

Appunti di Programmazione 2

Domande e Risposte

Capitolo 1: Astrazioni Strutturali e UML

1) Che cosa si intende per classe e per oggetto?

La **classe** è un modello o prototipo che definisce un tipo di oggetto, cioè un modello formale che specifica lo stato e il comportamento di tutte le sue istanze. L'**oggetto o (istanza)** è la rappresentazione concreta e specifica di una classe.

Esempio: Definizione di classe **Cane** e oggetti **Zoe** e **Drake**



2) Che cosa si intende per astrazione?

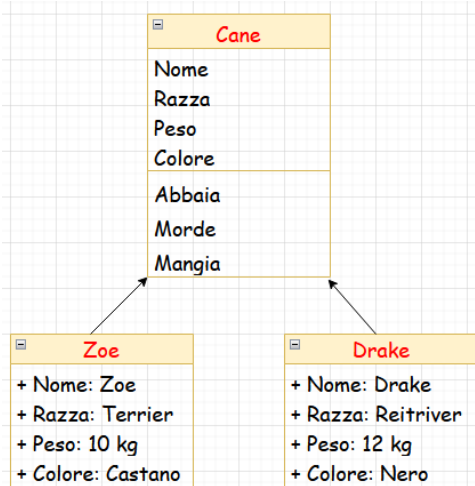
Un'**astrazione** è un procedimento concettuale per mezzo del quale si definisce un concetto mettendo in evidenza gli aspetti comuni e generali e rimuovendo dalla descrizione del concetto gli aspetti di dettaglio e particolari.

Esempio:

Cane è un concetto più astratto di **Drake** e **Zoe**.

Il legame è nei due sensi:

- 1 - **classificazione**, in cui si parte dall'oggetto e lo si classifica;
- 2 - **istanziamento**, in cui si parte dalla definizione di classe e si creano istanze (come creare Zoe come esemplare di Cane).



3) Come si applica al legame tra classe e le sue istanze?

→ Si definisce la classe osservando le istanze, raccogliendo a fattor comune le caratteristiche condivise e operando selezione.

→ L'**astrazione strutturale** modella oggetti e classi e relazioni fra di essi.

4) Descrivere l'astrazione classificazione "instance_of"

La **classificazione** **instance_of** è la tipologia di astrazione che si occupa di legare **istanze (oggetti)** e **classi**. La **classe** definisce le caratteristiche comuni degli **oggetti** di un insieme e l'oggetto della classe possiede le proprietà definite dalla classe.

Esempio:

Zoe è un esemplare di **Cane**.

5) Descrivere l'astrazione generalizzazione "is_a"

→ La **generalizzazione** lega una **classe genitore (superclasse)** a una o più **classi figlie (sottoclassi)** che ne sono sottoinsieme.

→ Una classe figlia **è una** classe genitore.

→ Ogni esemplare della **classe figlia** possiede tutte le proprietà definite nella **classe padre**, come ad esempio la classe **Cane** è una sottoclasse della classe **Mammifero**.

→ Ogni esemplare della classe figlia appartiene anche al superclasse (astrazione classificazione - **instance_of**), come ad esempio **Drake** è un **Cane**, quindi è anche un **Mammifero**.

Attenzione: vi è una distinzione netta tra **classificazione** e **generalizzazione**, ovvero:

→ **Drake** è un esemplare (**instance_of**) di **Cane**;

→ **Cane** è una sottoclasse di **Mammifero**.

6) Descrivere l'astrazione aggregazione "part_of"

→ L'**aggregazione** **part_of** lega una classe "**aggregato**" con un insieme di classi "**parti**". Ogni esemplare di Aggregato è costituito da esemplare delle classi Parti.

Attenzione:

→ un **Cane** è un **Mammifero** (**is_a**) - generalizzazione;

→ un **Cane** fa parte di un **Branco** (**part_of**) - aggregazione. Il **Cane** non è un **Branco**.

7) Descrivere l'astrazione associazione "has_a"

L'**associazione astrazione** **has_a** definisce una connessione logica fra oggetti di una **classe** di un'altra **classe**.

Esempi

→ **Classi**: Cane, Persona

• **Associazione**: una **Persona** è padrona di un **Cane**

→ **Cani**: Fido, Pluto - **Persone**: Mario, Pino

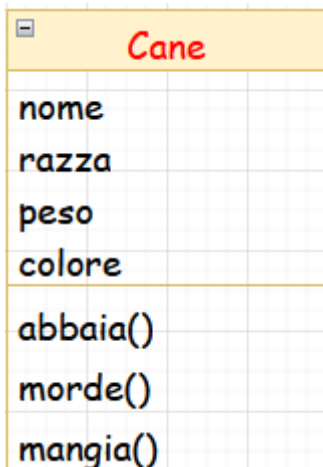
• Mario è padrone di Fido

• Pino è padrone di Pluto

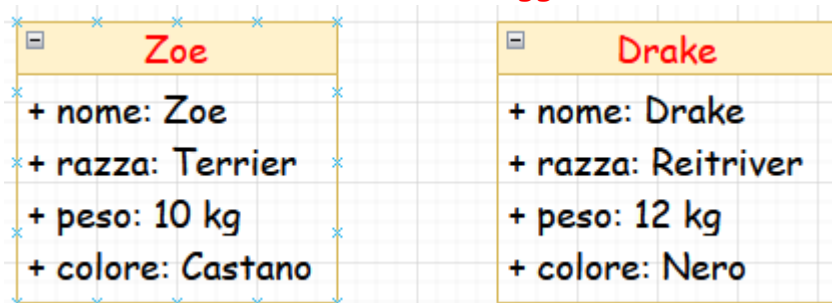
8) Cosa si intende per UML?

→ Il **diagramma UML** è un linguaggio di modellazione e di specifica basato sul paradigma orientato agli oggetti. Vengono creati per comprendere i progetti, l'architettura del codice e l'implementazione proposta di sistemi software complessi.

9) Come viene identificato una classe?

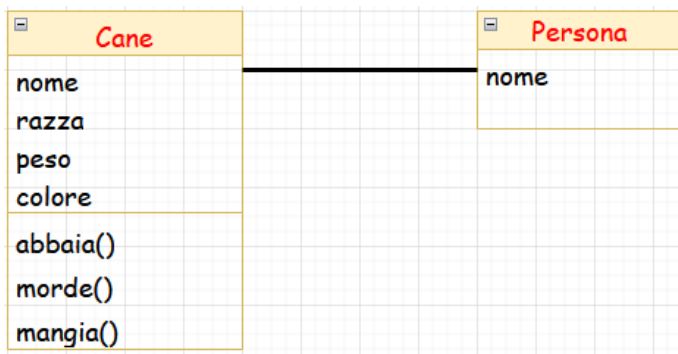


10) Come viene identificato un oggetto?



11) Come viene identificata un'associazione?

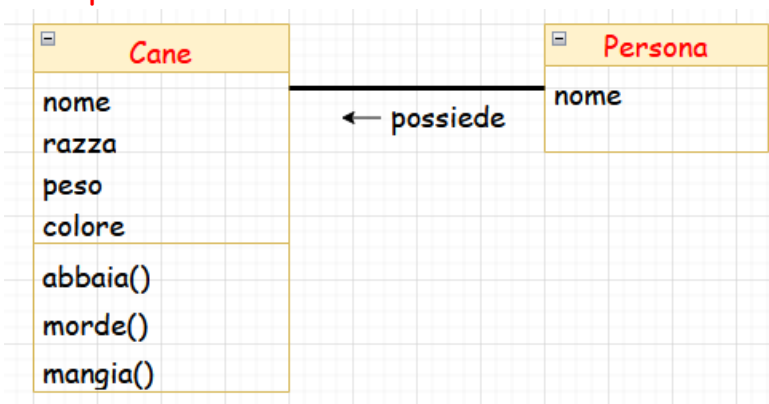
Un'**associazione** è una linea che collega le classi coinvolte.



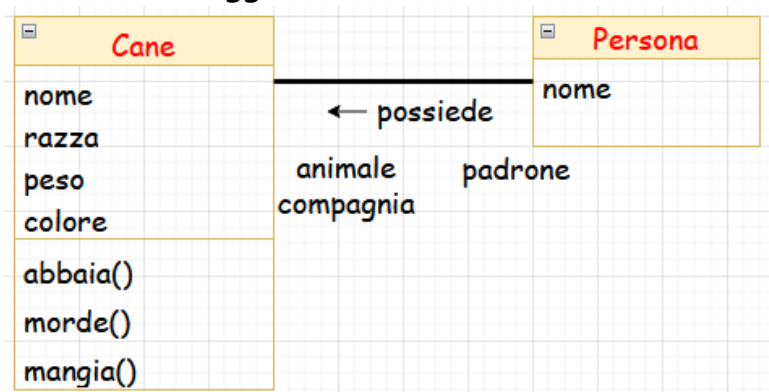
Caratteristiche:

→ Il **nome** dell'associazione esprime il significato dell'associazione (spesso è un verbo).

Esempio

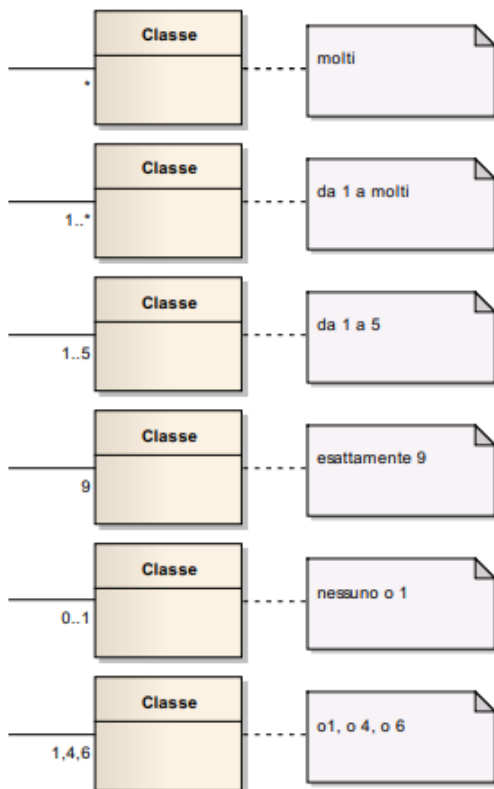


→ Il **ruolo** viene utilizzato per esprimere la funzione del partner (spesso è un sostantivo o aggettivo).



→ La **cardinalità** esprime quante istanze della classe possono essere associate all'altra classe.

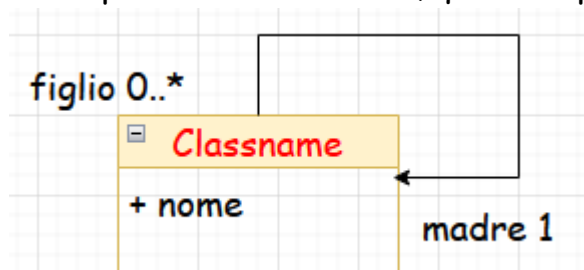
Esempi di cardinalità



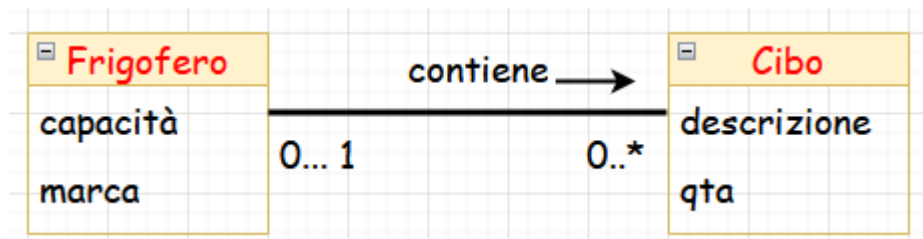
→ Un'**associazione multipla** esplicita più associazioni tra una coppia di classe per arricchirne la descrizione e viene utilizzata per rendere il diagramma più leggibile.



→ L'**associazione a cappio** specifica il fatto che una relazione può essere messa in relazione con sé stessa. In questo caso vi è un problema a definire la madre originale, in quanto madre ha cardinalità 1, ovvero ogni persona ha una madre, ma la prima non ne ha una, quindi si potrebbe correggere con cardinalità 0...1.



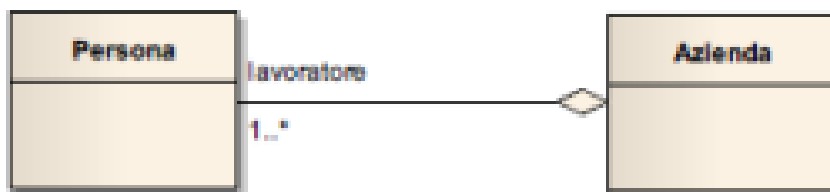
→ La **navigabilità** esplicita il fatto del chi vede cosa. Determina la modalità con cui viene realizzata l'associazione.



→ L'**aggregazione** rappresenta una relazione poco forte.

Le caratteristiche dell'aggregazione sono:

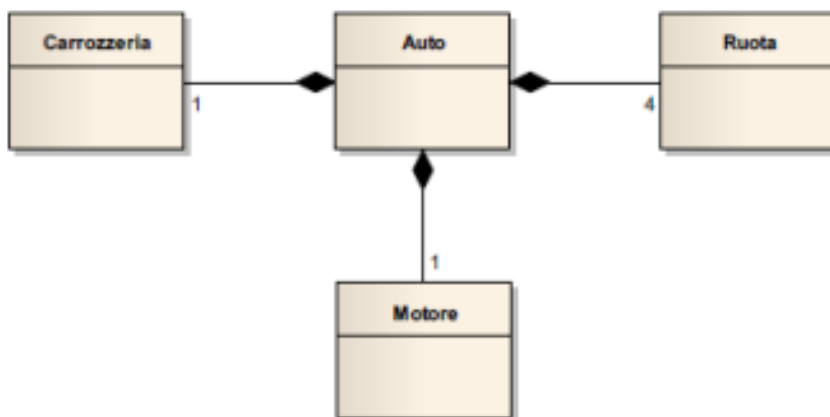
- a) le parti possono esistere indipendentemente dall'aggregato;
- b) il tutto può in alcuni casi esistere indipendentemente dalle parti, ma in altri casi no;
- c) il tutto è in qualche modo incompleto se mancano alcune delle sue parti;
- d) è possibile che più aggregati condividano una stessa parte.



→ La **composizione** rappresenta una relazione molto forte.

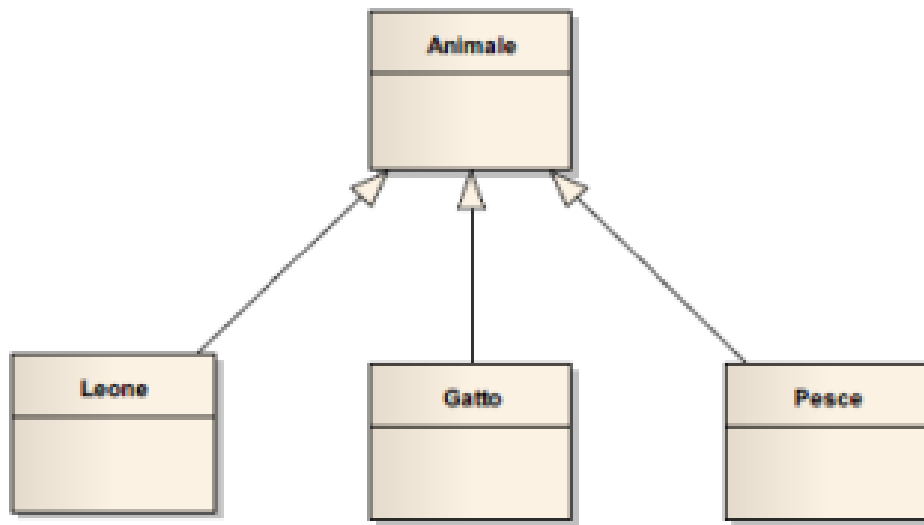
Le caratteristiche della composizione sono:

- a) ogni parte può appartenere ad un solo tutto per volta;
- b) il tutto è l'unico responsabile di tutte le sue parti: questo vuol dire che è responsabile della loro creazione e distruzione;
- c) il tutto può rilasciare una sua parte, a patto che un altro oggetto si prenda la relativa responsabilità;
- d) se il tutto viene distrutto, deve distruggere tutte le sue parti o cederne la responsabilità a qualche altro oggetto.



→ La **generalizzazione** rappresenta una relazione che ogni classe possiede con

la superclasse.



Capitolo 2: Programmazione Orientata agli Oggetti in Java

12) Definire il concetto di OOP

→ La **programmazione orientata agli oggetti (OOP)** è una metodologia di programmazione, in cui il programma viene considerato come una composizione di oggetti che interagiscono tra di loro per mezzo di azioni.

→ **Java** è un linguaggio di programmazione ad oggetti.

13) Che cosa si intende per oggetto?

→ Un **oggetto** è la rappresentazione di un'entità reale o concettuale e frutto di un processo di astrazione che porta ad estrarre solo le caratteristiche salienti per una data applicazione.

14) Quali sono le caratteristiche degli oggetti? Descriverne inoltre ciascuna di quelle

→ Un **oggetto** ha tre caratteristiche che sono **stato**, **comportamento** e **identità**.

→ Lo **stato dell'oggetto** è una delle possibili condizioni in cui esso può trovarsi ed è determinato dalle variabili definite, chiamate attributi o variabili di istanza che possono essere di tipi semplici o a loro volta degli oggetti.

→ Il **comportamento dell'oggetto** è la modalità con cui un oggetto risponde alle richieste da parte di altri oggetti, che comunicano tra di loro scambiandosi messaggi attraverso le rispettive interfacce. I messaggi attivano metodi che determinano il comportamento degli oggetti.

→ L'**identità** è la proprietà in cui due oggetti anche se si trovano nello stesso stato, sono comunque due entità ben distinte. Ogni **oggetto** ha un proprio **OID (Object Identifier)** che è univoco nel sistema e invariante nel tempo.

15) Che cosa si intende per classe?

La **classe** è quella "**struttura**" che definisce le caratteristiche delle sue istanze e rappresenta una e una sola astrazione a partire dalla quale è possibile creare **oggetti**. L'**istanza** o **oggetto** è un esemplare creato a partire dalla sua **classe**.

16) Come viene definita una classe?

Una **classe** deve:

→ specificare l'interfaccia che ogni suo oggetto offre verso gli altri con cui interagisce attraverso:

a) definizione (signature) dei comandi (o messaggi) (nome, tipo ritornato e parametri formali);

b) definizione degli attributi "visibili";

→ implementare attributi interni, metodi interni e operazioni dei metodi.

17) Cosa si intende per information hiding?

L' **information hiding** è un principio della programmazione orientata agli oggetti in cui alcuni attributi e metodi possono anche essere nascosti all'intero dell'oggetto, cioè resi invisibili agli altri oggetti. Attributi e metodi nascosti (o invisibili) sono detti **privati**.

Capitolo 3: Classi e istanze con Java

18) Cosa si intende per classe?

Una **classe** è la definizione di un tipo di **oggetto**. Una classe specifica il nome e il tipo delle variabili di istanza degli **oggetti**, ma non specifica il loro valore.

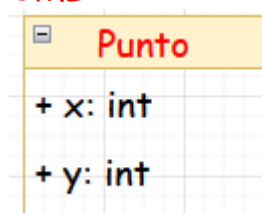
Essa specifica i metodi dei suoi **oggetti**. Un **oggetto** di una istanza della **classe**.

Struttura classe

```
public [altri modificatori] class NomeClasse {  
    //Definizione Attributi di istanza o classe  
    //Implementazione costruttori  
    //Implementazione Metodi di Istanza e/o di classe  
}
```

19) Come trasformare in codice una classe definita nel diagramma UML?

UML



Codice


```
public class Punto {  
    int x;  
    int y;  
}
```

20) Come viene creato un oggetto?

L'oggetto viene creato nella maniera seguente:

```
NomeClasse nomeReference = new NomeClasse();  
nomeReference.comando();  
nomeRefernce.attributo;
```

Esempio 1:

```
p1 = new Punto();  
p2 = new Punto();  
  
p1.x = 3;  
p1.y = 4;  
p2.x = 50;  
p2.y = 90;
```

Esempio 2

Creare due punti e assegnare i valori per x e y 3 e 4 e 50 e 90. Stampare a video quale punto ha x

```
public class Punto {  
    int x;  
    int y;  
}
```

```
public class PuntoDemo {
    public static void main(String[] args) {
        Punto p1,p2;

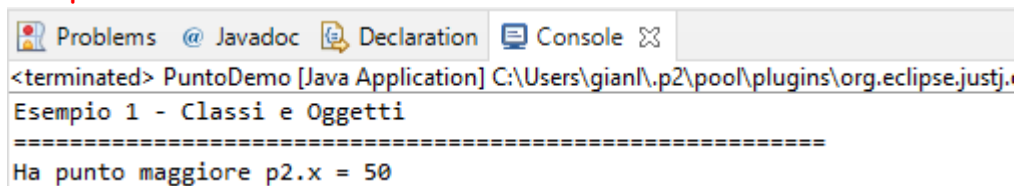
        System.out.println("Esempio 1 - Classi e Oggetti");
        System.out.println("=====");

        p1 = new Punto();
        p2 = new Punto();

        p1.x = 3;
        p1.y = 4;
        p2.x = 50;
        p2.y = 90;

        if (p1.x>p2.x)
            System.out.println("Ha punto maggiore " + p1.x);
        else {
            if (p1.x == p2.x)
                System.out.println("Sono uguali");
            else
                System.out.println("Ha punto maggiore " + p2.x);
        }
    }
}
```

Output



The screenshot shows the Eclipse IDE's console window. The title bar includes 'Problems', 'Javadoc', 'Declaration', and 'Console'. The console output is as follows:

```
<terminated> PuntoDemo [Java Application] C:\Users\gianl\p2\pool\plugins\org.eclipse.justj.  
Esempio 1 - Classi e Oggetti  
=====  
Ha punto maggiore p2.x = 50
```

21) Quali sono le problematiche dell'assegnamento e dell'uguaglianza di un oggetto?

- Gli operatori di assegnamento e uguaglianza si comportano con gli oggetti in maniera diversa rispetto alle variabili di tipo primitivo.
- L'operatore `=` assegna ad una variabile l'indirizzo di memoria in cui è allocato l'oggetto.
- L'operatore di uguaglianza `==` verifica se due oggetti sono memorizzati nella stessa area di memoria del computer.
- Un **identificatore dell'oggetto** contiene l'indirizzo di memoria (chiamato riferimento) in cui l'oggetto è memorizzato nella **HEAP**.
- Le variabili di tipo **oggetto** sono dette di tipo **riferimento** (**reference type** o

puntatori). Un tipo **riferimento** è un qualsiasi tipo le cui variabili contengono indirizzi di memoria al posto dei valori degli elementi.

22) Descrivere i riferimenti

→ I **riferimenti** sarebbero i **puntatori**, un tipo di variabile in cui viene memorizzato l'indirizzo di memoria.

→ In alcuni linguaggi di programmazione, come **C**, **C++**, i **puntatori** vengono gestiti dal programmatore, perché considerati linguaggi di basso livello. **Java** e altri linguaggi di alto livello, i puntatori vengono, invece gestiti dal sistema. Infatti, in **Java**, quando si creano gli oggetti questi vengono allocati nell'area di memoria dinamica (**HEAP**).

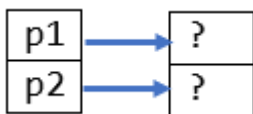
→ La differenza sostanziale tra **STACK** e **HEAP** è che lo **STACK** comporta un'allocazione lineare e sequenziale della memoria statica, mentre lo **HEAP** funge da pool di aree di archiviazione che allocano la memoria in modo casuale (allocazione dinamica della memoria).

→ La dimensione dello **HEAP** è pari a tutta la dimensione della memoria **RAM** libera sommato (grazie allo **swapping**) allo spazio libero su disco (**HDD**).

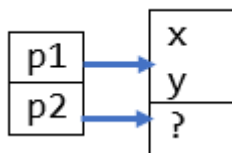
→ La **garbage collection** ("raccolta della spazzatura") è una tecnica che permette di liberare l'area di memoria dinamica. In alcuni linguaggi, come **Java** o **C#**, la **garbage collection** viene effettuata in automatico dal sistema (perché linguaggi di alto livello), mentre in altri (come **C**, **C++**) questa viene effettuata dal programmatore con l'istruzione **delete**.

23) Descrivere come funziona a livello pratico l'operatore =

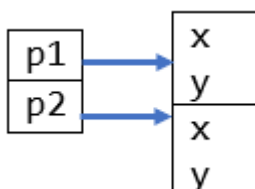
1) Punto p1, p2;



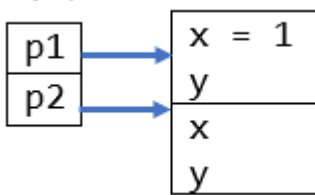
2) p1 = new Punto();



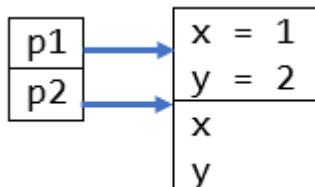
3) p2 = new Punto();



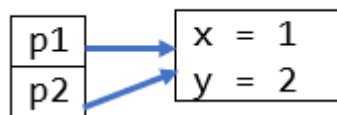
4) `p1.x = 1;`



5) `p1.y = 2;`



6) `p1 = p2;`



24) Descrivere il metodo main

Il metodo **main** è un metodo chiamato automaticamente al momento dell'esecuzione di un programma. Le variabili dichiarate all'interno di questo metodo sono allocate nello **stack** per tutto il tempo in cui il programma è in esecuzione.

25) Cosa si intende per null?

La **costante null** viene utilizzata per inizializzare un reference ad un valore di default. Viene usata anche per verificare se il **reference** è stato correttamente inizializzato dopo la dichiarazione.

Esempio:

```
Punto p1 = null;
if (!p1)
    p1 = new Punto();
else
    System.out.println("Oggetto istanziato!");
```

26) Cosa indica la keyword instanceof?

La **keyword instanceof** permette di verificare se un oggetto è stato istanziato a partire dalla classe specificata.

Esempio:

```
Punto p1 = new Punto();
if (p1 instanceof Punto)
    System.out.println("p1 referenzia un oggetto di tipo Punto");
```

Capitolo 4: Eclipse e Debug

Ambiente di sviluppo Java: Eclipse



27) Cosa identifica le sezioni Project e Workspace nell'ide di Eclipse?

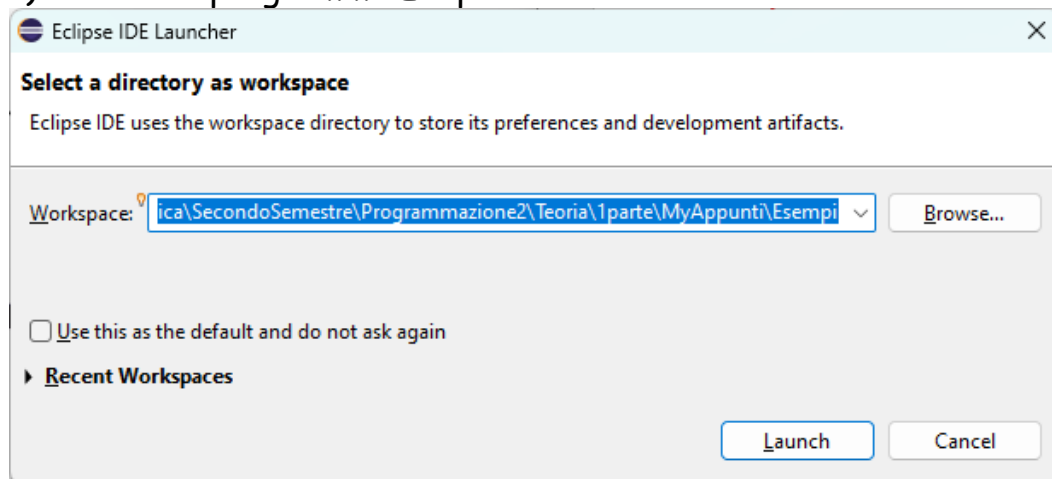
→ La sezione **Project** identifica l'insieme di file (**sorgenti**, **binari** e altri tipi di file) relativi allo sviluppo di un applicazione software.

→ La sezione **Workspace** rappresenta la cartella del file system in cui vengono memorizzati un **insieme di progetti (project) sviluppati attraverso Eclipse**. Ogni "project" corrisponde ad una specifica sotto-cartella dentro il "workspace".

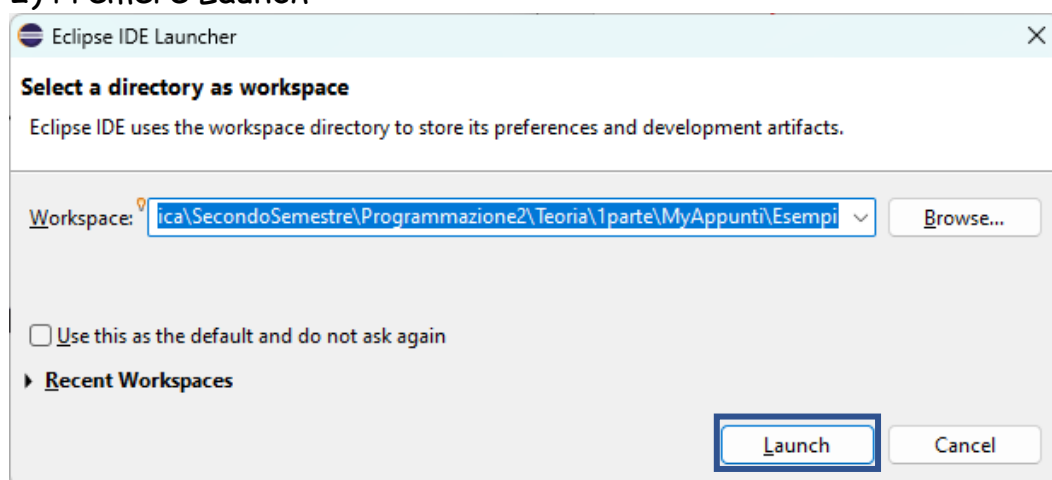
28) Descrivere la procedura di avvio di Eclipse

Per avviare Eclipse è necessarie eseguire il procedimento qui indicato:

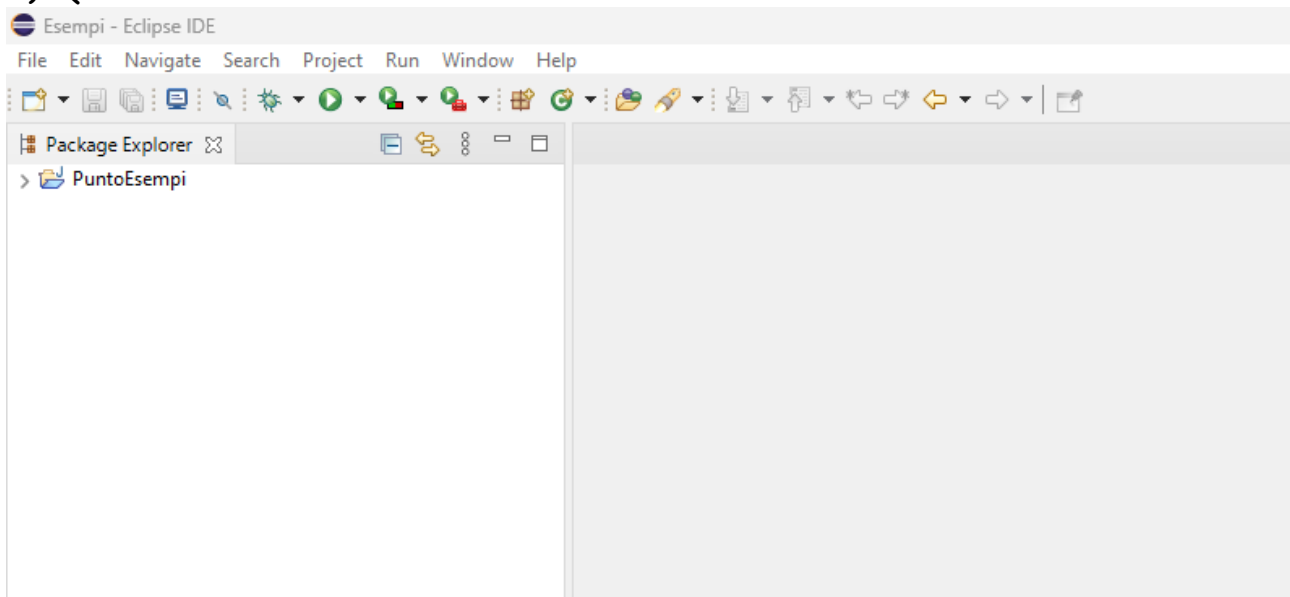
1) Avviare il programma Eclipse



2) Premere Launch

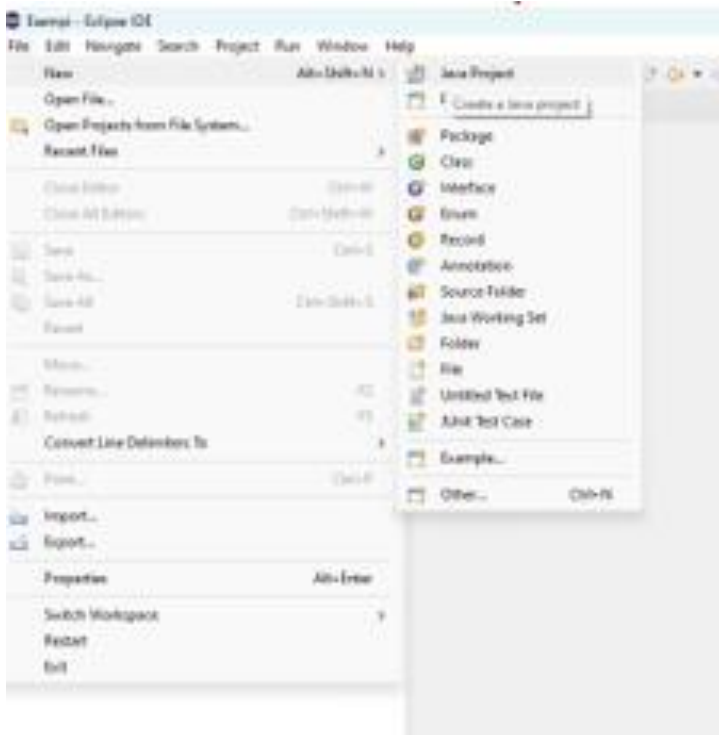


3) Questa è la schermata iniziale

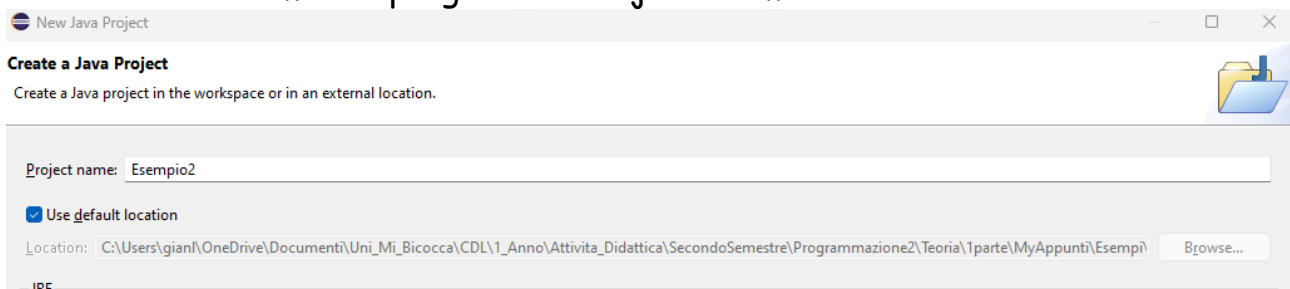


29) Descrivere la procedura di definizione di Project

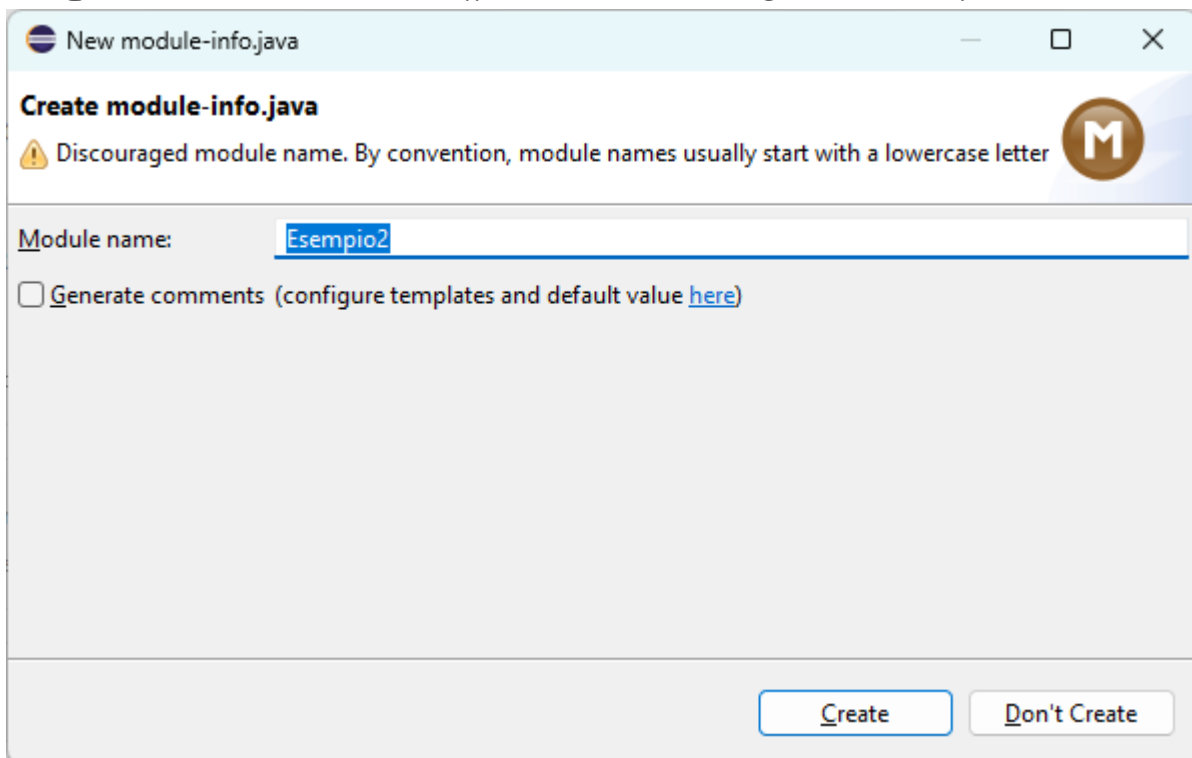
1 - Dalla sezione **File**, cliccare **New** e successivamente **Java Project**



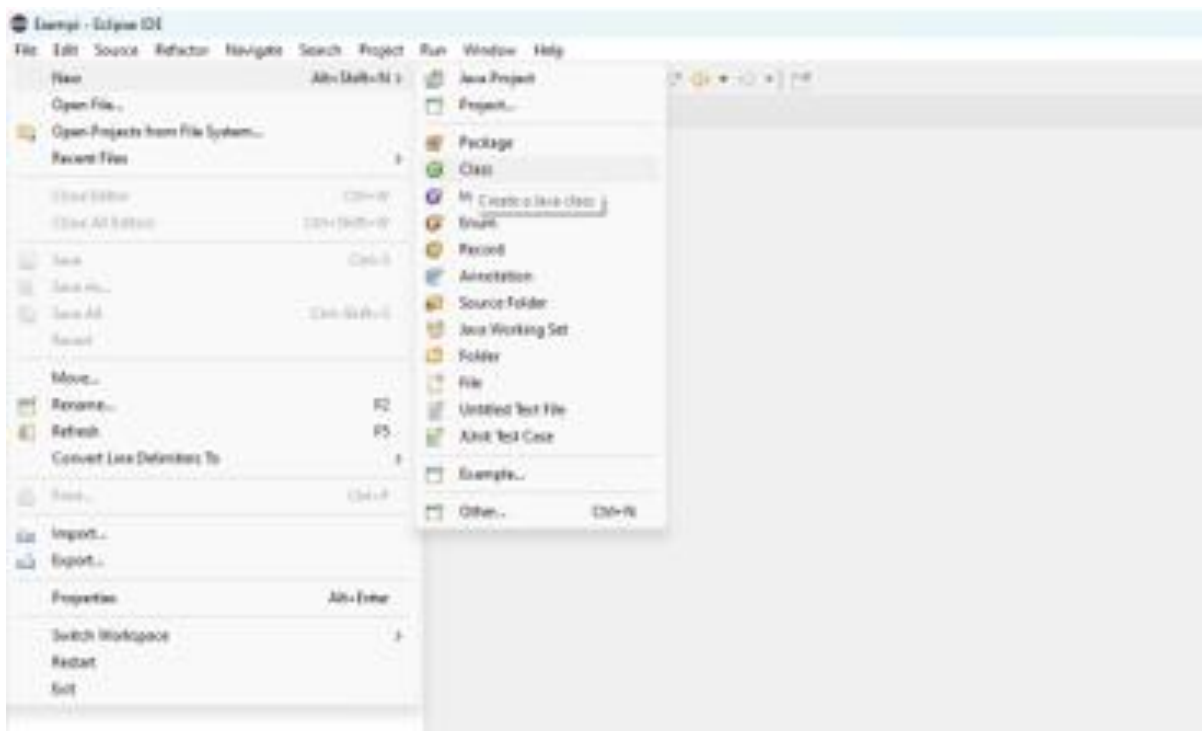
2 - Inserire il nome del progetto in Project Name



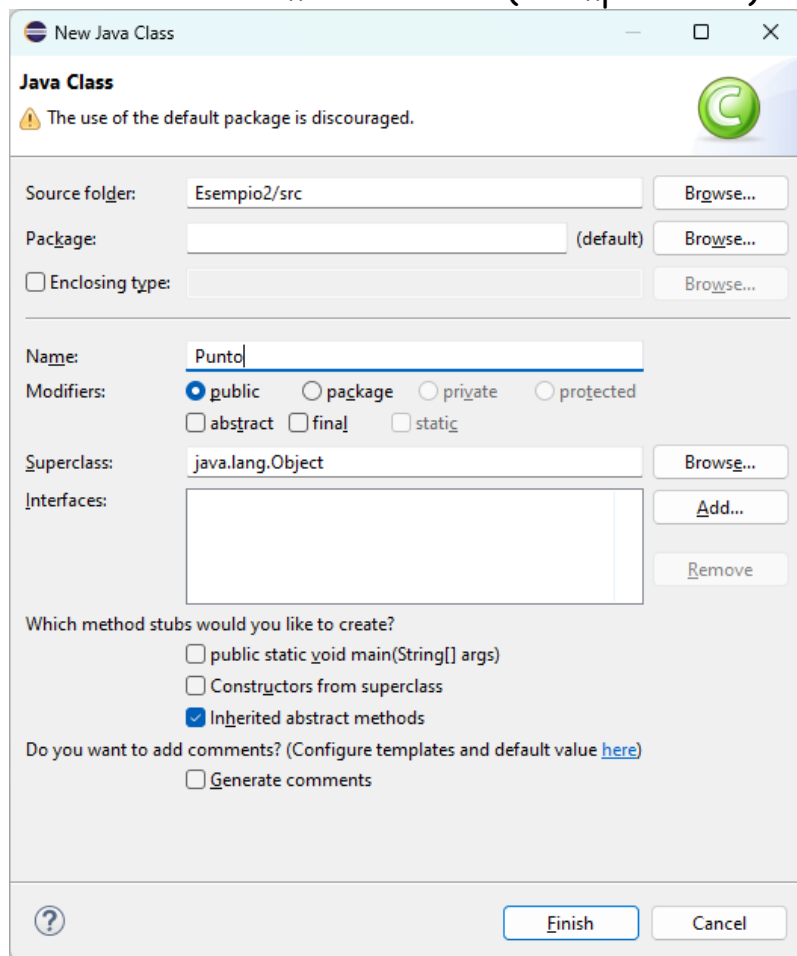
3 - Evitare la creazione del modulo attraverso Don't create.



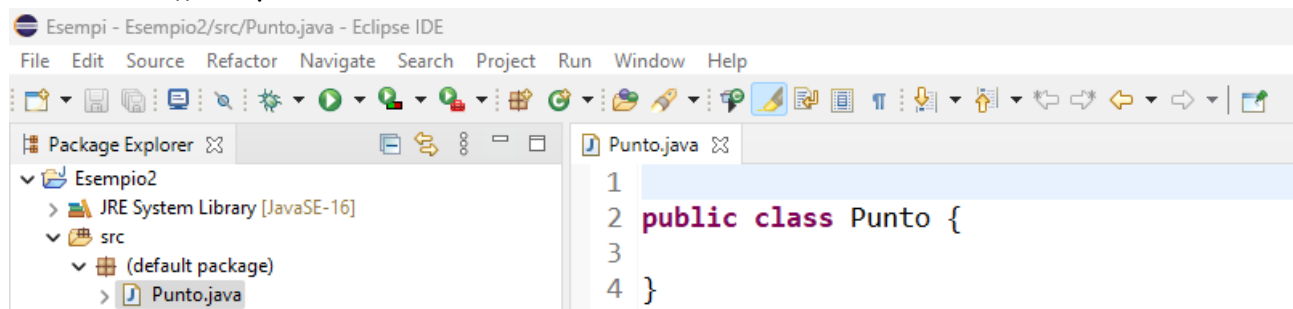
4 - Per creare una classe è necessario andare su File → New → Class



5 - Inserire il nome alla classe (esempio Punto)

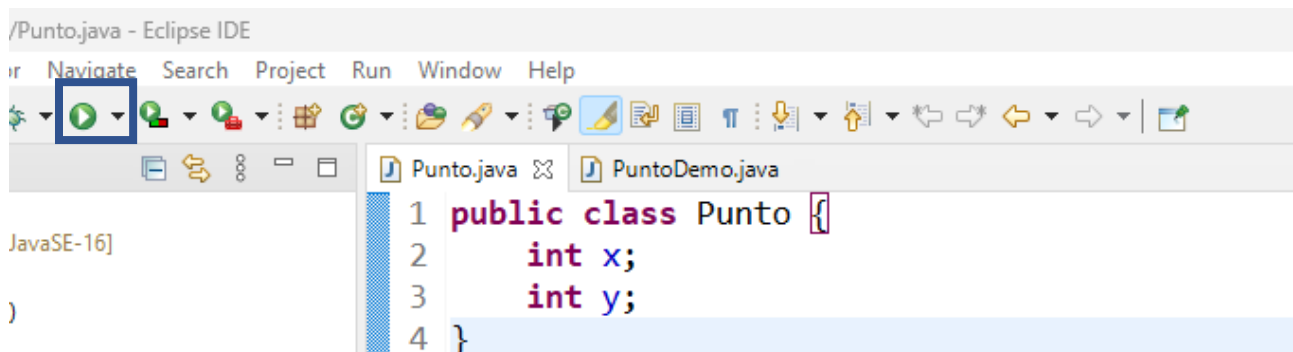


6 - Schermata finale



30) Come eseguire un programma Java su Eclipse?

Per mandare in esecuzione il programma Java su **Eclipse** è necessario premere il tasto **Run**




```

1 public class PuntoDemo {
2     public static void main(String[] args) {
3         Punto p1,p2;
4
5         System.out.println("Esempio 1 - Classi e Oggetti");
6         System.out.println("=====");
7
8         p1 = new Punto();
9         p2 = new Punto();
10
11         p1.x = 3;
12         p1.y = 4;
13         p2.x = 50;
14         p2.y = 90;
15
16         if (p1.x > p2.x)
17             System.out.println("Ha punto maggiore p1.x = " + p1.x);
18         else {
19             if (p1.x == p2.x)
20                 System.out.println("Sono uguali");
21             else
22                 System.out.println("Ha punto maggiore p2.x = " + p2.x);
23         }
24     }
25 }

```

Output

```

<terminated> PuntoDemo [Java Application] C:\Users\gianl\p2\pool\plugins\
Esempio 1 - Classi e Oggetti
=====
Ha punto maggiore p2.x = 50

```

31) Descrivere la procedura di import di una classe

Esistono due modalità per importare una classe:

→ copia;

→ import.

L'importazione della classe è composta dai seguenti passaggi:

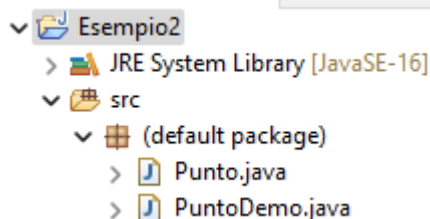
1)

Gianluca - Personale > Documenti > Uni_Mi_Bicocca > CDL > 1_Anno > Attività_Didattica > SecondoSemestre > Programmazione2 >					
<input type="checkbox"/> Nome	Stato	Ultima modifica	Tipo	Dimensione	
<input checked="" type="checkbox"/> Punto	✓	07/03/2023 17:49	File di origine Java	1 KB	
<input checked="" type="checkbox"/> PuntoDemo	✓	13/03/2023 22:30	File di origine Java	1 KB	

2)

Gianluca - Personale > Documenti > Uni_Mi_Bicocca > CDL > 1_Anno > Attività_Didattica > SecondoSemestre > Programmazione2 > Teoria > 1parte > MyAppunti > Esempi > Esempio2 > src					
<input type="checkbox"/> Nome	Stato	Ultima modifica	Tipo	Dimensione	
<input checked="" type="checkbox"/> Punto	✓	07/03/2023 17:49	File di origine Java	1 KB	
<input checked="" type="checkbox"/> PuntoDemo	✓	13/03/2023 22:30	File di origine Java	1 KB	

3)

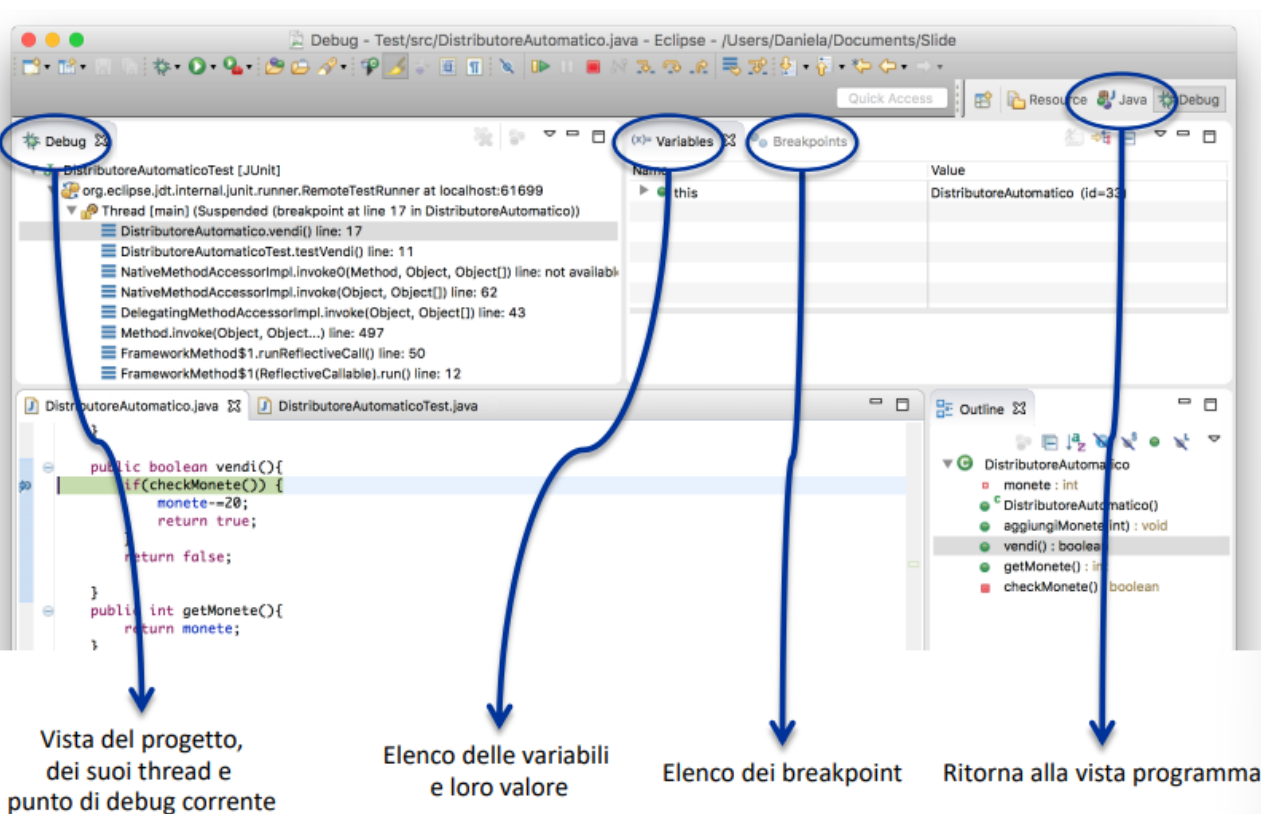
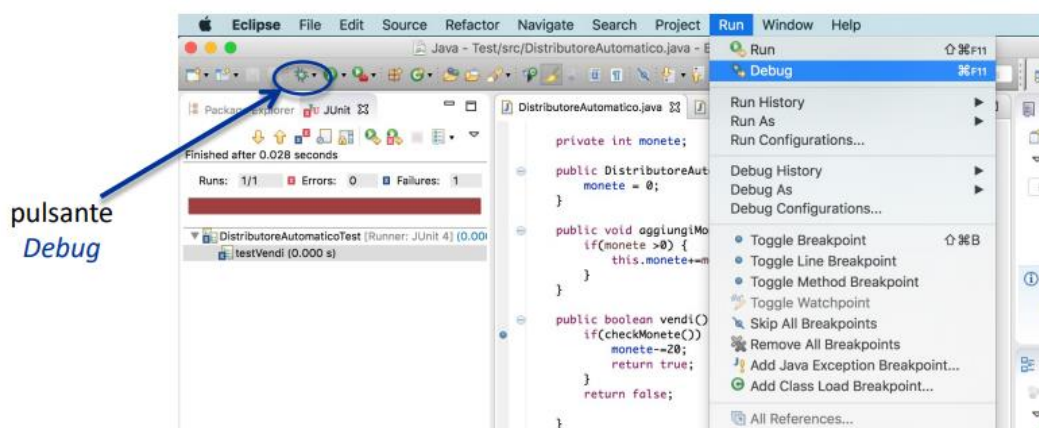


32) Cosa si intende per debugging e debugger?

Il **debugging** è il processo di individuare e risolvere errori o bug nel codice di origine di un software, mentre il debugger è uno strumento di sviluppo molto specializzato che si collega all'app in esecuzione e consente di controllare il codice.

33) Come effettuare il debug in Eclipse?

In Eclipse è possibile effettuare l'operazione di debug mediante il seguente procedimento:



Capitolo 5: Associazione

34) Cosa sono le associazioni?

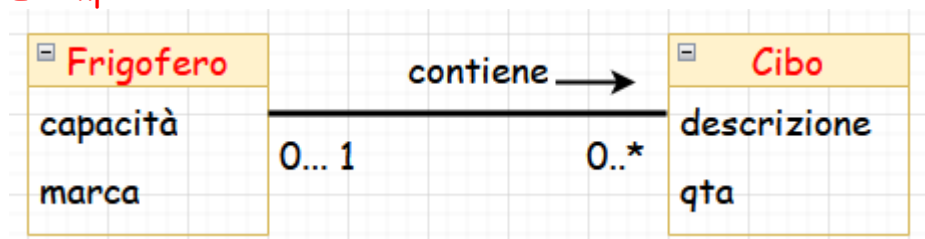
→ Le **associazioni** sono il mezzo in cui gli oggetti possono interagire e collegano classi.

→ Un'**associazione** è un legame semantico fra **classi**, ovvero che fra i corrispondenti oggetti c'è un legame (detto **link**).

→ Un **link** è un'istanza di associazione così come un oggetto è un'istanza di una classe.

→ Nonostante l'associazione sia bidirezionale, essa può essere resa unidirezionale.

Esempio 1:

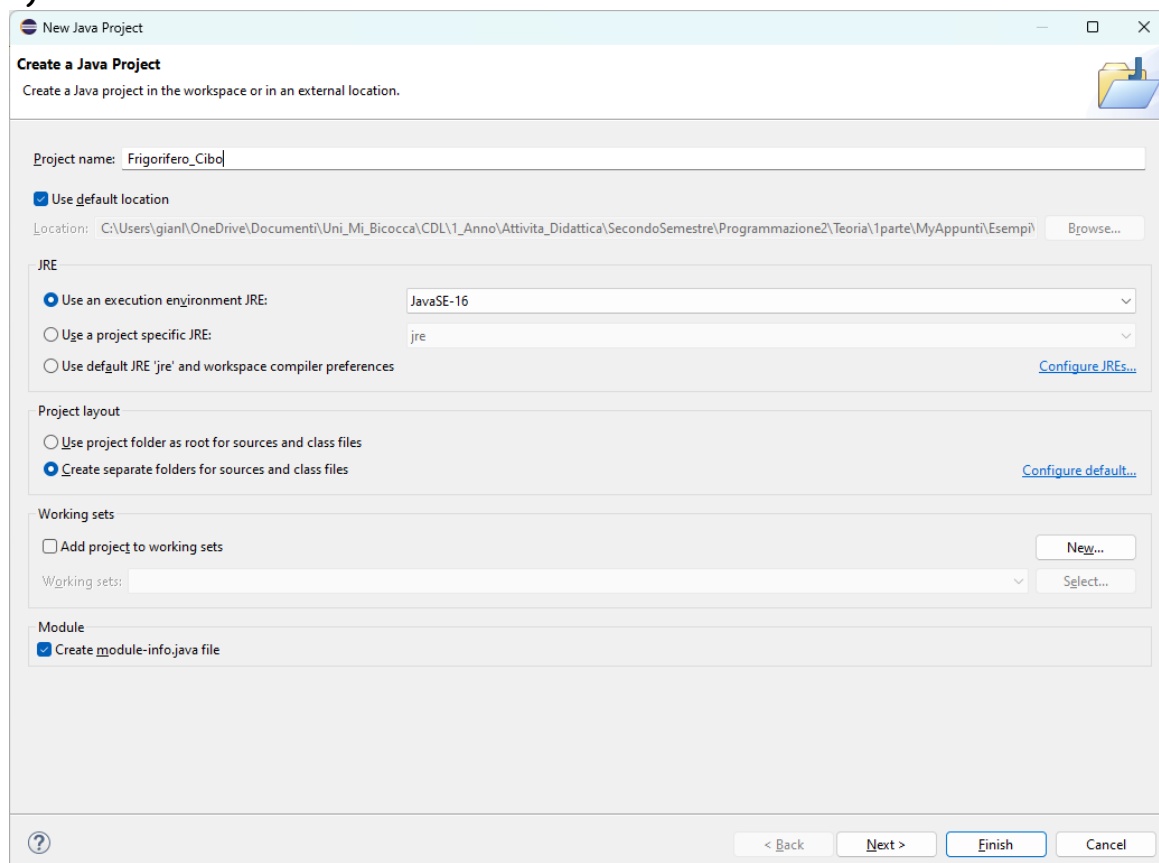


Vi è un'associazione tra **Frigorifero** e **Cibo**: ciò significa che fra ogni istanza della classe **Frigorifero** e **Cibo** c'è un link.

Oggetto : classe = link : associazione

35) Realizzazione

1)



2)

New Java Class

Java Class

⚠ The use of the default package is discouraged.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

3)

```
public class Cibo {  
    String descrizione;  
    int quantita;  
}
```

4)

```
public class Frigorifero {  
    String marca;  
    int capacita;  
    Cibo contenuto;  
}
```

5)

```
public class FrigoriferoDemo {
    public static void main(String[] args) {
        Cibo zucchina = new Cibo();
        Frigorifero f1 = new Frigorifero();

        System.out.println("Esempio 1 : Frigorifero");
        System.out.println("=====");

        zucchina.descrizione="Zucchina proveniente dall'Italia";
        zucchina.quantita=150;

        f1.capacita = 800;
        f1.marca = "Whirlpool";
        f1.contenuto = zucchina;

        System.out.println("Frigorifero - Dati");
        System.out.println("=====");
        System.out.println("ID = " + f1);
        System.out.println("Marca = " + f1.marca);
        System.out.println("Capacita = " + f1.capacita);
        System.out.println("Contenuto = " + f1.contenuto);
        System.out.println("Descrizione contenuto = " + f1.contenuto.descrizione);
        System.out.println("Descrizione quantita = " + f1.contenuto.quantita);

        System.out.println("=====");
        System.out.println("Cibo - Dati");
        System.out.println("=====");
        System.out.println("ID = " + zucchina);
        System.out.println("Descrizione contenuto = " + zucchina.descrizione);
        System.out.println("Descrizione quantita = " + zucchina.quantita);
        System.out.println("=====");
    }
}
```

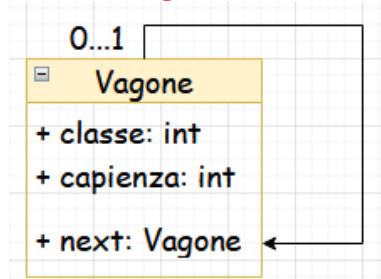
Output

```
Esempio 1 : Frigorifero
=====
Frigorifero - Dati
=====
ID = Frigorifero@156643d4
Marca = Whirlpool
Capacita = 800
Contenuto = Cibo@123a439b
Descrizione contenuto = Zucchina proveniente dall'Italia
Descrizione quantita = 150
=====
Cibo - Dati
=====
ID = Cibo@123a439b
Descrizione contenuto = Zucchina proveniente dall'Italia
Descrizione quantita = 150
=====
```

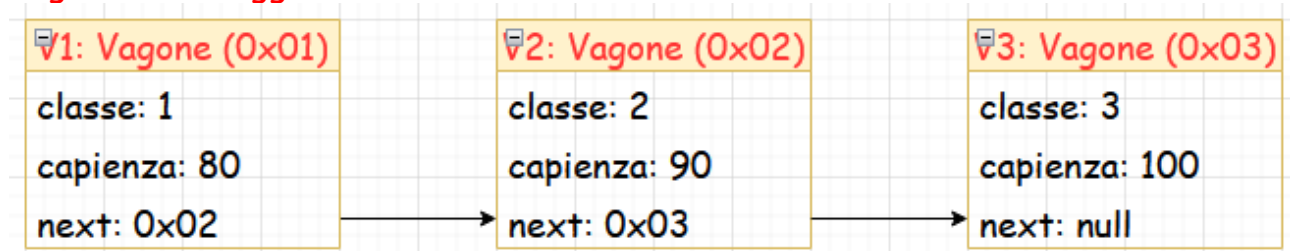
35) Come vengono implementate le associazioni a coppia?

Le associazioni a coppia vengono implementate utilizzando una struttura dati dinamica chiamata **lista**:

Classe Vagone



Vagoni treno: oggetti



Implementazione

```

public class Vagone {
    int classe;
    int capienza;
    Vagone next;
}
  
```

```

public class VagoneDemo {
    public static void main(String[] args) {
        Vagone v1 = new Vagone();
        Vagone v2 = new Vagone();
        Vagone v3 = new Vagone();

        System.out.println("Trenord");
        System.out.println("=====");

        v1.classe = 1;
        v1.capienza = 80;

        v2.classe = 2;
        v2.capienza = 90;

        v3.classe = 3;
        v3.capienza = 100;

        v1.next = v2;
        v2.next = v3;
        v3.next = null;
    }
}
  
```

```

System.out.println("Dettagli 1 vagone = v1");
System.out.println("=====");
System.out.println("ID Vagone 1 = " + v1);
System.out.println("Classe v1 = " + v1.classe);
System.out.println("Capienza v1 = " + v1.capienza);
System.out.println("Prossimo vagone = " + v1.next);
System.out.println("Prossimo prossimo vagone = " + v1.next.next);

System.out.println("=====");
System.out.println("Dettagli 2 vagone = v2");
System.out.println("=====");
System.out.println("ID Vagone 2 = " + v2);
System.out.println("Classe v2 = " + v2.classe);
System.out.println("Capienza v2 = " + v2.capienza);
System.out.println("Prossimo vagone = " + v2.next);

System.out.println("=====");
System.out.println("Dettagli 3 vagone = v3");
System.out.println("=====");
System.out.println("ID Vagone 3 = " + v3);
System.out.println("Classe v3 = " + v3.classe);
System.out.println("Capienza v3 = " + v3.capienza);
System.out.println("Prossimo vagone = " + v3.next);
System.out.println("=====");
}
}

```

Output

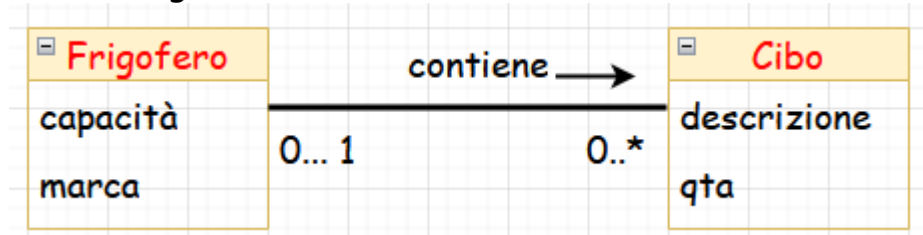
```

Trenord
=====
Dettagli 1 vagone = v1
=====
ID Vagone 1 = Vagone@5e265ba4
Classe v1 = 1
Capienza v1 = 80
Prossimo vagone = Vagone@156643d4
Prossimo prossimo vagone = Vagone@123a439b
=====
Dettagli 2 vagone = v2
=====
ID Vagone 2 = Vagone@156643d4
Classe v2 = 2
Capienza v2 = 90
Prossimo vagone = Vagone@123a439b
=====
Dettagli 3 vagone = v3
=====
ID Vagone 3 = Vagone@123a439b
Classe v3 = 3
Capienza v3 = 100
Prossimo vagone = null
=====

```


36) Se il frigorifero contenesse diversi cibi, come implementarlo?

Data la seguente situazione



È necessario utilizzare un **array di reference**.

L'implementazione è la seguente:

```

public class Cibo {
    String descrizione;
    int quantita;
}

public class Frigorifero {
    String marca;
    int capacita;
    Cibo contenuto;
}
  
```

Demo

```

import java.util.Scanner;

public class FrigoriferoDemo {
    public static void main(String[] args) {
        int quantoCiboinFrigo;
        Scanner in = new Scanner(System.in);
        Frigorifero f1 = new Frigorifero();

        System.out.println("Esempio 2 : Frigorifero con piu' cibo");
        System.out.println("=====");

        f1.capacita = 800;
        f1.marca = "Whirpool";

        System.out.print("Quanto cibo vi e' da inserire in frigo? (valore>0) ");
        quantoCiboinFrigo = in.nextInt();

        if (quantoCiboinFrigo<=0) {
            System.err.println("Non si accettano valori minori di 0");
            System.exit(-1);
        }
    }
}
  
```



```

else {
    Cibo[] food = new Cibo[quantoCiboinFrigo];

    for (int i=0;i<quantoCiboinFrigo;i++) {
        food[i] = new Cibo();
        System.out.println("=====");
        System.out.print("Descrivere il prodotto alimentare n.ro " + (i+1) + ": ");
        food[i].descrizione = in.next();
        System.out.print("Inserire la quantita' del prodotto alimentare n.ro " + (i+1) + ": ");
        food[i].quantita = in.nextInt();
    }

    System.out.println("=====");
    f1.contenuto = food;

    System.out.println("Frigorifero - Dati");
    System.out.println("=====");
    System.out.println("ID = " + f1);
    System.out.println("Marca = " + f1.marca);
    System.out.println("Capacita = " + f1.capacita);
    System.out.println("Contenuto = " + f1.contenuto);

    System.out.println("=====");
    System.out.println("Cibo - Dati");
    for (int i=0;i<quantoCiboinFrigo;i++) {
        System.out.println("=====");
        System.out.print("Descrizione prodotto alimentare n. " + (i+1) + ": ");
        System.out.println(food[i].descrizione);
        System.out.print("Quantita prodotto alimentare n. " + (i+1) + ": ");
        System.out.println(food[i].quantita);
    }
    System.out.println("=====");
}
}
}

```

Output

```

Esempio 2 : Frigorifero con piu' cibo
=====
Quanto cibo vi e' da inserire in frigo? (valore>0) 3
=====
Descrivere il prodotto alimentare n.ro 1: Carote
Inserire la quantita' del prodotto alimentare n.ro 1: 20
=====
Descrivere il prodotto alimentare n.ro 2: Zucchine
Inserire la quantita' del prodotto alimentare n.ro 2: 30
=====
Descrivere il prodotto alimentare n.ro 3: Peproni
Inserire la quantita' del prodotto alimentare n.ro 3: 40
=====
Frigorifero - Dati
=====
ID = Frigorifero@59a6e353
Marca = Whirpool
Capacita = 800
Contenuto = [LCibo;@7a0ac6e3
=====
Cibo - Dati
=====
Descrizione prodotto alimentare n. 1: Carote
Quantita prodotto alimentare n. 1: 20
=====
Descrizione prodotto alimentare n. 2: Zucchine
Quantita prodotto alimentare n. 2: 30
=====
Descrizione prodotto alimentare n. 3: Peproni
Quantita prodotto alimentare n. 3: 40
=====

```

Capitolo 6: Metodi di Istanza, Overloading, Costruttori, Incapsulamento

37) Descrivere le caratteristiche dei metodi

- Alcune sequenze di istruzioni possono dover essere ripetute più volte all'interno di un programma.
- Risulta quindi comodo poter scrivere tali sequenze una volta sola e poter far riferimento ad esse all'interno del programma tutte le volte che la loro esecuzione risulta necessaria.
- I **metodi** costituiscono lo strumento di programmazione che realizza quanto descritto.
- Il **metodo** è una porzione di codice riutilizzabile in diverse aree del programma. Raggruppa una sequenza di istruzioni che realizzano una funzionalità del programma e assegna loro un **nome**.
- Ogni qualvolta è necessario eseguire quella funzionalità, è sufficiente richiamarla attraverso il **nome**.
- Quando si utilizza un **metodo**, si dice che esso viene "**chiamato**" o "**invocato**".
- In **Java** esistono due tipologie di metodi:
 - 1) metodi che restituiscono un valore;
 - 2) metodi che eseguono delle istruzioni ma non restituiscono alcun valore (**void**).

38) Come si utilizzano i metodi?

- In primo luogo occorre definire il metodo scrivendo la sequenza di istruzioni e assegnando alla sequenza un nome.
- La definizione viene effettuata una sola volta e all'interno di una **classe**.
- Una volta definito il **metodo**, è possibile invocare il metodo usando il nome del **metodo**. Quando viene invocato un **metodo**, vengono eseguite le istruzioni definite al suo interno.
- Quando tutte le istruzioni del metodo sono state eseguite, l'esecuzione viene ripristinata nella posizione in cui era stata eseguita la chiamata al metodo.

39) Come vengono definiti i metodi?

- Un **metodo** è definito all'interno di una classe. Pertanto si dice che un metodo appartiene alla classe in cui è stato definito.
- Se un **metodo** viene definito **public** può essere invocato in ogni area del programma, anche in classi diverse da quelle in cui è stato definito.
- La parola chiave **static** è un altro modificatore che regola il modo con cui il metodo può essere invocato: **static** indica che è un metodo di classe.

40) Descrivere le caratteristiche dei metodi

- Un **metodo** di tipo **void**, viene utilizzato quando non si ha la necessità di restituire alcun tipo di valore. Dopo il termine **void**, il nome del metodo è seguito da una coppia di parentesi tonde (), dove all'interno delle quali **possono** essere elencati gli argomenti di cui il metodo necessita per poter eseguire il **corpo (body)** in esso definite.
- La definizione del metodo viene detta intestazione (**heading**) del metodo.
- Dopo l'intestazione viene riportata la parte rimanente della definizione di un metodo: il **corpo (body) del metodo**. Le istruzioni contenute nel corpo del metodo sono racchiuse tra parentesi graffe {}.
- Le variabili dichiarate all'interno del metodo vengono dette **variabili locali**.
- I metodi che restituiscono un valore vengono definiti in maniera simile ai metodi **void**, **con l'aggiunta della specifica del tipo di valore che restituiscono**. Il corpo della definizione di un metodo che restituisce un valore è come il corpo di un metodo **void**, con l'aggiunta dell'istruzione **return** al suo interno.

41) Come avviene l'invocazione del metodo?

- L'invocazione di un metodo **void** avviene semplicemente scrivendo un'istruzione che include il nome del metodo seguito da una coppia di parentesi e da un punto e virgola.
- Tra le parentesi è indicato il valore degli argomenti di cui il metodo necessita per eseguire le istruzioni in esso eseguite. Se il metodo invocato non richiede argomenti, tra le due parentesi non viene riportato nulla.
- Un **metodo** appartiene alla classe in cui è definito. Esso può essere invocato all'interno di altri metodi definiti nella sua stessa classe. Se il modificatore del metodo è di tipo **public**, il metodo può essere invocato anche al di fuori della classe in cui è stato definito.
- Un **metodo** che restituisce un **valore**, lo si può invocare in qualsiasi punto del codice in cui si potrebbe usare un elemento dello stesso tipo di ritorno del metodo.

42) Definizione di variabili locali

- Una **variabile locale** è una variabile dichiarata all'interno di un metodo e possono essere utilizzate solamente all'interno del metodo. Anche le variabili dichiarate all'interno del **main** sono locali al **main**.
- Le **variabili locali** aventi lo stesso nome, ma dichiarate in metodi diversi sono variabili a tutti gli effetti differenti (**non sono la stessa variabile**).

43) Descrizione dei metodi

- Tutti i metodi devono essere descritti e la descrizione dovrebbe contenere:
 - a) le **precondizioni**: indicano le condizioni che devono essere vere prima di invocare il metodo, che non deve essere invocato se le precondizioni non sono verificate (potrebbe fornire dei risultati non attesi).
 - b) le **postcondizioni**: descrivono gli effetti prodotti dall'invocazione del metodo.
- **Java** fornisce un comando **javadoc** per generare file in formato HTML che descrivono l'interfaccia pubblica della classe, esprimendo le precondizioni e le postcondizioni in maniera un po' più formale.

44) Definizione di Incapsulamento

- L'**incapsulamento** è una proprietà della programmazione orientata agli oggetti che permette di separare l'interfaccia dall'implementazione.
- l'**interfaccia** contiene ciò che è visibile all'esterno;
- l'**implementazione** contiene tutti i dettagli implementativi che si vogliono nascondere a chi utilizza la classe incapsulata.

45) Vantaggi incapsulamento

I vantaggi dell'**incapsulamento** sono:

- riuso del codice: il programmatore deve solo conoscere cosa il metodo fa e non come lo fa;
- possibilità di modificare l'implementazione senza che le classi che utilizzano quella modificata non se ne accorgano;
- stato dell'oggetto sempre consistente.

46) Come realizzare l'incapsulamento?

Per realizzare l'incapsulamento è necessario:

- distinguere fra interfaccia e implementazione: gli attributi di istanza sono un dettaglio implementativo, il significato è dato dalle operazioni che li manipolano;
- permettere solo stati validi per gli oggetti: lo stato degli oggetti deve contenere sempre valori validi.

47) Differenza tra modificatore private e public

- Un attributo di istanza di tipo **public** è visibile da qualsiasi parte del programma, mentre un attributo di istanza di tipo **private** è visibile solo all'interno della classe stessa.
- Dichiarare un'attributo d'istanza **private** comporta che è accessibile esclusivamente all'interno della classe in cui è definito.

→ Dichiarare gli attributi di istanza *private* obbliga a chi definisca la classe di dotarla di metodi che permettono l'accesso agli attributi stessi.

→ Un metodo di incapsulamento è semplicemente un metodo che permette ad un'altra classe di leggere o impostare il valore di un attributo di istanza dichiarato *private*.

48) Naming metodi

I metodi di incapsulamento seguono più o meno la seguente convenzione:

→ in modifica: `setNomeAttributo`;

→ in lettura: `getNomeAttributo`;

49) Descrivere la keyword *this*

La *keyword this* si riferisce all'oggetto corrente nel metodo o costruttore.

Viene utilizzato per eliminare la possibile "confusione" tra l'attributo di classe e il parametro con lo stesso nome.

50) Definizione formale di *overloading*

Operare l'*overloading* di un metodo vuol dire assegnare lo stesso nome a due o più definizioni differenti all'interno della stessa classe.

51) Come fa Java a sapere quale definizione deve eseguire?

Java determina la definizione corretta sulla base del numero e del tipo di argomenti.

52) Quando e perché si fa *overloading*? Non è più semplice definire metodi con nomi diversi?

Quando diversi metodi semanticamente fanno la stessa cosa ma con parametri in ingresso diversi. Non ha senso modificare il nome.

53) Definizione di costruttore

→ Un *costruttore* è uno speciale tipo di metodo destinato ad effettuare le inizializzazioni dell'oggetto.

→ Un *costruttore* permette di creare oggetti inizializzandoli a valori desiderati.

→ Il *costruttore* è un metodo speciale che viene invocato nel momento in cui viene creato un oggetto.

54) Descrivere le caratteristiche del costruttore

Le caratteristiche del metodo sono:

→ un costruttore esegue tutte le istruzioni specificate nella sua definizione;
→ quando si scrive un costruttore, occorre però ricordarsi che il suo scopo è quello di eseguire le azioni necessarie all'inizializzazione dell'oggetto;

→ viene dichiarato e invocato nella maniera seguente:

```
public NomeClasse([lista parametri formali]) {  
    //inizializzazioni  
}
```

```
NomeClasse ref = new NomeClasse([argomenti])
```

→ se non viene definito un costruttore all'interno di una classe, ne viene assegnato uno di default (costruttore di default) che non accetta in ingresso parametri e la cui implementazione è più o meno la seguente:

```
public class Automobile {  
    ...  
    public Automobile() {  
    }  
}
```

55) Overloading costruttori

Overloadare i costruttori vuol dire realizzare più costruttori che inizializzano l'oggetto variando la lista dei parametri formali. Nei costruttori in cui non viene richiesto il valore di inizializzazione di uno più attributi, devono, nell'implementazione, comunque inizializzare tali attributi.