LA DOCUMENTAZIONE AUTOMATICA CON JAVADOC

Come già detto è molto importante documentare il codice. La documentazione del codice serve per "spiegare come è stato scritto" il codice. La documentazione serve per comprendere il meglio il codice e quindi facilitare l'interazione con esso in operazioni di manutenzione correttiva o manutenzione evolutiva. Il commenti sono utili sia per eventuali altri programmatori che dovessero, in futuro, modificare il codice, sia per lo stesso programmatore che ha realizzato il codice per eventuali modifiche successive. Per rimediare al fatto che ogni programmatore documenta in maniera personalizzata il proprio codice, l'ambiente di sviluppo standard JDK ha messo a disposizione uno strumento che standardizza ed automatizza la **documentazione** delle **classi** realizzate. Questo strumento è un software chiamato **javadoc** che, automaticamente, riconosce le righe del codice opportunamente scritte utilizzando appositi **tag** (chiamati tag javadoc) e genera in automatico dei file HTML con la documentazione relativa al codice scritto.

I file HTML generati hanno lo stesso layout dei file di documentazione del linguaggio java forniti da Oracle e disponibili al seguente link:

https://docs.oracle.com/en/java/javase/21/docs/api/index.html

Per aggiungere dei commenti per javadoc si utilizza, nel codice Java, la seguente notazione che inizia con / e termina con */. Esempio:**

```
/**
 * Costruttore della classe Libro con titolo, autore, numero di
*pagine
 *
 */
public Libro (String titolo, String autore, int numeroPagine) {

     this.titolo=titolo;
     this.autore=autore;
     this.numeroPagine=numeroPagine;
}
```

Si devono indicare le informazioni utilizzando dei tag che vengono riconosciuti da javadoc:

@author

@version

@param nome parametro (descrizione del parametro di un metodo)

@return (descrizione del valore di ritorno di un metodo)

@exception (descrizione di una eccezione sollevata da un metodo)

@throws (indicazione di quando viene sollevata un'eccezione all'interno di un metodo)

@see (inserzione di un riferimento (un testo, un sito internet…))

Nel codice si aggiungono quindi:

- i commenti che descrivono la classe (prima della classe) aggiungendo @author e @version se necessario
- i commenti per la descrizione di ciascun metodo (con i relativi @param, @return, @exception @throws), prima del metodo da descrivere.

Documentare una classe

Esempio javadoic della classe String:
https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/String.html

```
/**
DOCUMENTAZIONE:
- COSA RAPPRESENTA LA CLASSE (da quali attributi è composta, se ci sono attributi statici, costanti o altri attributi particolari, fornire la spiegazione degli attributi. Non è necessario indicare il tipo di dato degli attributi. Indicare eventuali esempi di utilizzo di una classe)
- COSA CONSENTE DI FARE (se ci sono metodi che consentono di svolgere operazioni particolari e che è utile indicare per comprendere meglio cosa rappresenta la classe, la spiegazione dettagliata del funzionamento dei vari metodi non viene scritta qui poiché ciascun metodo sarà documentato con apposita documentazione)
- EVENTUALI ALTRE INDICAZIONI UTILI AL SUO UTILIZZO (vedi esempi successivi)
@author Dina Lampa
@version 1.0
*/
public class Classe { ... }
```

Esempio per la classe Libro:

```
/**
 *La classe Libro rappresenta un libro. I suoi attributi sono:
 * titolo: il titolo del libro
 * autore: il nome di colui che ha scritto il libro
 * numeroPagine: il numero di pagine di cui è costituito il
 * libro.
 * costoPagina: attributo statico, indica il costo di una
 * singola pagina del libro
 * COSTO_FISSO: è una costante, indica la parte di costo fisso
 * per la produzione di un libro
 * il prezzo finale di un libro sarà dato dalla somma fra il
 * COSTO_FISSO e numero di pagine moltiplicato per il
 * costoPagina
 *
 * @author Laini Gian Marco
 * @version 1.0
 */
public class Libro
{
    //Attributi
    private String titolo;
    private String autore;
    private int numeroPagine;
    private static float costoPagina=0.05f;
    private final double COSTO_FISSO=5.5;

    ……..
```

Documentare un metodo

```
/**
DOCUMENTAZIONE:
- QUAL'E'LA FUNZIONE DEL METODO (A COSA SERVE)
- ULTERIORI INFORMAZIONI UTILI PER COMPRENDERE COME
UTILIZZARE IL METODO.
- PRESENZA DI EVENTUALI CONTROLLI ED ECCEZIONI

@param x COSA RAPPRESENTA x?  (non è necessario indicare
il tipo di dato)
@return COSA RESTITUISCE
@throws Exception QUANDO VIENE LANCIATA?
*/
public int metodo(int x) throws Ex {...}
```

Esempio per il costruttore Libro:

```
/**
 * Costruttore della classe Libro. Consente di istanziare un
* nuovo libro
 * @param titolo:titolo del libro
 * @param autore: nome dell'autore del libro
 * @param numeroPagine: numero di pagine di cui è composto il
*  libro
*/
    public Libro(String titolo,String autore,int numeroPagine)
    {
        this.titolo=titolo;
        this.autore=autore;
        this.numeroPagine=numeroPagine;
    }
```

Esempio per il costruttore di copia del Libro:

```
/**
   * Costruttore di copia della classe Libro. Consente di
  * istanziare un nuovo libro
   * @param l: libro da cui verrà istanziato il nuovo libro.
  * Il libro istanziato sarà una copia del libro l
   */

    public Libro(Libro l)
    {
        titolo=l.getTitolo();
```

```
        autore=l.getAutore();
        numeroPagine=l.getNumeroPagine();
    }
```

Esempio per il costruttore di default:

```
 /**
     * Costruttore di default della classe Libro. Consente di
     * istanziare un nuovo libro con i seguenti
     * valori di default per gli attributi
     * titolo: stringa vuota
     * autore: stringa vuota
     * numeroPagine=0
    */

public Libro()
    {
        titolo="";
        autore="";
        numeroPagine=0;
    }
```

Esempio per i getter e i setter:

```
/**
     * Restituisce il titolo del libro
     * @return titolo
 */
    public String getTitolo()
    {
        return titolo;
    }

  /**
     * Assegna il titolo al libro
     * @param titolo
  */
    public void setTitolo(String titolo)
    {
        this.titolo=titolo;
    }

  /**
     * restituisce l'autore del libro
     * @return autore
   */
    public String getAutore()
    {
        return autore;
```

```
    }


/**
     * Assegna l'autore al libro
     * @param autore
*/
    public void setAutore(String autore)
    {
        this.autore=autore;
    }

    /**
     * Restituisce il numero di pagine di cui è composto il
     * libro
     * @return numeroPagine
    */
    public int getNumeroPagine()
    {
        return numeroPagine;
    }

    /**
     * Assegna il numero di pagine di cui è costituito il
     * libro
     * @param numeroPagine
     */
```

```
    public void setNumeroPagine(int numeroPagine)
    {
        this.numeroPagine=numeroPagine;
    }

/**
     * Metodo statico
     * Assegna il costo unitario per realizzare ciascuna
     * pagina del libro
     *
     * @param costo
  */
    public static void setCostoPagina(double costo)
    {
        costoPagina=costo;

    }

    /**
     * Metodo statico
     * Restituisce il costo unitario per realizzare ciascuna
     * pagina del libro
     * @return costoPagina
     */
    public static double getCostoPagina()
    {
        return costoPagina;
```

La descrizione in questo caso è più specifica perché sia il metodo sia l'attributo coinvolto sono meno intuitivi rispetto agli altri

La descrizione in questo caso è più specifica perché sia il metodo sia l'attributo coinvolto sono meno intuitivi rispetto agli altri

```
    }
```

Esempio per il metodo prezzo:

```
/**
     * Calcola e restituisce il prezzo di vendita del libro
     * @return prezzo
 */
  public double prezzo()
    {
        double p;
        p=COSTO_FISSO+costoPagina*getNumeroPagine();
        return p;
    }
```

Esempio per il metodo toString():

Specifico quali informazioni del libro verranno mostrate sotto forma di stringa

```
/**
   * Restituisce una stringa che rappresenta il libro
   * @return stringa con titolo, autore e numero di pagine del libro
 */
  public String toString()
  {
     String s;
     s=getTitolo()+";"+getAutore()+";"+getNumeroPagine()+" pagine";
     return s;
  }
```

Di seguito vengono riportati i commenti per la classe Scaffale e per alcuni suoi metodi:

```
/**
 *La classe rappresenta uno scaffale composto da NUM_RIPIANI ripiani.
 * Ogni ripiano è un oggetto di classe Mensola che è un aggregazione di oggetti di tipo
 * Libro.
 * Gli attributi sono:
 * ripiani (un array di  mensole)
 * NUM_RIPIANI (una costante che indica il numero di ripiani di cui è costituito lo
 * scaffale)
 * I metodi consentono di svolgere diverse operazioni sui libri che si trovano nello
* scaffale (aggiungere libri, rimuovere libri, ottenere dei libri, ottenere elenchi ordinati dei libri, ottenere i libri di uno specifico autore).
 * La posizione di ogni libro è individuata da da due numeri: ripiano e posizione.
 * Ripiano indica il ripiano in cui è posizionato il libro, posizione indica la posizione del
 * libro nel ripiano.
 *
 * @author Laini Gian Marco
 * @version 1.0
 */

public class Scaffale
{
……
….
}
```

```
/**
   * Costruttore di default
   * consente di istanziare uno scaffale con N_RIPIANI ripiani vuoti.
   *
   */
  public Scaffale()
  {
     ripiani=new Mensola[NUM_RIPIANI];

     for (int i=0;i<NUM_RIPIANI;i++)
     {
```

```
        ripiani[i]=new Mensola();
      }

  }



  /**
   * Costruttore di copia
   * Consente di istanziare uno scaffale copia dello scaffale passato come parametro
   * @param s lo scaffale da copiare
   * @throws ExceptionPosizioneNonValida viene sollevata quando si tenta di
    * accedere ad un ripiano inesistente
   * o ad una posizione inesistente all'interno del ripiano
   * @throws ExceptionPosizioneNonVuota Viene sollevata quando si tenta di
 * aggiungere un libro ad una posizione
   * non vuota dello scaffale
   */
  public Scaffale(Scaffale s) throws ExceptionPosizioneNonValida,
ExceptionPosizioneNonVuota
  {
      ripiani=new Mensola[NUM_RIPIANI];

      Libro l;
      for (int i=0;i<getNumRipiani();i++)
      {
         ripiani[i]=new Mensola();
         for(int j=0;j<ripiani[i].getNumMaxVolumi();j++)
         {
            if (s.getLibro(i, j)!=null)
            {
               l=s.getLibro(i, j);
               setLibro(l, i, j);
            }
         }
      }

  }
```

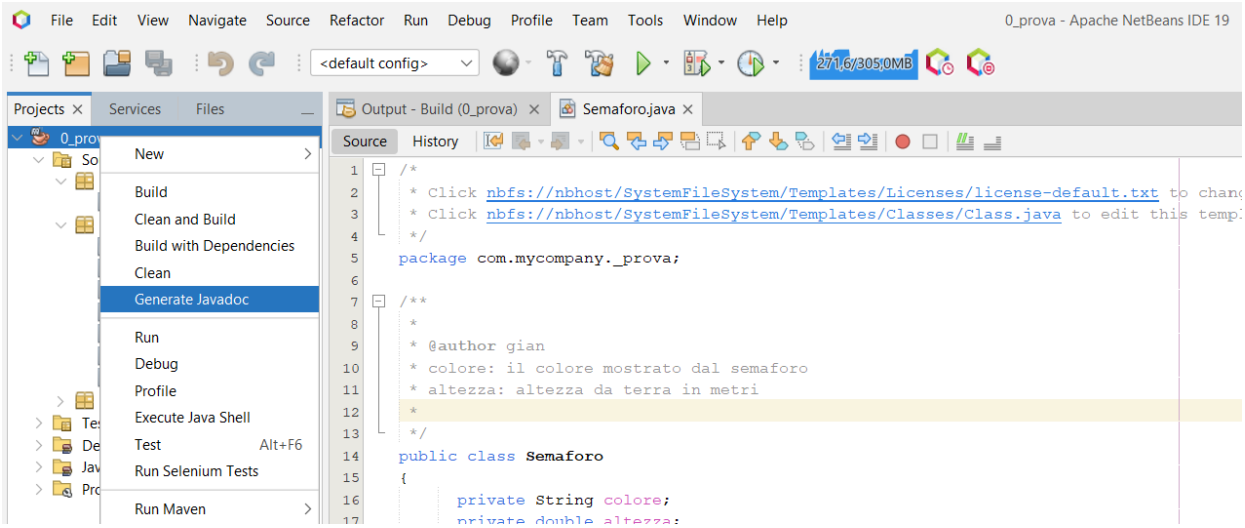
```
/**
  * Aggiunge un Libro allo scaffale in uno specifico ripiano e in una specifica posizione
  * all'interno del ripiano
  * @param libro il libro da aggiungere
  * @param ripiano il ripiano in cui aggiungere il libro (va da 0 a NUM_RIPIANI-1)
  * @param posizione la posizione, all' interno del ripiano in cui aggiungere il libro
  * @throws ExceptionPosizioneNonValida viene sollevata quando si tenta di
 * accedere ad un ripiano inesistente
  * o ad una posizione inesistente all'interno del ripiano
  * @throws ExceptionPosizioneNonVuota Viene sollevata quando si tenta di
 * aggiungere un libro ad una posizione non vuota dello scaffale
  */
  public void setLibro(Libro libro,int ripiano, int posizione) throws
ExceptionPosizioneNonValida, ExceptionPosizioneNonVuota
  {
     int inserimentoOk;
     try
     {
          inserimentoOk=ripiani[ripiano].setVolume(libro, posizione);
          if (inserimentoOk==-1)
             throw new ExceptionPosizioneNonValida(ripiano, posizione);
          if (inserimentoOk==-2)
             throw new ExceptionPosizioneNonVuota(ripiano, posizione);

     }
     catch(ArrayIndexOutOfBoundsException e1)
     {
        throw new ExceptionPosizioneNonValida(ripiano, posizione);
     }
  }

/**
  * Consente di ottenere il Libro che si trova nel ripiano e nella posizione specificati
  * @param ripiano il ripiano in cui si trova il libro che si vuole ottenere
  * @param posizione la posizione, all'interno del ripiano, in cui si trova il libro che si
 * vuole ottenere
  * @return il libro che si trova in (ripiano, posizione)
  * @throws ExceptionPosizioneNonValida viene sollevata quando il valore di ripiano
  * o il valore di posizione passati come parametro sono inesistenti nello scaffale.
  */
```

```
  public Libro getLibro(int ripiano, int posizione) throws ExceptionPosizioneNonValida
  {
     try
     {
        return(ripiani[ripiano].getVolume(posizione));
     }
     catch (ArrayIndexOutOfBoundsException e1)
      {
            throw new ExceptionPosizioneNonValida(ripiano, posizione);
      }
  }

  /**
   * Consente di ottenere il numero di libri presenti nello scaffale
   * @return il numero di libri presente nello scaffale
   */
  public int getNumLibri()
  {
     int numLibri=0;
     for(int i=0;i<getNumRipiani();i++)
     {
        numLibri+=ripiani[i].getNumVolumi();
     }
     return numLibri;
  }

/**
   * Consente di ottenere un array di String in cui sono riportati i titoli dei libri di uno
 * specifico
   * autore presenti nello scaffale
   * @param autore l'autore di cui si vogliono ottenere i titoli dei libri
   * @return l'aray di stringhe contenente i titoli dei libri dell'autore
   * @throws ExceptionPosizioneNonValida viene sollevata quando si tenta di
 * accedere ad un ripiano o a una posizione, all' interno del ripiano, inesistenti
   */
  public String[] elencoTitoliAutore(String autore) throws
ExceptionPosizioneNonValida
{
        ……
        …..
}
```

```
 /**
  * Consente di ottenere una String in cui sono riportati, per ogni scaffale, i libri
 * contenuti
  * suddivisi per ripianoo. Per ogni ripiano sono riportati i dati relativi ai libri presenti
 * e la posizione
  * in cui si trovano.
  * @return una sringa che descrve lo scaffale: quali libri sono presenti, suddivisi
 *  mensola per mensola con
  * indicazione della posizione in cui si trovano
  */
 public String toString()
{
        ……
        ……
}

/**
  * Consente di ottenere una String in cui sono riportati i titoli di ciascun libro
 * presente
  * nello scaffale in ordina alfabetico. Di ogni libro sono indicati anche il ripiano
  * e la posizione, all'interno del ripiano, in cui si trovano
  * @return un Stringa con i titoli dei libri presenti in ordine alfabetico, ripiano e posizione di ogni libro
  * @throws ExceptionPosizioneNonValida viene sollevata quando il valore di ripiano
 * o posizione di un
  * libro è inesistente.
  */
 public String elencoAlfabeticoLibri() throws ExceptionPosizioneNonValida
{
        ……
        ……
}
```

Per creare i file html di documentazione è necessario selezionare il progetto interessato nell' esplora risorse a sinistra (selezionare la scheda Project), selezionare con il tasto sinistro del mouse il progetto del quale si vuole generare la documentazione, e cliccare sul menu contestuale:

Generate Javadoc



Automaticamente, all'interno della cartella del progetto, si genererà la cartella **"target/site/apidocs"** che contiene i file html della documentazione. Tutte le pagine della documentazione sono raggiungibili a partire dalla pagina **index.html**. Se, dopo aver creato la documentazione, si apportano modifiche ai commenti e si vuole aggiornare la documentazione, è opportuno eliminare la cartella "site**" dal progetto e generare nuovamente tutti i file .html. Altrimenti la documentazione già presente non verrà aggiornata.**