1A

If you have not done so, create the directory structure specified in Lab1.

Start a new Bluej project called hw1a_arithmetic_draft in the cs46a/homework/hw01/draft folder. You can actually name the project anything, but this is a good naming convention that will help you keep things straight.

In the Bluej project, create a class called **`ArithmeticPrinter`**. (You must use this exact name to pass Codecheck). Copy and paste the start code from the 1A draft link below.

Finish the application so that it prints the sum of the first four even numbers. (2 + 4 + 6 + 8 ) and then on the next line, prints the product of the same four numbers. Product means multiply. You use the symbol '*' to multiply in Java

To have an application, you need a main method. It is provided for you in the starter code

Do not do the arithmetic on a calculator or in your head and simply display the answer. Actually have the computer do the work by doing the calculations in a System.out.println statement. (Look at Udacity lesson 1 "Text and Numbers" again if you have trouble). As an example, to print the sum of the numbers 1 to 5, you would use a statement like this

```
System.out.println(1+ 2 + 3 + 4 + 5);
```

For the draft, only print the sum of the first four even numbers.

To compile your code, click the Compile button in the top left. To run your program. right click on the ArithmeticPrinter rectangle in Bluej and choose **`void main(String[] args)`**

When you think your code is correct, copy the whole class from Bluej and paste it into the textarea in the Codecheck draft link. Click submit. This will generate a HTML report about the correctness of your solution. Click the Download Report button and save it until you have all three

When you are ready to work on the final, copy the Bluej project (hw1a_arithmetic_draft) into the cs46a/homework/hw01/final folder. Change the word *draft* to *final*. This will help you tell the draft and the final when both are open in Bluej. Open the copied project (hw1a_arithmetic_final) in Bluej and complete the assignment.

Note: It is important to remember that Codecheck just helps you get your code to work. It does not submit it for grading. **You must download and submit the report for grading in the Canvas Assignment area.** See Submission directions at the bottom of the page.

It is important that you name your class **exactly** as specified otherwise Codecheck will not be able to process your submission and you will get no credit.

If you use an IDE like Eclipse or Netbeans, **do not include a package statement**. Codecheck will not be able to process your submission and you will get no credit.

To run your draft in Codecheck, click on the draft link below. Copy and paste the code from Bluej into the textarea and click submit. If your code passes the test, Congratulations. If not, go back to Bluej, make any necessary changes, and repeat the process.

See the Submission directions at the bottom of the page.

Follow a similar process for the final version.

1A draft:

1A final:

1B

Start a new Bluej project called hw1b_year_draft in the cs46a/home works/hw01/draft folder.

In the project, create a new class called **`YearPrinter`** (you must use this exact name to pass Codecheck). In this application, you will display the year first in digits then in Roman numerals.

Go to the Codecheck draft link at the bottom of the page. Copy the **`YearPrinter`** starter code you are given there. Create a YearPrinter class in your Bluej project. Paste the code form Codecheck into the Blue class.

The completed final version will look **exactly** as shown.


For the draft, just print the first two lines like the image below. You will add the code for the draft inside the main method.


The "|" is called a pipe and is located above the Return (Enter) key

When you think your draft code is correct, copy the whole class from Bluej and paste it into the textarea in the Codecheck draft link. Click submit. This will generate a HTML report about the correctness of your solution. Click the Download Report button and save it until you have all three

When you are ready to work on the final, copy the Bluej project (hw1b_year_draft) into the cs46a/homework/hw01/final folder. Change the word *draft* to *final*. This will help you tell the draft and the final when both are open in Bluej. Open the final project (hw1b_doll_final) in Bluej and complete the assignment.

The third line in the printout needs more explanation. The third line does **not** consist of the English characters M, X, V, and I. These are actually special characters representing the Roman numerals. If you use the letters on the standard keyboard, your output will not match the Codecheck output.

So how do you get the Roman numerals? Well, every printable character in English and most other languages, along with many symbols is represented by a hexadecimal number (it is called Unicode). When you type a character on your keyboard, the computer software takes care of

translating the character to its Unicode representation. But when we want to display a character that is not on the keyboard like the Roman numerals, we have to supply the Unicode ourselves. We could also specify the English characters directly with Unicode, but it is much easier to let the computer take care of the translation. Here is a table of a few characters and their Unicode values

| Character | Unicode |
|---|---|
| Y | \u0059 |
| E | \u0045 |
| A | \u0041 |
| R | \u0052 |
| Ⅿ | \u216F |
| Ⅹ | \u2169 |
| Ⅶ | \u2166 |

In Java, the "\u" is an escape sequence which tells the compiler that the following characters have a special meaning, the Unicode for some character.
The following two statements will display the same results: the word "YEAR" in all uppercase. The first uses the Unicode values for the letters.

```
System.out.println("\u0059\u0045\u0041\u0052");
System.out.println("YEAR");
```

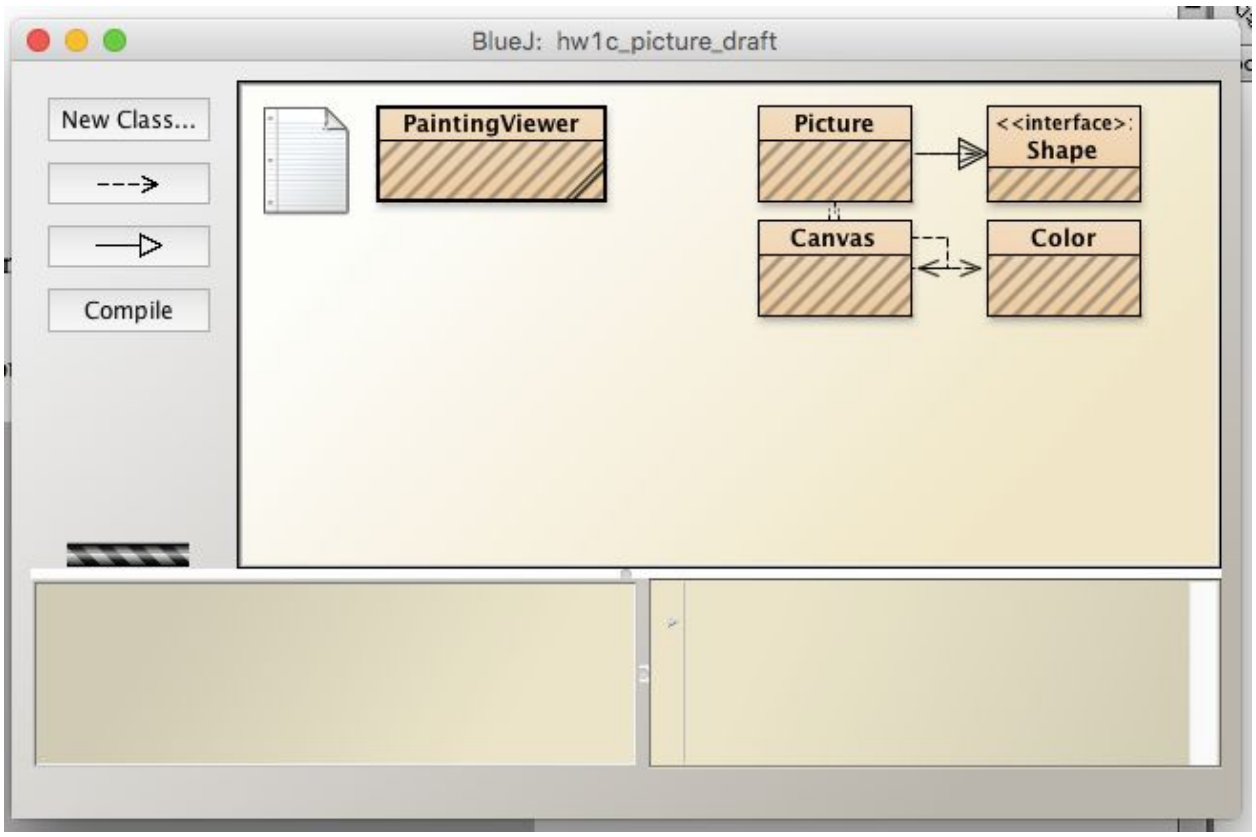
Note: For this assignment, you must use Unicode for the characters Roman numerals and not just paste the symbol in the println statement.
Note: It is important to remember that Codecheck just helps you get your code to work. It does not submit it for grading. **You must download and submit the report for grading in the Canvas Assignment area.** See Submission directions at the bottom of the page.
It is important that you name your class **exactly** as specified otherwise Codecheck will not be able to process your submission and you will get no credit.
If you use an IDE like Eclipse or Netbeans, **do not include a package statement**. Codecheck will not be able to process your submission and you will get no credit.

To test your draft, click on the draft link below. Copy and paste the code from Bluej into the textarea and click submit. If your code passes the test, Congratulations. If not, go back to Bluej, make any necessary changes, and repeat the process.
See the Submission directions at the bottom of the page.
Follow a similar process for the final version.
1B draft:
1B final:

## 1C

Here is a painting by a famous artist (Starry Night by van Gogh). I want to enlarge it so I can see more detail

Download hw1c_painting_draft.zip which contains the files you will need. Unzip it into your cs46a/homework/hw01/draft folder. (See Lab 1 Part B for directions on unzipping a file correctly in Windows.)
Open the folder. Double click package.bluej to open the project. Your workbench will look similar to this.



If it does not look like this image, STOP. We need to get you straightened out. You should not see a red folder that says "go up" That means you have not unzipped correctly
Complete the PaintingViewer class following these instructions

- In the main method, create a Picture object with the image painting.png that is in the downloaded folder
- Move the picture 100 pixels in the x direction and 50 pixels in the y direction. Use the translate method.
- The image is 106 x 86. Make the image twice as big by increasing the Picture's width and height using the grow method
- Draw the picture.

If you have forgotten how the translate and grow methods work, re-watch Udacity Lesson 2 "What does this method do?" and "Variables"
For the draft, just create the picture and draw it. Don't translate or grow it until the final version.
When you are ready to work on the final, copy the Bluej project (hw1c_painting_draft) into the cs46a/homework/hw01/final folder. Change the word draft to final. Open the project and complete the assignment.
1C draft:
1C final:
Submission
When you are finished with your code, submit it in Codecheck one final time. Notice at the bottom left there is a "Download" button. Click that and a .signed.zip file will be downloaded. That is the file you need to upload into Canvas.
When you are working on the draft, add the word "draft" to the end of the name, before .signed.zip after downloading. For example ArithmeticPrinter**_draft**.signed.zip. (Leave .signed.zip alone and do not add any extra dots)
When you are working on the final, add the word "final" to the end of the name, before .signed.zip. For example ArithmeticPrinter**_final**.signed.zip. This will help avoid submitting the wrong version.
Be sure to upload the .signed.zip file produced by Codecheck not the .java file you wrote. Do not open and alter the downloaded file in anyway. The files are digitally signed and the graders will check that they have not been altered.
For both the draft and the final, you will upload 3 signed.zip files. Upload all three programs at one time. Double check in Canvas that the files were uploaded
Warning: do not submit the final version as the draft. In order to be graded correctly, you must submit a program that does exactly what is specified for the draft - no more, no less.

2A
This semester, the first day of finals is December 13, 2017.
Write a program `FinalPrinter` which will calculate how many days from today (whatever day the program is run) until finals start.

Chapter 2 Worked Example 1 from your text gives you a `Day` class and discusses how to use it. A copy of the `Day` class itself is at the Codecheck URL below. You need to copy the class into a Bluej project along with the starter code.
Lesson 2 of the Udacity videos also discusses the `Day` object. Section *19: How Many Days* gives an example of working with the Day object in code.
Do the following:

- Create a `Day` object representing the current day. Use the constructor that does not take any parameters, like this. `new Day()`
- Assign that Day object to a variable called `today`. There is already a statement in the starter file to print the variable `today`. Do not change it. Note that no matter when the application is run, the value in `today` should be the current date.
- Create a `Day` object representing the first day of finals and assign it to a variable.
- Calculate how many days the first day of finals is from today (it will be a positive number). Print it.
- Now calculate the date 100 days from today.
- Print the year. (Use a method to get the year)
- Print the month. (Use a method to get the month)
- Print the day of the month. (Use a method to get the day)

For the draft, just add code to the starter to define the variable `today` and assign today's date to it. When you run the program, the date will be printed by the starter code.
2A draft:
2A final:
2B
In this problem you will use various String methods. Complete the class `StringsRUs`
You are given a word in the starter code. Do the following:

- print the two times the length of the given word. (use a String method. Don't just enter the number)
- print the uppercase version of the word (do not change the original word)
- replace "t" with "7", "a" with "4", and "e" with "3" in the **original** word. You will need to use the replace method multiple times. You will need to use the replace method multiple times.(Do not change the original word)
- print the string with replacements
- print the original word.

Remember that replace is an accessor method. It returns the changed version of the string. If you want to do something with the returned string - like use replace on it again - you need to save the return value in a variable.
You must use the String replace method on strings of one character. No credit for simply printing the word with the replacements or replacing the entire word.
Note: When you submit your code to Codecheck, it will first run with the given word, but then it will run again with a different word. You do not need to worry about that. It will happen automatically. If your code is correct,

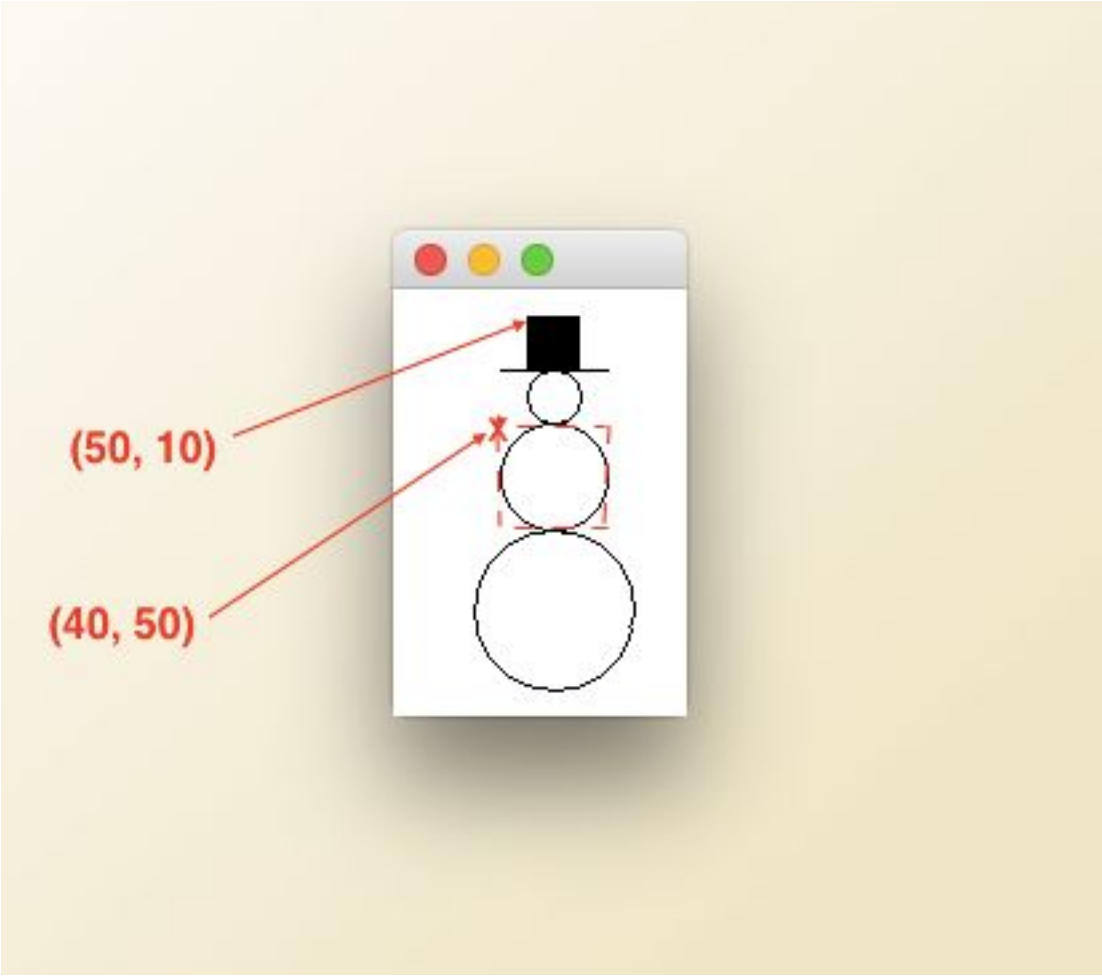
For the draft, just print the 2 times the length of the string. Note: There is a String method length() which returns the length of the String. Use it.

2B draft:

2B final:

2C

Finish the application **`SnowmanDrawing`** to draw a snowman consisting of 3 circles centered on top of one another with a hat on top. The centers of the circles fall on a vertical line. The drawing will looks like this:




The dotted rectangle is the bounding rectangle of the middle circle. It is not part of your drawing.
Specifications:

- The upper left hand corner of the hat is at 50, 10.
- The hatbox is 20 x 20. Fill it with black. (Do not draw it)
- The line for the hat brim at the bottom of the hat extends 10 pixels further than the hatbox on each side.
- The diameter of the small circle is 20. The upper left hand corner of its bounding rectangle is at (50, 30)
- The diameter of the middle circle is 40 The upper left hand corner of its bounding rectangle is a (40, 50)

- The diameter of the bottom circle is 60. You figure out this one.

For the draft, fill the hatbox and draw the brim.

For this problem, you will use the **Udacity graphics package.** After you create your project, you will need to import the graphics package into Bluej. Download the graphics package from Udacity and put it in a convenient location. (You will need this again in future assignments and exams). Select the workbench for your project. Click Import... Navigate to the graphics package and select it. You may get a warning. Just click Continue

2C draft:

2C final:

Note: Starting in this homework, we deduct points:

- if your code is not formatted properly. (In Bluej when the code editor has the focus, click Edit / Auto Layout to format your code)
- if you do not use camel casing for variable names
- if you do not use meaningful variable names
- if instance variables are not private
- for unnecessary instance variables

3A

Complete the `GroceryBill` class which models the checkout process at a grocery store. You are given starter code and a `GroceryBillTester class` which creates a `GroceryBill` and calls its methods. Copy both into Bluej.

In this class, we want to be able to add the cost of an item to the bill and remove a specific amount from the bill. We also want to able able to get the subtotal.

What does the class need to remember? That is the instance variable. What data type should it be (int, double, String)?

`GroceryBill` has a no-argument constructor which initializes the instance variable (subtotal) to 0. Remember that a constructor has the same name as the class.

It has the following methods:

- `public void add(double amount)` adds this cost to the subtotal for this GroceryBill
- `public void remove(double amount)` subtracts this cost from the subtotal for this GroceryBill
- `public double getSubtotal()` gets the subtotal for this GroceryBill

Provide Javadoc for the class, the constructor, methods, parameters and return values. Codecheck will run StyleCheck to tell you if your Javadoc is correctly formatted.

For the draft, provide the instance variable and implement the constructor to initialize the instance variable. Provide javadoc at the top of the class and for the constructor

3A draft:
3A final:
3B
HAL 9000 is a fictional character in Arthur C. Clarke's 2001: A Space Odyssey. HAL (Heuristically programmed ALgorithmic computer) is a AI (artificial intelligence) computer that controls the systems of the spacecraft and interacts with the ship's crew.
In this assignment you will finish the `Hal9000` class. You are also provided with a tester that uses the class and calls its methods. Copy both the starter code and the tester.
The `Hal9000` needs to remember the name of the crew member it is interacting with. This is the instance variable. The constructor takes a String parameter of the crew member's name and assigns it to the instance variable.
It has methods:

- `public String getName()` which gets the name of the crew member
- `public void setName(String newName)` which sets a new name for the crew member
- `public String greetCrewMember()` which returns a string consisting of "Welcome, *name*" where *name* is the name supplied in the constructor.
- `public String doCommand(String whatToDo)` which returns a string consisting of "I am sorry, *name*. I can't " + *whatToDo* where *name* is the name supplied in the constructor and *whatToDo* is the parameter for this method.

Supply Javadoc for the class, the constructor and all the methods
You can use the String concat method to join strings together, but I find the easiest thing to do is use the + operator. You will learn about that in lesson 4, but I can tell you now. Look at this example of combining two Strings to make a third. I want to combine the phrase "My cat is " with the variable containing my cat's name and then a period.

```
String cat = "Corky"
String message = "My cat is " + cat +".";
```

This is called concatenation.
For the draft, provide the instance variable and write the constructor to initialize (to a assign the value to) the instance variable. Provide getName() method and setName() methods. Provide stubs for all the other methods and supply the Javadoc for the class (at the top), the constructor and all four methods.
What is a stub? A stub has a method header complete with braces. If the method is void, that is all you need. If not, then you need to return a temporary dummy value: null for Strings and other objects, 0 for doubles and integers.
Here is the stub for whatDoYouNeed()

```
public String doCommand()
{
  return null;
}
```

In the Javadoc, describe what the final version of the method will do, not what the stub does now.

3B draft:
3B final:

3C

In this exercise, we are going to revisit the snowman from homework2c. In that assignment you wrote a class **SnowmanDrawing** with a main method and did all the drawing in it. But there is a problem with that design: If you wanted to draw a second snowman (or a third or a fourth), you would have to recalculate all the coordinates and repeat all the lines of code. Not good. The solution to this problem is to create a **Snowman** class that models a snowman at a given (x,y) coordinate and can draw itself. Then that class can be used in an application to draw an many **Snowman** objects as you want. I provide you with a class **SnowmanViewer** which is an application that uses the **Snowman** class. There is not starter class for **Snowman**. You will write all the code. But be sure to copy **SnowmanViewer** from Codecheck.

What does the class need to remember? Its starting coordinates. So those are the instance variables

**Snowman** has a constructor that takes the x and y coordinates of the upper left hand corner of the rectangular part of the hat as parameters and stores them in the instance variables. Do not draw in the constructor.

**Snowman** has a draw method that can draw the **Snowman** at its x,y coordinates. It takes no parameters

draw method specifications:

- The upper left hand corner of the hat is at x, y.
- The hatbox is 20 x 20 and filled with black
- The line for the hat brim at the bottom of the hat extends 10 pixels further than the hatbox on each side.
- The diameter of the small circle is 20. The upper left hand corner of its bounding rectangle is at (x, y + 20)
- The diameter of the middle circle is 40
- The diameter of the bottom circle is 60.

Provide Javadoc for the class itself, the constructor and the draw method.

Remember you need to import the graphics classes into the Bluej project.

If you had trouble with hw2c, then you can use the solution in the Modules on Canvas.

**Snowman's** draw method will have less that 20 lines of code. If you have a lot more than that. You are doing something wrong - even if it passes Codecheck.

**SnowmanViewer** draws several **Snowmans** at different locations.

For the draft, provide the constructor, instance variables and a draw method that draws the hat (hatbox and brim). Provide the Javadoc

3C draft:
3C final:

4A

At Windows, Inc., we make stain glass windows of one particular shape. The windows are rectangular with a semi-circle on top. There is an image below. We put a glaze on the surface of the window. The glaze costs $2.54 a fluid ounce, and a fluid ounce will cover 10 square **inches**. Complete the `Window` class. It has a constructor that takes the width and height (in feet) of the rectangular part of the window(in that order) as parameters. Instance variables are already defined

Implement the following methods:

- public double `getWidth()` Gets the width of this `Window` (provided)
- public double `getHeight()` Gets the height of this `Window` (provided)
- public void `setDimensions(double theWidth, double theHeight)` sets a new width and height for the window
- public double `getArea`() Gets the area of the window in square inches (updated sept 20)
- public double `getCostOfGlaze()` Gets the cost of the glaze needed for this window. So not calculate the area again in this method. Call another method in the class.

Note: the width of the rectangle is also the diameter of the semi-circle.

Note: you will have to do some units conversion.

Use Math.PI

Do not use magic numbers in your code. Use constants provided for number of square inches in a square foot, for the number of square inches an ounce will cover, and for cost of the glaze per ounce. The constant for $in^2/ft^2$ is below. You provide the one for cost of the glaze per ounce

```
public static final double COST_PER_OUNCE = 2.54;
public static final int SQUARE_INCHES_PER_SQUARE_FOOT = 144;
public final static int SQUARE_INCHES_PER_OUNCE_OF_GLAZE = 10;
```

For the draft, write the constructor and provide setDimensions method. Code the other methods as stubs. Provide Javadoc

If you have forgotten what a stub is, look here.

**Description**
**This stained glass window depicts St. Michael (archangel) casting out the great red dragon. Stained glass is typically found on church windows for decorative purposes.**

**Clipart courtesy FCIT**
**http://etc.usf.edu/clipart/**

4A draft:
4A final: (updated Sept 1)

4B

Write an application `InputApplication`. (An application has a main method.)

Create a `Scanner` object and do the following..

- Ask the user to enter a word using this exact prompt: `System.out.print("Enter a word: ");`
- Use Scanner's `next` method to get the word
- Print the last letter of the word
- Ask the user to enter a double using this exact prompt: `System.out.print("Enter a double: ");`
- Use `Scanner's nextDouble` method to get the double

- Now ask the user to enter a integer using this prompt: `System.out.print("Enter an integer: ");`
- Use `Scanner's nextInt` method to get the integer
- Calculate the product of the double and the integer (*Product* means multiply)
- Print the product on a new line.
- Print the integer portion of the product on a new line.(Think cast)

Only create one Scanner.
For the draft, create the Scanner and do the first three steps.
4B draft: (URL changed Sept14)
4B final:
4C
Sometimes people have their initials printed on things like towels, shirts, and glassware. These initials are called a monogram. You are to write a class `Monogram` which represents a person's initials. A `Monogram` class has String instance variables of a first name, middle initial (one letter. No period), and last name.
There is no starter file. You will write the whole class.
The `Monogram` class has two constructors (This is called overloading).

- `public Monogram(String theFirst, String theMiddleInitial, String theLast)` initializes the instance variables with the values of the parameters. The user is supposed to supply a single character for theMiddleInitial. But users often do not do what they should. A good program protects itself from user error. So use the substring method to get just the first character of theMiddleInitial and store that in the instance variable. Now your program will work, even it the user makes a mistake.
- `public Monogram(String theFirst, String theLast)` initializes the first and last names to the given parameters and initializes the middle initial to the empty String (two double quotes with no space like this "")

Letters class has these methods

- `public String getFirstName()`
- `public String getMiddleInitial()`
- `public String getLastName()`
- `public String getName()` gets the full name with spaces separating the names. Note: When the there is no middle initial, there will be 2 spaces between first and last name. That is okay.
- `public String getMonogram()` gets the letters comprising the initials (first initial, middle initial, last initial) with no spaces. Do not use an if statement. You do not need it. The middle initial is either a string of one character of the empty string.

Provide Javadoc (the class, the constructors, and the methods)
To get the first letter of a string, you can use substring

```
String firstLetter = word.substring(0, 1);
```

For the draft, provide the instance variables and the first constructor. Implement these methods: getFirstName, getMiddleInitial, and getLastName methods. Implement the other methods as stubs. Provide Javadoc. For the stubs, indicate what will be returned in the final not null.
4C draft:
4C final:

5A

Write an application called **`IfPractice`**. It will have a main method. There is no starter coder. In the main method: do the following:

- In then draft, ask the user for his/her first and last name using this prompt: **`System.out.print("Enter your first and last name: ");`**
- In the final, I made a mistake and so you need to use this prompt System.out.print("Enter your full name: "); Updated Sept 25
-
- Print Hello, *name*! where *name* is the name that was entered.
- Get just the first name.
- Print: I think I will call you *nickname* where *nickname* is the nickname in the table below.

| • **First Name** | • **Nickname** |
|---|---|
| • James or james | • Jim |
| • Robert or robert | • Bobby |
| • Ashwani or ashwani | • Ash |
| • Any other name | • Buddy |

- Prompt the user for his/her age. Prompt with **`System.out.print("How old are you? ");`**
- Get the age
- Based on the age, print one of the messages in the table followed by a comma, a space, and the nickname.
- Example: You are a child, Jim

| • T his age and over | • Under | • Message |
|---|---|---|
| • 0 | • 18 | • You are a child |
| • 18 | • 21 | • You can vote, but you can not drink |
| • 21 | • 65 | • You are a full adult |
| • 65 | • | • You can get Social Security |

- Print Goodbye, *name*! where *name* is the original name that was entered.

Hint: To get just the first name, you need the index of the space. You can get that with the String method **`indexOf`**. indexOf returns the index of the first occurrence of the substring or -1 if the substring is not found.

Use this code to get the index of the space

```
int indexOfSpace = name.indexOf(" ");
```

If the indexOfSpace is > -1, use substring to get all characters in the first name. Otherwise use the whole name entered as the first name

Notes:

- Use the proper if structure.
- No not use do nothing if or else clauses
- Do not do unnecessary tests in the if clause
- For example this is incorrect:
- `if (age >= 18 && age < 21)`
- Leave out the `age >= 18`. It HAS to be true because you have just determined that `age < 18` is false.

For the draft, do the first 2 steps.

5A draft:

5A final: (link updated Sept 19)

5B

We do not want users of our classes to be able to set invalid values for instance variables. Now that you know how to use if statements, you are going to add error checking to a a few classes from earlier homeworks. You can copy correct solutions from Canvas Modules if you need to.. You should not print error messages from constructors or methods of object. In the real world, you might throw an exception when the parameter is invalid, but we have not covered

exceptions yet. We will just be sure the data in our objects are valid. Follow the directions below to add error checking to each class.
Use proper if structure. Do not do unnecessary tests . Do not have empty if or else blocks

**`Window`** (hw4)

- In the constructor, if either the width or the height parameter is <= 0, set both width and height to 0
- In **`setDimensions`**, only change the width and the height if both are > 0. If either is 0, don't make any changes.

**`GroceryBill`** (hw3)

- In **`add`** method: only add the amount if it is greater than 0
- In **`remove`** method: only remove the amount if it is greater than 0 and also less than or equal to the subtotal

**`Hal9000`** (hw3)

- Modify both the constructor and the setName method as follows
    - Restrict the name to 8 characters. If the name contains more than 8 characters, set the first 8 characters as the crewmember's name
    - If the parameter is Robert, set the name to Bob

If you do not have a correct version of one of these classes, you can get it from the posted solutions.
For the draft, make the changes to **`Window`**
5B draft:
5B final:
5C
Write a class called **`TwoNumbers`** which will ask the user for two numbers and print out comments based on their values. (no starter file. You will write all the code)
Use this exact prompt to get input:

```
System.out.print("Enter two numbers (like this: 41.7 -22.5): ");
```

The user will enter both numbers on the same line at the same time. This is possible because the first call to **`nextDouble`** will skip any leading whitespaces and then read up to the next whitespace (the space), leaving that whitespace in the input stream. The second call to **`nextDouble`** will ignore leading whitespaces and then read up to the next whitespace (the newline) leaving the newline in the input stream. This is a handy way to request multiple inputs.
Here are the conditions and the comments to print.

| Condition | Comment |
|---|---|
| first is equal to second | The first number is equal to the second |
| first is greater than second | The first number is greater than second |

| | |
|---|---|
| first is less than second | The first number is less than second |
| the numbers are within .0001 of each other | The numbers are close together |
| the numbers are more that 10000 apart | The numbers are far apart |
| they have the same sign/different sign | The numbers have the same/different sign |

If the numbers are equal only print the one message. Otherwise print all that apply. Copy and paste the prompt and the replies. That way you will have exactly what Codecheck expects. Print the comments in the order given above

Use the appropriate if structure and do not just use all ifs.

Note for advanced students: Do not use `System.exit()`

For the draft, get the numbers from the user and print the appropriate one of the first 3 comments.

5C draft:

5C final:

6A

The Java Beach Resort has two types of rooms: Ocean Side and Street Side. The price is based on the number of occupants and the type. See the table. Complete the class `ResortRoom`.

| | **2 people** | **4 people** |
|---|---|---|
| **Ocean side** | $250.00 | $370.00 |
| **Street side** | $175.00 | $260.00 |

An int is used to represent the type of view. 0 is ocean view and 1 is street side. Use the defined constants to avoid errors

- `public static final int OCEAN_SIDE = 0;`
- `public static final int STREET_SIDE = 1;`

`ResortRoom` has a constructor that takes two parameters.

- `public ResortRoom(int type, int numberOfOccupants)`

If *type* is other than OCEAN_SIDE or STREET_SIDE, set the type to OCEAN_SIDE.

If the number of occupants is <= 0, set the occupants to 2.

Call the instance variable for the number of occupants *occupants*. That way the provided getOccupants method will work.
It also has a methods:

- `public double getCost()` - gets the cost of the specified room. Use nested if statements.
- `public int getOccupants()` gets the number of occupants in the room (provided)
- `public void setOccupants(int number)` - sets the number of occupants renting this ResortRoom
- `public String getType()` - gets type of this room, either "ocean" or "street" You will need to use an if statement to determine which string to return

The cost is based on the table above. One person costs the same as two people and three people cost the same as 4 people. For more than 4 people, the charge is $100 per each additional person for any room.
Define and use a constant for the cost for an extra person. (What should data type should it be?)
Provide Javadoc
For the draft, implement the constructor and getType method. Provide stub for the getCost and setOccupants methods so that your class will compile with the tester. Provide all the Javadoc
6A draft:
6A final:
6B
Write a class `FunWithLoops` that contains the methods specified below. There is no constructor or instance variables.

- `public int sumOdd(int value)` Return the sum of all the positive odd numbers less than the given value starting with 1. If *value* is less than or equal to 0, return 0. Use a loop
- `public double average(int count)` Asks the user for *count* integers and returns the average of the integers. If count is 5, you will prompt the user 5 times to enter a number. Use prompts that look like these in the loop:
-
- Enter integer 1:
- Enter integer 2:
- ...
- Enter integer 5:
-
- If count is <= 0, return 0. (We do not want to divide by 0 )
-
- The output might look something like this is, if count is 2
    - Enter integer 1: 25
      Enter integer 2: 4
      14.5

- `public double sumSeries(int value)` Calculates the sum of the following series up to a value of n (n must be odd). (changed 10/10)
- 
- 1 + 1/3 + 3/5 + 5/7 + ... (n-2)/n
- 
- if n is less than or equal to 0 or is even, return 0. If n is 1, return 1.

No starter. No Javadoc

For the draft, implement the `sumOdd` method

6B draft:

6B final:

6C

Complete the class `LoopyText` which has a constructor that takes a String of text (provided for you). Implement the following methods.

- `public String getEverySecondCharacter()` gets a string consisting of every other character, starting with the character at index 0
- `public int upperCaseCount()` gets the number of uppercase letters in the text. You can use the String `contains` method or the Character class' static method `isUpperCase`
- A method called `firstLetters` that returns a string consisting of the first character of every word in the string. If the string is the empty string return the empty String. You can use indexOf (" ") to control the loop and to determine where a new word starts.

Provide Javadoc

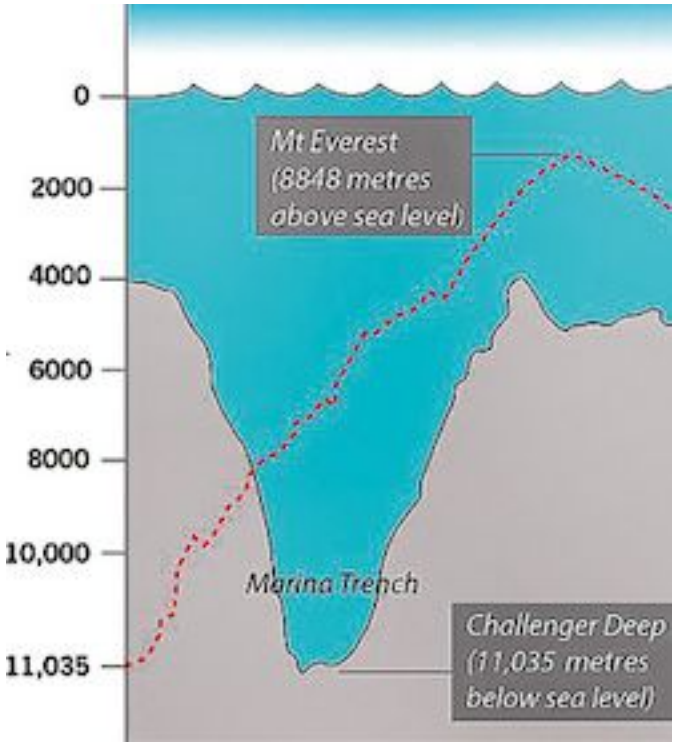
For the draft, implement the `getEverySecondCharacter()` method

6C draft:

6C final:

7A

Every place on earth has an elevation either above or below sea level from Mount Everest (29,035 feet above sea level) to the Challenger Deep in the Mariana Trench (36,070 feet below sea level)




Write an application (A program that has a main methods) called **`SeaLevelProcessor`** to read a collection of integers, each representing a (positive) distance above or a (negative) distance below sea level (elevation) of a point on the earth. The input will terminate when the user enters a non-integer.
Use this prompt: *System.out.print("Enter the sea level or Q to quit: ");*
Note: you will actually quit on any string that can not be converted  to an int.
Do the following:

- find and print the number of elevations greater than 0 (above sea level).
- find and print the highest elevation
- find and print the average elevation

Only print the numbers if at least 1 integer was entered. Otherwise print "*No values entered*"
You will read the inputs one at a time and do all the processing as you go. No Arrays for this
The outputs should be on separate lines and in this order

- number of elevations above 0
- highest elevation
- average elevation

Do not initialize the maximum to some 0 or a negative number. Initialize to the first input. You can use a Boolean variable as a flag as discussed in the videos.
No Javadoc required when there is only a main method.
For the draft, read the numbers and print them, one to a line. Quit when the input can not be converted to a int. (That means hasNextInt() returns false.) Use the prompt given above. After exiting the loop, print "All done" (only in the draft)
7A draft:
7A final:

7B

Write a class StringsAndThings. It  has a constructor that takes a String parameter.
Write methods:

- **`countNonLetters`** that will count how many of the characters in the string are not letters. You can use Character's **`isLetter`** method
- **`moreVowels`** which returns true if the String parameter has more vowels than consonants. Otherwise it returns false. Ignore punctuation and digits. Add the **`isVowel`** method from Lab6 to your class. You can use **`isLetter`** from the Character class
- **`noDuplicates`**  which returns a string where each character in the phrase appears exactly once. Consider upper and lower case letters as different letters. Note that spaces and punctuation are also characters. This is a good place to use String's **`contains`** method. Hint: Remember if dup is true then !dup is false. Do not use a do-nothing if clause. There are a lot of weird solutions on the Internet. Do not use anything we have not covered. If the string is Mississippi, the return value will be Misp. Hint: it is easier to use substring here than charAt.

No starter file. Provide Javadoc for the class, the constructor, and all methods.
For the draft: implement the class with constructor and instance variable and the **`countNonLetters`** method. Provide Javadoc for the class the constructor and the method.
7B draft:
7B final:

7C

A digital image is made up of pixels. Each pixel is a tiny square of a given color. The Picture class has methods to manipulate pixels. Each pixel has x, y coordinates. You can think of the image as being rows of pixels. The x coordinate is the row and the y coordinate is the position within the row. The x coordinate varies from 0 to the width of the image, and the y coordinate varies from 0 to the height of the coordinate. You can use nested loops to pick up each pixel in the image and then do something with it. Worked Example 6.2 gives an example of iterating through all the pixels in an image.
Your task is to write an application **`Framer`** that can put a round frame around a Picture. See the output below. You should set all the pixels outside the circle to black. The center of the circle is the center of the image, and the radius is 40 percent of the width or height, whichever is smaller. Find the center by dividing the width and the height of the image by 2. Get the width and height by using methods so that the code will work if you use any image. That is, do not assume that the dimensions of the image are 200 x 180.
You will need to import the Udacity graphics package into your project.
Hint: If the distance of the pixel from the center of the circle is greater than the radius, set the pixel color at that x, y to Color.BLACK. Think Pythagorean Theorem
Here is the original image of oliver_bed.jpg

Here is the a screenshot of the output

For the draft, make the first row of pixels red (We are using red because it is more visible on the image than black. You will use black in the final). The y coordinate of every pixel in the first row is 0. Use Picture's **`getWidth`** method to get the width of the image.

7C draft:

7C final:

8A

Complete the application **`RandomSquares`** which uses a loop to draw 25 squares with random length sides at random locations. Make the x coordinate a random number between 0 (inclusive) and 200 (exclusive), Make the y coordinate a random number between 0 and 300

(exclusive). This x,y is the upper left hand corner of the rectangle (square). Make the length of the sides a random value between 20 (inclusive) and 100 (exclusive). Declare and use constants for:

- maximum x coordinate
- maximum y coordinate
- maximum length
- minimum length
- number of squares

Some constants are already defined for you. You do the rest

In the starter file, I have created a **`Random`** object for you with a seed. Be sure to use it.

Because of the seed, the **`Random`** object will produce the same sequence of numbers every time the program is run so your drawing will always look the same (and will pass Codecheck). Do not change the seed.

Generate the random values in this order: x coordinate, y coordinate, length of the side.

You must use a loop. Do not have 25 lines of code each of which draws a square

Draw the squares in green. After you have drawn all the squares, fill the largest square with blue. Use the predefined colors from the Color class. If more than one square has the largest side, use the first one with that side as the largest. You will need to keep track of the largest as you go.

You will need to import Dr. Horstmann's graphics package into your Bluej project.

For the draft, draw one random square in green using the predefined color from the Color class.

8A draft:

8A final:

8B

There is no start file this time. You create a class called **`PracticeWithArrayLists`** with a main method In the main method, do the following

- Create an ArrayList of Strings called zoo
- **`Add`** "tiger", "lion", "elephant", "kangaroo" in that order.
- **`Add`** "giraffe" at position 1
- Replace the element at index 4 with "zebra" Use the **`set`** method
- Replace the next to the last element with "wallaby". Do this in a manner that would replace the next to the last element, no matter the size of the array list. Think about how to find the last element in an array list, then find the index of the next to the last element and use the **`set`** method.
- Remove "lion"
- Get and print the 0th element followed by "***"
- Print the elements on one line in the [xxx, yyy, ...] format
- Print the entire array list one element to a line (use the enhanced for loop)

Note: You must use an ArrayList and its methods. Do not fake it by just printing the correct strings. You will know how to use these methods in a situation where the output can not be faked.

Note: There is a version of remove method that will remove a specific object. Use that. Here is the code to remove target from the ArrayList called stuff

```
stuff.remove(target);
```

Note: To print an ArrayList, use its toString method like this

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

For the draft,

- create the array list of of strings call zoo,
- Add "tiger", "lion", "elephant", "kangaroo" in that order.
- Print the elements one to a line. Use an enhanced for loop.

8B draft:

8B final:

8C

Finish the class **`Rectangles`**. It contains methods that manipulate an ArrayList of Rectangles Rectangles has an instance variable ArrayList<Rectangle> Call the instance variable *list*. It has a constructor that takes no parameters but initializes the instance variable to an empty ArrayList It has methods

- **`public void add(Rectangle r)`** adds this Rectangle to the ArrayList of Rectangles
- **`public void swap(int index1, int index2)`** - swaps the element at index1 with the element at index2. If either index is out of bounds, do not changing anything.
- **`public Rectangle largest()`** gets the Rectangle with the largest area. If more than one Rectangle has the same areas, return the first. If the Rectangles object is empty, return null
- **`public String toString()`** gets a string representation of the ArrayList - provided. (This is why the instance variable must be called *list*.)

You will need to import the Udacity graphics package in order to use Rectangle
Provide Javadoc
Note that if you do not implement the methods in order, you may not get the expected results until all the methods are implemented
For the draft, implement the constructor and the add method

8C draft:

8C final:

9A

In the **`PointsGalore`** class, finish the drawRectangle method.

**`PointsGalore`** has a constructor that takes an ArrayList of Points as a parameter. The constructor is provided

The `drawRectangle` method fills small circles at the points (provided) and then draws the smallest Rectangle that contains the points (you will do this)
Note: Points that fall on the rectangle boundary will not look like they are entirely inside because the points are drawn as ellipses with width and height while a point does not actually have dimensions.
The `Point` object is defined in java.awt.Point. The only thing you need to know about the class is that you have to import it and that it has `getX` and `getY` methods which return doubles.
You will need to import the Udacity graphics package into
Best programming practices dictate that you should use several helper methods rather than have one long method. That means you should write helper methods to get the maximum and minimum x coordinate and the maximum and minimum y coordinate
For the draft, draw a rectangle whose upper left-hand corner is at the coordinates of the Point at index 0 and which has a width of 50 and a height of 30.
9A draft:
9A final:
9B
Finish the class `RectanglesArray`. It contains methods that manipulate an array (not an ArrayList)
Rectangles has an instance variable Rectangle[] Call the instance variable *list*. It has a constructor that takes an array of Rectangles as a parameter and initializes the instance variable with it.
It has methods

- `public double averageArea()` finds the average area of a Rectangles in this array. Use the helper method `area` described below. Do not calculate the area in this method.
- `public void swap(int index1, int index2)` - swaps the element at index1 with the element at index2. If either index is out of bounds, do not changing anything.
- `public Rectangle largest()` gets the Rectangle with the largest area. If more than one Rectangle has the same areas, return the first. Use the helper method `area` described below. Do not calculate the area in this method.
- `public double area(Rectangle r)` gets the area of the rectangle parameter
- `public String toString()` gets a string representation of the array - provided. (This is why the instance variable must be called *list*.)

You will need to import the Udacity graphics package in order to use Rectangle
`largest` and `swap` are similar to the methods in 8c. Those processed an ArrayList. These work with arrays.
Provide Javadoc
For the draft, implement the constructor and helper method `area()`
9B draft:
9B final:

9C

`Util2DInteger` manages a 2d array of ints. It has a constructor that takes a 2d array as a parameter and assigns it to the instance variable.
You will write the whole class.
Provide the constructor and instance variable.
Provide these methods

- `getSmallest()` Gets the smallest integer in the array
- `public int product()` gets the product of the elements in the array. It would probably be easiest to initialize product to 1. [Note: product means multiply]
- `last()` Gets the element in the last column of the last row
- `contains(int target)` Returns true if the target is in the array, otherwise false

Provide Javadoc
For the draft, provide the constructor and the getSmallest method. Implement the other methods as stubs. Remember that a stub for a boolean method returns false. Provide Javadoc. Remember the Javadoc should say what the method will do in the final not what it does in the draft.
9C draft:
9C final:

10A

In this problem you will write several static methods to work with arrays and ArrayLists. Remember that a static method does not work on the instance variables of the class. All the data needed is provided in the parameters. Call the class `StaticPractice`. Notice how the methods are invoked in `StaticPracticeTester`.
`StaticPractice` is a utility class. It has no instance variables and should not have a constructor

- `public static int max(int[] numbers)` gets the maximum value in the array. **Use the enhanced for loop**
- `public static int max(ArrayList<Integer> numbers)` Gets the maximum value in the ArrayList as an int. You can assume the ArrayList contains at least one element. **Use the enhanced for loop**
- `public static boolean containsTwice(int[] list, int target)` determines if the target is in the array at least two times. Returns true if it is, otherwise returns false

- `public static boolean containsTwice(ArrayList<Integer> list, int target)` determines if the target is in the ArrayList at least two times. Returns true if it is, otherwise returns false

Notice that there are two methods called `max` and two methods called `contains`. These are examples of overloading - methods with the same name but different number and type of parameters. The compiler tells them apart because in each case, one takes an array and one an ArrayList as a parameter.
When using the enhanced for loop, the implementation of the pairs of methods will be very similar. The exact same loop works for both
Provide Javadoc. Look at the documentation produced for `StaticPractice`. The Javadoc utility works on static methods, too.
For the draft, implement first the max method

10A draft:
10A final:

10B

Now we are going to use the design pattern for collecting objects. We are going to model a `GardenStore` with plants. A `GardenStore` uses an `ArrayList` to keep track of `Plant` objects. You will write both a `GardenStore` class and a `Plant` class.
A `Plant` has a name and a price. Provide a constructor that takes name and price, in that order. Provide getters and setters for the instance variables. This is the design pattern for managing properties of objects.
A `GardenStore` has a constructor that takes no parameters. Remember it must initialize the instance variable
It has methods

- `add()` Adds the specified `Plant` to the `GardenStore`
- `sum()` Finds the total cost of all `Plants` in the `GardenStore`
- `contains()` determines if a `Plant` with a given name is in the `GardenStore`. Returns true if a `Plant` with that name is in the `GardenStore`. Otherwise false.
- `plantList()` gets an ArrayList names of the `Plants` in the `GardenStore`.

Provide Javadoc for both classes.
For the draft: implement the `Plant` class
Note: The `Plant` class will not change in the final, but you will need to submit it again so that `GardenStore` can find it.
10B draft:
10B final:

10C

In this problem you will use the design pattern for maintaining state. Write a `Zebra` class. A `Zebra` has 4 states. You will define and use these static constants to represent the states.

- `public static final int NOT_HUNGRY = 1;`
- `public static final int SOMEWHAT_HUNGRY = 2;`

- `public static final int HUNGRY = 3;`
- `public static final int VERY_HUNGRY = 4;`

While in your code you can not assume what the value is for any of the constants, you can assume that they the values are consecutive integers. That is, VERY_HUNGRY will be 1 greater than HUNGRY, etc.
Zebras roam the Serengeti Plain in Tanzania.

(*image from Wikipedea*)
A `Zebra` runs across the plain and as it runs, it becomes more hungry. If it is `NOT_HUNGRY`, it becomes `SOMEWHAT_HUNGRY`. If it is `SOMEWHAT_HUNGRY`, it becomes `HUNGRY` and so on. When the `Zebra` sees food, if it is in any of the "hungry states", it will eat and become one level less hungry. If the `Zebra` is `VERY_HUNGRY` when it sees food, it will eat and become `HUNGRY`. The next time it sees food, it will eat again and become `SOMEWHAT_HUNGRY`. If it is `NOT_HUNGRY`, it does not eat and its state does not change.
The `Zebra` can not be less than `NOT_HUNGRY` or more than `VERY_HUNGRY`.
The constructor takes no parameters. A `Zebra` is very hungry when it is creates so the constructor must initialize the state to `VERY_HUNGRY`.
Provide methods:

- `public void run()` the `Zebra` becomes more hungry if not already `VERY_HUNGRY`
- `public void seeFood()` the `Zebra` will eat if it is hungry and become less hungry
- `public int getState()` Gets the integer representing the state, an integer 1 through 4.

- **`public String getHungerLevel()`** Gets a string describing the current hunger state of the **`Zebra`**: "Not hungry", "Somewhat hungry ,"Hungry", or "Very hungry"

Provide Javadoc

For the draft, provide the static constants, implement the constructor, and the **`getState()`** method. Implement the other methods as stubs. Provide Javadoc

10C draft:

10C final:

11A

The **`GeometricShape`** interface will have one method, **`area`**, which takes no parameters and returns a double.

You are provided with three classes: **`Parallelogram`**, **`Circle`**, and **`RightTriangle`**. Modify the classes so that they implement the **`GeometricShape`** interface. Supply the appropriate method for each class.

Use Math.PI.

Notice in InterfaceRunner for the final version that the objects are added to an **`ArrayList`** of **`GeometricShapes`**.

For the draft: write the **`GeometricShape`** interface and then modify **`Parallelogram`** class to implement the **`GeometricShape`** interface.

11A draft:

11A final:

11B

Modify the **`Plant`** class from last week (hw10b) to implement the **`Comparable`** interface. If you need it, you can get a working version from the solutions

**`Plants`** are ordered first by price with the smallest coming first. If two **`Plants`** have the same price, the **`Plants`** are ordered alphabetically by name. If this **`Plant`** should come before the other **`Plant`**,compareTo returns a negative integer. If this **`Plant`** should come after the other **`Plant`**, it returns a positive integer. Otherwise, it returns 0.

Remember about using Double.compare to compare doubles.

Remember that the **`compareTo`** method take a **`Object`** as parameter not a **`Plant`** (That means you will have to cast)

For the draft, add the "implements Comparable" clause to the **`Plant`** class header and implement compareTo as a stub

Note: You will get a compiler warning about "unsafe or unchecked operations" when you compile **`PlantTester`**. You can safely ignore it for now.

11B draft:

11B final:

11C

Write a class to model a `Computer`. A `Computer` has a constructor that takes the brand as a String. It also takes a double, gigahertz. Call the instance variables brand and ghz. Gigahertz is a measure of the clock speed of a computer.
`Computer` implements the the `Comparable` interface. `Computer`s are ordered by gigahertz. If two `Computer`s have the same speed (gigahertz), then the `Computer` with the brand that comes first in the lexicographical (alphabetical) order is the smaller. For Example, "Apple" comes before "Surface" alphabetically and so is the smaller.
`Computer` has methods `getGhz()` and `getBrand()`.
A `toString` method is provided for you.
Provide Javadoc
Remember that the `compareTo` method take a `Object` as parameter not a `Computer` (That means you will have to cast)

For the draft, supply the instance variables, the constructor and the `getGhz()` and `getBrand()` methods.
11C draft:
11C final:

12A
Many people want to follow a low carbohydrate diet. In order to do that, they need to know how many carbohydrates are in the foods they choose.
Make a subclass `Food` of the `Product` class we created in class. Besides the description and price inherited from `Product`, `Food` also has an instance variable carbs (an int). Do not redefine price and description variables. Call the super class to initialize them
`Food` has a methods:

- `public void setCarbs(int grams)`
- `public double getCarbs()`
- `public boolean isLowCarb()` which tells if this `Food` is considered low carb. A `Food` is considered low carb if it has 10 or fewer grams of carbohydrates. Return true if the `Food` has 10 or fewer carbs. Otherwise, return false.
- `public String getDescription()` overrides the `getDescription` method in `Product` to also include the number of carbs. Call the getDescription method in the super class and add the new information on the end. The return string will look like this: **oatmeal carbs=8.3**

Provide Javadoc.
For the draft, make `Food` a subclass of `Product`. Provide the new instance variable, carbs. Write the constructor that takes price, description, and carbs, in that order. Be sure to initialize all the instance variables. Provide `getCarbs`. Implement the the `setCarbs` and

`isLowCarb` methods as stubs. Remember that if a stub's return type is boolean, it should return false. Do not include `getDescription` in the draft.

12A draft:

12A final:

12B

Write a subclass `SpecialRectangle` of the `Rectangle` class A `SpecialRectangle` has a special set of colors. It has a String instance variable called paintColor. **Do not add any instance variables except the paintColor.** You will get lose credit if you do.

Valid paintColors and their red, green, blue values are:

vanilla (255, 255, 248)

blue_green (204, 255, 255)

purple (159, 0, 197)

burnt_orange (227, 117, 00)

Provide two constructors:

Each constructor will need to set the paintColor instance variable and call the super class `setColor()` method so the `SpecialRectangle` will be drawn in the correct color.

- The first constructor takes 4 parameters: the x, y coordinate of the upper left-hand corner and the width and the height as ints. This constructor sets a value of "silver" (204, 204, 204) to the instance variable
- A second constructor takes 5 parameter: the x, y coordinate of the upper left-hand corner and the width and the height and a String paintColor. If the String is any other value besides the four valid strings, set paintColor to "silver" (204, 204, 204)

Provide methods to set and get the paintColor.

For the draft, provide the instance variables and implement the constructor with 4 parameters. . Implement the getPaintColor() method. Code setPaintColor as a stub.

12B draft:

12B final:

12C

Animals need energy to move, which they get from eating food.

Create a class `Animal` with a constructor that takes no parameters and has an instance variable

```
private int energy;
```

When an animal is "born" it has no energy.

It has the methods

- `public void eat(int amount)` - which increases the amount of energy the animal has by amount, but only if amount > 0
- `public void move(int amount)` - which decreases the amount of energy the animal has by amount. Energy changes only if amount > 0. The energy can never be 0. If an `Animal` has an energy of 2 and tries to move 5, its energy will be
- `public int getEnergy()` - which returns the amount of energy the animal has left

Notice there is no setEnergy method. Energy is only changed by eating or moving.

When an animal is "born" it has no energy.
It isn't realistic for animals to be able to gather infinite amounts of energy. Create a subclass `ImprovedAnimal` which has a cap on the amount of energy an animal can have. The constructor takes a parameter that sets a maximum for energy. If the amount the `ImprovedAnimal` eats would set its energy above the max, the energy level is only increased to the max.
For the final, `Animal` will not change, but you need to submit it.
Provide Javadoc for both classes.
For the draft, write the Animal class. Include Javadoc
12C draft:
12C final: