Metodologie di Programmazione

Simone Lidonnici

29 aprile 2024

Indice

1

Object-Oriented Programming

La programmazione ad oggetti è un modo di scrivere codice modulare e riutilizzabile utilizzando oggetti che interagiscono tra loro tramite delle relazioni di interdipendenza:

- **Composizione**: un oggetto è composto da tanti oggetti di un altro tipo (oggetto Libro composto da tanti oggetti Capitolo)
- **Specializzazione**: un oggetto è un tipo più specifico di un altro oggetto (oggetto Macchina è un tipo specifico rispetto all'oggetto Veicolo)
- **Utilizzo**: un oggetto ha bisogno di un altro oggetto per funzionare (oggetto LettoreDVD ha bisogno di un oggetto DVD)

Il linguaggio di programmazione che rappresenta meglio questo metodo di scrittura di codice è **Java**.

1.1 Classe

Una classe è un tipo di dato che rappresenta un oggetto che possiede determinate caratteristiche (attributi) e svolge determinate operazioni (metodi). Le classi devono rappesentare un singolo concetto (coesione).

1.1.1 Modificatori di visibilità

Determinano quali altre classi possono accedere ai vari metodi e attributi di una classe:

Modificatore	Classe stessa	Package	Sottoclassi	Tutti
public	✓	✓	✓	✓
protected	✓	✓	✓	
default	✓	✓		
private	✓			

Se non viene messo nessun modificatore viene assegnato automaticamente quello di default.

1.1.2 Modificatori static e final

I modificatori **static** e **final** non servono per indicare da chi può essere visto un attributo o metodo ma per creare costanti o metodi non sovrascrivibili.
Modificatore static:

- Attributo: l'attributo diventa condiviso da tutte le istanza di quella classe
- Metodo: permette di chiamare il metodo senza dover istanziare un oggetto di quella classe

Modificatore final:

- Attributo: l'attributo diventa una costante
- Metodo: non permette al metodo di essere sovracritto
- Classe: non permette alla classe di avere Sottoclassi

1.1.3 Metodi

I metodi possono essere di due tipi:

- Metodo accessor: svolge delle funzioni sull'oggetto senza modificarlo (i metodi get)
- Metodo mutator: modifica l'oggetto (i metodi set)

Una classe si dice immutabile se non ha metodi mutator (la classe String).

1.2 Principi SOLID

I principi SOLID sono dei principi da seguire quando si programma con il metodo object oriented e sono 5, rappresentati dalla sigla:

1. Single responsability principle: Ogni classe deve avere una sola responsabilità e ogni compito va affidato ad un'istanza diversa.
2. O: Open/closed principle: Un software deve essere aperto alle estenzioni ma chiuso alle modifiche, quindi non deve avere attributi troppo specifici che ne impediscono l'estenzione.
3. Liskov substitution principle: Una classe deve poter essere sostituita da una sua sotto-classe senza alterare il programma.
4. Interface segregation principle: Si preferiscono interfacce specifiche rispetto a interfacce generiche.
5. Dependancy inversion principle: Una classe deve dipendere dalle astrazioni, il codice di una classe deve essere scritto basandosi sulla superclasse.

1.3 Java code convention

La java code convention impone che vengano rispettate delle regole per i nomi di classi, variabili e metodi:

- Classi: PascalCase (Persona, PersonaDisoccupata)
- Metodi: camelCase (getVariabile)
- Attributi: camelCase (longVariable)
- Attributi costanti: tutto maiuscolo (MAX_HEIGHT)

2

Ereditarietà

L'ereditarietà permette di creare una sottoclasse che eredita tutti i metodi e gli attributi della superclasse che estende. La sottoclasse può poi aggiungere altri metodi o sovrascrive quelli ereditati.

2.1 Polimorfismo

Il polimorfismo permette ad un'espressione di assumere più forme, per esempio si può usare un oggetto di una classe come se fosse un oggetto della superclasse. Si applica tramite l'**overriding** e l'**overloading**.

2.1.1 Overriding

Sovrascrittura di un metodo ereditato dalla superclasse, con lo stesso nome, return e tipi di dati in input.
Esempio:

```
public class Square{
    private double base;

    public Square(double b){
        base=b;
    }

    public void getInfo(){
        System.out.println("base:"+base);
    }
}
```

```
public class ColoredSquare extends Square{
    private String color;

    public ColoredSquare(double b, String c){
        super(b);
        color=c;
    }

    public void getInfo(){
        super.getInfo();
        Sysem.out.println("color:"+color);
    }
}
```

2.1.2 Overloading

Creazione di più metodi con lo stesso nome con però dati in input e return differenti.
Esempio:

```
public class Calculator{
    public int somma(int a, int b){
        return a+b;
    }
    public int somma(int a, int b, int c){
        return a+b+c;
    }
}
```

3

Interfacce

Un'interfaccia è un dato astratto che viene utilizzato per definire il comportamento di più classi. Ha una struttura simile ad una classe ma contiene metodi astratti che devono essere per forza sovrascritti dalla classe cha la implementa, non ha un costruttore e non può essere istanziata.
Esempio:

```
public interface Measurable{
    double getMeasure();
}


public class Rectangle implements Measurable{
    private double base;
    private double height;

    public double getMeasure(){
        return base*height;
    }
}
```

4

Eccezioni

Le eccezioni sono degli errori nel programma che si verificano durante l'esecuzione, in java alcune di queste eccezioni possono essere catturate e gestite senza far interrompere il programma. Le eccezioni sono di due tipi, controllate e non controllate.

4.1 Eccezioni controllate

Le eccezioni controllate sono errori dovuti a circostanze esterne che il programmatore non può evitare, ma può controllare tramite un try-catch. Si inserisce nel try la parte che può generare un errore e nel catch il tipo di errore e come va gestito.
Esempio:

```
try{
    File f = new File("path");
}catch(FileNotFoundException e){
    System.out.println("File not found");
}
```

Le eccezioni possono anche essere lanciate, cioè si segnala tramite throws il tipo di eccezione che potrebbe essere generata e se succede viene interrotto il metodo.

```
public void read() throws FileNotFounfException{
    File f = new File("path");
}
```

4.2 Eccezioni non controllate

Le eccezioni non controllate sono errori dovuti al programmatore, che non possono essere gestite, come se si controlla un indice di un array inesistente.

```
ArrayList<String> array = new ArrayList<String>();
array.add("ciao");
System.out.println(array[1]);
```

5
Insiemi di dati

5.1 Array

Gli array sono insiemi di elementi ordinati statiche (non possono cambiare grandezza) in cui gli elementi non sono indicizzati. Non implementano nessuna interfaccia.

5.2 List

Le liste sono insiemi di elementi ordinati che possono contenere ripetizioni e ce ne sono di diversi tipi. Implementano l'interfaccia List.
Metodi che possono essere usati sulle List:

- `add(e)`: aggiunge l'elemento alla fine della List
- `remove(e)`: elimina l'elemento dalla List
- `size()`: ritorna il numero di elementi
- `for(Object e:lista)`: permette di iterare per ogni oggetto della List

Ci sono diversi tipi specifici di List.

5.2.1 ArrayList

L'ArrayList è una List con delle caratteristiche specifiche:

- Struttura dinamica
- Accesso agli elemeni idicizzato
- Possibile aggiungere ed eliminare elementi dovunque

5.2.2 LinkedList

La LinkedList è una List con delle caratteristiche specifiche:

- Struttura dinamica
- Accesso agli elemeni sequenziale (ogni elemento ha il collegamento con quello dopo)
- Possibile aggiungere ed eliminare elementi solo all'inizio o alla fine

Implementano anche l'interfaccia Queue oltre a quella List.
Metodi che possono essere usati sulle LinkedList:

- `addFirst(e)`: aggiunge l'elemento all'inizio della LinkedList
- `addLast(e)`: aggiunge l'elemento ala fine della LinkedList
- `removeFirst()`: elimina l'elemento all'inizio della LinkedList
- `removeLast()`: elimina l'elemento ala fine della LinkedList

5.2.3 Stack

Lo Stack è una List con delle caratteristiche specifiche:

- Struttura dinamica
- Accesso solo al primo elemento
- Possibile aggiungere ed eliminare elementi solo all'inizio secondo una politico FILO

Metodi che possono essere usati sugli Stack:

- `pop()`: elimina l'elemento in cima allo Stack
- `push(e)`: aggiunge l'elemento in cima allo Stack

5.3 Set

I set sono insiemi disordinati di elementi unici, che implementano l'interfaccia Set e possono essere di due tipi in base alla loro implementazione:

- **HashSet**: implementati tramite una tabella hash
- **TreeSet**: implementati attraverso un albero binario di ricerca

Metodi che possono essere usati sui Set:

- `add(e)`: aggiunge l'elemento al Set
- `remove(e)`: elimina l'elemento dal Set
- `size()`: ritorna il numero di elementi
- `for(Object e:set)`: permette di iterare per ogni oggetto del Set

5.4 Map

Le map sono insiemi disordinati di coppie chiave-valore in cui le chiavi devono essere tutte diverse tra loro. Implementano l'interfaccia Map e possono essere implementate in due modi:

- **HashMap**: implementate tramite una tabella hash
- **TreeMap**: implementate attraverso un albero binario di ricerca

Metodi che possono essere usati sulle Map:

- `put(key,value)`: aggiunge alla map la coppia chiave-valore, se si passa una chiave già presente verrà modificato il valore
- `get(key)`: ritorna il valore associato alla chiave
- `remove(key)`: elimina la coppia chiave-valore
- `keySet()`: ritorna un Set contenente tutte le chiavi