Come funziona la programmazione O-O

La programmazione orientata agli oggetti utilizza i concetti di **astrazione** e di **information hiding**

Astrazione: la visione di un oggetto che si concentra su informazioni che sono utili all'implementazione di un programma ignorando tutte le altre che non sono necessarie. Quindi si ha una semplificazione di una certa realtà, andando a considerare le sole caratteristiche utili alla risoluzione di un problema. Una stessa realtà può avere astrazioni diverse a seconda del problema da risolvere.

Information hiding: tecnica di sviluppo software in cui ciascuna interfaccia di ciascun modulo (sottoprogramma) rivela il meno possibile su come il modulo lavora. Nella programmazione O-O il modulo è rappresentato dalle classi.

Il concetto di astrazione è strettamente legato ai modelli, cioè ad una semplificazione di una realtà. Gli elementi di un modello sono gli **oggetti**, i quali appartengo ad una classe e ne costituiscono un'istanza, mentre le caratteristiche sono i dati i quali sono passivi e di solito privati (variabili d'istanza). Quando vengono assegnati i dati, allora viene creato un oggetto di quella classe (è il concetto del costruttore, cioè un metodo che non ha nessun valore di ritorno)

Una categoria di oggetti forma una **classe.** Una classe descrive i comportamenti (i metodi in Java) comuni a tutti gli oggetti che ne fanno parte

Gli oggetti sono elementi attivi di una classe, perché eseguono delle azioni. Essi sono attivati dalla ricezione di un messaggio. I messaggi a cui un oggetto può rispondere vengono determinati da una classe attraverso la dichiarazione di metodi

Le classi hanno un nome, mentre gli oggetti no, ma sono identificati da riferimenti (reference)

Come funziona Java

Java è un linguaggio di programmazione orientato agli oggetti che viene utilizzato per sviluppare applicazioni per diverse piattaforme, come desktop, mobile, web e cloud.

Quando si scrive un codice Java, si creano classi che contengono variabili e metodi. Le variabili rappresentano gli stati di un oggetto mentre i metodi rappresentano le azioni che un oggetto può compiere. Una volta che le classi sono state create, è possibile creare oggetti di quelle classi, ognuno con le proprie variabili e metodi.

Java utilizza un compilatore per convertire il codice sorgente in bytecode, che viene eseguito dalla Java Virtual Machine (JVM). La JVM è presente sulla maggior parte dei sistemi operativi e garantisce che il codice Java viene eseguito in modo uniforme su diverse piattaforme.

Java è sia compilato che interpretato. Nel processo di traduzione un codice viene prima **compilato** e si genera un file scritto in un linguaggio intermedio che è il JAVA BYTECODE il quale è codice binario ma non linguaggio macchina. Successivamente interviene la JAVA VIRTUAL MACHINE, specifica per la macchina utilizzata, che **interpreta** il bytecode (lo traduce in linguaggio macchina) e lo esegue.

- **Java Virtual Machine (JVM)**: la JVM è un'implementazione di Java che esegue il byte code Java su una specifica piattaforma.
- **Java Development Kit (JDK)**: il JDK è un insieme di strumenti che include il compilatore Java e altre utility necessarie per sviluppare applicazioni Java.

- **Java Runtime Environment (JRE)**: il JRE è un sottoinsieme del JDK che include solo la JVM e le librerie Java necessarie per eseguire applicazioni Java.
- **Classi**: le classi sono i modelli per gli oggetti in Java e possono contenere variabili e metodi.
- **Oggetti**: gli oggetti sono istanze di una classe e possono avere le proprie variabili e metodi.
- **Variabili**: le variabili sono usate per memorizzare gli stati di un oggetto.
- **Metodi**: i metodi sono usati per rappresentare le azioni che un oggetto può compiere.

Metodi

Un metodo è un comportamento che può avere un oggetto di una classe. Due oggetti comunicano tra loro attraverso l'invio e ricezione di messaggi, quindi attraverso i metodi
Esistono due tipi di metodi:

- **ACCESSORI**: NON modificano lo stato del parametro implicito Esempio: esempio il calcolo della media dei valori nel caso in cui la media non sia una variabile d'istanza

- **MUTUATORI**: modificano lo stato del parametro implicito Esempio: un filtro invocato da un oggetto di una classe gestore

Casting

Il casting è la capacità di trattare un oggetto di una classe come se fosse di un'altra classe. Ad esempio:

```
Animale a = new Gatto();
Gatto g = (Gatto) a;
```

In questo esempio, l'oggetto **a** di tipo **Animale** viene trattato come se fosse di tipo **Gatto** e viene assegnato alla variabile **g** di tipo **Gatto**.
Il casting può essere utilizzato anche per trattare un oggetto di una classe come se fosse di un tipo primitivo, ad esempio:

```
int i = (int) 3.14;
```

In questo caso, il valore **3.14** viene trattato come se fosse di tipo intero e viene assegnato alla variabile **i**.

Specificatori di accesso

Private

- Indica che una risorsa è accessibile solo all'interno della classe in cui viene dichiarato
- Inaccessibile da altre classi o sottoclassi
- Utilizzato per nascondere l'implementazione dei dettagli di una classe

- Esempio: private int numero; (la variabile di istanza numero è privata e può essere utilizzata solo all'interno della classe in cui viene dichiarata)

Protected

- Indica che una risorsa è accessibile solo all'interno della classe in cui viene dichiarato e dalle sue sottoclassi
- Inaccessibile da altre classi
- Utilizzato per consentire l'accesso limitato ai membri di una classe da parte delle sue sottoclassi
- Gli elementi dichiarati protected sono accessibili da classi che sono presenti nello stesso package della classe padre e figlio. Inoltre, gli elementi protected sono accessibili anche dalle sottoclassi della classe padre, indipendentemente dal package in cui si trovano.
- Esempio: protected String nome; (la variabile di istanza nome è protetta e può essere utilizzata solo all'interno della classe in cui viene dichiarata o dalle sue sottoclassi)

Public

- Indica che una risorsa è accessibile da qualsiasi parte del codice
- Utilizzato per rendere visibili i membri di una classe a tutto il programma
- Esempio: public static final double PI = 3.141592653589793; (la costante PI è pubblica e può essere utilizzata da qualsiasi parte del programma)

Se non si specifica nessun specificatore di accesso per un elemento di una classe in Java (ad esempio, un campo o un metodo), allora l'elemento sarà considerato "package-private" di default. Ciò significa che l'elemento sarà accessibile solo all'interno del package in cui è definito.

Static e abstract

Static e **abstract** non sono specificatori di accesso in Java.

Cosa significa che un membro di classe è statico?

Significa che appartiene alla classe stessa e non alle istanze della classe, e può essere utilizzato indipendentemente dall'esistenza di istanze della classe.

Qual è l'effetto del modificatore di accesso static sulla visibilità di un membro di classe?

Non ha alcun effetto sulla visibilità del membro di classe. La visibilità di un membro statico è determinata dallo specificatore di accesso (ad esempio, public, private, etc.).

Qual è la differenza principale tra i membri di classe statici e i membri di istanza?

I membri di classe statici appartengono alla classe stessa e non alle istanze della classe, mentre i membri di istanza appartengono alle istanze della classe. Inoltre, i membri di classe statici possono essere utilizzati indipendentemente dall'esistenza di istanze della classe, mentre i membri di istanza possono essere utilizzati solo a seguito della creazione di un'istanza della classe.

Posso accedere ai membri di istanza all'interno di un metodo statico?

No, i metodi statici possono accedere solo ai membri di classe statici e non ai membri di istanza. Se si tenta di accedere a un membro di istanza all'interno di un metodo statico, il compilatore segnalerà un errore.

Posso invocare un metodo non statico utilizzando il nome della classe anziché l'istanza della classe?

No, i metodi non statici possono essere invocati solo a seguito della creazione di un'istanza della classe e utilizzando l'istanza stessa. Tentare di invocare un metodo non statico utilizzando il nome della classe genererà un errore del compilatore.

Abstract è un modificatore di classe che viene utilizzato per indicare che una classe è astratta. Una classe astratta è una classe che non può essere istanziata e deve essere estesa da una sottoclasse concreta. Una classe astratta contiene almeno un metodo astratto, cioè un metodo senza implementazione che deve essere implementato dalle sottoclassi.

Garbage Collector

Il garbage collector (GC) di Java è un componente del sistema di runtime Java che si occupa di gestire l'allocazione e la liberazione della memoria. Quando un oggetto non viene più utilizzato nell'applicazione, il garbage collector può liberare la memoria occupata dall'oggetto per poterla riutilizzare in seguito.

In particolare il GC vede gli oggetti non più raggiungibili, cioè che non sono più referenziati, e recupera la memoria occupata da questi

Il periodo di vita (scope) dell'oggetto inizia con la creazione e continua ad esistere finché c'è una variabile di riferimento che si riferisce ad esso

Read process loop pattern

Sono cicli del tipo:

```
leggi il primo oggetto
while(un oggetto valido è stato letto){
  esegui delle operazioni con l'oggetto
  leggi il prossimo oggetto
}
```

Questo tipo di ciclo viene chiamato **read process loop pattern**

Viene utilizzato nelle classi gestore per popolare una collezione di una classe base

Per usare questo pattern è necessario definire:

- il metodo per leggere l'oggetto
- la condizione che indica che è stato letto un oggetto valido
- il codice per elaborare l'oggetto

Responsabilità delle classi

La responsabilità di una classe può essere definita come l'insieme di operazioni che la classe è in grado di eseguire e di informazioni che è in grado di gestire. Ad esempio, una classe "Libro" potrebbe essere responsabile di gestire informazioni come il titolo, l'autore e il numero di pagine di un libro, mentre operazioni come il prestito o il ritorno di un libro saranno gestite dalla classe “Biblioteca”.

Il concetto di responsabilità delle classi è importante poiché aiuta a definire il ruolo di ogni classe all'interno di un programma e a garantire che ogni classe svolga solo il compito per cui è stata progettata, evitando il sovraccarico di funzionalità. Inoltre, il rispetto delle responsabilità delle classi contribuisce a garantire la coerenza del programma e a rendere il suo codice più facile da comprendere e da mantenere.

Ecco un esempio di classe che non rispetta la responsabilità:

```
// costruttore
public Studente(String nome, String cognome, int matricola, double mediaVoti, String password) {
  this.nome = nome;
  this.cognome = cognome;
  this.matricola = matricola;
  this.mediaVoti = mediaVoti;
  this.password = password;
}

// metodi
public void cambiaPassword(String vecchiaPassword, String nuovaPassword) {
  if (vecchiaPassword.equals(this.password)) {
  this.password = nuovaPassword;
  }
}

public void inviaEmail(String destinatario, String oggetto, String testo) {
// codice per inviare un'email
}

public void prenotaLibro(String titolo, String autore) {
// codice per prenotare un libro in biblioteca
}

public void pagaTassa(double importo) {
// codice per effettuare un pagamento online
}

private String nome;
private String cognome;
private int matricola;
private double mediaVoti;
private String password;
```

Questa classe "Studente" ha diverse responsabilità, come gestire il nome, il cognome, la matricola e la media dei voti di uno studente, e fornire il metodo "cambiaPassword" per modificare la password dello studente. Tuttavia, la classe ha anche altre responsabilità, come inviare email, prenotare libri in biblioteca e effettuare pagamenti online, che non sono strettamente correlate alla sua principale responsabilità di gestire le informazioni di uno studente. Questo potrebbe rendere la classe più difficile da comprendere e da mantenere e potrebbe creare confusione su quali siano le sue effettive responsabilità.

Diagrammi UML

Diagrammi UML

Un diagramma UML (Unified Modeling Language) è una notazione grafica utilizzata per modellare e documentare sistemi software. In particolare, il diagramma UML viene utilizzato per descrivere la struttura e il comportamento di un sistema software, rappresentando le classi, gli oggetti, le interazioni e le relazioni tra di essi.

Diagrammi di Classi in UML

- Rappresentano la struttura del sistema software, descrivendo le classi e le relazioni tra di esse
- Vengono utilizzati per modellare l'architettura del sistema e il suo funzionamento

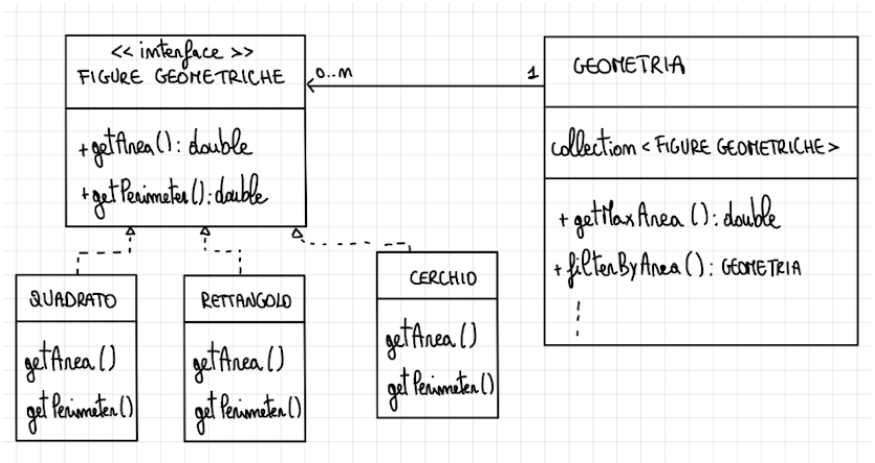
Elementi di un Diagramma di Classi in UML

- Nome della classe: viene visualizzato in alto e descrive il tipo di oggetto che la classe rappresenta
- Attributi (variabili d'istanza): viene visualizzato subito dopo il nome della classe e descrive le proprietà della classe
- Metodi: viene visualizzato nella parte inferiore della classe e descrive le operazioni che la classe può eseguire
- Prima di un attributo o di un metodo si utilizza:
    - "+" per indicare che è public (si usa anche per static)
    - "-" per indicare che è private (si usa anche per abstract)
    - "#" per indicare che è protected
    - "~" per indicare che è final

Relazioni tra Classi in UML

- Si utilizza una linea tratteggiata quando si vuole implementare un'interfaccia in una classe
- Si utilizza una linea continua con freccia piena per indicare la dipendenza tra classi, poi va specificato il tipo di dipendenza(1:1, 1:N, N:M)
- Si utilizza una linea continua con freccia vuota per indicare una relazione di ereditarietà ("IS-A" oppure "È-UN")

Esempio:



Array

Sono un tipo di dato che permette di memorizzare una sequenza di elementi di uno stesso tipo. Sono considerati una struttura di dati poiché permettono di organizzare e gestire i dati in modo efficiente. Gli elementi di un array sono accessibili tramite un indice, che inizia da 0. La **dimensione** di un array è **fissa** e non può essere modificata una volta che è stata assegnata. Inoltre, dato che array è un tipo di dato e non una classe, **NON** ha metodi

Esempio "**length**":

lenght non è un metodo, ma una **proprietà**, infatti non ha le ()

```
int[] array = {1,2,3};
int size = array.lenght; //a size verrà assegnato 3
```

Per copiare un array si utilizza il metodo arraycopy:
arraycopy(int[] fromArray, int fromPosition, int[] toArray, int toPosition, int count) della classe **System**

fromArray: quale array bisogna copiare;

fromPosition: da quale posizione dell'array si vuole iniziare la copia;

toArray: in quale array bisogna effettuare la copia;

toPosition: la posizione del nuovo array in cui iniziare ad inserire gli elementi;

count: indica quanti elementi si vogliono copiare.

Esempio:

```
double[] data = {0,1,2,3,4,5,6,7,8,9};
double[] newData = new double[10];
System.arraycopy(data, 3, newData, 2, 4);
//Allora newData sarà: { , ,3,4,5,6, , , , }
```

Questo metodo è usato anche per raddoppiare la dimensione di un array.

L'algoritmo è:

```
double[] data = new double[10];
//Creare un nuovo array
double[] newData = new double[2 * data.length];
//Copiarvi tutti gli elementi
System.arraycopy(data, 0, newData, 0, data.length);
//Memorizzare il riferimento
data = newData;
```

Array in 2D

- Struttura di dati che permette di immagazzinare una matrice di elementi

- Gli elementi di un array in 2D sono accessibili tramite due indici, il primo per le righe e il secondo per le colonne
- La dimensione di un array in 2D è fissa e non può essere modificata una volta che è stata assegnata

```
int[][] array2D = new int[][] {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12}
};
```

Liste

In Java, List è un'interfaccia che rappresenta una sequenza di elementi ordinata e modificabile.

Un List può contenere elementi duplicati e mantiene l'ordine di inserimento degli elementi. Esistono diverse implementazioni dell'interfaccia List in Java, come ArrayList, LinkedList e Vector.

Ecco alcune caratteristiche delle liste in Java:

- Possono contenere elementi duplicati
- Mantengono l'ordine di inserimento degli elementi
- Possono essere modificate dinamicamente (ad esempio, aggiungendo o rimuovendo elementi). Quando una lista viene descritta come "modificabile dinamicamente", significa che è possibile aggiungere o rimuovere elementi dalla lista in modo dinamico durante l'esecuzione del programma, cioè quando viene aggiunto un elemento viene allocata memoria e quando viene rimosso viene deallocata memoria. Ciò si differenzia da una struttura di dati statica, che ha una dimensione fissa e non può essere modificata durante l'esecuzione del programma.

ArrayList

Struttura di dati che permette di immagazzinare una sequenza di elementi di uno stesso tipo in modo dinamico. Gli elementi di un **ArrayList** sono accessibili tramite un indice, che inizia da 0. La dimensione di un **ArrayList** può essere modificata dinamicamente, aumentando o diminuendo il numero di elementi in esso contenuti. Quando si vuole aggiungere un elemento in un **ArrayList** e questo è saturo, l'**ArrayList** in automatico raddoppia la sua dimensione.

Creare un ArrayList

Per creare un **ArrayList** di un determinato tipo di dati, è possibile utilizzare la sintassi seguente:

```
ArrayList<TipoDati> nomeArrayList = new ArrayList<>();
```

Ad esempio, per creare un **ArrayList** di interi:

```
ArrayList<Integer> numeri = new ArrayList<>();
```

Per creare un **ArrayList** di stringhe:

```
ArrayList<String> parole = new ArrayList<>();
```

Aggiungere elementi ad un ArrayList
Per aggiungere un elemento a un **ArrayList**, è possibile utilizzare il metodo **add()**:

```
nomeArrayList.add(elemento);
```

Rimuovere elementi da un ArrayList

Per rimuovere un elemento da un **ArrayList**, è possibile utilizzare il metodo **remove()**:

```
nomeArrayList.remove(elemento);
```

Iterare sugli elementi di un ArrayList

Per iterare sugli elementi di un **ArrayList**, è possibile utilizzare un ciclo **for** o un **for generalizzato**:

```
for (int i = 0; i < nomeArrayList.size(); i++) {
    TipoDati elemento = nomeArrayList.get(i);
    // Codice da eseguire per ogni elemento
}
```

oppure:

```
for (TipoDati elemento : nomeArrayList) {
    // Codice da eseguire per ogni elemento
}
```

Ad esempio, per stampare gli elementi dell'**ArrayList** di stringhe:

```
for (String parola : parole) {
    System.out.println(parola);
}
```

Associazioni (o dipendenza tra le classi)

- **Associazione molti a molti (N:M)**: in questo tipo di associazione, entrambe le classi hanno una relazione con zero o più istanze dell'altra classe. Ad esempio, in un sistema di gestione dei libri di una biblioteca, la classe "Libro" potrebbe avere una relazione con la classe

"Persona", poiché un libro può essere prestato a più persone e una persona può prestare più libri.

- **Associazione uno a uno (1:1)**: in questo tipo di associazione, una classe ha una relazione con una sola istanza di un'altra classe. Ad esempio, in un sistema di gestione dei contratti di un'assicurazione, la classe "Auto" potrebbe avere una relazione con la classe "Contratto", poiché ogni auto ha un solo contratto.
- **Associazione uno a molti (1:N)**
  In questo tipo di associazione, una classe ha una relazione con zero o più istanze di un'altra classe. Ad esempio, la classe "Proprietario" ha una relazione con la classe "Contratto", poiché un proprietario può avere più contratti.

Ereditarietà

In Java l'ereditarietà è il meccanismo che permette ad una classe (detta classe figlia o sottoclasse) di ereditare tutti i metodi e le variabili pubbliche o protette di un'altra classe (detta classe padre o superclasse).

L'ereditarietà non può essere sempre utilizzata in quanto si basa sulle associazioni "**is - a**" cioè "**è - un**". Esempi di queste associazioni:

- Cane -> Animale;
- Cerchio -> Figura Geometrica;
- Alunno -> Persona.

Per utilizzare l'ereditarietà in una classe bisogna usare la keyword **extends**.

```
public Class ClasseFiglia extends ClassePadre
```

Così facendo la classe figlia erediterà tutti i metodi e le variabili della classe padre, ma avrà accesso solo alle risorse dichiarate come public o protected.

Un esempio di implementazione di un'associazione "È-Un" (ereditarietà) in Java:

```
public class Animale {
    // Variabili di istanza e metodi comuni agli animali
}

public class Cane extends Animale {
    // Variabili di istanza e metodi specifici dei cani
}
```

In questo caso, la classe **Cane** "è un" tipo di **Animale** ,poiché estende la classe **Animale** ereditandone le proprietà e i metodi.

Packages

- Sono dei contenitori utilizzati per organizzare le classi e le interfacce in modo logico
- Vengono utilizzati per evitare conflitti di nomi tra le classi e per rendere più facile l'utilizzo delle classi da parte degli sviluppatori (infatti possono esistere classi con lo stesso nome solo se sono situati in package diversi)

Classi Astratte

Che cos'è una Classe Astratta in Java

- Una classe astratta è una classe che contiene almeno un metodo astratto, cioè un metodo che non ha implementazione
- Una classe astratta in Java non può essere istanziata e deve essere estesa da una sottoclasse
- Una classe astratta contiene metodi astratti o anche metodi con implementazione

Come creare una Classe Astratta in Java

Utilizzare la keyword **abstract** prima della keyword class seguita dal nome della classe astratta

```
public abstract class Animale {
    public abstract void emettiSuono();
    public void cammina() {
        // implementazione del metodo
    }
}
```

Come estendere una Classe Astratta in Java

- Utilizzare la keyword extends seguita dal nome della classe astratta nella dichiarazione della sottoclasse
- La sottoclasse deve implementare tutti i metodi astratti della classe astratta

```
public class Cane extends Animale {
    public void emettiSuono() {
        // implementazione del metodo astratto
    }
}
```

Differenze tra Classi Astratte e Interfacce

- Una classe può estendere solo una classe astratta, mentre può implementare più interfacce
- Una classe astratta può avere variabili d'istanza e costruttori, mentre un'interfaccia può avere solo variabili statiche e finali, quindi una classe astratta modella una certa realtà, mentre l'interfaccia non può farlo.

Polimorfismo

Pagina 35 appunti Programmazione 2.

Il polimorfismo è un concetto della programmazione orientata agli oggetti che si riferisce alla capacità di un oggetto di assumere diverse forme. In Java, un esempio di polimorfismo è dato da metodi overridden.

Overriding: l'overriding è la capacità di una classe figlia di ridefinire un metodo presente nella classe padre. Ad esempio:

```
class Animale {
    public void muovi() {
        System.out.println("Muovo le zampe");
    }
}

class Gatto extends Animale {
    @Override
    public void muovi() {
        System.out.println("Muovo la coda");
    }
}
```

In questo esempio, la classe **Gatto** ridefinisce il metodo **muovi** presente nella classe **Animale**. Quando viene chiamato il metodo **muovi** su un oggetto di tipo **Gatto**, verrà eseguito il codice presente nella ridefinizione del metodo nella classe figlia.

Binding

Il binding è il collegamento tra l'oggetto che chiama il metodo e il metodo stesso da eseguire. In Java, esistono due tipi di binding:

1. **Early binding**: Il binding a tempo di compilazione (o statico) si verifica quando il compilatore conosce il tipo di oggetto su cui invocare il metodo a tempo di compilazione. In questo caso, il compilatore può risolvere il binding in modo statico, ovvero prima che il codice venga eseguito. Ad esempio:

   ```
   Name name = new Name();
   name.print();
   ```

2. **Dynamic binding**: Il binding a tempo di esecuzione, al contrario, si verifica quando il programma conosce il tipo di oggetto su cui invocare il metodo solo a tempo di esecuzione. In questo caso, il binding viene risolto in modo dinamico, ovvero durante l'esecuzione del

codice. Ad esempio:

```
Name name = new ExtendedName();
name.print();
```

Fattorizzazione

Definizione di factoring:

L'ereditarietà può essere anche usata per mettere a fattor comune un comportamento condivisod a due o più classi in una singola superclasse

Quando si hanno una superclasse e una sottoclasse e all'interno della sottoclasse si vuole implementare un metodo già presente nella superclasse ma con qualche aggiunta, è possibile richiamare il metodo della superclasse senza dover riscrivere tutto il codice.

Esempio: il metodo read è il metodo fattorizzato

```
public class Opera {

  public Opera(String author, String title, Date year, String collocation) {
    this.author = author;
    this.title = title;
    this.year = year;
    this.collocation = collocation;
  }

  public static Opera read(Scanner sc) throws ParseException {
    if(!sc.hasNextLine()) return null;
    String author = sc.nextLine();
    if(!sc.hasNextLine()) return null;
    String title = sc.nextLine();
    if(!sc.hasNextLine()) return null;
    String yearString = sc.nextLine();
    Date year = Constants.sdf.parse(yearString);
    if(!sc.hasNextLine()) return null;
    String collocation = sc.nextLine();
    return new Opera(author, title, year, collocation);
  }
}
```

```
public class Book extends Opera{

  public Book (String author, String title, Date year, String collocation) {
    super(author, title, year, collocation);
  }

  public static Book read(Scanner sc) throws ParseException {
    Opera op = Opera.read(sc);
    return new Book(op.getAuthor(), op.getTitle(), op.getYear(), op.getCollocation());
  }

}
```

CallBacks

Definizione: consente ad una classe di delegare un apposito metodo per ottenere informazioni diverse dagli oggetti di una stessa classe o di classi diverse in modo neutro

Inner Class

In Java, un'inner class è una classe definita all'interno di un'altra classe. Le inner class sono spesso utilizzate per creare classi ausiliarie che sono utilizzate solo all'interno della classe padre (e quindi anche dalle classi figlie).

Ecco un esempio di inner class in Java:

```
public class OuterClass {
    private int x;

    public class InnerClass {
        public void printX() {
            System.out.println(x);
        }
    }
}
```

In questo esempio, la classe **InnerClass** è definita all'interno della classe **OuterClass**. La classe **InnerClass** può accedere a tutte le variabili e i metodi privati della classe **OuterClass**, come mostrato nel metodo **printX()**.

Exception

Sito Lorenzo: https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/

Appunti Programmazione 2 p.40

Le **eccezioni** si riferiscono ad un evento indesiderato e/o inaspettato che si verifica durante l'esecuzione di un programma e se non vengono gestite, provocano l'interruzione dell'esecuzione

Un'eccezione rappresenta un problema che può verificarsi durante l'esecuzione di un programma e che può essere gestito in modo esplicito dal programma. Ad esempio, una eccezione può essere generata quando si tenta di accedere a un indice non valido di un array o quando si tenta di aprire un file che non esiste.

Esistono due tipi di eccezioni:

- CHECKED: sono dovute a circostanze esterne che il programmatore non può prevenire. Possono essere provocate da eventi esterni come un errore del disco oppure l'interruzione del collegamento di rete. Il compilatore richiede che sia aggiunto **throws Exception** nell'intestazione dei metodi

- UN-CHECKED: sono le eccezioni che si verificano a RunTime. Derivano da problemi di programmazione che si possono evitare, correggendo il programma. Si gestiscono con i blocchi **try-catch-finally**. Esempi di runtime exception:
    - NullPointerException: uso di un riferimento null
    - IndexOutofBoundException: accesso ad elementi esterni ai limiti di un array

Differenza tra errori ed eccezioni

Sito Lorenzo: https://www.geeksforgeeks.org/errors-v-s-exceptions-in-java/

In Java, sia **Errori** che **Eccezioni** sono sottoclassi di java.lang.Throwable.

L'**errore** si riferisce a un'operazione non corretta eseguita dall'utente che provoca un funzionamento anomalo del programma. Gli errori di programmazione spesso non vengono rilevati finché il programma non viene compilato o eseguito. Alcuni errori impediscono la compilazione o l'esecuzione del programma. Pertanto, gli errori devono essere rimossi prima della compilazione e dell'esecuzione.

Invece le **eccezioni** si riferiscono ad un evento indesiderato e/o inaspettato che si verifica durante l'esecuzione di un programma e se non vengono gestite, provocano l'interruzione dell'esecuzione

Chat GPT:

Le eccezioni in Java sono eventi che possono interrompere il flusso normale di esecuzione di un programma. Possono essere generate da situazioni inaspettate o indesiderate, come ad esempio tentativi di accedere a un indice di array non valido o di aprire un file che non esiste. Le eccezioni possono essere gestite utilizzando il meccanismo della gestione delle eccezioni di Java, che consente di continuare l'esecuzione del programma anche se si verifica un'eccezione.

Gli errori in Java, al contrario, sono problemi dovuti ad un malfunzionamento di sistema che non possono essere gestiti. Ad esempio, un errore di memoria potrebbe verificarsi se il programma ha bisogno di più memoria di quanta sia disponibile. Gli errori non possono essere gestiti utilizzando il meccanismo della gestione delle eccezioni e interrompono l'esecuzione del programma.

Iterator (legame con for generalizzato)

Definizione: un iteratore è un oggetto che consente di scorrere gli elementi di una raccolta di dati e di accedervi in modo sequenziale. Gli iteratori sono spesso utilizzati per eseguire l'iterazione sui dati contenuti in una raccolta, ad esempio una lista o un set.

Gli iteratori forniscono un modo conveniente per accedere agli elementi di una raccolta senza dover conoscere i dettagli di implementazione della raccolta stessa. Inoltre, gli iteratori consentono di modificare gli elementi della raccolta durante l'iterazione, ad esempio eliminando elementi o aggiungendo nuovi elementi.

In Java, gli iteratori sono oggetti che implementano l'interfaccia Iterator. L'interfaccia Iterator definisce i metodi che possono essere utilizzati per iterare attraverso gli elementi di una raccolta, ad esempio hasNext() e next().

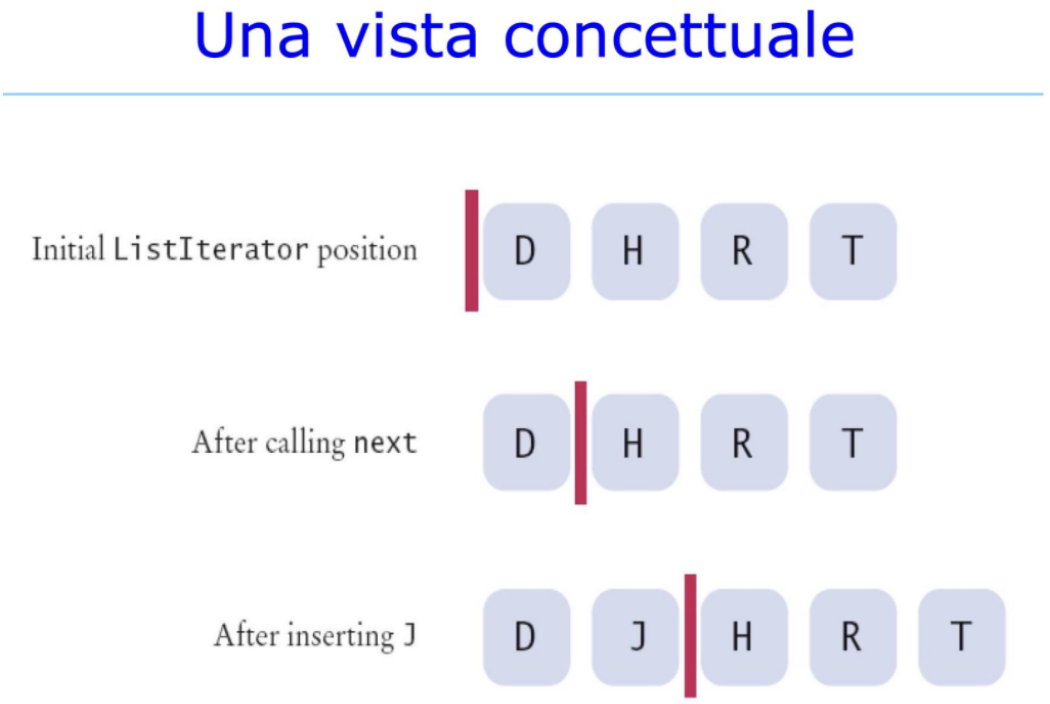
In Java, è possibile utilizzare un iteratore su qualsiasi tipo di raccolta che implementi l'interfaccia Iterable. L'interfaccia Iterable definisce un metodo che restituisce un oggetto Iterator, che può essere

utilizzato per iterare attraverso gli elementi della raccolta. Le raccolte comunemente utilizzate in Java che implementano Iterable includono le classi ArrayList, LinkedList, HashSet e TreeSet

COME FUNZIONA ITERATOR:

- Inizialmente posizionato prima del primo elemento
- Il metodo **next** sposta l'iterator: iterator.next(); next lancia una eccezione NoSuchElementException se si è già oltrepassato il limite della lista
- **hasNext** è vero se la lista ha ancora elementi

```
if (iterator.hasNext())
  iterator.next();
```



Esempio:

```
Iterator<Contract> iterator = contracts.iterator();
while (iterator.hasNext()) {
  iterator.next().print();
  System.out.println();
}
```

equivale a:

```
for (Contract contract : contracts) {
  contract.print();
  System.out.println();
}
```

Comparable

Comparable è un'interfaccia presente in Java che permette di definire un ordinamento per una classe. L'interfaccia **Comparable** contiene il metodo **compareTo**, che prende in input un oggetto e restituisce un valore intero.

Se il valore restituito è negativo, significa che l'oggetto corrente è "meno" dell'oggetto passato come argomento; se il valore è positivo, l'oggetto corrente è "maggiore" dell'oggetto passato; se il valore è zero, l'oggetto corrente è "uguale" all'oggetto passato.

Flashcards:

1. Cosa fa l'interfaccia **Comparable** in Java?
   Definisce un ordinamento per una classe
2. Quale metodo contiene l'interfaccia **Comparable**?
   Contiene il metodo **compareTo**
3. Cosa fa il metodo **compareTo**?
   Prende in input un oggetto e restituisce un valore intero: negativo se l'oggetto corrente è "meno" dell'oggetto passato come argomento, positivo se l'oggetto corrente è "maggiore", zero se l'oggetto corrente è "uguale" all'oggetto passato

Binary Search Tree (BST)

Utilizzati nell'implementazione di TreeSet e TreeMap. Un BST mantiene l'ordine delle chiavi in modo che le chiavi più piccole siano a sinistra e quelle più grandi siano a destra. Permette di effettuare ricerche veloci, inserimenti e cancellazioni di elementi.

Ogni nodo del BST contiene:

- al massimo due riferimenti a Nodo (figli sinistro e destro);
- un campo di informazione di tipo Comparable perché deve essere necessario confrontare gli oggetti

Un BST è ottimale quando ha un numero di nodi proporzionale al logaritmo del numero di elementi.

Un albero è bilanciato se ogni nodo ha un numero di discendenti a destra paragonabile a quello dei discendenti a sinistra. Quindi per un albero bilanciato, l'inserimento di un nuovo elemento ha complessità logaritmica

- Per l'**inserimento**:
  - Per ogni riferimento a nodo **non-null** si analizza il valore
  - Se il valore è maggiore di quello da inserire, il processo continua con l'albero di sinistra (quindi il valore da inserire è più piccolo di quello del nodo analizzato)
  - Se il valore è minore di quello da inserire, il processo continua con l'albero di destra (quindi il valore da inserire è più grande di quello del nodo analizzato)
  - Quando si raggiunge un riferimento **null**, allora si aggiunge il nuovo nodo
- Per la **rimozione**:
  - È facile rimuovere un nodo foglia, basta deallocare il nodo
  - Se si rimuove un nodo con un solo figlio, il figlio rimpiazza il nodo rimosso

- Se si rimuove un nodo con due figli, il nodo è rimpiazzato con il nodo più piccolo del sottoalbero destro

Hash Tables

Hashing: una tecnica di memorizzazione che consente di identificare velocemente un oggetto in una struttura, senza necessità di visite sequenziali Una hash table: ◦ utilizza la tecnica di hashing ◦ può essere utilizzata come base per l'implementazione di set e map

Una funzione hash calcola un valore intero (detto hash code) a partire da un oggetto Calcolare il codice hash di un oggetto x : **int h = x.hashCode();**

Una buona funzione deve minimizzare le collisioni

Collisione: si ha una collisione quando due oggetti generano lo stesso hash code

L'implementazione di una hash table è molto semplice:

- Creare un array
- Generare il codice hash degli oggetti
- Inserire ogni oggetto nella posizione relativa al suo codice

Tuttavia potrebbero esserci dei problemi:

- potrebbe esserci un array virtualmente infinito
- non si gestiscono le collisioni

Per risolvere il problema della dimensione dell'array si ricorre alla normalizzazione,

cioè l'hash code viene ricalcolato in base alla dimensione dell'array nel seguente modo:

h = h % size;

Però così facendo occorre gestire le collisioni

Per farlo l'hash table utilizza una lista di nodi, detta bucket

Inoltre per evitare un elevato numero di collisioni l'hash code viene implementato utilizzando un moltiplicatore, tipicamente un numero primo perché si è visto che moltiplicando per un numero primo si generano meno collisioni.

Esempio: due stringhe possono generare lo stesso hashcode se sono una l'anagramma dell'altra. Quindi l'hash code verrà calcolato in base alla posizione dei caratteri (notazione posizionale, cioè si tiene conto del "peso" della posizione del singolo carattere)

Quindi **l'hash table è un array di LinkedList**

HashSet e TreeSet

Set

- Set è una struttura dati implementata in java tramite un'interfaccia e rappresenta un insieme di elementi distinti.
- Offre metodi per aggiungere, rimuovere e verificare la presenza di elementi.
- Non consente elementi duplicati e non mantiene l'ordine di inserimento degli elementi.
- Le implementazioni di Set includono HashSet, LinkedHashSet, EnumSet e TreeSet.

HashSet

- Un HashSet è un'implementazione di Set che utilizza una tabella hash per immagazzinare gli elementi.
- Si utilizza quando è necessario gestire molti elementi univoci.
- Non mantiene l'ordine degli elementi come inseriti.
- Offre prestazioni veloci per le operazioni di inserimento, ricerca e cancellazione.

TreeSet

- Un TreeSet è una implementazione di Set che utilizza un albero binario di ricerca per immagazzinare gli elementi.
- Si utilizza quando è necessario gestire pochi elementi univoci.
- Per poter utilizzare un TreeSet è necessario che gli oggetti che lo compongono implementino l'interfaccia Comparable oppure che sia definito un opportuno tipo di Comparator.
- Mantiene l'ordine degli elementi in base al **compareTo** o **comparator** implementato.
- Offre prestazioni più lente rispetto a HashSet per le operazioni di inserimento, ricerca e cancellazione, ma permette di effettuare ricerche range e di ottenere elementi in ordine.

HashMap e TreeMap

Map

- Map è un'interfaccia della collezione di Java che rappresenta una mappa di chiavi-valori.
- Offre metodi per aggiungere, rimuovere e recuperare i valori in base alla chiave.
- Non consente chiavi duplicate e non mantiene l'ordine delle coppie chiave-valore.
- Le implementazioni di Map includono HashMap, LinkedHashMap, EnumMap e TreeMap.

HashMap

- Un HashMap è una implementazione di Map (un'interfaccia della collezione di Java) che utilizza una tabella hash per immagazzinare le coppie chiave-valore.
- Si utilizza quando è necessario gestire molti elementi.
- Le chiavi sono hashed, cioè l'hash code è calcolato sull'oggetto chiave.
- Non mantiene l'ordine delle coppie chiave-valore come inserite, ma le inserisce nella posizione calcolata sulla base dell'hash code della chiave.
- Offre prestazioni veloci per le operazioni di inserimento, ricerca e cancellazione.
- Inoltre, l'hash table di un'HashMap ha un ulteriore parametro detto **load factor** che indica il grado di riempimento della tabella oltre il quale essa raddoppia la dimensione. Di default è

impostato a 0,75, ma può essere anche scelto all’atto della dichiarazione di un’HashMap passando un float come parametro esplicito al costruttore

TreeMap

- Una TreeMap è un’implementazione di Map (un'interfaccia della collezione di Java) che utilizza un albero binario di ricerca per immagazzinare le coppie chiave-valore.
- Si utilizza quando è necessario gestire pochi elementi.
- Per poter utilizzare una TreeMap è necessario che la chiave implementi l’interfaccia Comparable oppure deve essere definito un oggetto Comparator per le chiavi. NON c’è nessun vincolo sui valori.
- Mantiene l'ordine delle coppie chiave-valore in base al loro valore.
- Offre prestazioni più lente rispetto a HashMap per le operazioni di inserimento, ricerca e cancellazione, ma permette di effettuare ricerche range e di ottenere coppie chiave-valore in ordine.