ITSC 1213 Module 1 Lab
Java Documentation and Debugging

Introduction

An important aspect of programming is utilizing code that has been written by others (reuse) which is often facilitated by documentation. In this lab we will review some Java basics as we utilize Java libraries. We will review accessing Java libraries and their documentation and how to create our JavaDoc for the classes you create. We will also review creating objects, calling methods, outputting to the console, and how to use the debugger tool in NetBeans.

Concepts covered in this lab:

- Accessing Java API Documentation
- Importing packages
- Creating objects and call methods
- Generating console output in Java
- Creating Java Documentation
- Using the debugger tool in NetBeans
- Capturing user input in Java

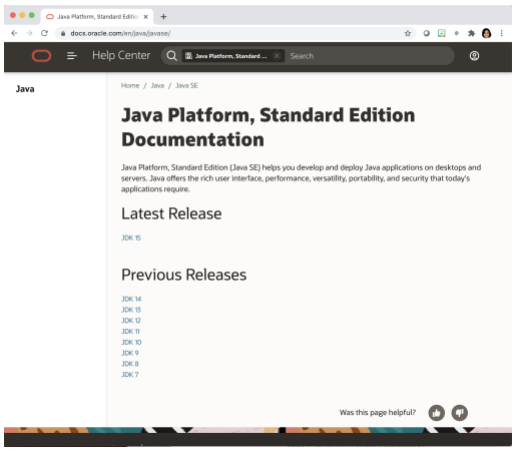
Required files or links

- Circle.java
- DebugExample.Java

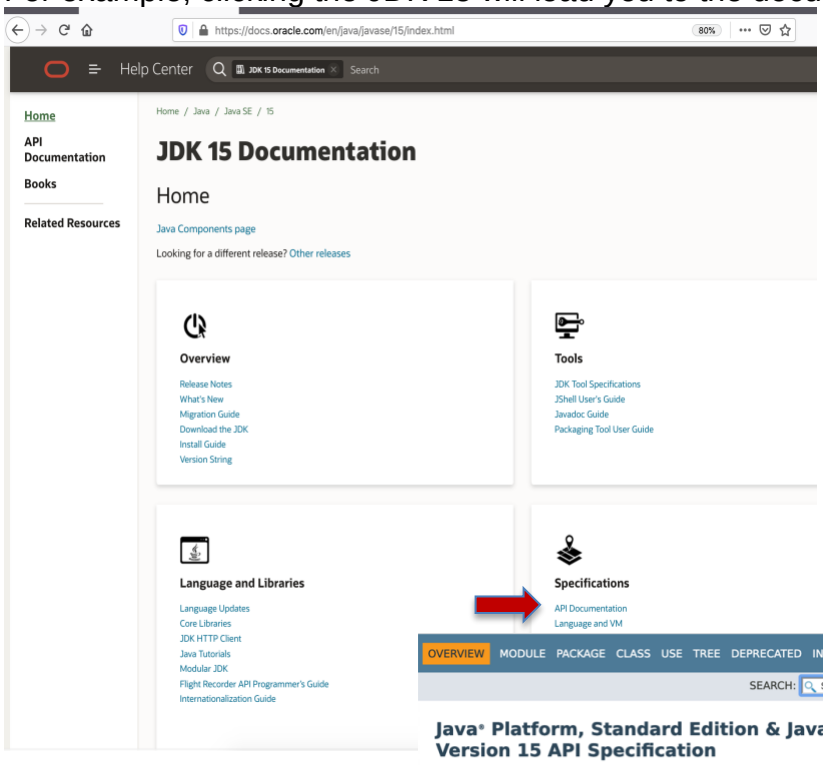
Part A: Browsing the Java API Documentation

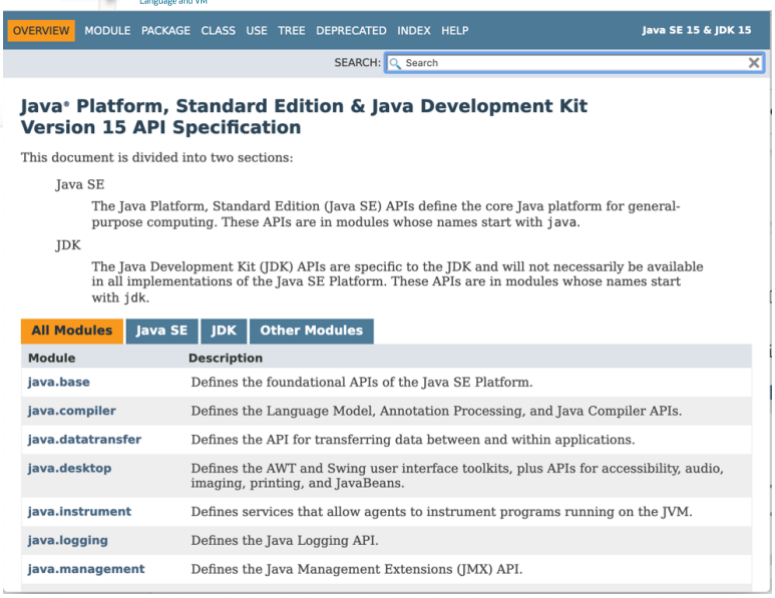
Objective: Be able to use the Java API documentation for different Java classes

1) Java provides thousands of built-in classes that you can use. It is neither necessary nor useful to memorize the details of each class. Instead, it is important to get comfortable with using the Java API Documentation, which describes how to use these classes. You can find the API documentation at: https://docs.oracle.com/en/java/javase/



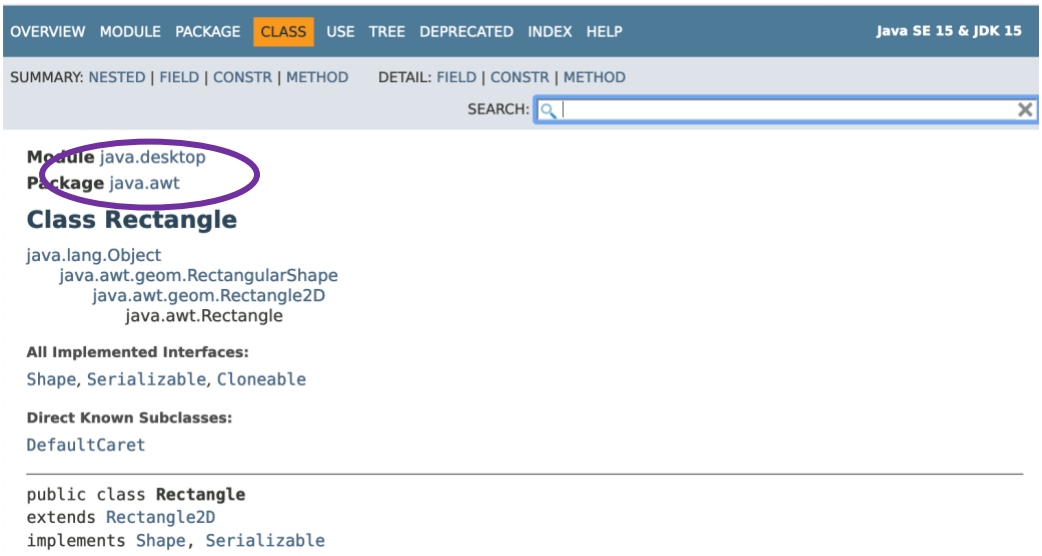
For example, clicking the JDK 15 will lead you to the documentation for that version



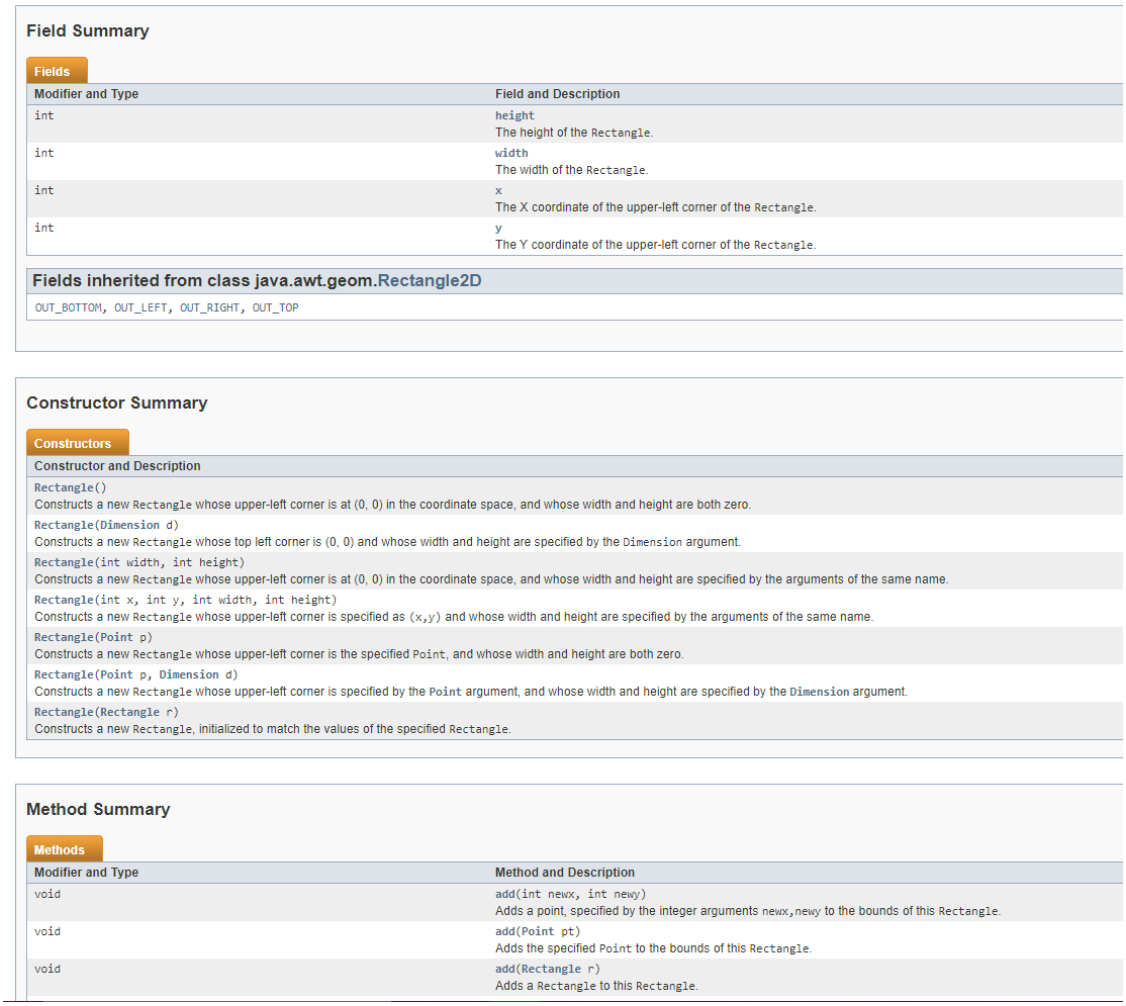


2) Given the large number of built-in classes, Java organizes related classes into **packages**, and related packages into **modules**. It is important to know the package that a given class belongs to because to use the class, one needs to write an **import** statement that includes the package name.

3) Now let's take a look at the Javadoc for the Rectangle class. To do this, go to the above website and select the JDK version you are using Then, in the search bar on the top right corner of the page type "Rectangle".
4) Look through the Javadoc for the Rectangle class. As you can see, it starts with the names of the module and the package that the class belongs to. Note that, for this class, the package is "java.awt". To use this class in a Java program, you will need to include the statement "import java.awt.Rectangle;" or "import java.awt.*;". The difference is that the first import statement only enables use of the Rectangle class, while the second enables use of any class defined in this package. The asterisk is commonly used to represent a "wild-card" type of value and here it means include all the classes that are a part of the java.awt package.



5) In addition, the Javadoc includes a section that describes the purpose of the class, a list of fields, and a summary of the constructors and methods. You can click on any method to get more details, which include: method declaration, a description of the method, a list of parameters (if any), and a description of the method's return value.



Field Summary

Fields

Modifier and Type	Field and Description
int	height The height of the Rectangle.
int	width The width of the Rectangle.
int	x The X coordinate of the upper-left corner of the Rectangle.
int	y The Y coordinate of the upper-left corner of the Rectangle.

Fields inherited from class java.awt.geom.Rectangle2D

OUT_BOTTOM, OUT_LEFT, OUT_RIGHT, OUT_TOP

Constructor Summary

Constructors

Constructor and Description
Rectangle() Constructs a new Rectangle whose upper-left corner is at (0, 0) in the coordinate space, and whose width and height are both zero.
Rectangle(Dimension d) Constructs a new Rectangle whose top left corner is (0, 0) and whose width and height are specified by the Dimension argument.
Rectangle(int width, int height) Constructs a new Rectangle whose upper-left corner is at (0, 0) in the coordinate space, and whose width and height are specified by the arguments of the same name.
Rectangle(int x, int y, int width, int height) Constructs a new Rectangle whose upper-left corner is specified as (x,y) and whose width and height are specified by the arguments of the same name.
Rectangle(Point p) Constructs a new Rectangle whose upper-left corner is the specified Point, and whose width and height are both zero.
Rectangle(Point p, Dimension d) Constructs a new Rectangle whose upper-left corner is specified by the Point argument, and whose width and height are specified by the Dimension argument.
Rectangle(Rectangle r) Constructs a new Rectangle, initialized to match the values of the specified Rectangle.

Method Summary

Methods

Modifier and Type	Method and Description
void	add(int newx, int newy) Adds a point, specified by the integer arguments newx,newy to the bounds of this Rectangle.
void	add(Point pt) Adds the specified Point to the bounds of this Rectangle.
void	add(Rectangle r) Adds a Rectangle to this Rectangle.

6) Spend several minutes reading the Javadoc of the Rectangle class and answer each of the following questions:
   - In which package is the Rectangle class located?
   - What do you need to do in order to use the Rectangle class in your program?
   - How many constructors does the Rectangle class have?
   - Find the errors in the following statements:

```
r = new Rectangle();
r.translate(10, "Hello");
```

7) Show your answers to the instructional team to be checked off and proceed to **Part B.**

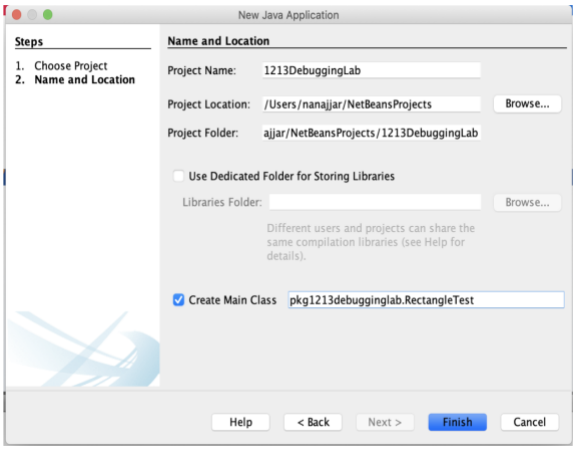
Part B: Using the Rectangle class

Objective: Be able to create and work with objects of the Rectangle class.

In part A, you got familiar with browsing the Java API Documentation. In this part, you will consult the Javadoc and write a program using the Rectangle class. Note that the Rectangle class is usually used in conjunction with a GUI (Graphical User Interface). Today, you will use it in a regular console application. The goal of this part is for you to get more practice with using built-in Java classes.

1) Open NetBeans, choose **File** → **New Project**…, then choose 'Java with Ant' (or "Java" in Netbeans 8), then choose '**Java Application**', and click '**Next >**'.

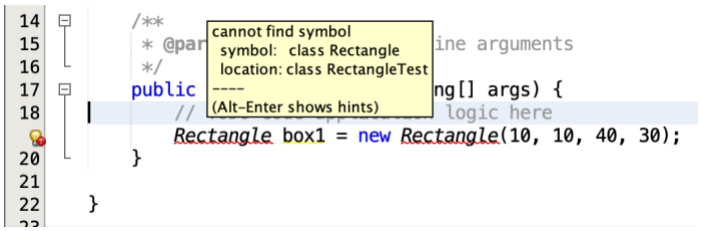
2) Create a project called **1213DebuggingLab**. Leave the '**Create Main Class**' checkbox checked. Name the class **RectangleTest** instead of **Main** and then click '**Finish**'. This will create the package and the file under the **src** folder and opens the RectangleTest.java source code file in the editor pane.



3) Now, inside of the main method, type the following statement to create a rectangle.

```
Rectangle box1 = new Rectangle(10, 10, 40, 30);
```

4) You will see that "Rectangle" has a red underline and if you move your cursor to the lightbulb on the left. You will see the following message: cannot find symbol: class Rectangle. This is because the Rectangle class is defined in a different package, and the compiler does not know where to locate it.



5) There are two ways to fix this error
   - add an import statement right below: `package pkg1213debugginglab;`
   - click on the lightbulb and select the first suggestion, which will add the import statement automatically.

6) Once you fix the error, check if the box is created properly by adding the following statement:

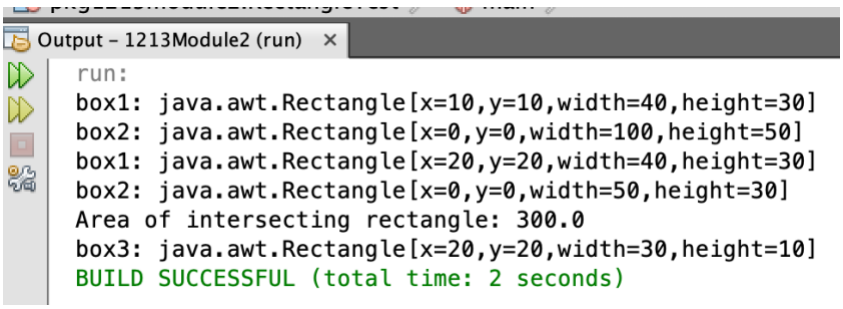
```
System.out.println("box1: "+ box1);
```

7) Run the **RectangleTest** by clicking '**Run** → **Run File**' or right clicking inside the file viewer and then Run File. You should see this:

```
box1: java.awt.Rectangle[x=10,y=10,width=40,height=30]
```

8) Now use the Rectangle class to complete the following tasks:
   - Create another object of the Rectangle class named box2 with a width of 100 and height of 50. Note that we are not specifying the x and y position for this Rectangle object. Hint: look at the different constructors)
   - Display the properties of box2 (same as step 7 above).
   - Call the proper method to move box1 to a new location with x of 20, and y of 20.
   - Call the proper method to change box2's dimension to have a width of 50 and a height of 30.
   - Display the properties of box1 and box2.
   - Call the proper method to find the smallest intersection of box1 and box2 and store it in reference variable box3.
   - Calculate and display the area of box3. Hint: call proper methods to get the values of width and height of box3 before calculating the area.
   - Display the properties of box3.

9) Sample output of the program is as follow:

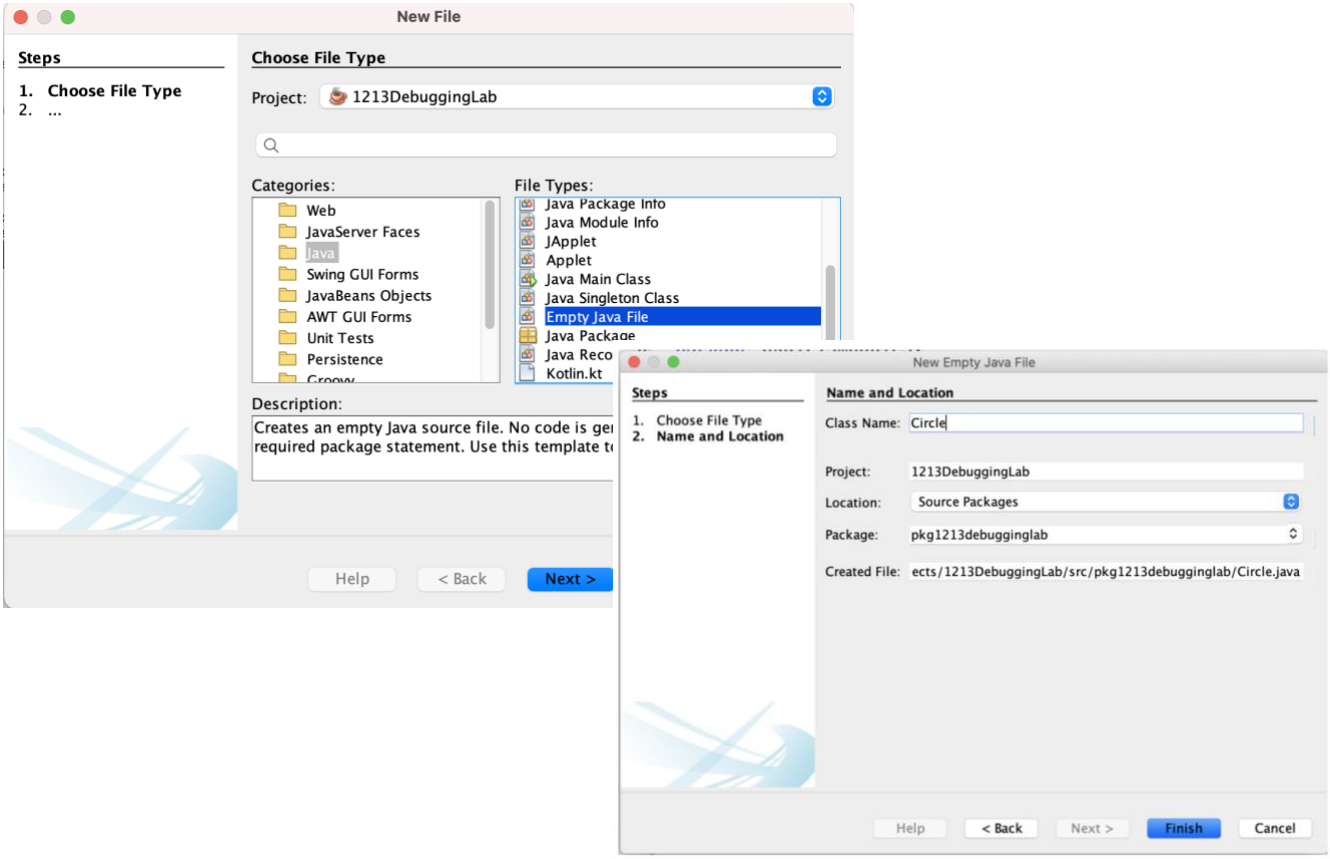


10) When you are satisfied with the output, show your work to the instructional team to be checked off and proceed to Part C.

Part C: Creating Java Documentation

Objective: Be able to add JavaDoc comments and create the Java documentation for a class

1) Create a new Empty Java class called **Circle.java**. Make sure that it is added to the correct package.



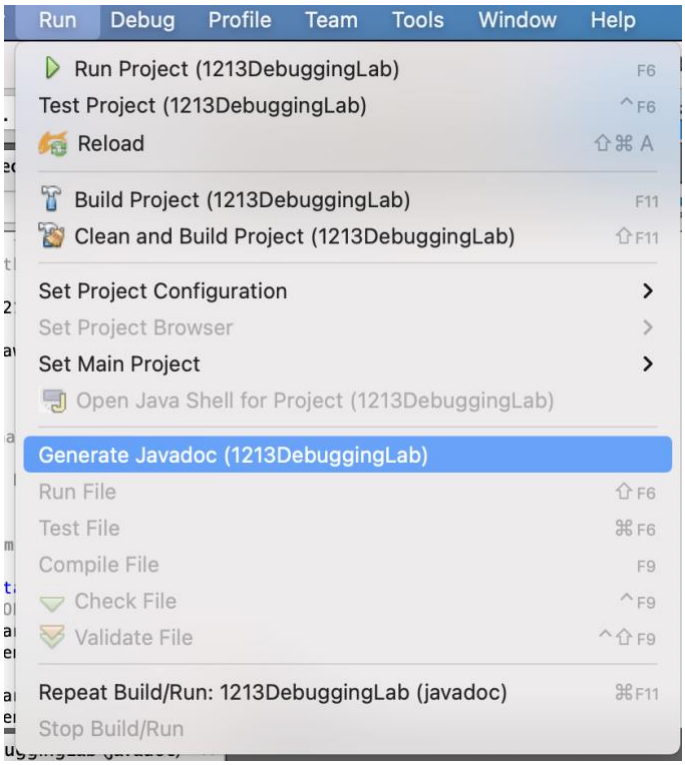
2) Copy the contents from [this code snippet] into the **Circle.java** source code file.

3) Make sure to add the package declaration to the beginning of the file
   `package [packagename];`

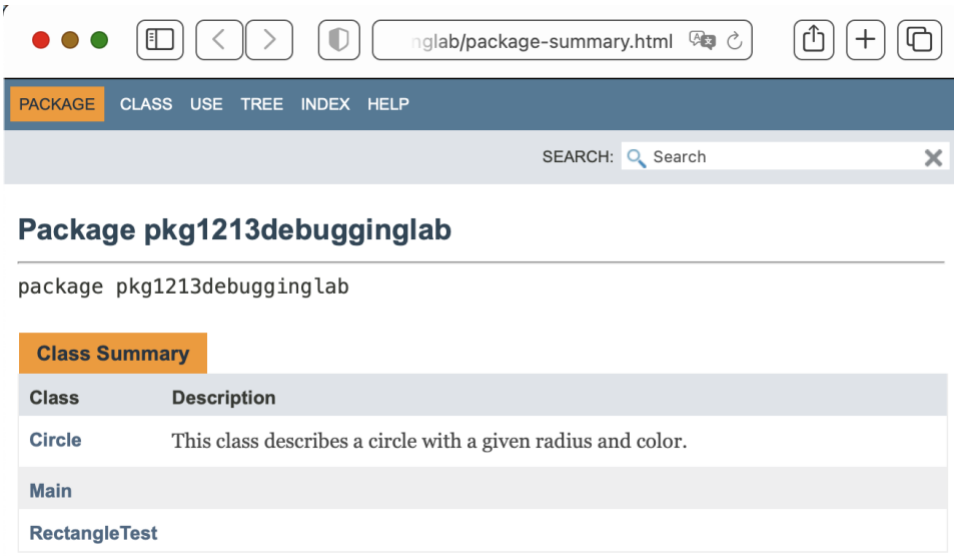
4) Examine the code and develop an understanding of the code you just copied. At this point you should be able to follow along each line in this code and know what it means and its purpose inside this class. Take a few minutes to note the different components of this class and identify all the fields (instance variables), constructors, and methods.

5) Now, add in the appropriate Java comments. Be sure to include:
   - A documentation comment for the overall class, including an @author tag
   - Documentation comments for each of the methods and constructors (a summary that describes the purpose of the method)
   - @param tags for all parameters (describes each of the parameters in the parameter list (if any)).
   - @return tags for all return values (describes the information that is returned to the calling statement (if any)).

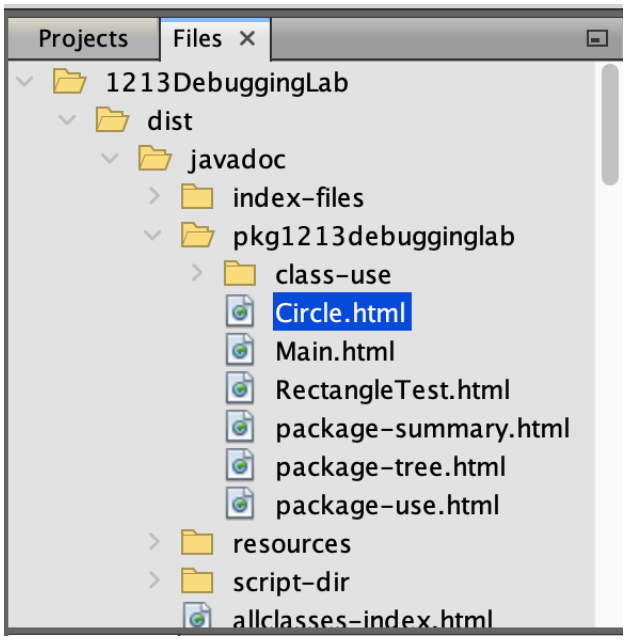
6) A description of anything that someone using the class might need to know.
   When you are done adding comments to the **Circle** class you are ready to generate the Java doc. NetBeans supports generating HTML files that can be viewed using any web browser. To generate the Java doc select your project in the Projects Pane, then choose **Run → Generate Javadoc ([PROJECT NAME])**.



7) This will open a web browser window. On the left panel of this window will be a list of classes. Select the **Circle** class to load its Java doc. Check the Java doc to ensure your comments were put in correctly.



8) In the future, if you need to access the Javadoc of the **Circle** class, simply click on the Files tab in the upper left (beside the Projects tab). Expand **[Project Name]** → **dist** → **javadoc** → **[Project Name]**, right-click **Circle.html** then click View.



9) When you are satisfied with the JavaDoc, show your work to the instructional team to be checked off and proceed to Part D.

Part D: Using the Debugger

Objective: Be able to use the debugger tool in NetBeans to inspect code during execution

Most development environments, including NetBeans, have a *debugger*, a tool that lets you execute a program in slow motion. You can observe which statements are executed and you can peek inside variables. Debuggers can be complex, but fortunately there are only three commands that you need to master:

1. Set breakpoint
2. Single step
3. Inspect variable

In this lab, we will use the debugger that is a part of NetBeans.

1) For this Part, create a new Java class with a main method called **DebugExample.java**.

2) Copy the contents from this code snippet into the **DebugExample** main method.
3) Notice that there are few **compilation errors** in the program.
   Note: The compiler is always running behind the scene. Thus, there is no "Compile" option in this environment to choose from.

4) The very **1st compilation error** is in the **main** method.

5) Change **int** to **void** in the **main** method to remove the compilation error. The **main** method is always **void** as it **doesn't return any value**.
   public: so that it can be accessed outside the class.
   static: so that it can be invoked without creating any objects.
   void: as it doesn't return any value.

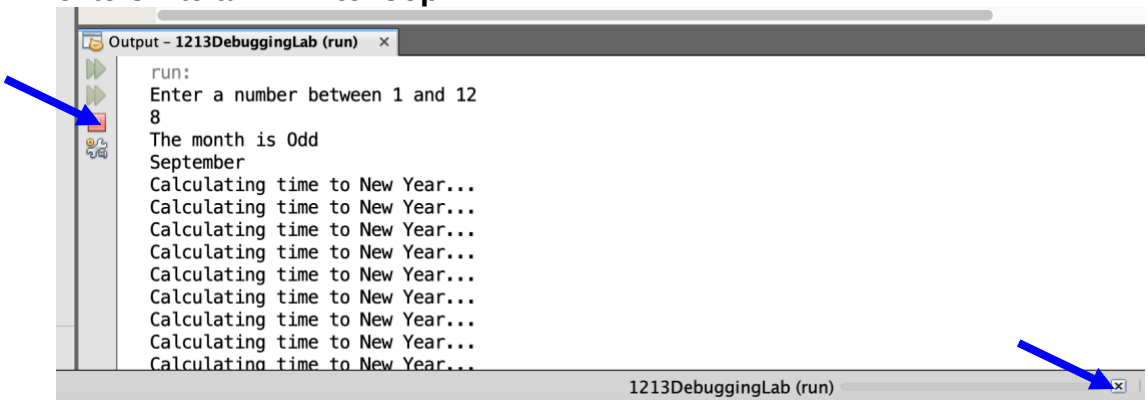
6) The **2nd compilation error** is in the call to the method `getMonthName(...`. This error is because of the fact that this method expects an integer (int) argument not a float. And since this parameter is used in the switch statement which does not work with float it makes sense to change the variable **monthNum** at the start of the main method to the data type **int** from float.

   Note: A **switch** works with the **byte**, **short**, **char**, and **int** primitive data types.

7) To remove other errors, right click in the source editor window and select **Fix Imports**.
   Make sure that **java.util.Scanner** appears under import statements in the pop up box. Click **OK**. Note that now there are no compilation errors in the program.

8) Save the class and run the program by selecting Run File from the Run tab on the menu bar or by right clicking on the file and selecting **Run File**.

When the program prompts for the input, enter **8** in the Output pane at the bottom of the IDE and hit **Enter**. Note that despite the entered month as even (August), the output being displayed is **The month is Odd** and **September**. And then the program enters into an **infinite loop**.



To exit from the infinite loop, you can either click the red stop button on the side margin or click on the close button at the bottom right margin of the IDE.

9) We're now going to track down the cause of the problem using the **debugger**. Do not solve the problem just by looking at the code. The point of this exercise is to learn how to use the debugger to solve more difficult problems than this one.

   Unfortunately, the debugger cannot "go back", so you can't simply go to the point of failure and backtrack. Instead, you first run your program at full speed until it comes close to the point of failure. Then you slow down and execute small steps, watching what happens.

   To do this, we can first set a ***breakpoint*** in the code. A *breakpoint* is simply a flag that tells the IDE "Stop here when we are debugging." It is a debugging mechanism which tells the program that when you reach this point you want to pause the program. To set a *breakpoint* you go to the line of code that you wish to set it on and then click on the grey border on the left. This will put a *breakpoint* on this line.

10) Let's trace the program to see why these errors occur. We know that the first few statements run as expected and we were able to enter a value as input. It won't be too interesting to debug those lines. Instead, let's start at the if statement. We'll set a *breakpoint* at that line, so that the debugger stops as soon as it reaches the line.

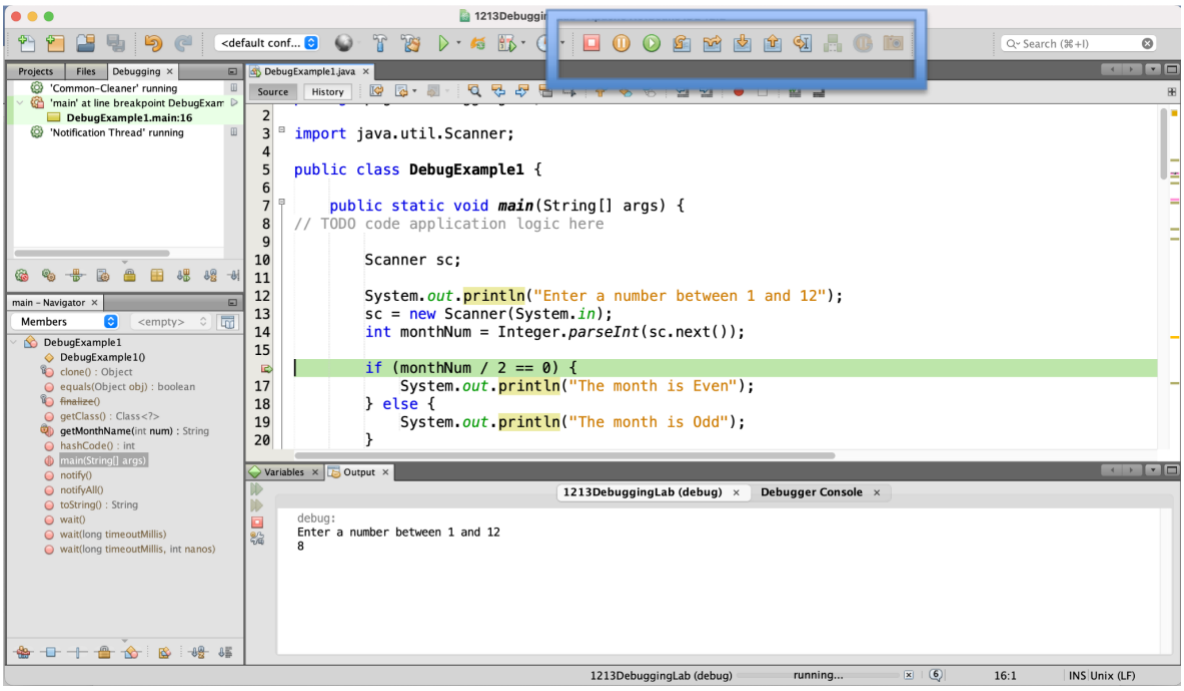
    Move the cursor to the third call to test. Then right click and select **Toggle Line Breakpoint** from the menu. You will see a small red square to the left of the line, indicating the breakpoint.

```
11
12          System.out.println("Enter a number between 1 and 12");
13          sc = new Scanner(System.in);
14          int monthNum = Integer.parseInt(sc.next());
15
            if (monthNum / 2 == 0) {
17              System.out.println("The month is Even");
18          } else {
19              System.out.println("The month is Odd");
20          }
```

11) Now right-click on the **DebugExample** class in the package explorer window. Select the menu option **Debug File**. This can also be done by right clicking the file in the editor window or through the **Debug** main menu item. The debugger now runs your program. When it hits the first breakpoint, you will see that the execution stops and the line with the breakpoint is highlighted with a green color.

    Launch the debugger as described. Notice the new options that now appear in the toolbar.

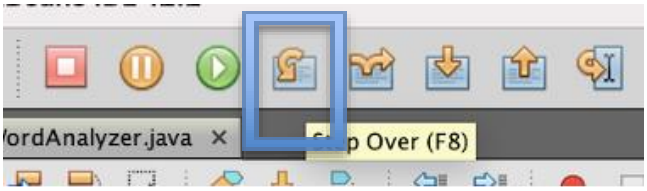
12) Once again, when prompted enter **8** in the Output pane at the bottom of the IDE and hit **Enter**.



13) You should now see that the execution is paused and a green line shows where the debugger has stopped.

14) To walk through the program step by step we'll use the "Step Over" command instead. NetBean gives you three ways of executing it:

- Select **Debug -> Step Over** from the menu
- Hit the F8 key
- Click the "step over" icon in the Debug toolbar



Execute the "Step Over" command and see what happens.

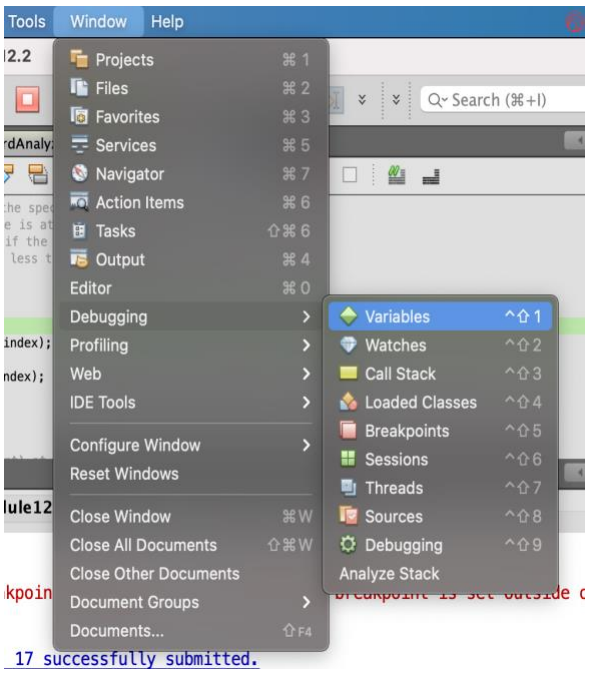
```
15
        if (monthNum / 2 == 0) {
17          System.out.println("The month is Even");
18      } else {
            System.out.println("The month is Odd");
20      }
21
```

Doing this we find that instead of the if statement in the main method, the else statement is being executed. Which indicates that we have a problem in our condition.

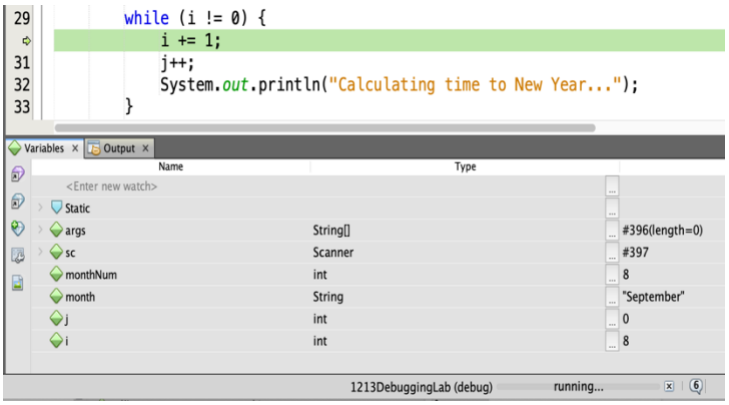
15) Click on Step Over and continue with the clicking of Step Over and you will also find that the `getMonthName(...)` method returns the string value September instead of August and the program never exits the while loop (value of `i` keeps increasing!) and therefore this loop is being executed infinite times.

In fact, click on the **Variables** tab at the bottom of the IDE and you can observe the values of `i` goes from 8, to 9, to 10, to 11, etc., not going towards zero at all! This is a classic infinite loop! This process of variable watching allows you to observe the behavior of your program easily.

16) Look inside the Variables window. You can see the contents of the variables declared so far



17) The triangle next to `args` and `src` indicates that you can expand the variable. Click on the triangle next to `src`. What do you see? Why are there no triangles next to `j` or `i`?



18) Now keep executing the Step Over command. Watch what happens to the `j` and `i` values. You'll see `i` and `j` increase each time the loop is executed.

As we have now narrowed down the source of the problems, stop the debugging by clicking on **Finish** icon on the top of the IDE.

Let's try and fix the first error. We found out that the if condition is not properly written.To fix this error, replace the '/' operator in the if statement with '%' operator.

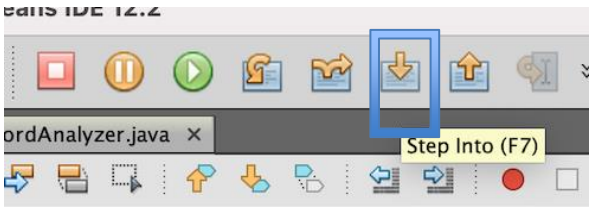
The second error with the `getMonthName()` requires a little more debugging so we will leave that for now and come back to it after we fix the infinite loop.

Look closely and you will find that as we need to calculate the months remaining until the new year, the **`i != 0`** doesn't make any sense to decide when to exit the loop. What we actually want is to terminate once `i` reaches 12. Change that to be **`i != 12`**.

Now let's try and narrow down the problem with the name of the month. We will do this demonstrating other features of the debugging tool. Add a breakpoint at the line with the method call. Start the debugger again and when you reach the first breakpoint (the if statement) rather than performing a Step Over we can tell the debugger to continue running the program up to the next breakpoint. Let's try that, select the **Debug -> Continue** menu option or click the "continue" icon in the Debug tool bar

19) Your program runs again at full speed, until it hits the next breakpoint or until it terminates.

20) Now we want to step through the `getMonthName` method in slow motion. Similar to stepping over NetBean gives you three ways of executing stepping into:



- Select **Debug -> Step Into** from the menu
- Hit the F7 key
- Click the "step into" icon in the Debug toolbar

21) Execute "Step Into". Now you are inside the `getMonthName` method. You should get to the lines

```
String month = "";
```

22) Remember, we want to find out why we get the wrong month name. Should you execute "Step Into" or "Step Over" at this point?

23) Execute "Step Over" a couple of times. You should get to the line

```
month = "August";
```

Executing a “Step Over” a few times we notice that along with case 8 in the switch statement, case 9 is also being executed before we reach the end of the method. This is because of the fact that there is no **break;** statement in case 8 and therefore soon after execution of case 8, the control shifts to case 9 which assigns the value “September” to the variable that gets returned. To fix this error, we just need to include the **`break;`** in **case 8** of the switch statement.

Now that we found out the exact cause of this error we can exit the debugger and fix it.

After fixing the error, run the project again by clicking on the **Run** icon on the top of the menu bar or by right clicking on the project and select **Run**.

Enter any month of your choice and verify that you get the correct output without any more errors!

You now know how to use the debugger. You have learned how to set breakpoints, how to single-step, step into other classes and methods, and how to view variables.

24) When you are satisfied with how the program works show your work to the instructional team to be checked off this lab.

In summary, in this lab, we discussed:

1) How to browse Java API documentation
2) How to read Javadoc of a class
3) Why and how to use import statement
4) How to create an object of a class
5) How to call methods on an object
6) How to add and generate JavaDoc to a class
7) How to use the Scanner class to capture user input
8) How to use the debugger in NetBeans to step through code using breakpoints