

Terminology

Object-oriented programming	Identifiers	Compiler	Syntax
Java API (class library)	Reserved words	Interpreter	Semantics
Documentation	White space	Source code	Compile-time error
Comments	Machine language	Java bytecode	Run-time error
Class	Assembly language	IDE	Logic error
Method	High-level language	Debugger	
	Editor	SDK / JDK	

Comments in Java

Java has three types of comments: single line (`//`), multiple line (`/* */`), and Javadoc (`/** */`).

`// single line comment - goes through the end of the line`

`/* multiple line comment - begins on this line but may end on another line; ends with */`

`/** Javadoc comment - begins on this line but may end on another line; ends with */`

Course Coding Standard

A **coding standard** consists of guidelines that are used by developers when writing source code to ensure a consistent format. The course coding standard is supported by Checkstyle, which is an automated tool that works with the jGRASP IDE and the Web-CAT program grading system to automatically detect style errors based on the rules or checks adopted for the course. The first two rules are introduced below.

- For activity and project assignments, all public classes and public methods require Javadoc comments. In the HelloWorldCommented program below, the text in *italics* describes the information that goes on the line. The information shown should be replaced by your own information.

```
/**
 * Simple program to print Hello World. Description of your program goes here.
 *
 * Activity 1 (your activity or project number goes here).
 * @author your name — course — section as appropriate
 * @version date
 */
public class HelloWorldCommented {
    /**
     * Prints Hello World one time. Usually shorter description here.
     * @param args Command line arguments — not used.
     */
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

- Lines of code must not exceed 80 characters (including indentation). You can continue string literals on a new line by using string concatenation which creates a string from two or more other strings).

For example:

```
System.out.println("Suppose this is a long output string.");
```

Can be rewritten as:

```
System.out.println("Suppose this is a long "
    + "output string.");
```

The output of these two statements is the same; i.e., the concatenation does not change the output.

Goals

By the end of this activity you should be able to do the following:

- Create a program then compile and run it
- Add Javadoc comments to your program
- Generate documentation for your program
- Correct your source code structure using Checkstyle
- Submit your completed program to Web-CAT for grading



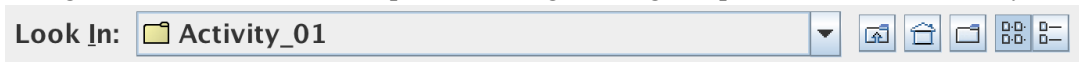



Directions

Part A: Using jGRASP and Checkstyle


- **Set up file folder** – Open a file browser on your computer (e.g., File Browser on Windows or Finder on MacOS), and set up an appropriately named file folder for this activity (e.g., Activity_01).
- **Open jGRASP** – e.g., On Windows, click the **Start** button in the lower left corner of the screen, then in the Search window, enter **jgrasp** then when **jGRASP** selected. On MacOS, enter Cmd-space, enter **jgrasp**, then double-click on it.
- Using the top menu in jGRASP, open a new Java window (**File > New > Java**). Enter the following Java statements to create a class called StudentInfo:

```
public class StudentInfo
{
}







```

- Click the Save  button on the top menu, then in the Save As dialog, use the buttons to the far right of “Look In” near the top of the dialog to navigate up  to the folder where you

want to save your program. If you have not yet created a course folder (and within it a folder for this activity) you can do that now by clicking the folder  button to create a new folder. Click once on the name “New folder” to select it then click on it again and change the name to number of this course. Double-click the folder to open it, then (as above) create a folder for this activity and open it. If your program code is correct, you should see StudentInfo.java as the file name. If this is not the case, enter the file name manually. Now click the Save button at the bottom of the dialog.
- Click the Compile button , and fix any compile-time errors.
- Generate your CSD by either pressing F2 or by clicking the Generate CSD button  on the toolbar at the top of the jGRASP desktop. Now turn on Auto Generate CSD (View > then check the **Auto Generate CSD** box), so that the CSD will be generated each time you Load or Compile a file. You may turn this off/on at any time.
- Add a main method to the class. Be sure to replace the blank in the code below with the method name (don’t forget to re-generate the CSD):

```
public class StudentInfo
{
    public static void _____(String[] args)
    {
    }
}
```

- Click the Compile button , and fix any compile-time errors.
- Now add the Javadoc comments for the class and the main method of this program by adapting the Javadoc comments on page 1 so that they describe this program and have your name, course info, and today's date.
- Inside the main method, write three or more print statements: (1) the first prints **Name :** followed by your name (first and last) on the first line, (2) the second that prints **Previous Computer Courses :** on the second line, (3) the third prints three spaces followed by the name of a computer course you have had (or **NONE** if this is your first course), (4) if you have had more than one computer course, print an additional line for each course (be sure to begin each course line with three spaces). Don't forget to put double quotes around the String literals in the println statements.

```
public static void main(String[] args)
{
    System.out.println("Name: _____");
    System.out.println("Previous Computer Courses:");
    System.out.println("    _____");
}
```

- Click the Compile button , and fix any compile-time errors. Note that if Auto Save is on, which is the default, your program is saved each time you compile.
- Click the Run button . Ensure that your output is correct.
- Click the Run Checkstyle button  on the jGRASP toolbar and correct any issues identified by Checkstyle. *If you don't see the Checkstyle button then Checkstyle has likely not been installed on your machine. On your personal computer, you will need to download and install (unzip) Checkstyle. You may also need to configure Checkstyle in jGRASP (**Tools > Checkstyle > Configure**) so that jGRASP can find the folder containing the Checkstyle JAR file.*
- Toggle line numbers on/off by clicking  on the toolbar. Most users leave line numbers on.
- Click the Browse button  on the toolbar. This opens the jGRASP Browse tab on the folder that contains the file in the CSD window. You should see StudentInfo.java underlined in the Browse tab. You should also see the corresponding .class file (StudentInfo.class) that was generated by the compiler if your file compiled successfully. This file contains the bytecode for your program that is used to run your program when you click the Run button .

In Part B you will develop a second program CourseInfo.java and then submit both StudentInfo.java and CourseInfo.java at the same time to Web-CAT.

Part B: Using Checkstyle, Web-CAT, and jGRASP Projects

1. Using Checkstyle to Find and Correct Style Errors

- Download the Part B zip file and save it in your folder for this activity. Then extract the CourseInfo.java file (Windows: right-click on the zip file then select **Extract All ...**; Mac OS X: just open the zip file).
- Open CourseInfo.java by double-clicking on the file in the Browse tab. You will be responsible for correcting style errors and logic errors that are present in the program. The first step is to modify the program to adhere to the course coding standard. To do this, run Checkstyle and correct all of the formatting issues that appear.
 - HINT: Checkstyle states that the main method is missing a Javadoc comment, and this is true. Hint: Make sure that you know the difference between a *// to end of line comment*, a */* single or multiple line comment */*, and a */** Javadoc comment */*.
 - HINT: For dealing with a source line over 80 characters, see page 1 of this activity.
- The program's **expected output** is shown below (the line numbers on the left are **not** part of the expected output):

1	This course provides an introduction to Java and
2	object-oriented programming.
3	
4	The course also introduces the basics of software development.
5	

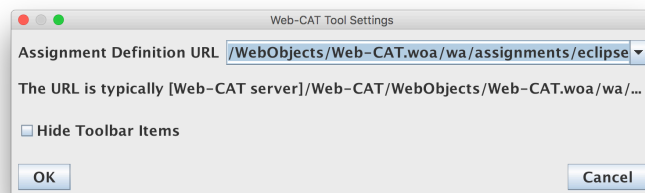
The original author of the program claims the program is correct because it "gets the point across," but you know that it is incorrect because the **actual output** of the program does not match the **expected output** above when you run it during testing. Correct the output errors by modifying the program. *Hint: Look for issues in formatting/spacing, spelling, capitalization, etc. One way to skip a line is print an empty line with the statement:*

```
System.out.println();
```

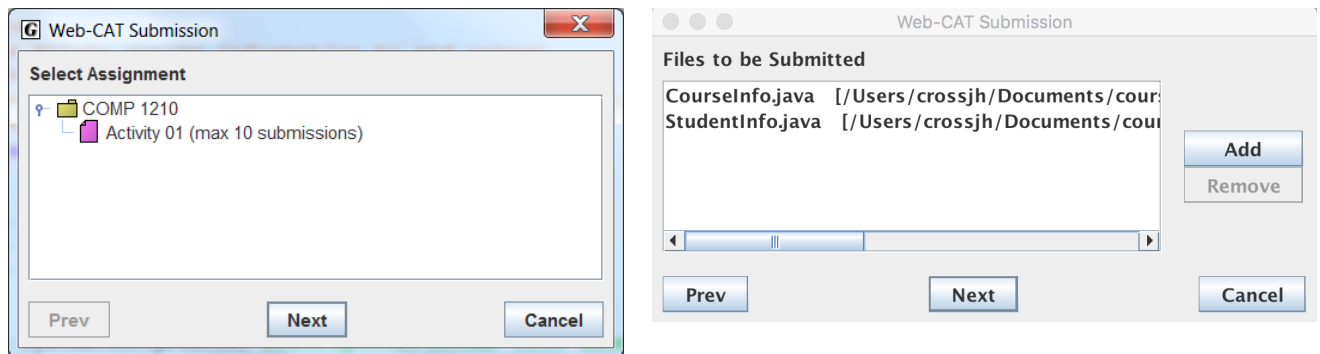
2. Using Web-CAT to Grade Your Program

In jGRASP, navigate to **Tools > Web-Cat > Configure**. In the Web-CAT Tool Settings dialog, delete the current entry (if any) for Assignment Definition URL. Select (rather than click) the URL below, right-click and select **Copy**, and then, right click and select **Paste**. Finally, click OK. You should now have a Web-CAT button 🐱 on the toolbar. (See the Web-CAT documentation on the class website for more information.)

<https://webcat2.eng.auburn.edu:8443/Web-CAT/WebObjects/Web-CAT.woa/wa/assignments/eclipse>



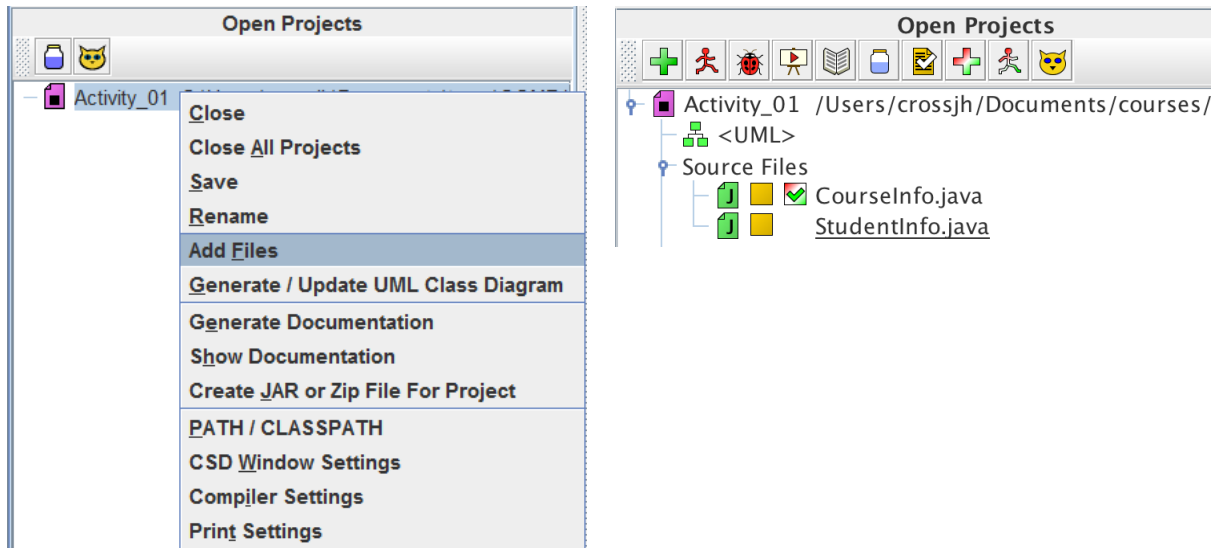
- Open the Java file that you want to submit then you can submit your program via Web-CAT. Click the Web-CAT toolbar button 🐱 (or go to **Tools > Web-Cat > Submit File**), and the Web-CAT submission dialog will open with a list of courses and assignments open for submission. If multiple courses are listed, be sure to select the one in which you are enrolled; otherwise, your submission will fail. For example, in the figure below the COMP 1210 course folder is shown opened with the assignment labeled **Activity 01**. Click **Activity 01** to select it, click **Next**; click **Add** to add the other file for this activity so that both CourseInfo.java and StudentInfo.java are listed. Then click **Next**, login to Web-CAT, and click **Submit**. A web page should open indicating your Web-CAT submission is being processed and usually within a few seconds you should see your results.



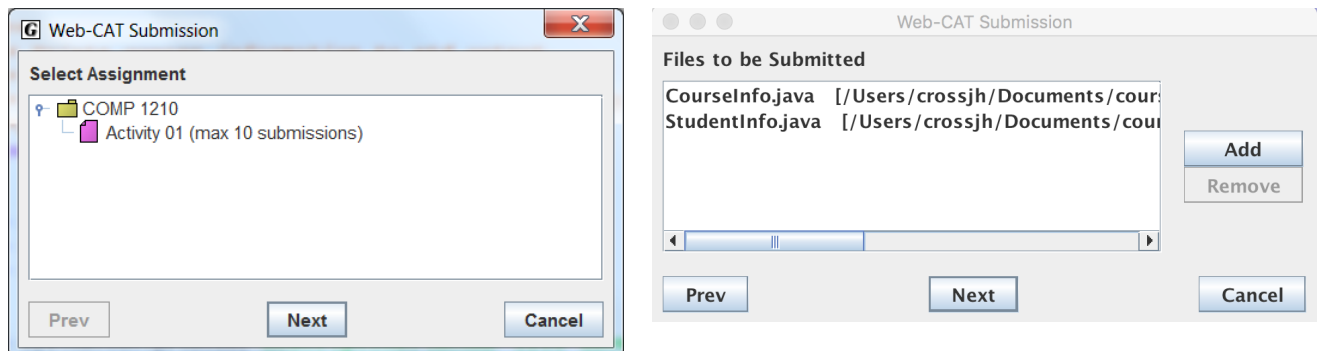
- You should try for the maximum number of points in Web-CAT for Style/Coding + Correctness/Testing (TA points will be assigned latter). You will have 10 tries so if you make changes/corrections, be sure that your files compile/run as expected in jGRASP and that they pass the Checkstyle audit before you submit your files again to Web-CAT.
 - Remember, in order to submit the two or more files, you can press the Web-CAT 🐱 button on one file and then add the other files prior to submission (as described above) OR you can create a jGRASP project (see below).
3. **Creating a jGRASP Project and submitting to Web-CAT (recommended when submitting multiple files)**

You can create a jGRASP project, add both Java files to the project, and then submit the project to Web-CAT. This will simplify the submission process and save time, especially as the number of files in assignments increases.

- To create a project in jGRASP, go to **Project > New** and make sure that you are in the same directory as your source code files. For *Project Name*, enter a project name (e.g., **Activity_01**) and click **Next**. Click **Next** again to create the project.
- Now in the lower part of the Browse tab you should see an **Open Projects** section. Right-click the project name and select **Add Files**. Add your java files to the project.



- Once all files are added, click the Web-CAT symbol 🐱 on the **Open Projects** tab.
- Select the **Activity 01** assignment, click **Next**, make sure that all files in the are included, click **Next**, login to Web-CAT (if not already logged in), and click **Submit**.



- Automatically **generate project documentation** in the form of linked web pages from your Javadoc comments. As a last exercise for this activity, find the Open Projects toolbar in Browse tab click the Show/generate project documentation button . A web page should open showing your project documentation. Most software organizations require Javadoc (or similar) comments in all programs to support the automatic generation of documentation from the programs, where each program may range from a few files to thousands of files.
- In the future, you should always begin each activity or project by creating a separate folder for it. After you have created one or more .java files and saved them to the folder, you should create a jGRASP project and add the .java files. In addition to saving time when you submit to Web-CAT, using jGRASP projects has a number of other advantages that we will explore as the course progresses.