

# Laboratorio di Informatica

## Lezione 5

Cristian Consonni

11 novembre 2015

# Outline

- 1 Introduzione alla Programmazione Orientata agli Oggetti (OOP)
- 2 Oggetti e Classi
- 3 Utilizzo del debugger
- 4 Esercizi

Cristian Consonni

- **DISI - Dipartimento di Ingegneria e Scienza dell'Informazione**
- **Pagina web** del laboratorio:  
<http://disi.unitn.it/~consonni/teaching>
- **Email:** [cristian.consonni@unitn.it](mailto:cristian.consonni@unitn.it)
- **Ufficio:** Povo 2 - Open Space 9
  - Per domande: scrivetemi una mail
  - Ricevimento: su appuntamento via mail

# Outline for section 1

- 1 Introduzione alla Programmazione Orientata agli Oggetti (OOP)
- 2 Oggetti e Classi
- 3 Utilizzo del debugger
- 4 Esercizi

# Programmazione Orientata agli Oggetti (I)

- La Programmazione Orientata agli Oggetti (o *Object Oriented Programming*, OOP, in inglese) è un paradigma di Programmazione basato sui concetti di **oggetto** e **classe**.
- In questa lezione introdurremo i concetti di base legati agli oggetti e li utilizzeremo come una **struttura dati**, ovvero una entità che permette di gestire un insieme di dati (detti *attributi* o *membri*).

# Programmazione Orientata agli Oggetti (II)

Finora **ad eccezione delle String** abbiamo usato delle variabili di uno dei tipi **primitivi** (o *atomici*):

- `int`
- `double`
- `float`
- ...

utilizzando un tipo primitivo sappiamo che i dati verranno rappresentati in memoria in un modo predefinito:

- `int` → 4 byte (32 bit)
  - min:  $-2^{31}$ ;
  - max:  $2^{31} - 1$ ;
- `double` → 8 byte (64 bit)
  - $max \equiv |min|$ :  $1.7976931348623157 \cdot 10^{308}$ ;
  - $\epsilon$ :  $4.9 \cdot 10^{-324}$ ;

# Programmazione Orientata agli Oggetti (III)

Immaginiamo di volere scrivere un programma che memorizza i dati (nome e cognome, luogo di nascita, età) di una persona.

```
String nome1 = "Alice Rossi";  
String luogo1 = "Milano";  
int eta1 = 19;  
  
String nome2 = "Roberto Verdi";  
String cognome2 = "Roma";  
int eta2 = 20;  
...
```

Problemi di questo approccio:

- È molto scomodo creare nuove istanze dell'entità (concettuale) "Persona";
- È molto facile commettere errori;

# Programmazione Orientata agli Oggetti (IV)

Una **oggetto**:

- è un'*entità software* costituito da un insieme di stati e comportamenti che si riferiscono a uno stesso concetto;  
(«*An object is a software bundle of related state and behavior.*»)

<https://docs.oracle.com/javase/tutorial/java/concepts/>

- consiste in una regione di memoria allocata;



# Programmazione Orientata agli Oggetti (V)

Una **classe**:

- è un prototipo a partire dalla quale gli oggetti vengono **creati** (o **istanziati**, o **istanziati**);  
(«A class is a blueprint or prototype from which objects are created.»)  
<https://docs.oracle.com/javase/tutorial/java/concepts/>
- è la specifica che descrive quali sono i possibili stati e comportamenti di un oggetto creati a partire da esso;
- è un insieme di variabili (anche dette **membri** o **attributi**) e di funzioni i metodi (dette **metodi**);
- è anche chiamata *Abstract Data Type* (ADT).

# Programmazione Orientata agli Oggetti (VI)

Classe:

```
public class Persona {  
  
    private String nome;  
    private String luogoNascita;  
    private int eta;  
  
}
```

Oggetti:



Photo credit: CC-BY-SA Mark Sebastian @ Wikimedia Commons <http://bit.ly/1kip83N>

# Programmazione Orientata agli Oggetti (VI)

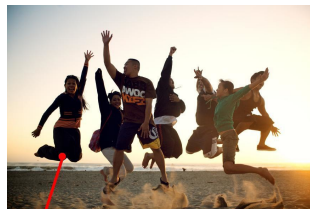
Classe:

```
public class Persona {  
  
    private String nome;  
    private String luogoNascita;  
    private int eta;  
  
}
```

...

```
Persona Alice;
```

Oggetti:



Alice

# Programmazione Orientata agli Oggetti (VI)

Classe:

```
public class Persona {  
  
    private String nome;  
    private String luogoNascita;  
    private int eta;  
  
}
```

...

```
Persona Alice;  
Persona John;
```

Oggetti:



Alice

John

# Programmazione Orientata agli Oggetti (VI)

Classe:

```
public class Persona {  
  
    private String nome;  
    private String luogoNascita;  
    private int eta;  
  
}
```

...

```
Persona Alice;  
Persona John;
```

Oggetti:



Alice

John

## Oggetti vs. Classi

Un oggetto è un'istanza di una classe

# Classi vs. Oggetti (I)

## Classe:

- descrizione delle proprietà *comuni* a una tipologia di variabili;
- è un “concetto”;
- è una parte di un programma;

- 1 Persona
- 2 Album
- 3 Canzoni

## Oggetti:

- rappresentazioni delle proprietà di una singola istanza;
- Un “fenomeno”/“manifestazione”.
- una parte dei dati nell'esecuzione di un programma.

- 1 Hillary Clinton, Rafael Nadal, Lewis Hamilton;
- 2 Thriller, Back in Black, The Dark Side of the Moon
- 3 Thriller, Beat it, Billie Jean, Hells Bells, Shoot to Thrill, Back in Black, ...

# Classi vs. Oggetti (II)

- Nella programmazione ad oggetti (OOP) si scrivono classi;
  - il codice sorgente che scriviamo contiene delle **classi**;
  - una classe è “statica”;
  - “Una”;
- Gli oggetti sono creati a partire dalle classi;
  - una classe è come una ricetta di una torta, gli oggetti sono le torte prodotte a partire dalla ricetta.
  - un oggetto è “dinamico”
  - “Molti”

## Ricetta (Classe) **Muffin**:

- 1 3 uova;
- 2 380 g di farina;
- 3 ...



Photo credit: CC-BY Sara K @ Flickr <http://bit.ly/1kioQdd>

# Outline for section 2

- 1 Introduzione alla Programmazione Orientata agli Oggetti (OOP)
- 2 Oggetti e Classi
- 3 Utilizzo del debugger
- 4 Esercizi



# Classi: costruttore (I)

Una classe necessita di un **costruttore** per potere essere istanziata:

```
public class Persona {  
    // attributi della classe -> variabili dell'oggetto  
    private String nome;  
    private String luogoNascita;  
    private int eta;  
  
    // Costruttore con parametri  
    public Persona(String nome, String luogoNascita, int eta) {  
        ...  
    }  
  
    // Costruttore senza parametri con valori di default  
    public Persona() {  
        ...  
    }  
}
```

# Classi: costruttore (II)

Il **costruttore** è il *metodo* che crea la classe:

```
public class Persona {  
    ...  
    // Costruttore con parametri  
    public Persona(String nome, String luogoNascita, int eta) {  
        this.nome = nome;  
        this.luogoNascita = luogo;  
        this.eta = eta;  
    }  
  
    // Costruttore senza parametri con valori di default  
    public Persona() {  
        this.nome = "Anonimo";  
        this.luogoNascita = "Sconosciuto";  
        this.eta = -1;  
    }  
}
```

## Classi: costruttore (III)

La parola riservata **this** serve per specificare se ci si sta riferendo ai parametri del costruttore o ai membri della classe.

```
...  
    public Persona(String nome, String luogoNascita, int eta) {  
        this.nome = nome;  
        this.luogoNascita = luogo;  
        this.eta = eta;  
    }  
...
```

# La parola riservata this

- Costruttore con parametri:

```
public Persona(String nome, String luogoNascita, int eta) {  
    this.nome = nome;  
    this.luogoNascita = luogo;  
    this.eta = eta;  
}
```

```
Persona anna = Persona("Anna Rossi", "Milano", 19);
```

- Costruttore senza parametri:

```
public Persona() {  
    this.nome = "Anonimo";  
    this.luogoNascita = "Sconosciuto";  
    this.eta = -1;  
}
```

```
Persona anon = Persona();
```

# Accessibilità delle variabili (I)

Le parole riservate **private**, **public**, **protected** modificano l'accessibilità delle variabili:

- 1 **public**: sono accessibili da chiunque. (**più accessibili**)
- 2 **protected**: sono accessibili solo all'interno dallo stesso package dalla stessa classe e dalle sottoclassi.
- 3 default (nessun modificatore aggiuntivo): sono accessibili solo all'interno dallo stesso package dalla stessa classe.
- 4 **private**: sono accessibili solo dalla stessa classe. (**meno accessibili**)

public e private

Noi useremo solo **public** (**più accessibili**) e **private** (**meno accessibili**)

# Accessibilità delle variabili (I)

Le parole riservate **private**, **public**, **protected** modificano l'accessibilità delle variabili:

- 1 **public**: sono accessibili da chiunque. (**più accessibili**)
- 2 **protected**: sono accessibili solo all'interno dallo stesso package dalla stessa classe e dalle sottoclassi.
- 3 default (nessun modificatore aggiuntivo): sono accessibili solo all'interno dallo stesso package dalla stessa classe.
- 4 **private**: sono accessibili solo dalla stessa classe. (**meno accessibili**)

## public e private

Noi useremo solo **public** (**più accessibili**) e **private** (**meno accessibili**)

## Accessibilità delle variabili (II)

La tabella seguente riassume l'accessibilità delle variabili dichiarate con i vari modificatori.

Modificatore	Class	Package	Sottoclassi	Mondo
<b>public</b>	<b>Y</b>	<b>Y</b>	<b>Y</b>	<b>Y</b>
protected	Y	Y	Y	N
default (no modificatore)	Y	Y	N	N
<b>private</b>	<b>Y</b>	<b>N</b>	<b>N</b>	<b>N</b>

Si veda anche:

<https://docs.oracle.com/javase/tutorial/java/java00/accesscontrol.html>

# Accessibilità delle variabili: metodi getter e setter (I)

Dato che le variabili **private** non sono accessibili al di fuori della classe è necessario creare dei metodi che restituiscono e modificano il loro valore. Questi metodi sono detti rispettivamente:

- **getter**: restituisce (**get**) il valore della variabile, ad es. `getNome()`;
- **setter**: imposta (**set**) il valore della variabile, ad es. `setNome()`;

di solito i nomi sono `getNomeVariabile()` e `setNomeVariabile()`;

## Variabili private vs. public

Se una variabile è **public** non è necessario avere getter e setter, ma è buona norma usare variabili **private** nelle classi



# Accessibilità delle variabili: metodi getter e setter (I)

Dato che le variabili **private** non sono accessibili al di fuori della classe è necessario creare dei metodi che restituiscono e modificano il loro valore. Questi metodi sono detti rispettivamente:

- **getter**: restituisce (**get**) il valore della variabile, ad es. `getNome()`;
- **setter**: imposta (**set**) il valore della variabile, ad es. `setNome()`;

di solito i nomi sono `getNomeVariabile()` e `setNomeVariabile()`;

## Variabili private vs. public

Se una variabile è **public** non è necessario avere getter e setter, ma è buona norma usare variabili **private** nelle classi

# Accessibilità delle variabili: metodi getter e setter (II)

Esempio di metodi **getter** e **setter** per la variabile nome nella classe Persona:

```
public class Persona {  
  
    private String nome;  
    ...  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setName(String nome) {  
        this.nome = nome;  
    }  
  
}
```

# Metodo toString (I)

- il metodo toString() restituisce una **stringa** ovvero la “rappresentazione testuale” dell’oggetto su cui è invocato (da usare ad esempio quando si stampa l’oggetto).
- la definizione default del metodo nella classe Object è  
`< nome_classe > @ < hashCode >;`
- possiamo ridefinire questo metodo per stampare un oggetto nel modo che vogliamo;

# Metodo toString (II)

Esempio:

```
public class Persona {  
  
    private String nome;  
    private String luogoNascita;  
    private int eta;  
  
    ...  
  
    public String toString() {  
        return nome + " (" + luogoNascita + "), " + eta;  
    }  
}
```

⇒ `println(anna);` ⇒ Anna Rossi (Milano), 19

# Accessibilità delle classi

A livello delle classe è possibile usare:

- 1 **public**: classe accessibile anche da altri package.
- 2 default (nessun modificatore aggiuntivo): classe accessibile solo all'interno dello stesso package.

# Struttura di un progetto (I)

In un progetto esistono più classi:

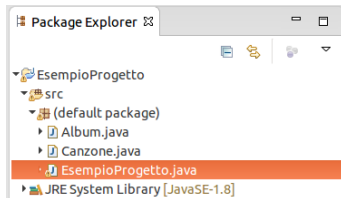
- Una classe principale che contiene il metodo **main**;
- altri file contenenti le altre classi “secondarie”: create un nuovo file per ogni classe (ad esempio `Persona.java` per la classe `Persona`);
- il nome di una classe ha l'iniziale maiuscola e si usa la notazione CamelCase/PascalCase;
- non si possono avere due classi con lo stesso nome all'interno dello stesso progetto;

# Struttura di un progetto (II)

Esempio:

- creiamo un progetto `EsempioProgetto` che riguarderà la musica (album e canzoni);
- La classe principale si chiama `EsempioProgetto` e si trova nel file `EsempioProgetto.java`;
- Esistono due classi `Album` (`Album.java`) e `Canzone` (`Canzone.java`) che possono essere usate dentro `EsempioProgetto`.

# Struttura di un progetto (II)



Classe:

```
public class EsempioProgetto {  
  
    public static void main(String[] args) {  
  
        Album thriller = new Album("Thriller", ...);  
        Canzone hb = new Canzone("Hells Bells", "AC/DC", "5:13");  
    }  
}
```



# Outline for section 3

- 1 Introduzione alla Programmazione Orientata agli Oggetti (OOP)
- 2 Oggetti e Classi
- 3 Utilizzo del debugger
- 4 Esercizi

- un **bug** (o **baco**) indica un errore o un comportamento inaspettato in un programma software;
- l'operazione di individuazione e correzione degli errori in un programma si chiama **debug/debugging**;
- uno degli strumenti per facilitare l'operazione di debugging è il **debugging**

# Bug (intermezzo storico) (I)

Piccola nota storica:

- in inglese **bug** significa *insetto*;
- A livello teorico l'idea che un programma potesse contenere errori è stata avanzata per la prima volta nel 1843 da **Ada Lovelace** nelle sue note sulla **macchina analitica** di **Babbage**;
- pare che l'espressione “bug” fosse in uso per indicare malfunzionamenti meccanici sin dal fine del 1800 (Thomas Edison, 1878)



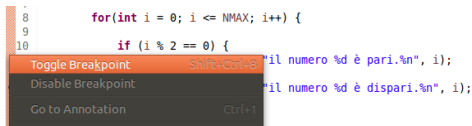


# Debugging (I)

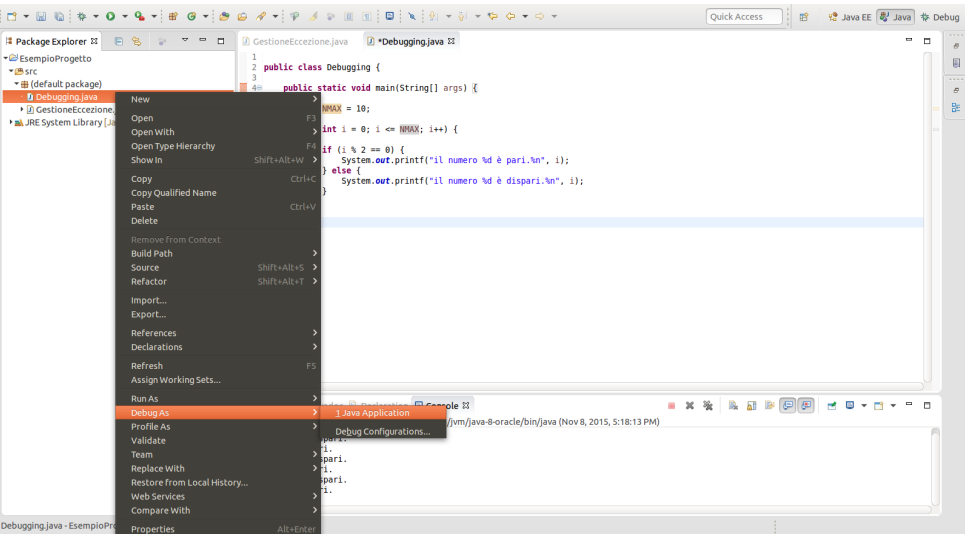
- durante il debugging un programma viene eseguito in modo interattivo per potere osservare l'esecuzione del codice e il valore delle variabili passo dopo passo;
- si possono definire dei “punti di controllo”, detti **breakpoints** e **watchpoints**;
  - nei **breakpoints** l'esecuzione di un programma viene interrotta per poterlo ispezionare;
  - nei **watchpoints** l'esecuzione di un programma viene interrotta solo se una variabile viene letta o modificata;

# Debugging (I)

- Eclipse permette di lanciare un programma in **debug mode** e di usare una **debug perspective** allo scopo di eseguire il debugging;
- è possibile inserire un breakpoint cliccando all'inizio della riga con il tasto destro e selezionando **Toggle Breakpoint**;

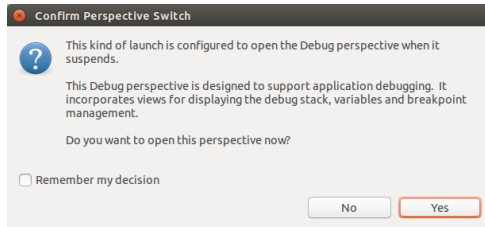


- Per lanciare il programma premere con il tasto destro sul nome del file e selezionare **Debug As > Java Application**;



# Debugging (II)

Confermare l'apertura della *debug perspective*:





The screenshot shows an IDE with the following components:

- Top Toolbar:** Contains various icons for file operations, running, and debugging.
- Debug Tab:** Shows a list of terminated sessions:
  - <terminated>Debugging [Java Application]
  - <terminated>Debugging at localhost:42284
  - <terminated, exit value: 0>/usr/lib/jvm/java-8-oracle/bin/java (Nov 8, 2015, 5:32:18 PM)
- Variables and Breakpoints Panel:** Currently empty.
- Code Editor:** Displays the source code for `Debugging.java`. The code is as follows:

```
1 public class Debugging {
2
3
4     public static void main(String[] args) {
5
6         int NMAX = 10;
7
8         for(int i = 0; i <= NMAX; i++) {
9
10             if (i % 2 == 0) {
11                 System.out.printf("il numero %d è pari.\n", i);
12             } else {
13                 System.out.printf("il numero %d è dispari.\n", i);
14             }
15         }
16     }
17 }
```
- Outline Panel:** Shows the class structure with `main(String[]): void` highlighted.
- Console Panel:** Shows the output of the program:

```
<terminated> Debugging [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Nov 8, 2015, 5:32:18 PM)
il numero 5 è dispari.
il numero 6 è pari.
il numero 7 è dispari.
il numero 8 è pari.
il numero 9 è dispari.
il numero 10 è pari.
```
- Bottom Status Bar:** Displays 'Writabile', 'Smart Insert', and the time '17:5'.

# Debugging (II)

Il debugger può essere comandato attraverso le icone nella barra o i tasti funzione:

- **F5** esegue la linea selezionata e va alla riga successiva. Se la linea selezionata è una chiamata a funzione allora il debugger entra nella funzione indicata;
- **F6** “step over”: esegue un metodo senza entrare esplicitamente in esso con il debugger;
- **F7** “step out”: ritorna alla funzione che ha chiamato il metodo corrente, ovvero termina l'esecuzione del metodo corrente e ritorna al chiamante;
- **F8** riprende l'esecuzione del programma fino a che non viene incontrato un nuovo breakpoint;

# Debugging (III)

Dal pannello **variabili** è possibile:

- ispezionare il valore della variabile;
- (click destro) impostare un nuovo valore per la variabile;

# Outline for section 4

- 1 Introduzione alla Programmazione Orientata agli Oggetti (OOP)
- 2 Oggetti e Classi
- 3 Utilizzo del debugger
- 4 Esercizi**

# Esercizi (I)

Creare un nuovo progetto chiamato ProgettoMusica che:

- contenga una classe principale Raccolta (Raccolta.java);
- contenga una classe secondaria Canzone (Canzone.java);
- la classe Canzone contiene i seguenti attributi **private**:
  - una variabile nome di tipo String (nome della canzone);
  - una variabile artista di tipo String (nome dell'autore della canzone)
  - una variabile durata di tipo String (durata della canzone) (scrivere la durata come una stringa del tipo XmYYs, ovvero X minuti e YY secondi);
  - una variabile rating di tipo **int** (voto della canzone da 1 a 5);

# Esercizi (II)

La classe Canzone ha:

- un costruttore con parametri che imposta i valori di nome, artista, durata e rating;
- un costruttore senza parametri che imposta i valori di nome a "Nessun nome", nome a "Nessuno", durata a "0m00s" e rating a 0;
- i metodi getter e setter per le variabili;
- un metodo toString() che stampa i dati contenuti nella classe come segue:
  - `< nome > di < artista > (< durata >), < rating > stelle`
  - Thriller di Michael Jackson (5m58s), 5 stelle

# Esercizi (III)

- Nella classe principale `Raccolta` creare un vettore di 5 canzoni (oggetti di tipo `Canzone`) inizializzandoli con dati a vostra scelta e successivamente stamparli;
- Successivamente, modificare il programma per stampare solo le canzoni che hanno un rating superiore a 3;
- Infine, modificare nuovamente il programma per stampare solo le canzoni che hanno `"Michael Jackson"`;

# Esercizi (IV)

## Suggerimento:

- Ricordate che per verificare l'uguaglianza tra due stringhe dovete usare il metodo `equals`;
- Ricordate che c'è differenza tra maiuscole minuscole;

```
String mj = "Michael Jackson";
```

```
// mj.equals(c1.getArtista()) ritornerà True
```

```
Canzone c1 = new Canzone("Thriller", "Michael Jackson",  
                          "5m58", 5);
```

```
// mj.equals(c2.getArtista()) ritornerà False
```

```
Canzone c2 = new Canzone("Back in Black", "AC/DC",  
                          "4m15", 5);
```