Documentation

CSC02A2

Outline

Outline

1. Basic Code Comments
2. **JavaDoc** Comments
   - **JavaDoc** Commenting
   - **JavaDoc** Tags
   - **JavaDoc** Tool
   - **JavaDoc** Example
3. UML Class Diagrams
   - UML Class Diagrams
   - Class Diagram Example
   - UML Class Relationships
   - UML Class Relationship Example

Basic Code Comments

Basic Code Comments

Most programming language have some way to indicate a comment in the source code.

Inline Comments Comments that are on a single line of code. Usually above or inline with the related source code.

Block Comments Comments that take up more space than a single line of code.

```
1 // This is an example of an inline comment
2
3 /*
4  * This is an example of a block comment
5  */
```


JavaDoc Comments

JavaDoc Commenting

JavaDoc comments are a way of specifying special comments that describe the classes, methods and usage. JavaDoc comments are specified with specific comment syntax that is different than regular code comments.

```
/**
 * JavaDoc comment for class
 */
class MyClass
{
  /**
   * JavaDoc comment for constructor or method
   */
  public MyClass()
  {
    // Regular inline comment
  }
  /*
   * Regular block comment
   */
}
```


JavaDoc Tags

JavaDoc tags are used to specify extra details about classes or methods.

`@author`	Author/Contributor to the class
`@param`	Parameter in a method or constructor. Used to provide extra information about the parameter Needs to match the variable name specified in the method or constructor
`@return`	Return value of a method. used to provide extra information about the value returned
`@see`	Used to specify other classes or methods to look at when using this specific class or method
`{@link}`	Used to link to other classes or methods. Example:`{@link MyClass#myMethod}`
`@version`	Used to specify version information. The @since tag specifies since which version a particular class or method was available.

JavaDoc Tool

The javadoc command-line tool can be used to compile **JavaDoc** comments found in source code files into a set of web pages (HTML). An additional parameter can be specified to ensure that files output by the tool is put into a specific directory.

```
1 javadoc -sourcepath %PRAC_SRC% -classpath %PRAC_BIN%;%PRAC_LIB% -use
  ↪  -version -author -d %PRAC_DOCS%\JavaDocs -subpackages csc2a
```


JavaDoc Example I

```java
/**
 * This class stores information about a ShippingBox. {@link ShippingBox#ShippingBox}
 * @author Mr MaganLal -- Your details should appear here
 * @version PL01
 * @see Warehouse A related class
 */
public class ShippingBox
{
  // Shipping Label is unique
  private final String shippingLabel;
  // Text description of the contents of the box
  private String contents;

  /**
   * Basic constructor for a ShippingBox instance
   * @param label The unique shipping label
   * @param contents The contents of the ShippingBox
   */
  public ShippingBox(String label,String contents)
  {
    this.shippingLabel = stdNo;
    this.contents = contents;
  }

  /**
   * Method to open a ShippingBox
   * @param code The code to open the box
   * @return Returns true if box successfully opened, false if the code is incorrect
   */
  public boolean open(int code)
  {
    // logic for opening
  }
}
```


JavaDoc Example II

Constructor Detail

ShippingBox

```
public ShippingBox(java.lang.String label, java.lang.String contents)
```

Basic constructor for a ShippingBox instance.

Parameters:

label - The unique shipping label

contents - The contents of the ShippingBox

JavaDoc Example III

Method Detail

open

```
public boolean open(int code)
```

Method to open a ShippingBox

Parameters:

code - The code to open the box

Returns:

- Returns true if box successfully opened, false if the code is incorrect

UML Class Diagrams

UML Class Diagrams

UML Class Diagrams allows programmers to visualise how classes are related. Typically *nouns* representing important entities in the problem will be represented by ***classes*** while the *verbs* associated with single entities will become ***operations***. Packages can be represented by placing the set of class diagrams in container with the package name in the top left corner.

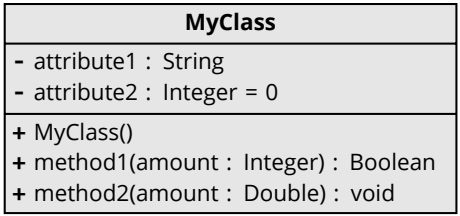
Class diagrams consists of three sections:

Class Name	Name of the class with any additional modifiers.
Attributes	List of attributes of the class showing name, visibility and type.
Operations	List of operations showing visibility, name, parameters with types and return type.

Attributes and operations can have specific visibility modifiers:

Public	Represented by **+**
Private	Represented by **-**
Protected	Represented by **#**
Package	Represented by **~**

Class Diagram Example



UML Class Relationships

Relationship	Type	Usage	Symbolic
Association	...	Named Association	
Dependency	*uses-a*	Open arrow on dependant	
Aggregation	*has-a*	Containment - Diamond on container	
Composition	*owns-a*	Containment - Diamond on container	
Generalisation	*is-a*	Arrow on base class	
Realization	*is-kind-of*	Arrow on interface	

UML Class Relationship Example