World");
System.out.println(greeting);
```

- I dati forniti in ingresso ad un metodo si chiamano parametri.

- Ci sono parametri espliciti e parametri impliciti
  - il parametro implicito è l'oggetto con cui viene invocato il metodo stesso.
- Alcuni metodi richiedo un parametro esplicito altri no, inoltre alcuni restituiscono un dato in uscita altri no.
  - il valore restituito da un metodo è il risultato che il metodo ha calcolato perché venga utilizzato nel codice che ha invocato il metodo stesso.

Es. : `int n = greeting.length();`

Oggetti, classi e metodi

- Il valore restituito da un metodo può anche essere utilizzato direttamente come parametro di un altro metodo.

  Es. :

  ```
  System.out.println(greeting.length());
  ```

- Non tutti i metodi restituiscono un valore, per es. il metodo println() interagisce con il sistema operativo per la stampa a video della stringa immessa.

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://javamattone.4it.it
- Tutorial Java:
  - http://docs.oracle.com/javase/tutorial/java/index.html
- Documentazione della API Java:
  - http://docs.oracle.com/javase/7/docs/api/
- Tool per iniziare a programmare:
  - BlueJ (http://www.bluej.org/)

Si applica a tutte queste slide se non diversamente specificato

Classi

ICAR CNR

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*.
- Se il metodo è private, può essere invocato solo da metodi della stessa classe
- Se il metodo è protected può essere invocato dai metodi delle classi figlie
- **Regola generale: metodi pubblici** (e **attributi privati**)
  - 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:
  - Invocare 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 dell'attributo 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;
```

Esempio con 3 errori

```
public miaclassetipo Grandierrori
```

Gli attributi si dichiarano privati

I nomi delle classi iniziano per lettera maiuscola

I nomi degli attributi iniziano per lettera minuscola

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

- 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 String 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:

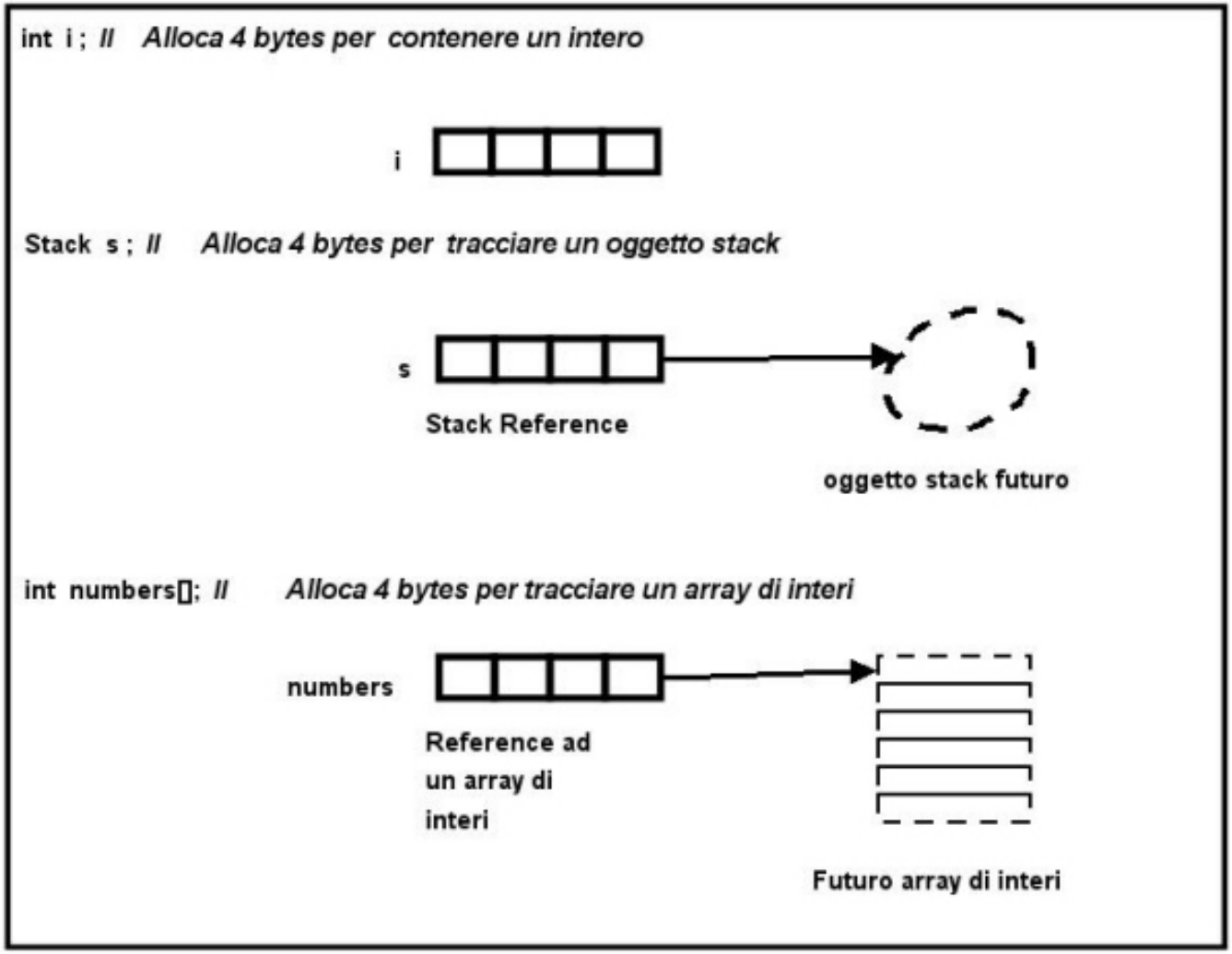
```
int numbers[];
```

- 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

ICAR CNR

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 metodo 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.

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 completo

- A questo punto è possibile 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> ([tipoparam
   nomeparam][, tipoparam nomeparam])
{
   …
}
```

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.
- Sintassi:

```
nomeTipo nomeVariabile = valore;
```

Variabili nei metodi

- Quando ogni oggetto istanziato 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

ICAR CNR

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

```
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 una variabile di un metodo:

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

- per un attributo di una classe:

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

Principi fondamentali dell'OOP

I 3 principi fondamentali dell'OOP

- Astrazione
  - Cattura di un oggetto solo i dettagli che sono significativi per la prospettiva scelta
- Ereditarietà
  - Una modalità per il riuso del codice
  - L'altra è la composizione
- Polimorfismo
  - Uno stesso metodo può essere invocato con parametri diversi

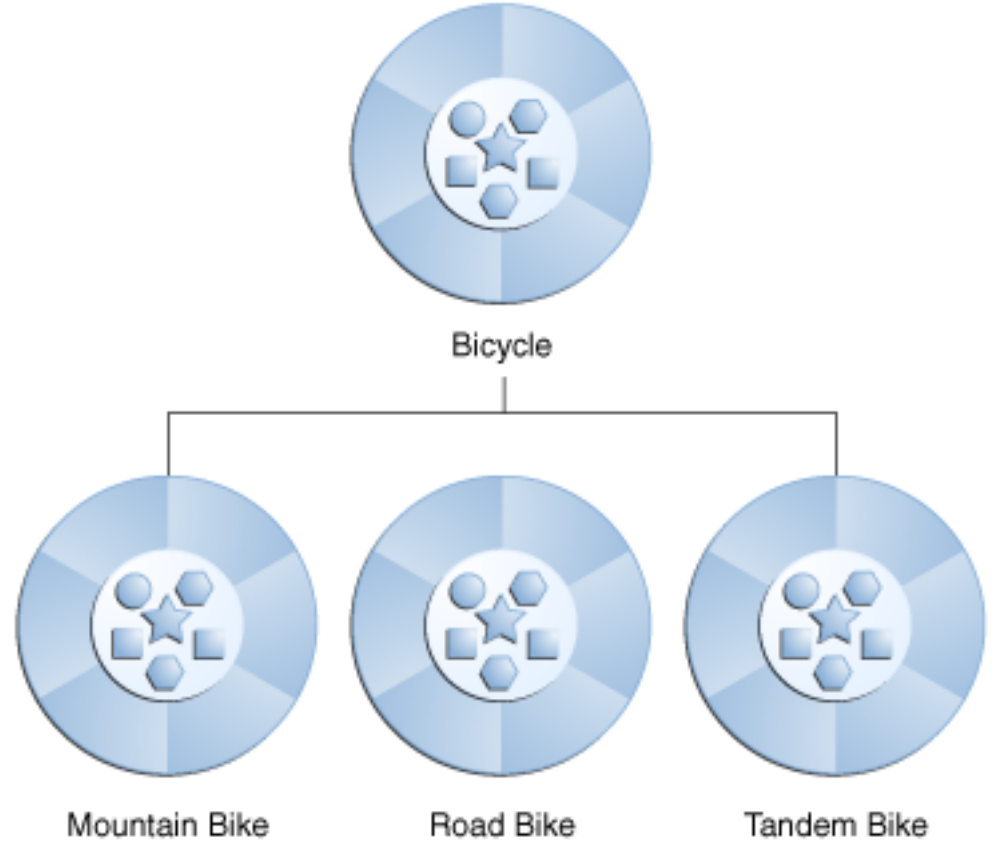
Vantaggi dell'astrazione

- Semplifica il modello nascondendo i dettagli non significativi
  - Vedremo vari esempi nella analisi dei requisiti (cap. 5 libro Bruegge-Dutoit)
  - Spesso avviene tramite l'uso di due relazioni:
    - Is-a –> alla base della relazione di ereditarietà
      - Un pastore tedesco è un cane
        - » dal concetto di cane eredita alcune proprietà come il numero di zampe e lo scodinzolare
        - » Ha un aspetto specifico che non è proprio del cane in generale ma unico del pastore tedesco
    - Has -> alla base della relazione di composizione
      - L'automobile ha quattro ruote ed è in grado di frenare
- Rischio: trascurare dettagli significativi

Ereditarietà

- Tipi differenti di oggetti spesso hanno un certo contenuto informativo in comune.
  - Mountain bike, bicilette da corsa, tandem, per esempio, condividono le caratteristiche di una bicicletta: velocità istantanea, cadenza istantanea di pedalata, marcia utilizzata in un certo istante.
- Eppure queste biciclette differiscono in alcuni aspetti:
  - i tandem hanno due sedili e due manubri, le bici da corsa hanno il manubrio ricurvo, alcune mountain bike hanno gli ammortizzatori
- La OOP permette di ereditare le caratteristiche di una classe (attributi e comportamento) da altre classi.
- La bicicletta è una superclasse da cui si specializzano MountainBike, RoadBike, TandemBike.
- In Java, ogni classe può avere una sola superclasse.

Esempio di ereditarietà



Da http://docs.oracle.com/javase/tutorial/java/concepts/inheritance.html

Sintassi per l'ereditarietà

- La sintassi per creare una sottoclasse è semplice:

```
class MountainBike extends Bicycle {

    // new fields and methods defining
    // a mountain bike would go here

}
```

- In questo modo la classe MountainBike eredita tutti i campi e metodi della classe Bicycle. Così nel programmare la classe MountainBike ci si può concentrare sulle parti di codice che la rendono unica, lasciando alla classe Bicycle il compito di implementare le parti che sono comuni.

Polimorfismo

- Il polimorfismo serve a creare legami più laschi (disaccoppiare) in termini di tipi
- Il polimorfismo si applica ai metodi
  - Possono esistere più metodi con lo stesso nome ma con parametri diversi

```
public class Circle {
…
public void drawCircle(int x,int y,int raggio)
    {
        // put your code here
    }
public void drawCircle(int x1,int y1,int x2,int y2, int x3, int y3)
    {
        // put your code here
    }
}
```

Interfacce

- Gli oggetti definiscono la loro interazione con il mondo esterno attraverso i metodi che espongono.
- I metodi formano l'interfaccia dell'oggetto con il mondo esterno.
- La definizione più comune di interfaccia è un gruppo di metodi con il corpo vuoto:

```
interface Bicycle {
    //  wheel revolutions per minute
    void changeCadence(int newValue);
    void changeGear(int newValue);
    void speedUp(int increment);
    void applyBrakes(int decrement);
}
```

- Per implementare un'interfaccia bisogna usare una classe con un nome diverso dalla interfaccia e la parola chiave 'implements'

```
class ACMEBicycle implements Bicycle {
    // remainder of this class
    // implemented as before
}
```

- Le interfacce formano un **contratto** tra la classe e il resto del mondo.
- Tutti i metodi definiti dall'interfaccia devono apparire nel codice sorgente della classe che la realizza prima che la classe possa essere compilata con successo

Altri aspetti fondamentali dell'OOP

- Modularità
  - Il codice sorgente di un oggetto può essere creato e modificato indipendentemente da quello di un altro oggetto.
  - Le dipendenze sono specificate tramite le interfacce
- Riuso
  - Se un oggetto esiste, esso può essere riusato (anche in altri programmi) tramite:
    - Composizione
    - Ereditarietà