Javadoc

Documentació

La documentació és tot text, diagrama, taula, etc. que acompanya a un projecte.

És molt important, ja que la seua finalitat és facilitar el desenvolupament, comprensió o ús del projecte.

Tenim diferents tipus de documentació:

- Documentació de les especificacions del projecte. En ella es plasmen les funcionalitats que desenvoluparà el projecte. És un document on el desenvolupador i el client es posen d'acord sobre el que ha de realitzar el projecte. És fonamentalment descriptiu (el client ha de poder entendre-ho). Es descriu la informació que es manejarà, el maquinari necessari, les funcionalitats que es desenvoluparan, els controls, els límits, etc.
- Documentació del disseny. En ella s'especifiquen les estructures de dades que s'utilitzaran, els objectes per a representar la informació que es manejarà. les interfícies que s'utilitzaran, etc. És un document per als desenvolupadors i programadors.
- Documentació del codi font. En ella s'especifiquen detalls del codi que s'està implementant. Normalment són comentaris inclosos en el codi font per a aclarir el codi, però es poden posar per a comentar errors a corregir o criteris per a realitzar proves. És un document per als programadors.
- Documentació per a l'usuari. En ella es descriu com s'utilitza el projecte que s'ha desenvolupat. Està dirigida a qualsevol tipus d'usuari, per tant la seua redacció s'haja d'adequar al tipus d'usuari.

Un parell de cites:

"*Si el teu programa no val la pena comentar-lo, probablement no val la pena executar-lo*" (J. Nagler)

"*Escriu la documentació abans d'escriure el codi*" (S. W. Ambler)

Javadoc

Un Javadoc és documentació de codi i s'utilitza per a comentar una classe, un camp, un constructor o un mètode, i així facilitar el seu ús.

Un Javadoc s'escriu, just, abans de l'element a comentar i entre els delimitadors /** i */ .

S'escriu en HTML, i el seu contingut serveix a l'eina Javadoc per a generar el API del projecte de manera automàtica. En NetBeans la tenim en l'opció de menú Run > Generate Javadoc

Té dues parts, una descripció seguida per un bloc d'etiquetes.

```
/**
 * Returns an Image object that can then be painted on the screen. The url
 * argument must specify an absolute URL. The name argument is a specifier
 * that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the image exists.
 * When this applet attempts to draw the image on the screen, the data will
 * be loaded. The graphics primitives that draw the image will incrementally
 * paint on the screen. </p>
 *
 * @param url an absolute URL giving the base location of the image
 * @param name the location of the image, relative to the url argument
 * @return the image at the specified URL
 * @see Image
 */
public Image getImage(URL url, String name) {
    try {
        return getImage(new URL(url, name));
    } catch (MalformedURLException e) {
        return null;
    }
}
```

Les etiquetes comencen amb @, la primera etiqueta finalitza el bloc de descripció. En l'exemple, la primera etiqueta és @param.

Un Javadoc pot tindre només una de les dues parts, només descripció o només etiquetes.

Es pot tindre qualsevol quantitat d'etiquetes i les etiquetes es poden repetir.

Els * inicials de cada línia es poden obviar, sols són obligatoris el /** inicial i el */ final.

Primera frase

La primera frase d'un Javadoc ha de ser una frase que resumeix el contingut de l'element del API, és una descripció concisa però completa. Aquesta frase acaba en el primer punt seguit d'un espai en blanc, un tab, un fi de línia, o amb la primera etiqueta.

```
/**
* This is a simulation of Prof. Knuth's MIX computer.
*/
```

La primera línia acaba en la paraula “Prof”.

També es pot utilitzar les entitats d'HTML per a substituir aqueixos caràcters.

```
/**
* This is a simulation of Prof. Knuth's MIX computer.
*/
```

Ara, la primera línia acaba en la paraula “computer”.

Descripció

La descripció ha de ser independent de la implementació.

Ha de definir clarament el que es requereix i el que pot variar.

La descripció ha de ser completa, incloent els límits de les col·leccions, els rangs dels paràmetres, els casos particulars.

Si el comportament depén de la plataforma s'indica mitjançant “On nomPlataforma” a l'inici del paràgraf que defineix eixe comportament especial. L'exemple defineix el comportament quan s’executa en Windows.

```
/**
* “On Windows” les vores no es mostren.
*/
```

Heretar

El Javadoc s'hereta, els mètodes dels fills reben el Javadoc del pare

```
/**
 * escriu una ressenya.
 * @param text a escriure
 * @param autor del text
 */
 public void escriu(String text, String autor) {
   System.out.println(text + " escrit per " + autor);
}
```

Si classe filla sobreescriu el mètode i no escriu el Javadoc, llavors es mostra el Javadoc del pare.

Si la classe filla vol canviar el Javadoc, aquest s'ha d’escriure abans de l'anotació @Override

```
/**
 * escriu una ressenya completa.
 * @param text text a escriure
 * @param autor autor de la ressenya
 */
@Override
public void escriu(String text, String autor) {
    System.out.println("ressenya \"" + text + "\" escrita per " + autor);
}
```

Si no s'escriu alguna de les etiquetes, llavors es rep el comentari escrit en el Javadoc del pare.

```
/**
 * escriu una ressenya completa.
 * @param autor autor de la ressenya
 */
```

```
@Override
public void escriu(String text, String autor) {
   System.out.println("ressenya \"" + text + "\" escrita per " + autor);
}
```

Guia d'estil

Per a escriure un Javadoc existeixen unes convencions.

Escriu els noms de paquet, classe, interfície, camps i mètodes, les paraules reservades de Java i els elements de codi dins de les etiquetes <code> i </code>

```
/**
* @return       <code>true</code> si el valor està entre els límits.<br>
*               <code>false</code> en qualsevol altre cas.
*/
```

Omet els parèntesis quan et referisques a un mètode o constructor

calculaIVA preferible
calculaIVA() evitar

Si tens un mètode amb múltiples signatures, llavors, entre parèntesis escriu únicament el tipus del paràmetre, no el seu nom (els tipus són els que distingeixen les signatures dels mètodes)

```
add(Object)
add(int, Object)
```

Usa frases curtes, en lloc d'oracions completes, sobretot en la primera línia i en els comentaris dels paràmetres.

Obté l'etiqueta preferible
L'objecte obté l'etiqueta del botó evitar

Usa la tercera persona en lloc de l'infinitiu, és el mètode el que realitza l'acció.

Obté l'etiqueta preferible
obtenir l'etiqueta evitar

La descripció d'un mètode comença amb un verb.

```
Llig de teclat.
Calcula la mitjana de les notes.
```

Per a una classe, interfície o camp, en les descripcions pot ometre's el subjecte i simplement definir l'estat de l'objecte, aquestes API sovint descriuen les coses en lloc d'accions o comportaments

Una etiqueta d'un botó preferible
Aquest camp és una etiqueta d'un botó evitar

Usa "aquest" (this) en lloc de "el" en referir-se a un objecte creat a partir de la classe actual.

Obté el conjunt d'eines d'aquest component preferible
Obté el conjunt d'eines del component evitar

Els millors comentaris de la API es "auto-documenten", és a dir, et diuen el que fa la API de manera simple.

```
/**
* Defineix el text que es mostrarà en un tooltip. El text
* es mostra quan el cursor es deté en el component.
* @param text la cadena de text per a mostrar. Si el text és nul,
* el tooltip està desactivat per a aquest component.
* /
public void setToolTipText (String text) {  …  }
```

és millor que el següent

```
* Defineix el tooltip.
* @param text el text per al tooltip.
* /
public void setToolTipText (String text) {  …  }
```

Evita els comentaris que no afigen res a la simple lectura del nom de l'element que es vol comentar.

Etiquetes

Una etiqueta s'escriu a l'inici de la línia, té una @ seguida per una paraula (sense separació) i a continuació el comentari associat a l'etiqueta.

Es distingeixen majúscules de minúscules.

Un nom d'etiqueta es pot repetir. Per convenció, les etiquetes amb el mateix nom s'agrupen.

Hi ha dos tipus d'etiquetes

- de bloc obrin un bloc de text amb un significat específic
- en línia s'inclouen en línia dins dels comentaris

Etiquetes en línia

Les etiquetes en línia s'escriuen en el text de qualsevol etiqueta de bloc, el tetx s'escriu entre { }

@code

La etiqueta `@code` mostra el text en codi font sense interpretar el text com a HTML. Això li permet utilitzar els caràcters < i > en lloc de les entitats HTML < i >.

el seu format és {@code text}

Per exemple, el text {@code Un <B>C} en el Javadoc apareix en la pàgina HTML com "Un <B>C", és a dir el <B> no s'interpreta com a negreta.

@literal

Fa el mateix que @code però s'usa per a representar literals, té el mateix format {@literal IVA}

@link

Insereix un enllaç amb una etiqueta de text visible que apunta a la documentació del nom especificat de paquet, classe o membre d'una classe a la qual es fa referència.

el seu format és {@link package.class#member label}

Per exemple,

```
@deprecated usa {@link oovv.Empleat#setCompSegurs(java.lang.String)} en el seu lloc
```

visualitza la línia següent

Deprecated. usa *Empleat.setCompSegurs(java.lang.String)* en el seu lloc

El text *Empleat#setCompSegurs(java.lang.String)* és un enllaç, en fer clic sobre ell se salta a la documentació d'aqueix mètode.

Si l'element al qual se salta està en la mateixa pàgina, llavors es pot escriure únicament des de la #

```
@deprecated usa {@link #setCompSegurs(java.lang.String)} en el seu lloc
```

@value

Es pot obtindre un valor, públic o protegit. En el Javadoc s'escriu el nom de l'element i aquest se substitueix pel seu valor.

El seu format és {@value paquet.Classe#element}

Pot ser un camp o una constant, ha d'estar definit a nivell de la classe.

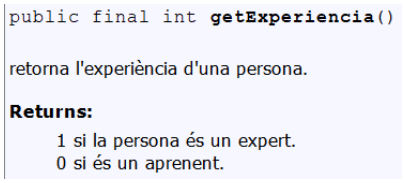
Aquestes són dues constants, que representen un expert i un aprenent. El mètode següent retorna l'experiència d'una persona.

```
public static final int EXPERT = 1;
public static final int APRENENT = 0;

/**
 * retorna l'experiència d'una persona.
 *
 * @return {@value #EXPERT} si la persona és un expert.<br>
 * {@value #APRENENT} si és un aprenent.
```

```
 */
public final int getExperiencia() {
    return this.experiencia;
}
```

Visualitza per al mètode



Les etiquetes de bloc

Per convenció l'ordre d'escriptura de les etiquetes és el següent

@author (etiqueta per a classes i interfícies)
@version (etiqueta per a classes i interfícies)
@param (etiqueta per a mètodes i constructors)
@return (etiqueta per a mètodes)
@exception (@throws és un sinònim)
@see
@since
@serial, @serialField o @serialData
@deprecated

@author

L'etiqueta @author permet definir l'autor de la classe o interfície.

Si hi ha diverses etiquetes d'autor, aquestes s'han d'escriure en ordre cronològic, amb el creador més “antic” en la part superior.

L'etiqueta d'autor no és crítica, ja que no s'inclou en la generació de la documentació del API, i només és visible per aquells que accedeixen al codi font.

@version

L'etiqueta @version permet definir la versió del codi de la classe o interfície.

La convenció de programari de Java per a l'etiqueta @version és una cadena amb el format de SCCS (Source Code Control System) del tipus %I%, %G%.

%I% és un número que s'incrementa cada vegada que s'edita un arxiu. Quan es crea un arxiu, %I% s'estableix en 1.1, quan es modifica, s'incrementa a 1.11

%G% és la data, i és del tipus mm/dd/aa

```
@version 1.39, 02/28/97
```

indica que la versió és la 1.39 del 28 de febrer de 1997.

@param

L'etiqueta @param permet definir cada paràmetre que rep un mètode o constructor.

L'etiqueta @param està seguida pel nom del paràmetre (no el tipus de dada), i després per la descripció del paràmetre.

Per convenció, en la descripció el primer substantiu és el tipus de dades del paràmetre. (int és una excepció)

```
@param ch el caràcter a testar
@param observador l'observador d'imatge al qual cal notificar
```

Les etiquetes @param tenen el mateix ordre que els arguments en la signatura del mètode.

Si la descripció és una frase, comença amb minúscules, si és una sentència, comença amb una majúscula i acaba amb punt. Després de la primera frase o sentència, es pot escriure més sentències.

```
@param x coordenada x, mesurada en píxels
@param x Especifica la coordenada x, mesurada en píxels.
```

@return

L'etiqueta @return permet definir el valor retornat per un mètode.

L'etiqueta @return ha d'aparèixer en els mètodes que retornen un valor, fins i tot si el seu contingut és totalment redundant amb la descripció del mètode.

Una etiqueta @return fa més fàcil perquè algú trobe el valor de retorn d'un mètode.

Sempre que siga possible, cal comentar els valors de retorn per a casos especials, en l'exemple, s'especifica el valor retornat si es produeix un fora de límits en la matriu.

```
/**
* Retorna un client de la llista de clients.
* @param numClient l'índex del client
* @return el client. <br>null si l'índex està fora de la matriu.
* /
public Client getClient (int numClient) {    …    }
```

L'ús de les majúscules i de la puntuació és la mateixa que per als paràmetres.

@throws

L'etiqueta @throws permet definir les excepcions que es poden produir en un mètode.

Una etiqueta @throws ha d'incloure's per a qualsevol excepció comprovada.

També, es poden incloure per a les excepcions no comprovades, i amb l'excepció de NullPointerException .

Els errors no s'han de documentar, ja que són impredictibles.

Si hi ha més d'un @throws, aquests han d'escriure's en ordre alfabètic dels noms d'excepció.

També, es pot expressar amb @exception

@see

L'etiqueta @see permet definir referències a altres elements del Javadoc.

Podem usar diversos formats

- @see text — mostra el text, solament.
- @see text — crea un enllaç mitjançant el "text"
- @see package.class#member text — crea un enllaç mitjançant el "text"

En l'exemple següent tenim un mètode que ordena una llista de clients, en el @see indiquem que mire el Javadoc de Collection i de List

```
/**
* ordena la llista de clients.
* @param clients la llista de clients
* @see     Collection
* @see     List
*/
public static void ordenaClients(List<Client> clients) {     }
```

En l'exemple següent tenim un enllaç a l'identificador section de la pàgina spec.html

```
@see <a href="spec.html#section">Java Spec</a>
```

En l'exemple següent tenim un enllaç al mètode equals de la classe String

```
@see String#equals(Object) equals
```

Si hi ha més d'un @see, aquests s'ordenen de l'accés més pròxim al més llunyà, del menys qualificats (ruta més curta) fins al totalment qualificat (ruta més llarga).

```
@see #field
@see #Constructor(Type, Type...)
@see #method(Type, Type,...)
@see Class
@see Class#field
@see Class#Constructor(Type, Type...)
@see Class#method(Type, Type,...)
@see package.Class
@see package.Class#field
@see package.Class#Constructor(Type, Type...)
@see package.Class#method(Type, Type,...)
```

@since

L'etiqueta @since permet definir des que versió l'element es va afegir al API.

Si un paquet, classe, interfície o membre ha sigut afegit a la plataforma Java 2, Standard Edition, en la versió 1.2, llavors s'escriu

```
@since 1.2
```

Quan s'afig un paquet, s'escriu un @since en la seua descripció i en cadascuna de les seues classes, però no es posa en els mètodes.

@serial, @serialField, @serialData

Les etiquetes @serial, @serialField i @serialData permeten documentar els camps, dades i classes serialitzables.

@deprecated

L'etiqueta @deprecated permet definir un mètode com a obsolet.

La primera frase ha de dir-li a l'usuari quan va quedar desfasat i el que ha d'usar per a reemplaçar-lo.

```
/**
* @deprecated As of JDK 1.1, replaced by {@link #setBounds(int,int,int,int)}
*/
```

Només la primera frase apareix en la secció de resum i l'índex del Javadoc. Sentències posteriors poden explicar per què s'ha quedat obsolet.

En l'exemple es crea un enllaç al mètode que substitueix a aquest.

Tens més informació en

https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

Exercici 1

Tenim l'aplicació amb les classes següents. Realitza els Javadocs per a tots els elements

```
public class Principal {
  public static void main(String[] args) {
    try {
     System.out.println("12345 és capicua: " + SocUtil.esCapikua(12345));
     System.out.println("1221 és capicua: " + SocUtil.esCapicua(1221));
     System.out.println("1234321 és capicua: " + SocUtil.esCapikua(1234321));
    } catch (EsNegatiuEX ex) {
    }
  }
}
```

```
public class EsNegatiuEX extends Exception {
    public EsNegatiuEX () {
```

```
        super("el valor no pot ser negatiu");
    }
    public EsNegatiuEX (String msg) {
        super(msg);
    }
}
```

```
public class SocUtil {
    public static boolean esCapicua(int numero) throws EsNegatiuEX {
        if (numero < 0) {
            throw new EsNegatiuEX();
        }
        int numAlReves = 0;
        int còpia = numero;
        while (numero > 0) {
            numAlReves = numAlReves * 10 + numero % 10;
            numero /= 10;
        }
        return còpia == numAlReves;
    }
    public static boolean esCapikua(int numero) throws EsNegatiuEX {
        if (numero < 0) {
            throw new EsNegatiuEX ();
        }
        String cadNum = numero + "";
        String numAlReves = (new StringBuilder(cadNum)).reverse().toString();
        return cadNum.equals(numAlReves);
    }
    public static boolean esPrimer(int numero) throws EsNegatiuEX {
        if (numero < 0) {
            throw new EsNegatiuEX();
        }
        int limit = numero / 2 + 1;
        int div = 2;
        while (div < limit) {
            if (numero % *div == 0) {
                return false;
            }
            div++;
        }
        return true;
    }
    public static long getFactorial(int numero) throws EsNegatiuEX {
        if (numero < 0) {
            throw new EsNegatiuEX("no es pot calcular el factorial d'un número negatiu");
        }
        long fact = 1L;
        while (numero > 1) {
            fact *= numero;
            numero --;
        }
        return *fact;
    }
}
```

Per a les classes SocUtil i EsNegativoEX defineix l'autor i la versió.

Per als mètodes, defineix la descripció (títol i algun comentari sobre l'algoritme del mètode), els paràmetres, el valor retornat i les excepcions llançades. Si ja hi ha Javadoc en anglès, passar-ho a castellà i millorar-ho.

Tots els mètodes que llancen una excepció, mostren el constructor de l'excepció utilitzada (@see).

En el mètode getFactorial posar un @see amb el text “Sangaku Maths” amb un enllaç que obri la pàgina "http://www.sangakoo.com/es/temas/el-factorial-de-un-numero" en una altra pestanya.

Marca el mètode esCapicua com a obsolet (@deprecated) des de la versió 1.35 i que s'ha reemplaçat pel mètode esCapikua.

Exercici 2

A partir de la classe següent, crea el Javadoc per a tots els elements de la classe Password.

Per a la classe Password crea una descripció, con una frase general englobe tota la funcionalitat de la classe i després una xicoteta descripció. Recorda que s’ha d’escriure en HTML. Afig l’autor i la versió, que serà la 1.5.

Per als mètodes, defineix en una frase el algoritme, encara que siguen el getters i setters. També has de posar els paràmetres i el valor retornat en cas de què existeixen.

El mètode generaPassword, ha de mostrar (@see) els mètodes random() i floor() de la classe Math.

En el mètode esFuerte has de fer referència (@see) al mètode charAt() de la classe String.

Marca el mètode esFort() com a obsolet (@deprecated) des de la versió 1.2. Es reemplaçà per el mètode esFuerte().

Per últim, per a facilitar la comprensió del codi, fes els comentaris que siguen necessaris.

```
public final class Password {
    private final static int LONG_DEF=8;

    private int longitud;

    private String contraseña;

    public int getLongitud() {
        return longitud;
    }

    public void setLongitud(int longitud) {
```

```
        this.longitud = longitud;
    }


    public String getContraseña() {
        return contraseña;
    }

    public String generaPassword (){
        String password="";
        for (int i=0;i<longitud;i++){
            int eleccion=((int)Math.floor(Math.random()*3+1));

            if (eleccion==1){
                char minusculas=(char)((int)Math.floor(Math.random()*(123-97)+97));
                password+=minusculas;
            }else{
                if(eleccion==2){
                    char mayusculas=(char)((int)Math.floor(Math.random()*(91-65)+65));
                    password+=mayusculas;
                }else{
                    char numeros=(char)((int)Math.floor(Math.random()*(58-48)+48));
                    password+=numeros;
                }
            }
        }
        return password;
    }

    public boolean esFuerte(){
        int cuentanumeros=0;
        int cuentaminusculas=0;
        int cuentamayusculas=0;

        for (int i=0;i<contraseña.length();i++){
                if (contraseña.charAt(i)>=97 && contraseña.charAt(i)<=122){
                    cuentaminusculas+=1;
                }else{
                    if (contraseña.charAt(i)>=65 && contraseña.charAt(i)<=90){
                        cuentamayusculas+=1;
                    }else{
                        cuentanumeros+=1;
                        }
                    }
            }
            if (cuentanumeros>=5 && cuentaminusculas>=1 && cuentamayusculas>=2){
            return true;
        }else{
            return false;
        }
    }
public boolean esFort(){
        int cuentanumeros=0;
        int cuentaminusculas=0;
        int cuentamayusculas=0;

        for (int i=0;i<contraseña.length();i++){
                if (contraseña.charAt(i)>=97 && contraseña.charAt(i)<=122){
                    cuentaminusculas+=1;
                }else{
```

```
            if (contraseña.charAt(i)>=65 && contraseña.charAt(i)<=90){
                cuentamayusculas+=1;
            }else{
                cuentanumeros+=1;
                }
            }
        }
        if (cuentanumeros>=3 && cuentaminusculas>=1 && cuentamayusculas>=1){
        return true;
    }else{
        return false;
    }
}

public Password (){
    this(LONG_DEF);
}

public Password (int longitud){
    this.longitud=longitud;
    contraseña=generaPassword();
}
```

Exercici 3

Crea una aplicació Java en la que uses com a mínim els següents elements, vists a la unitat anterior:

- Label
- Button
- CheckBox
- RadioButton
- Combobox
- Popup Menu
- JOptionPane, almenys tres
- Almenys tres finestres diferents
- JDialog per a confirmar tancar una finestra
- Canvi de color del background de la finestra
- Canvi del icon de la finestra

Es tracta de aprofitar les classes que haveu fet als exercicis de programació i crear una aplicació que permeta realitzar diferents tasques, emprant com a mínim els elements que s'indiquen amunt.

Després fes el Javadoc del codi, de manera similar als anteriors exercicis.

```
Per a entregar l'activitat has de crear un repositori públic a
GitHub i afegir l'enllaç a la tarea.
```