UNIVERSITY OF ALBERTA

AUCSC 112 LAB

Assignment #1

(Due before midnight on day before the next lab)

Goals:

- To program in Java – use sequencing, simple methods, and simple loops (use `while` and `for-each` only).
- To increase in computational thinking ability, and be able to explain why code works.
- To name descriptively, to maintain proper code indentation, and to begin to subdivide program code.

Reference:

Heise, chapters 0 and 1.

GitHub Site:

https://github.com/AUCSC-112/Lab-1-Assignment

Instructions:

Write a variety of Java methods and code, as outlined below (Part 1, Part 2, and Part 3), in three separate files (and separate projects). You may always create more than the requested number of methods; you are even encouraged to do so.

This is an INDIVIDUAL assignment - the code you write should be entirely your own. Do not use code from anyone else (including from the internet). Do not submit any parts of this assignment to any internet site, including homework help sites. When you are helping someone else, do not show your own code. You may help others by finding bugs or explaining a concept, but do not share code.

Do not import any libraries. Do not access Java's String library methods (i.e. no dot messages to Strings), except the concatenation operator (+) is allowed.

Ensure that you name all files and methods EXACTLY as specified, remembering that Java is case-sensitive.

Part 1:

Filename: **`Main.java`** (this will be the default file name, in a regular project under the IntelliJ IDE)
This file must contain the following components:

1) A file header with information about the code, including file summary, author, class, ID number, and date. The file header should be the first thing in the file.

2) A method: **`main(String[]) → void`**
   This method prints "`Hello Java Programmer` ☺" to the IDE's output window. A smiley can be made by copying the symbol or with `\u263A`.

3) A method: **`sayWelcomeTo(String) → void`**
   This method takes someone's name and prints out a message, with that person's name filled in. Include the symbols (`\u2603` = ☃; `\u2744` = ❄).

For example: `sayWelcomeTo("Mickey");`

```
Hi there Mickey. Nice to see you!
Hope you enjoy coding in Java.
```

Indentation must match the example. If the String parameter is the null String or the empty String, still print the welcome message, but omit the name, making sure there is no extra space before the period.
For example: `sayWelcomeTo(null);`

```
Hi there. Nice to see you!
Hope you enjoy coding in Java.
```

Note that in Java, you may correctly compare a String to null or empty using "==" and "`!=`". The usual `.equals`, which would be a String library method, is not necessary.

4) Extend `main` so that it prints the welcome message to each element in this array:

```
String[] classList = {"Mickey Mouse", "Minnie
      Mouse", "Donald Duck", "Daisy Duck", "Pluto",
      "Santa Claus", "", "Wile E Coyote", "Road
      Runner", "Bugs Bunny"};
```

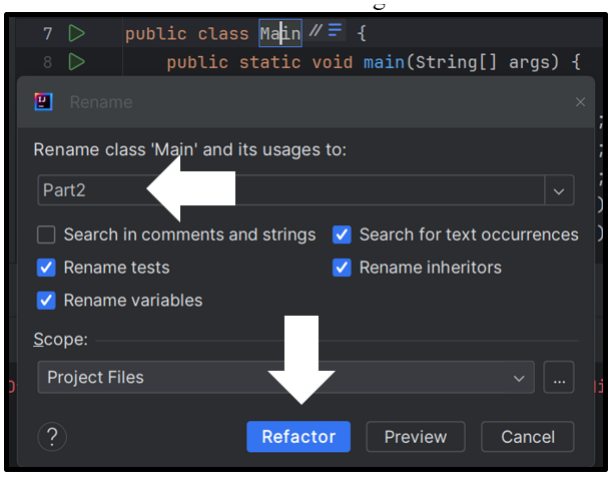
Do this by using Java's enhanced `for` loop, the `for-each` loop, which matches Python's most common loop. Make sure you have a blank line between the messages to each person. This blank line should come from `main`, not from your `sayWelcomeTo` method.

Do not forget to include proper Javadoc method headers on all of your methods.

Part 2:

Filename: **`Part2.java`**

Note: to change the filename, click on the class name "Main", then from the top menu tabs select "Refactor" → "Refactor This…" → "Rename" → "Refactor" → "Refactor This…" → "Rename". (You need to do the sequence twice to get the actual dialog box to change both the filename and class name at the same time.) Put in the new name for the class (and file happens automatically) and click the "Refactor" button. You should see a dialog similar to this:



This file must contain the following components:

1) A file header with information about the code, including file summary, author, class, ID number, and date.

2) A method: **`main(String[]) → void`**
This method will contain the testing that you write for the following method.

3) A method: **`printNumChars(int, char) → void`**
This method takes a number (as a primitive integer) and a character, and repeatedly prints that character the given number of times. Do not move to the next line. Make sure that your method handles sizes 0 or negative appropriately by printing absolutely nothing. For any loop that you create, use only a `while` loop.

For example: `printNumChars(4, '&');`
`&&&&` and does not move to new line nor end with a space.

4) A method: **`printHouse(int) → void`**
This method prints a house to standard output, using the character that is the first letter of your first name. The house must be of the size specified by the parameter. This size is used to determine key aspects of the house:

- The number of rows making up the wall of the house matches this size exactly, when including the base in this number.
- The number of rows in the roof of the house is half of this size (rounding down for an odd number).
- The height of the door of the house (the door width is always 3).
- The base of the house has width `2 * size - 1`.

Do not print any houses smaller than a size of 5. For any loop that you create, use only a `while` loop. You should make use of your `printNumChars` method wherever appropriate.

My name starts with "`R`"; remember to use the letter that your name starts with.

For example: `printHouse(5);`

```
  R
 R R
R RRR R
R R R R
R R R R
RRRRRRRRR
```

For example: `printHouse(8);`

```
   R
  R R
 R   R
R     R
R     RRR     R
R     R R     R
R     R R     R
R     R R     R
R     R R     R
R     R R     R
R     R R     R
RRRRRRRRRRRRRRR
```

Number of rows in roof is `size / 2`

Number of rows in main part of house is `size` (including the base)

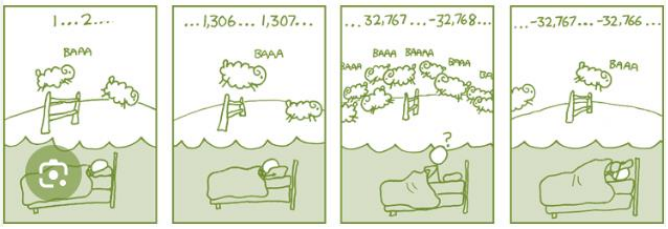
Number of letters in base is `2 * size - 1`

For example: `printHouse(-25);`
`Sorry, cannot print a house so small`

A perfect solution would break this problem into smaller methods, for example, having a method to print the roof and a method to print the general sides.

Reminder: all methods need method headers.

Part 3:



Filename: **`Part3.java`**
This file must contain the following components:

1) A file header and method headers for any method that you make.

2) A method: **`main(String[] args)` → `void`**
This method will contain the testing that you may write for the following method.

3) A method: **`visualInt(int)` → `String`**
This method returns a String, which is the integer in visual form, using the correct number of stars to represent each digit. Place a single blank character between each digit and keep the sign in front (as just '`-`') when the number is negative (no sign on a positive number). If one of the digits is 0, use the underscore ('`_`') to signify that. The lead digit will never be 0, unless the entire number is 0.

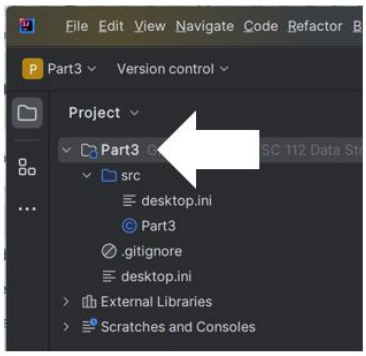
   For example: `visualInt(-5723)` → `-***** ******* ** ***`
   For example: `visualInt(123456)` → `* ** *** **** ***** ******`
   For example: `visualInt(102)` → `* _ **`
   For example: `visualInt(0)` → `_`

   Reminder: Do not use any String methods except concatenation (+). Notice also that this method needs to return a value (and not to print anything from inside the method itself – any testing that you create, inside `main`, could print the returned result).

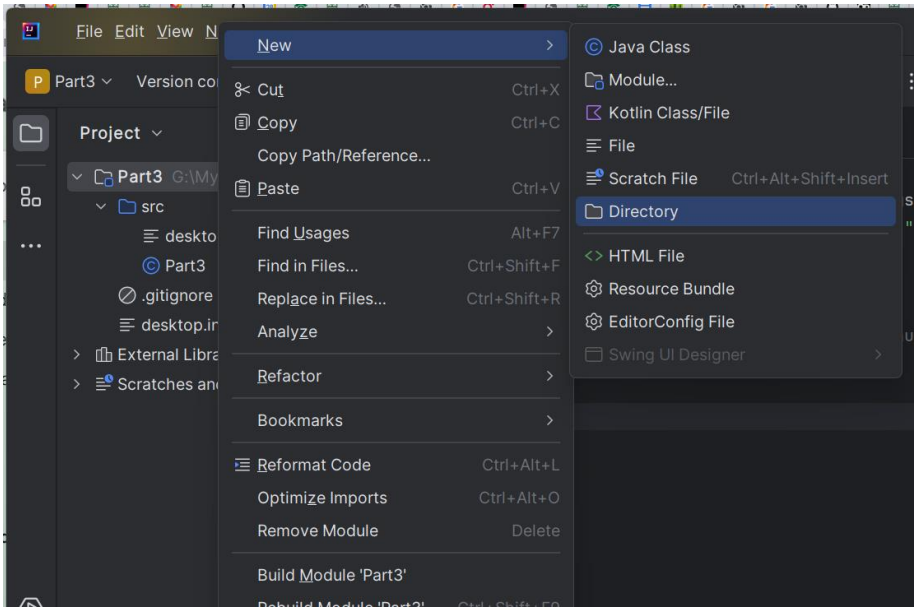
   Hint: What does a base-10 integer mean? You can use the division and modulus operators to extract any digit from a number.

Once you have made the `visualInt` method, test it in the IntelliJ IDE, by creating a testing file "`Part3Test.java`", and copying in the contents of the same file from our class GitHub website. Here are instructions for how to set up and perform the testing properly in IntelliJ:

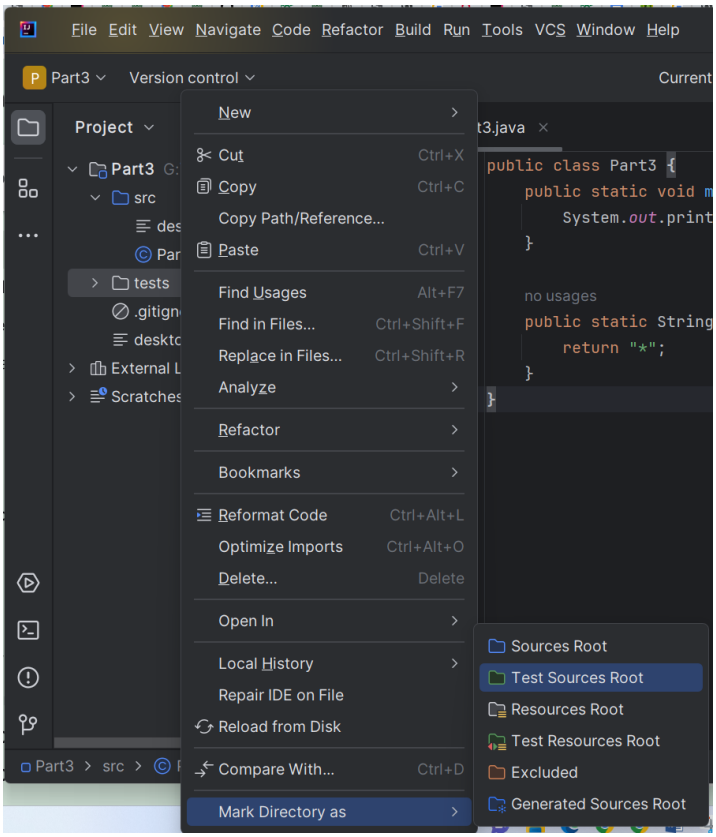
1) Make a directory for the testing:
   a) Right-click on your project at the left side:



b) "New" → "Directory" → fill in the directory name as "tests" and press "Enter"



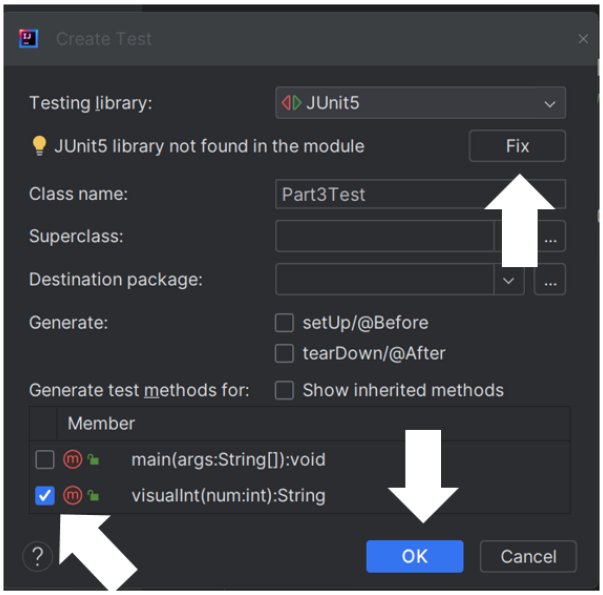
c) Right-click the tests directory → "Mark Directory as" → "Test Sources Root"



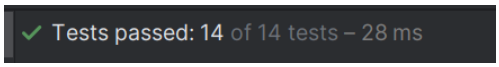
2) Make the testing file:

- **a)** Click on the class declaration line, "`public class Part3`".
- **b)** Press Alt-Enter or Right-click and select "Show Context Actions".
- **c)** Click "Create test".
- **d)** Click the "Fix" button if the library is not found.

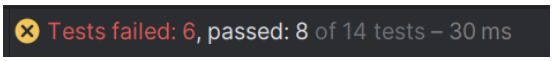
e) Select visualInt, and click "OK".



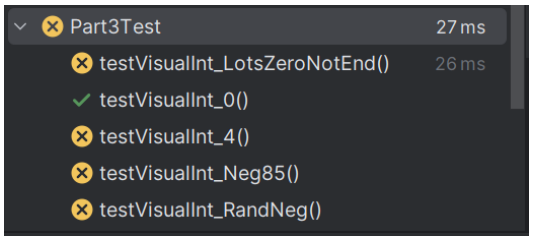
3) Copy the contents from GitHub into this new file. Then run the tests by simply running the testing file. If all of the tests pass, then you will see something similar to this:



Otherwise you might see something like this:



If any tests have failed, then you may modify your code to attempt to correct your solution. The bottom, leftmost window shows which tests have failed (with a yellow "X") and which have passed (with a green checkmark). For example:



You can run the tests as many times as you want. Note that you should not write code that only passes the given set of tests, but that rather works for all possible test cases, even those not included in this particular testing file.

How to Hand In Your Work:

Submit six files on eClass:

1) Three `.java` files
- **a)** `Main.java`
- **b)** `Part2.java`
- **c)** `Part3.java`

Remember that they must be named exactly with matching case and proper extension. The name of your class (inside the file) must match too.

2) A `.pdf` of each Java file
   a) `Main.pdf`
   b) `Part2.pdf`
   c) `Part3.pdf`

Practice Questions – Assignment 1 Concepts:

These are self-check questions; do not hand in solutions. These questions are provided so that you can check whether you have learned the concepts we are expecting you to learn. No solutions are provided, and we encourage you to discuss these questions with other students.

1) How do you start writing a Java program?
2) How is a comment put into a Java program? State two ways.
3) Where does program execution start in Java?
4) Why does the `main` method have a `void` return type?
5) What is `String[] args`?
6) What is the shortcut for running a Java program in your IDE?
7) What are three things you can say about `System.out.println`?
8) What is a method (= function = subprogram)? List all the methods you wrote.
9) What are the parameters of a method?
10) What is the formal parameter in your `sayWelcomeTo` method? What is the actual parameter?
11) What is the difference between writing a method and calling a method?
12) How is the return type of a method specified?
13) What is the difference between printing from a method and returning a value from a method?
14) `classList` is an array. What does it mean to be an array?
15) What is the data type of each element in `classList`?
16) What is a way to create repetition in Java code?
17) How do you make a Python-like for loop in Java?
18) What is the difference between a for-each loop and a while loop?
19) What is "`\n`", and what does this tell you about characters in general?
20) What does it mean to declare a variable?
21) What is the difference between a local variable and a parameter?
22) How is division in Java different from division in Python?
23) Suppose that an integer with magnitude greater than 10 is stored in the variable `x`. Write the Java code to get the 2nd last digit of this number.
24) What is a syntax error? Give an example of a syntax error you made.
25) What is a run-time error? Give an example of a run-time error you experienced.
26) What is a logic error? Give an example of a logic error you made.
27) How do methods help with the programming process?
28) Where is the source file of your programs located? Where is the object file located?

Image Credits:
House: https://www.alamy.com/stock-photo/cartoon-house.html?sortBy=relevant
Comic: https://www.approxion.com/grokking-integer-overflow/
Question Mark: https://www.freepik.com/free-photos-vectors/question-mark-cartoon
Hands in: https://www.shutterstock.com/search/cartoon-high-five-hand