DUE DATE: FEB 7TH, 2020 AT 11:59PM

Instructions:

- Read the **`SubmissionProcedures.PDF`** and the **`Standards.java`** carefully BEFORE you start.
- Let me know about any typos, clarifications or questions you may have right away. Don't wait.
- Read the whole assignment before you start.
- Start right away. The sooner you get stuck, the sooner you can get help to get un-stuck.

Assignment overview

Learning outcomes:

You will use custom instance objects as data types to store information, become familiar with static and instance variables and methods, and learn many of the basic skills you will need to successfully develop software. This will likely be the largest single piece of software many of you have written and managing that complexity is part of the challenge. Be sure to test your methods as you go and comment well.

Additional Notes:

Read **Standards.java** and **SubmissionGuidelines.PDF** Carefully & before you start.

The classes **PartySim.java and RandomGeneration.java** class will be provided but will need to be modified.

Please provide detailed commenting!!!

Assignment Notes

Be sure to read the assignment carefully. Include a **`readme.txt`** document that specifies the latest commit that will compile and run. I will **NOT** mark your assignment without the **`git`** repo included so do not "forget" to do it. This assignment is as much about using git, learning good formatting, commenting, testing and refactoring practices as it is about Java. Do not skip steps. Also, I am aware not all the methods you create are directly used in this assignment. You still need to create and test them, we will use them in the future.

You also may want to make backups if you are unsure about git. You can take the whole folder (**`.git`** file included) and put it in a **`.zip`** file if you are worried about breaking things.

Git Tips:

- Do not commit any file that should not be there. It is harder to remove files that have been added then to not add them in the first place. The commit is meant to be a reflective phase where you review your work. You CANNOT UNDO a commit and anything that gets put in will be in the record forever.
- The append option on a commit will allow you to revise or add additional files to the LAST commit you made, after that it is in the permanent record.
- Each git commit should include the Phase and a clear and complete summary of what code was added.
- If you find a bug, include that bug as its own commit. You should include the specific error (eg. Null reference) and how you fixed it. This will be a useful reference when you encounter the same error again.
- You may need to enable hidden files in your Windows/Mac file explorer in order to see the **`.git`** and **`.gitignore`** files.

Tips:

Start early. Read the assignment fully before beginning. We will go through it in the first class after it has been handed out. That is your chance to ask questions

Seriously, start early. I know you are busy but it is much easier if you get stuck early and can get help troubleshooting.

Notes on Academic Misconduct:

Your code should be done entirely by you. Any suspicion of Academic Misconduct or plagiarism will be fully investigated and can lead to serious academic penalties including 0 on the assignment for a first office, or an F in the course. Third offences could lead to expulsion from ICM. If you have any questions as to what qualifies as Academic Misconduct, please discuss it with me.

Things that are considered Academic Misconduct include:

- Copying code from old assignments
- Copying code from classmates or other people
- Telling or being told the specific steps required to solve a problem
- Copying code off the internet.

What you are allowed to do:

- Discuss examples given as handouts, from the textbook or class notes with others.
- Get together to study and help each other with basic understanding of concepts (eg. What is an object).
- I am considering Git to be a tool you are using, so feel free to help each other with usage of SourceTree and git.

Phase 0:

When you first start your project, before writing any code, you will create a new **`git`** repo in the same folder. You should ignore any **`.class`** files, or any files associated with your text editor (basically ignore every file type that appears other then **`.java`** or **`.txt`**)

Create a new local **`git`** repo, then make your first commit here, which will simply be the **`.gitignore`** file that was created with a message (such as "Initial commit" with a list of the types of files ignored.

Phase 1: Basic Objects

Create the following Object class files:

`Place.java`
`Person.java`
`Thing.java`

For now, these will all have the same structure with a **`String name`** and **`String description`**. We will customize them later. Each class should have a **`toString()`** method that returns the **`name`** and a **`constructor`** that sets both variables.

Make your second commit here. It should include a descriptive and clear message.

Phase 1b:

Refactor and comment your code. Fix any typos, ensure the indentation is neat and add commenting. Each variable, method and class should each have a description of what it is for. This description should be **CLEAR** and provide helpful context, usage notes or anything else that could be helpful later. It should NOT be the assignment requirements pasted in. Write it yourself and put some thought into it.

Make your third commit, detailing how you improved your code in the last step. Again, an appropriate message should be included.

Phase 1c. Add accessor and mutator methods. More commenting.

Add a **`getDescription`** accessor method that simply returns the description.
You should also have a **`setDescription`** method in each class that has a **`void`** return type and accepts a **`String newDescription`**. This method will replace the current **`description`** with **`newDescription`**.

Test the classes out by creating your own **`TestClass.java`** class to create Objects of each type and calling all the methods. Double check the spelling and case (methods are **`lowercase`**, classes are **`Uppercase`**) against the assignment doc.

Create a text file that includes your test output (call it **`test_output.txt`** or something equally obvious) and commit that along with your test class itself.

Phase 2: PersonList class

Since you are going to have many arrays of type **`Person`** objects, we can leverage the power of **`Objects`** by creating a **`PersonList`** class to manage groups of people. This will simply be a class that has the purpose of containing a **`private array`** of type **`Person`**. You will create methods that allow you to manipulate this **`array`** more easily.

> eg. **`Place`** will contain multiple **`Persons`**, while each **`Person`** will also contain a list of other **`Persons`**.

Create the following methods in the **`PersonList`** class.

`A constructor, accepting no parameters`

> The **`PersonList`** constructor will create the **`array`** object itself. For now, it will contain only **`null`** values. Choose some sufficiently high number (say **`100`** for now) as the length of the **`array`**. You will also add an instance variable to the class of type **`int`** that will keep track of the current number of **`Person`** objects the **`array`** is holding. Note that this is not the same as the length of the **`array`**, and this **`counter`** should be used to avoid accessing **`null`** values in the **`array`**.

`public void addPerson(Person newPerson)`

> Add a new **`Person`** in the next open spot of the **`array`** (ie if there are two **`Persons`** in index **`0`** and index **`1`**, you will add the new **`Person`** at index **`2`**). It should also update the **`counter`** (so you know how many **`Person`** objects there are).

`public int size()`

> Return the current number of **`Person`** objects currently stored in the array (not the length).

`public String toString()`

Return a single **`String`** value that includes all of the names of the **`Person`** objects stored in the list with the format. You should remove the last comma as well, just don't add any additional bugs while you do it. Specifically watch for errors when your list is either empty or contains only one Person.

[**`Name1, Name2, Name3`**]

At this point, you should compile and test your work.

When you are satisfied that it is working, commit your **`PersonList`** class, your updated test file, and add the output generated to your **`test_output.txt`** file (don't erase the old stuff, and make sure its clear which phase each is from.)

You should test it by creating some **`Person`** objects, creating a **`PersonList`**, then adding the **`Person`** objects to the **`PersonList`**, checking that the **`size`** is correct, then outputting the list as a **`String`**.

Some other things to check for:

Does your **`toString`** method work for size **`0`** and size **`1`** lists?

Don't worry about filling up the array yet, we will deal with that later.

Phase 2b:

Continuing with your **`PersonList`** class, add the following methods.

`private int findIndex(Person )`

This method returns the index of a **`Person`** if they are found, **`-1`** if they are not

Since we are looking for **`Person`** objects which will all be unique, we can compare if they are the same using (**`A == B`**) to test if they are the same instance. In this case, two **`Person`** objects with identical **`names`** and **`descriptions`** would be considered different **`Persons`**.

You may notice this method is **`private`**. This means it can only be used within this **`class`**. You may also notice that both of the following methods use variations of searching a list for a **`Person`**. Reuse your existing methods whenever possible.

At this point, test your method and commit your code (including tests and output) as before.

Phase 2c: More Methods

Add the following methods to **`PersonList`**

`boolean containsPerson(Person key)`

Iterate through the array and return true if the person is found and false if they are not.

`Person removePerson(Person key)`

Remove the **Person** from the list and return a reference to that **Person** object. All the **Person** objects stored at indexes higher than the **Person** found will need to be moved one position left (lower) in the **array** to fill the gap. You can assume (for now) that the **Person** exists in the list if it is passed as a parameter.

// ie. [A, B, C, D] // where each letter is a Person object in a given array
// Remove Person C
// Array becomes [A,B,D] with a size() of 3.

Test and commit with the same diligence as before.

Phase 2d:

Revisit your class before moving on. Fix any formatting issues (indentation), minor bugs, typos and ensure your commenting is complete. I will be marking you on the commenting included at the appropriate times and will not be looking favorably on code that does not include commenting until the very end.

Phase 3: Player Generation

Add player generation (**Person** objects with random values). To do this, add another constructor to **Person** that accepts no parameters. Calling this constructor should return a new Person with a randomly selected **name** and **description**.

You can use the **RandomGenerator** as a starting point and example. Add a new method **getRandomPersonDescription()** to the **RandomGenerator** that works the same way as **getRandomLocationName** but uses arrays of **verbs** and **nouns** that could describe **Persons**. Please keep the language appropriate and not insulting. Put some effort in to this. You should have at least **5 nouns** and **5 verbs** at a minimum (but more is better).

Note, you should NOT break or otherwise remove functionality from the **RandomGenerator** class. You should ONLY add functionality.

Again, test this method by creating **10** random **Person** objects, adding them to a **PersonList** and printing them out. Add the output to your **test_output.txt** file before proceeding and be sure to comment and format your code as per before.

Phase 4: A Place to Party

Add a private variable of **PersonList** called **people** to the **Place** class. Add the following methods to **Place**:

public String getPeople()

Return a **String** of the **Person** objects at this **Place**, formatted the same way as **PersonList** returns.

public void addPerson(Person toAdd)

Add a **Person** to the **people** list

`public boolean removePerson(Person toRemove){`

Attempt to remove the given **`Person`** from the list.
Check if **`Person`** exists in the list first.
If the **`Person`** was found, then remove them and return **`true`**, otherwise don't change anything and return **`false`**.

`public int countPeople()`

Returns a count of the current number of people at this **`Place`**.

`public boolean contains(Person toFind)`

Does this **`Place`** contain a given **`Person`**?

5. Conversations

Complete the **`mingle`**, **`createPeople`** and **`wholePartyToString`** methods in the **`PartySim`** class as described in their method comments.

To test, (in the **`PartySim`** main method) generate a **`Place`** and a **`PersonList`** with 5 random **`Person`** objects. Add all the **`Persons`** created to the **`Place`**. Keep a copy of this list around as you will need it later (for outputting all **`Persons`**).

Call **`mingle()`** on the **`Place`**, then **`remove`** one **`Person`** at random from the **`Place`**. Keep calling **`mingle,`** then removing a random **`Person`** until there is only 1 **`Person`** left at the **`Place`**. At this point the party is over.

Then, print out the complete list of all guests at the party (the original **`PersonList`**), including who each **`Person`** talked to. You will probably (definitely) want to do some additional formatting to help make this output clearer.

Make a commit including all the same best practices we have done before.

Phase 6.

Revisit all your classes, improving the commenting, formatting and fixing any last minute bugs. Include additional testing including edge cases (eg methods shouldn't crash if there are not people in the list, etc). Include a **`Readme.txt`** file that includes clear instructions to run your code, student name and ID, and any outstanding issues (ie bugs, things that were not sufficiently completed, comments.) I will be looking for the Phase 6 commit so don't skip this step.

Hand in instructions:

Place all your **`.java`** files, the **`.git`** repo and the **`.gitignore`** files in the hand-in folder on Moodle with your **`readme.txt, honesty declaration and test_output.txt`**

DO NOT:
- DO NOT place handin files in a Folder or zip file.
- DO NOT hand in **`.class`** files
- DO NOT hand in **`bluej`** files or other various editor files **`(.vs, .proj, etc).`**
- DO NOT ignore the hand in instructions.

Congratulations, you are done. Be proud of yourself and relax.

7. Bonus (Optional)

**** Not worth a lot of marks but good if you want a challenge!***

Wait, what are you still doing here? Do you want more?

a) Complete the **`Interests`** and **`Friends`** feature outlined in the **`PartySim`** comments (at the bottom)

b) Consider and suggest additional features that we could add. Include notes about how you would design the feature and what the expected output would be.

Hand In Additional Notes:

- It is especially important that your classes and methods are very accurate. Make sure your class names are **`Uppercase`** and your **`methodNames`** are lowercase and don't contain typos. **If your program doesn't compile for the marker you will lose significant marks**, even if it works on your machine.
- Also be sure you remove any package headers added by IntelliJ or Eclipse etc as these will break compilation.
- See the **`SubmissionGuidelines.pdf`** document for full instructions. See the **`Standards.java`** document for formatting guidelines.
- Include a **`README.txt`** file when you hand in the assignment to tell the marker which phases were completed, so that only the appropriate test can be run. This can also include messages about errors you are encountering.
- For example, if you say that you completed Phases 1-3, then the marker will compile your files to check for completion of those phases. If it fails to compile and run, you will lose all of the marks for the test runs (at least half of the assignment). The marker will not try to run anything else, and will not edit your files in any way.
- Submit a completed Honesty Declaration with EACH assignment. Your assignment will not be marked without it. I will provide an easy text based version you can use so you don't have to mess around signing PDFs.

Good luck! Don't Panic!