# *Programming I (420-B10-HR)*

# *Lab 7 – Drawing UML Diagrams and Introduction to the if Statement*

Date assigned: Tuesday, October 6, 2015

Date due: **Tuesday, October 6, 2015**

**Learning Objectives**

Upon successful completion of this lab exercise, the student will be able to:

1. Draw a UML class diagram using LucidChart;
2. Read a UML Sequence Chart;
3. Use a simple if statement to compare two values.

Students will review how to:

1. Create a driver class containing a main method.
2. Instantiate objects and call methods.

**Statements Used**

1. Simple if ... statement:

**if** condition

true statement;

**To be handed in**

1. Your ***username*\_B10\_L07\_UML\_Diagrams** folder should be zipped and uploaded to **Moodle**.
2. The class diagram for the **TaxWhiz** program should be printed and handed in.

**To Start**

1. Download and unzip the **B10\_L07\_UML\_Diagrams** folder from **Moodle** and rename it to ***username*\_B10\_L07\_UML\_Diagrams**.

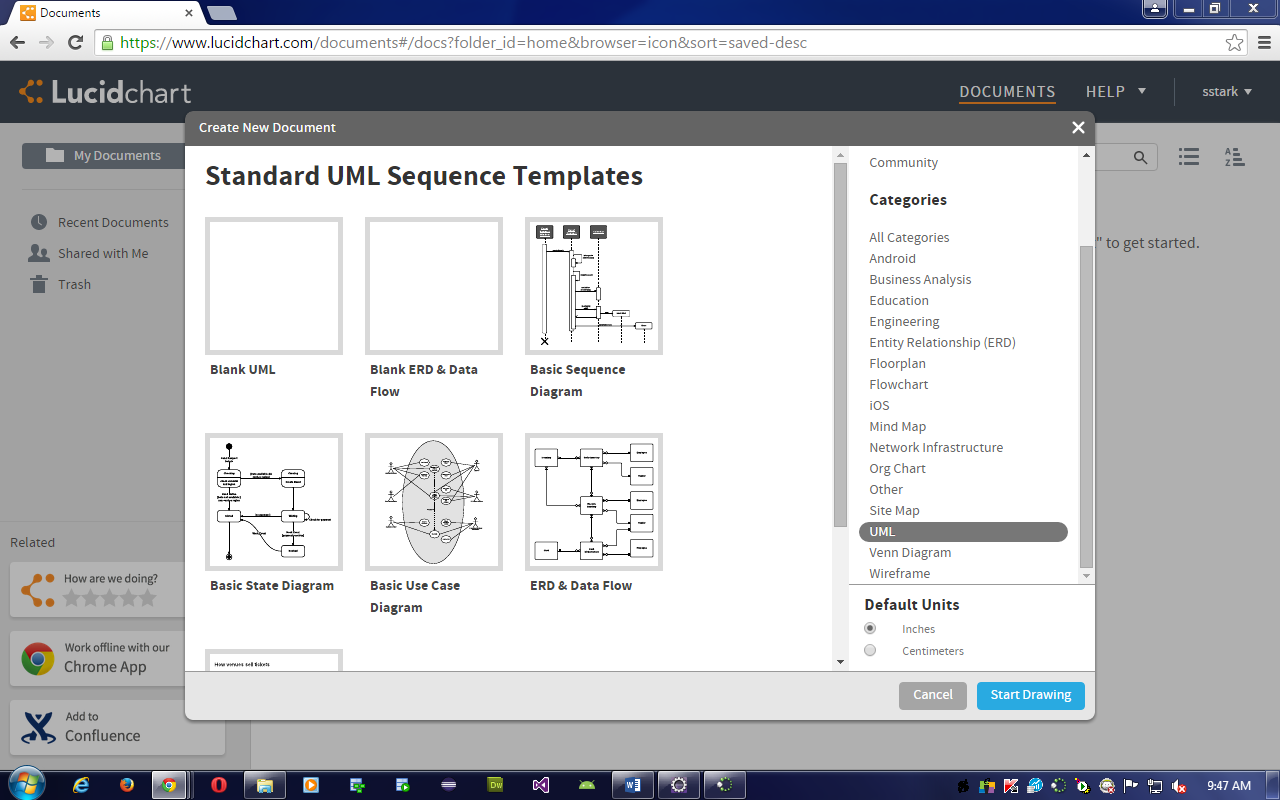
# Drawing a Class Diagram with LucidChart

***Purpose:*** Learn to use **LucidChart** to draw a UML class diagram.

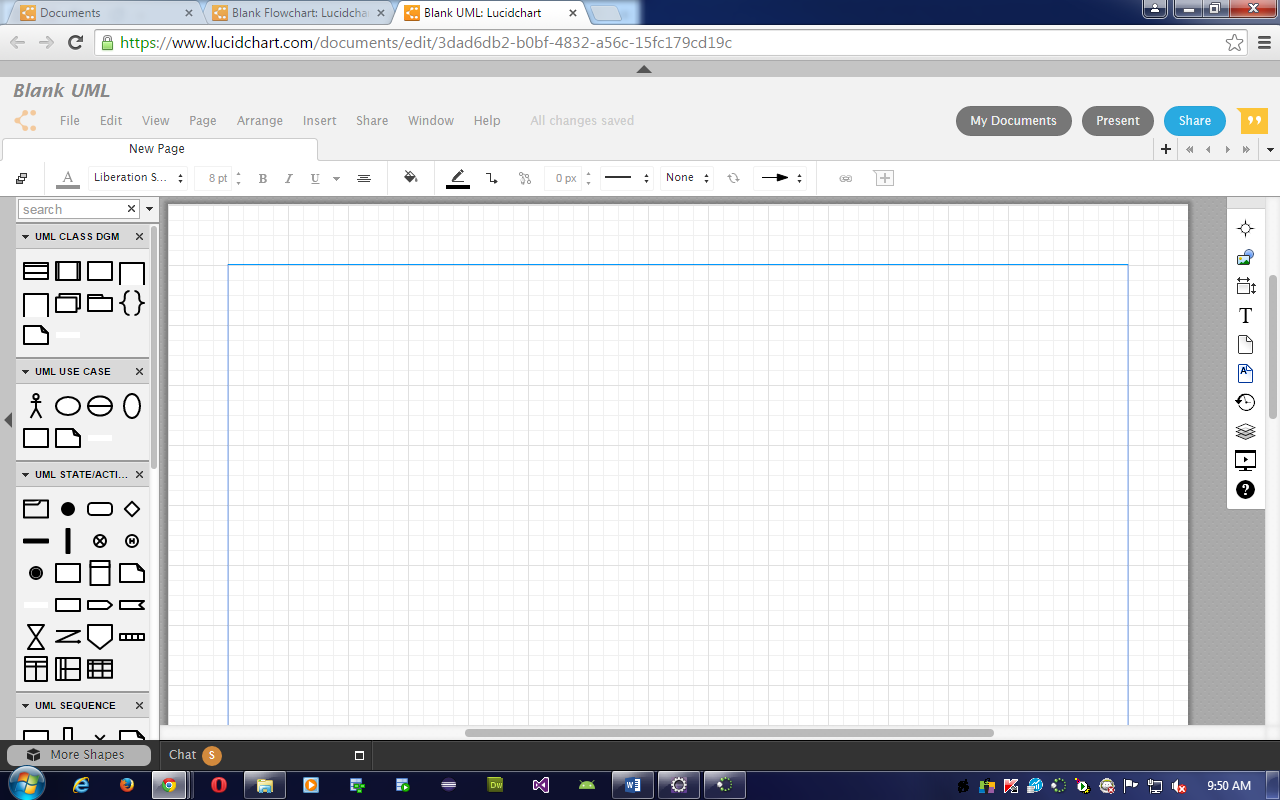
***To Do:***

## Create a free account for LucidChart. Go to <https://www.lucidchart.com/users/registerLevel?t13=A&t4=A&t5=A&tP=1&t3=E&t10=A> and select Start Free Account.

## Create a new document under the category **UML**. Select the **Blank UML Template**. Click the **Start Drawing** button in the bottom right corner.



## A blank drawing window should be open. Close the Flowchart, Containers, and Shapes symbols on the left-hand side of the window so that you are left with the UML class diagram shapes showing:



***Add Classes to a diagram***

## Add the **RectangleUser** and **Rectangle** classes to your diagram:

### Select the **Class** icon from the toolbar on the left of the diagram and drag it to the drawing space.

### The **Class** symbol will be placed on the drawing space. Double click the symbol and type ***RectangleUser*** in the **Class** field.

### Repeat the previous two steps for the **Rectangle** class. Place the Rectangle class under the **RectangleUser** class.

***Add Attributes to a class***

## Double-click on the icon for the ***Rectangle*** class and modify the instance variables field to have two **private** attributes – length:double and width:double.

***Add Methods to a class***

## Double-click on the icon for the ***Rectangle*** class and modify the operations field to have five operations: **setLength()**, **setWidth()**, **getLength()**, **getWidth()** and **calculateArea()**. Note the following:

### The **setLength()** and **setWidth()** methods have no return type and have one parameter which has a type of **double**.

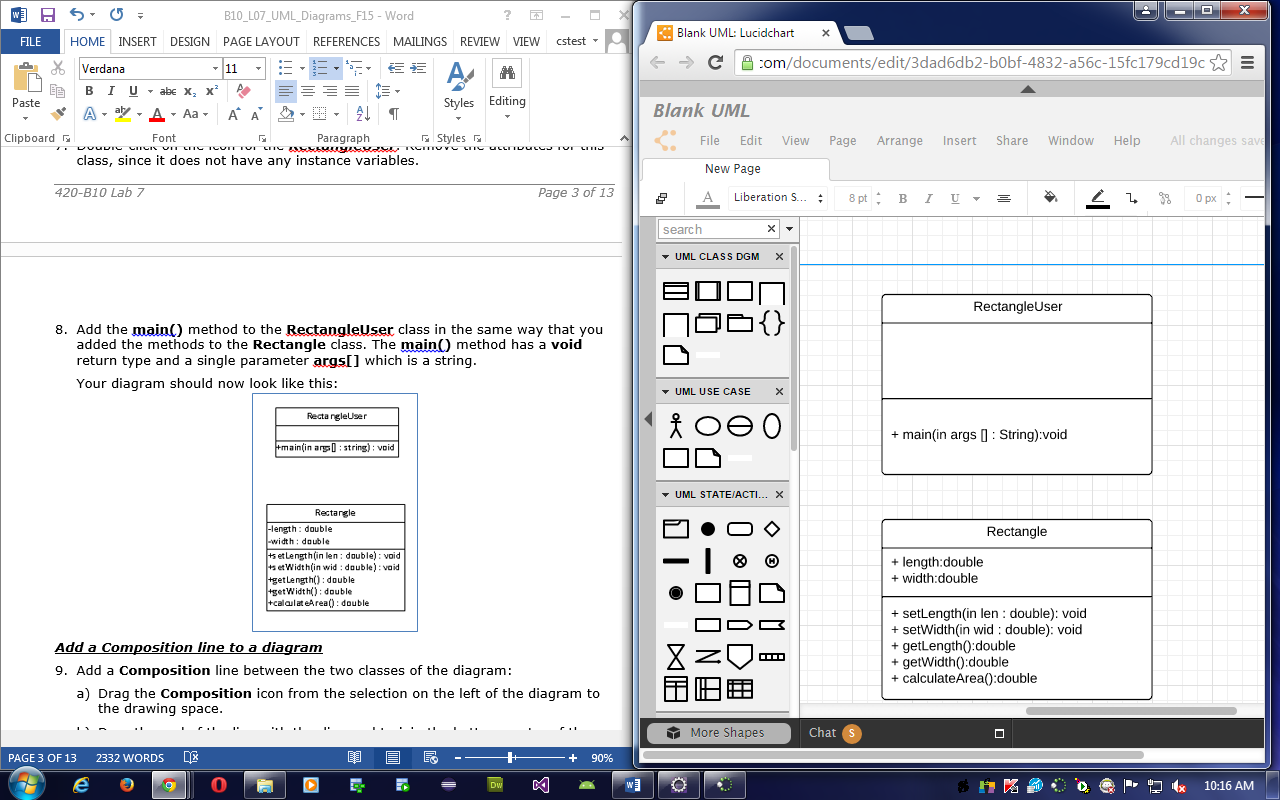
### The **getLength()**, **getWidth()** and **calculateArea** method have a return type of **double** and have no parameters. Add them now.

### Once you have defined all the methods, click **OK** to close the **UML Class Properties** window.

## Double-click on the icon for the ***RectangleUser***. Remove the attributes for this class, since it does not have any instance variables.

## Add the **main()** method to the **RectangleUser** class in the same way that you added the methodsto the **Rectangle** class. The **main()** method has a **void** return type and a single parameter **args[]** which is a string.

Your diagram should now look like this:



***Add a Composition line to a diagram***

## Add a **Composition** line between the two classes of the diagram:

### Draw a line from the bottom of the **RectangleUser** class to the top of the **Rectangle** class by selecting the **RectangleUser** class and dragging down to the **Rectangle** class. You should end up with a diamond at the **RectangleUser** end of the diagram.

### Double-click on the composition line and type **rectangle1** in the text field.

### Repeat the previous two steps and add a second composition line called **rectangle2**.

***Align the shapes in your model***

## Highlight all the shapes on your diagram by dragging the mouse from the upper left-hand corner to the lower right-hand corner. (A light blue rectangle should surround your two classes.)

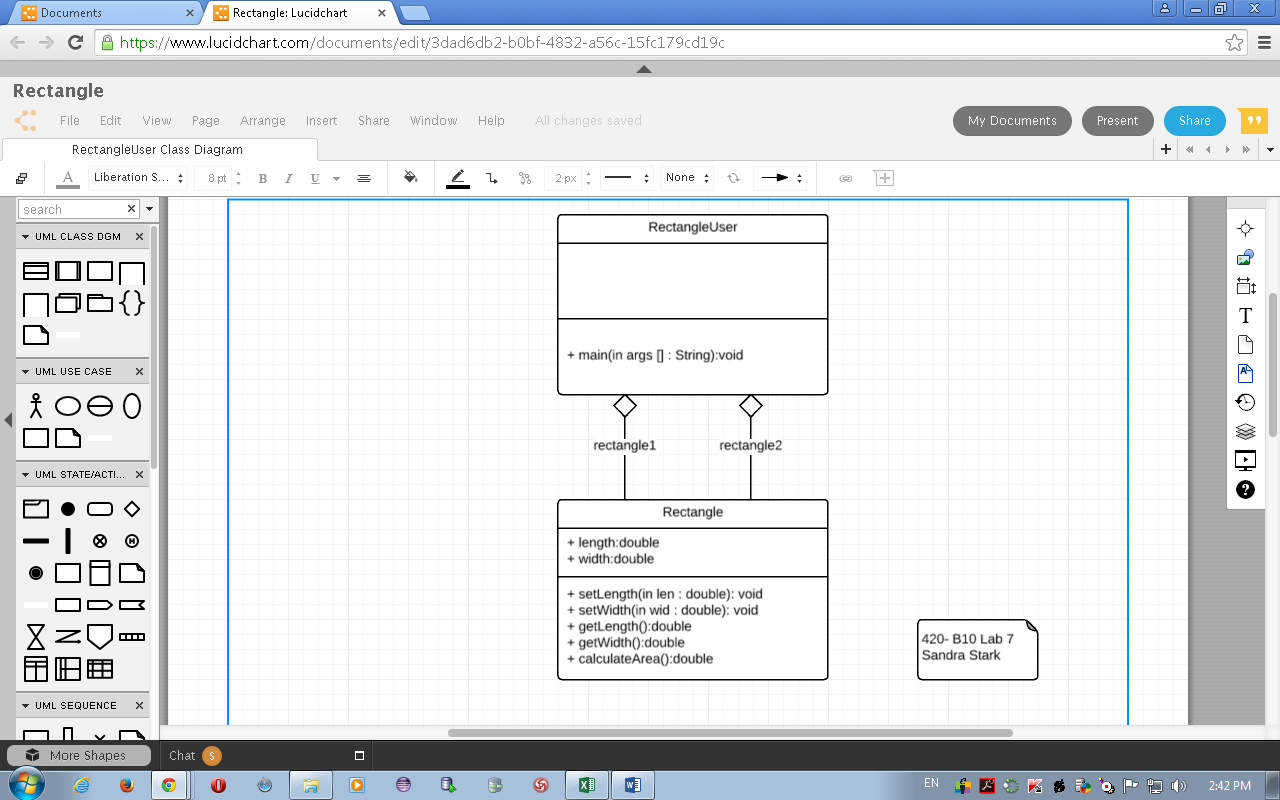
## Select Arrange->Align Objects->Center from the menu at the top.

***Add a title and a note to a diagram***

## Add a title:

### Select the tab at the top of the window that says **New Page**, and rename it to **RectangleUser Class Diagram**.

## Drag a Note to your diagram. It shouldcontain "420-B10 Lab 7" on one line and your name on the next line. Your final diagram should be similar to the one below.



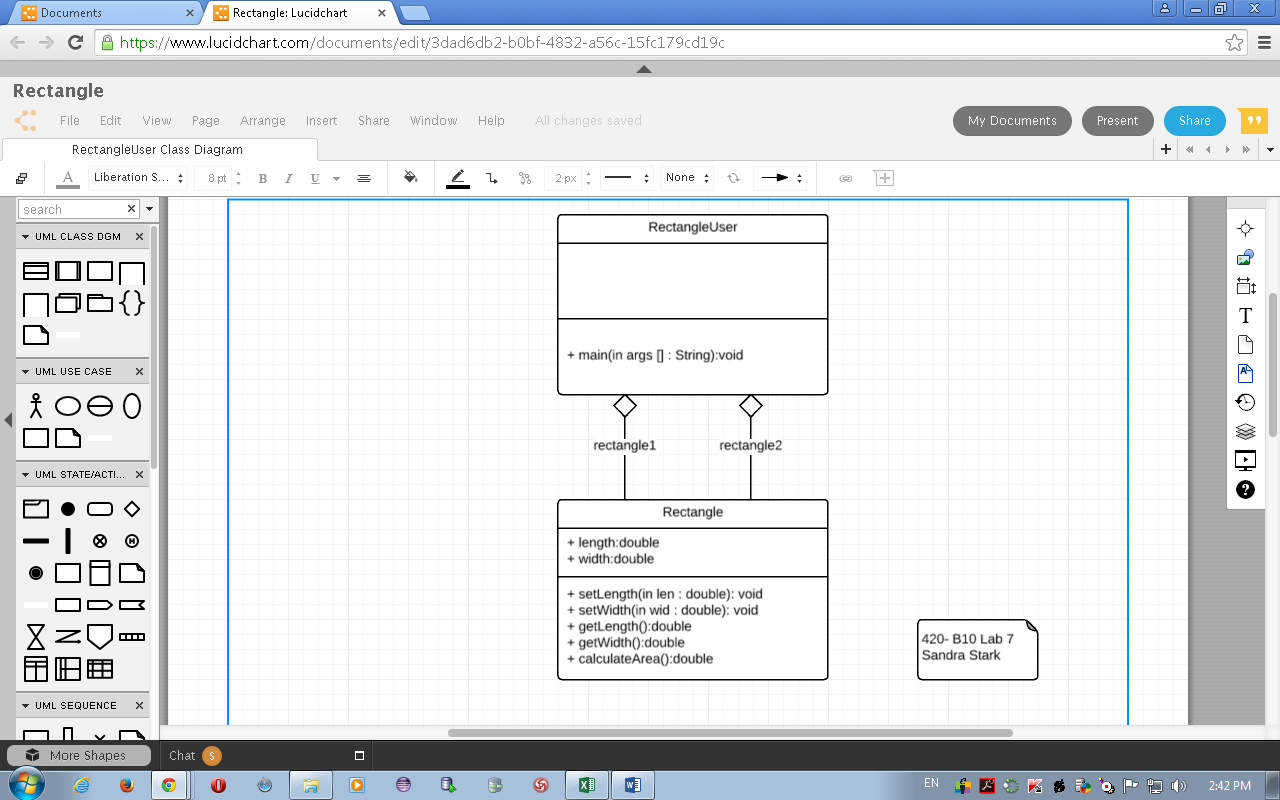
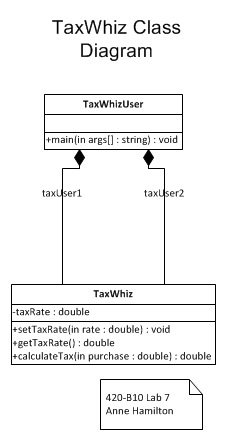
## Save your diagram:

### Select ***File***, ***Save*** and save it as ***username*\_B10\_L07\_RectangleUser.** Select ***File, Download As*** todownload the document as a pdf and save it in the ***username*\_B10\_L07\_UML\_Diagrams** folder you created at the beginning of the lab.

## Close the diagram.

On Your Own:

## Use **LucidChart** to create the class diagram for the **TaxWhiz** program that is shown here. Save it as ***username*\_B10\_L07\_TaxWhiz**.



## Print your class diagram.

# Java Class Coding Review

***Purpose:*** Review how to convert a class diagram to a Java class.

***To Do:***

## Start **Eclipse**. Use your **H:\420-B10\Labs** folder as the workspace.

## Create a **New Java Project** called ***username*\_B10\_L07\_UML\_Diagrams**. At the moment, an error shows in the **taxWhiz** package.

## Create a new class called **TaxWhiz** in the **taxWhiz** package. Add the attributes and methods shown in the class diagram you created in **LucidChart**.

## The sequence diagram for **TaxWhiz** is shown below for information.



## Implement the **TaxWhiz** methods as follows:

### The **setTaxRate()** method should set **taxRate** to **rate**.

### The **getTaxRate()** method returns **taxRate**.

### The **calculateTax()** method multiplies the **purchase** parameter by the **taxRate** instance variable divided by 100 and returns the result.

## Open and run the **TaxWhizUser** class that is included in the project. The output should be:

The tax on a purchase of $10 at a tax rate of 15.0% is $1.5

The tax on a purchase of $15 at a tax rate of 14.0% is $2.1

# Class Object Instantiation Review

***Purpose:*** Review how to create a driver class containing a main method and instantiate objects and call methods.

***To Do:***

## Open the **OneRowNim.java** class.

## Create a new class in the **oneRowNim** package called **TestOneRowNim**. It should contain a **main()** method. Put a comment block at the top of the class with your name. The main method should instantiate a **OneRowNim** object called **game**. Itshould call the **startGame()** method with 7 sticks. It should then play the game as documented in the sequence chart on the next page. After each turn display the number of sticks left and the next player. A sample run of the game is:

At the start of the game, there are 7 sticks.

It is player 1's turn.

There are now 4 sticks.

It is player 2's turn.

There are now 3 sticks.

It is player 1's turn.

There are now 2 sticks.

It is player 2's turn.

There are now 0 sticks.

The game is over! Player 1 won!

Use the **getPlayer()** method to get the player number for each turn. Use the appropriate **take()** method to remove sticks. Use the **getNumSticks()** method to get the number of sticks remaining. For example, to display the second and third lines of the display, you would code:

System.out.println("It is player " + game.getPlayer() + "'s turn.");

game.takeThree();

System.out.println("There are now " + game.getNumSticks() + " sticks.");



# Using the Simple if Statement

***Purpose*:** Use a simple **if** to compare two values.

***To Do:***

## Open **Grader.java** in the **Grader** package. Look at the **getStatus()** method.

## Open **GraderUser. java**. It creates a **Grader** object, inputs a mark from the user and calls the **getStatus()** method to determine the message output.

## Run **GraderUser** 6 times using the input shown in the table below. Complete the table with the output produced for each run.

| ***Mark*** | ***Output*** |
| --- | --- |
| **59.0** | **poor** |
| **60.0** | **error** |
| **61.0** | **passed** |
| **75.5** | **passed** |
| **100.0** | **perfect** |
| **101.0** | **Passed** |
| **-5** | **poor** |

Look at the **getStatus()** method in **Grader.java** and complete the following table:

|  |  |
| --- | --- |
| **Symbol** | **Meaning** |
| < | Is less than |
| <= | Is less than or equal to |
| == | Is equal to |
| != | Is not equal to |
| >= | Is greater than or equal to |
| > | Is greater than |

## Modify the **Grader** class so that it returns **Passed** if a mark of 60.0 is entered.

## Modify the **Grader** class to return **Error** if a mark greater than 100 is entered. (***Note***: you will have to put the statement before the statement that compares to 60.0. Otherwise it will still return **Passed**.)

## Modify the **Grader** class to return **Error** if a mark less than 0 is entered. (***Note***: you will have to put the statement at the beginning of the method. Otherwise it will still return **Poor**.)

## Change the author of the **Grader** class to yourself.

## Test your changes using the test data above.

***On your own:***

## Open **OneRowNim**. Add a method called **isGameOver()**. It should return a **boolean** value. Add statements to the method so that if **numSticks** is less than or equal 0, it returns true. If **numSticks** is greater than 0, it returns false. You will have to add another return false at the end for the method to work.

## Add a void method called **takeSticks()** to the **OneRowNim** class. It should have one integer parameter called **nSticks**. If **numSticks** is less than **nSticks**, the method should return. Otherwise, it should subtract **nSticks** from **numSticks** and change the player. (See the **takeOne()**, **takeTwo()** or **takeThree()** method to see how to change the player.)

## Open **TestOneRowNim**. Add a **Scanner** object to the class. Instead of executing the **takeOne()**, **takeTwo()** and **takeThree()** methods:

### ask the user how many sticks to take;

### read the number of sticks into an integer variable called **numSticks**;

### call the **takeSticks()** method using **numSticks** as the argument.

## Add a if statement before the last **println()**. If the value returned by the **gameOver()** method is true, execute the **println()** statement. If the value returned is false, print a message stating that the game is not over yet and nobody won. Two sample runs follow:

At the start of the game there are 7 sticks.

It is player 1's turn.

How many do you want to take? 3

There are now 4 sticks.

It is player 2's turn.

How many do you want to take? 2

There are now 2 sticks.

It is player 1's turn.

How many do you want to take? 1

There are now 1 sticks.

It is player 2's turn.

How many do you want to take? 1

There are now 0 sticks.

The game is over! Player 1 won!

At the start of the game there are 7 sticks.

It is player 1's turn.

How many do you want to take? 3

There are now 4 sticks.

It is player 2's turn.

How many do you want to take? 5

There are now 4 sticks.

It is player 2's turn.

How many do you want to take? 1

There are now 3 sticks.

It is player 1's turn.

How many do you want to take? 1

There are now 2 sticks.

The game is not over. Nobody won.

# To Finish:

## Zip your ***username*\_B10\_L07\_UML\_Diagrams** folder and upload it to **Moodle**.

## Hand in your printed **TaxWhiz** class diagram.