Javadoc Class and Method Documentation

Every public class and public method in your program should have a Javadoc comment. The Javadoc comment for the class specifies the programs purpose, the activity or project number, the author, and the date. Tags that start with the `@` symbol represent special information that is read differently when documentation is generated. The Javadoc comment for each method includes the purpose of the method as well tags such as `@param` for each parameter and `@return` for the return value unless the the return type is `void`. Make sure that your Javadoc comments begin with `/**` and that your class Javadoc class comment is placed directly before your class declaration but after any import statements.

```java
/**
 * Simple program to print Hello World.
 *
 * Activity 1
 * @author your name – course - section as appropriate
 * @version date
 */
public class HelloWorldCommented {

   /**
    * Prints Hello World one time.
    *
    * @param args Command line arguments – not used.
    */
   public static void main(String[] args) {

      System.out.println("Hello World");
   }
}
```

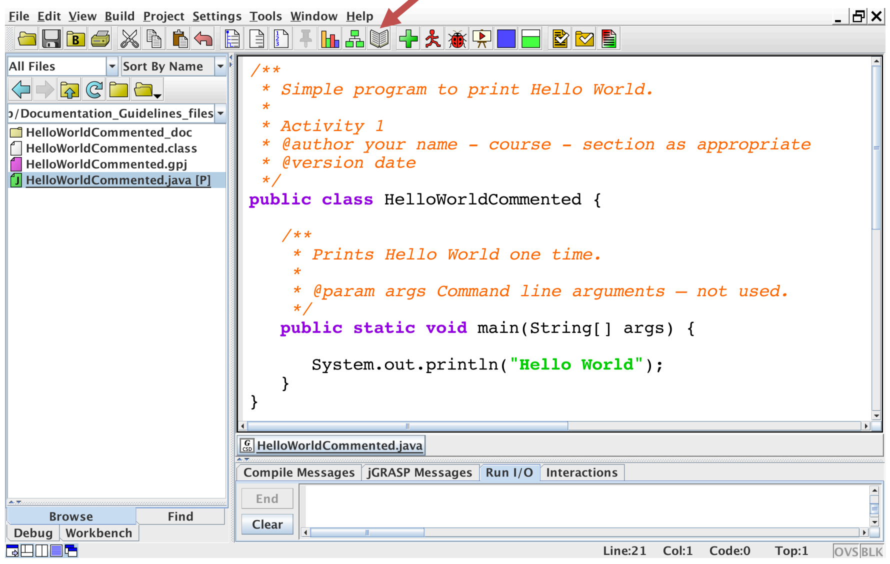
Class description – indicate what your class does.

Activity or Project number and/or name

Name and date

Class header

Checking your documentation: In order to make sure that your documentation is properly formatted, press the Generate Documentation ( ) button in jGRASP:

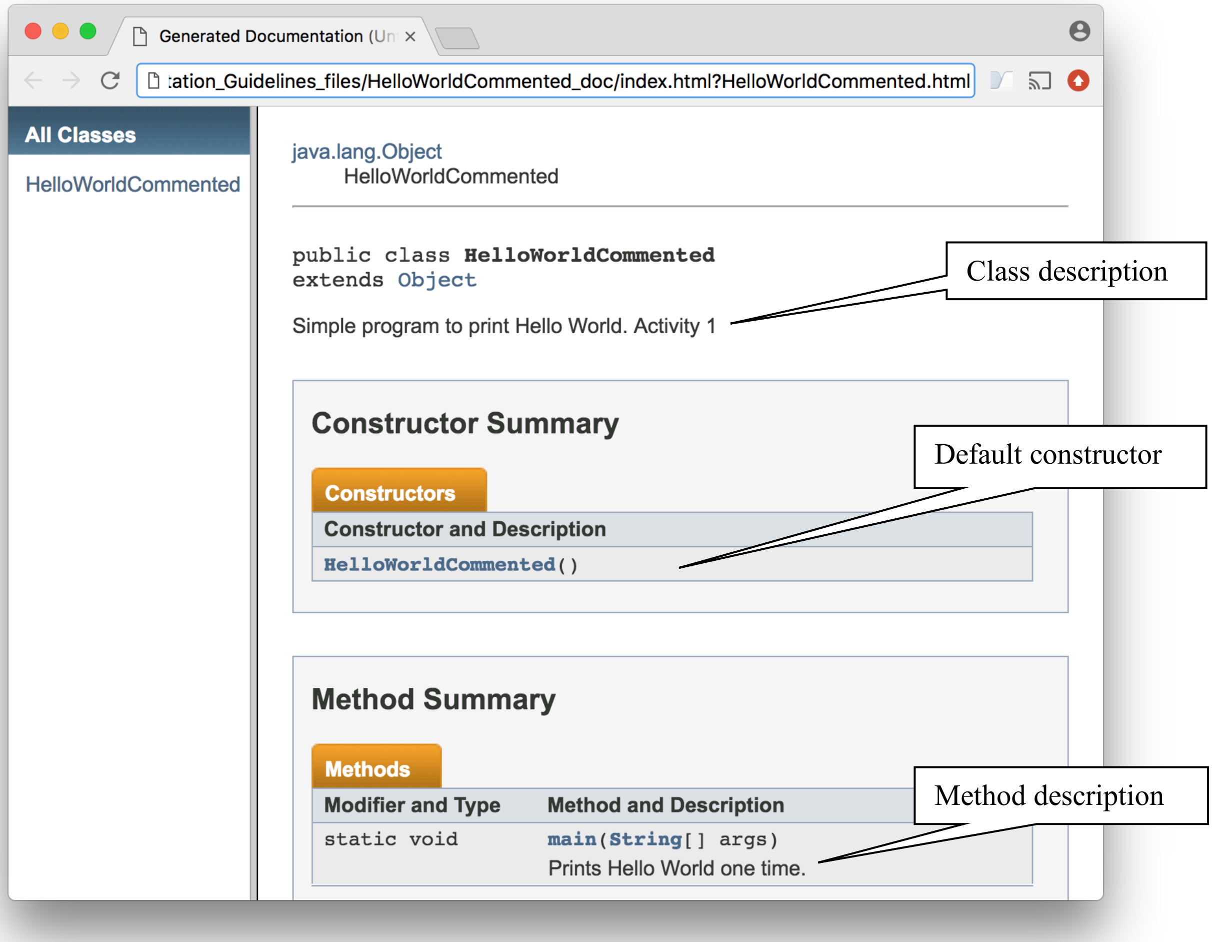


```java
/**
 * Simple program to print Hello World.
 *
 * Activity 1
 * @author your name - course - section as appropriate
 * @version date
 */
public class HelloWorldCommented {

   /**
    * Prints Hello World one time.
    *
    * @param args Command line arguments – not used.
    */
   public static void main(String[] args) {

      System.out.println("Hello World");
   }
}
```

Javadoc utility will generate a set of HTML files and then will automatically open the root HTML file (e.g., HelloWorldCommented.html) in your default Internet browser. You should review the documention to make sure that your descriptions for class and method are correctly displayed; note that your name and the date are not included when the documentation is generated. The figure below shows the class description along with the Constructor Summary and Method Summary. Notice that although this class did not contain a constructor, Java provides a default parameterless constructor, which is shown in the constructor summary.


Generated Documentation (Un
:ation_Guidelines_files/HelloWorldCommented_doc/index.html?HelloWorldCommented.html
All Classes
HelloWorldCommented
Class description
Default constructor
Method description

java.lang.Object
HelloWorldCommented

```
public class HelloWorldCommented
extends Object
```

Simple program to print Hello World. Activity 1

Constructor Summary

Constructors

Constructor and Description
`HelloWorldCommented()`

Method Summary

Methods

Modifier and Type	Method and Description
`static void`	`main(String[] args)` Prints Hello World one time.

A Few More Detail for Documenting the `main` Method: Your main method should also have a Javadoc comment. Note that the Javadoc comment for main includes an `@param` tag for the parameter `args`; you can just copy the tag from below for your program, since command-line arguments and parameters in general will be discussed at a later date.

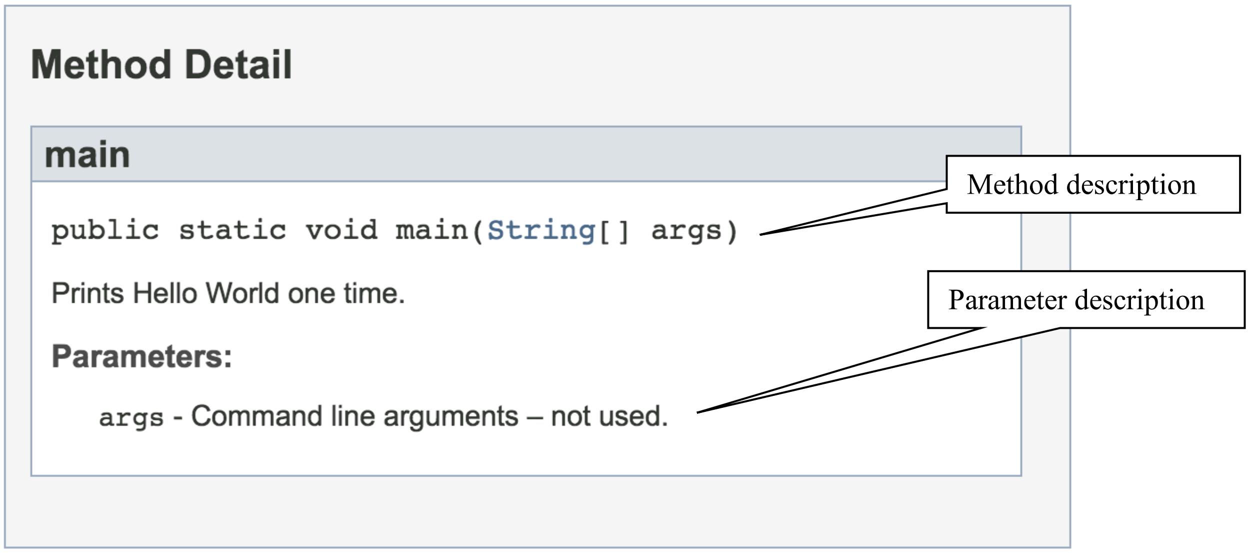
Method description

Parameter description

```
/**
 * Prints Hello World one time.
 *
 * @param args Command line arguments – not used.
 */
public static void main(String[] args) {

   System.out.println("Hello World");
}
```

Checking your documentation: In the documentation that was generated previously, scroll down to find the Method Detail section as shown below. It should contain your method description as well as your `@param` description.



Method Detail

main

```
public static void main(String[] args)
```

Prints Hello World one time.

Parameters:

`args` - Command line arguments – not used.

Method Documentation (when writing methods other than `main`)

As with your main method, all other public methods in your program need to have a Javadoc comment before the method header. The Javadoc comment should describe exactly what the method does, and should have the following tags for your formal parameters and return values:

`@param variableName` – Description of the parameter
`@return` – Description of the return value.

Consider the Javadoc comment for the `spend` method of the CreditCard class; note that the `spend` method has a `void` return type so there is no `@return` tag in the Javadoc comment.

```java
/**
 * Adds an amount spent to the balance of the credit card.
 * Also adds one travel mile for every full dollar spent.
 *
 * @param amountSpent The amount spent by the user.
 */
public void spend(double amountSpent) {
   balance += amountSpent;
   travelMiles += (int)amountSpent;
}
```

The method takes one parameter called amountSpent, and so the `@param` tag is used to specify the parameter name and what the parameter represents:

$\texttt{@param}\ \underset{\text{Variable name}}{\underset{\uparrow}{\texttt{amountSpent}}}\ \underset{\text{Description}}{\underset{\uparrow}{\underline{\texttt{The amount spent by the user}}}}\texttt{.}$

If your method has more than one parameter, you will have to have an `@param` tag for each parameter. Take a look at the following example:

```java
/**
 * Adds an amount spent to the balance of the credit card.
 * Adds one travel mile for every full dollar spent as well as any
 * additional (bonus) miles specified.
 *
 * @param amountSpent The amount spent by the user.
 * @param additionalMiles Additional (bonus) travel miles.
 */
public void spend(double amountSpent, int additionalMiles) {
   balance += amountSpent;
   travelMiles += (int)amountSpent + additionalMiles;
}
```

You will also need to specify what each method returns. If the method does not return a value (`void`) as in the methods above, you do not need an `@return` tag. Take a look at the following method which has a return type but no parameters:

```java
/**
 * Returns a String representation of the object,
 * including customer name, interest rate, balance,
 * and accumulated travel miles.
 *
 * @return String representation of credit card object.
 */
public String toString() {
   String output = "Name: " + customer + "\n" +
      "Interest rate: " + interest +"%\n" +
      "Balance: $" + balance +"\n" +
      "Travel Miles: " + travelMiles +"\n";
   return output;
}
```

Note that the method description indicates exactly what is included in the output of the `toString` method and a description of the return after the `@return` tag (you should never have a variable name in your `@return` tag).

$$\texttt{@return }\underbrace{\texttt{The String representation of credit card object.}}_{\text{Description only}}$$

If a method has both a return and multiple parameters, then you will need to have an `@param` tag for each parameter and an `@return` tag for the return. Example:

```java
/**
 * Adds interest to a card by multiplying the balance by the
 * interest rate and adding that amount to the old balance.
 * Returns the balance of the card after the interest has been
 * added.
 *
 * @param interestRate The interest rate in decimal format.
 * @return The current balance after adding the interest.
 */
public balance chargeInterest(double interestRate) {
   balance += balance*interest;
   return balance;
}
```

Constructor Documentation (when writing your own class from which objects will be created/instantiated)

You should document your constructor exactly as you would a method. Two things to take into consideration when documenting a constructor are as follows:

- You will never have an `@return` statement for a constructor, as constructors do <u>not</u> have a return type, not even `void`.
- You need to describe what happens when an object is set up. This is especially important when values are initialized in the constructor with no parameters.

See the example below. Note that it describes how the `balance` and travel miles are initialized even though there are no inputs given for those values.

```
/**
 * Creates a new credit card object with a given customer name
 * and interest rate. The balance of the account and the number
 * of travel miles are initialized to 0.
 *
 * @param customerName The name of the customer.
 * @param interestRate The interest rate on the credit card.
 */
public CreditCard(String customerName, double interestRate) {
   customer = customerName;
   interest = interestRate;
   balance = 0;
   travelMiles = 0;
}
```

The @throws Tags for Documentation (introduced with File I/O)

If a method or constructor calls method that *throws* a checked exception. For example, when a `Scanner` is created on a `File`, a `FileNotFoundException` is thrown if the file is not found. The calling method or constructor must either catch the checked exception in a try-catch statement <u>or</u> it must add a throws clause to the method head indicating that it may throw the exception. When the latter is the case, the Javadoc comment must contain an `@throws` tag which names the exception (e.g., `@throws FileNotFoundException`). The `@throws` tag usually follows the `@param` and `@return` tags.