Lab 05: West Hall Simulator

COSC 102 - Spring '22

In this lab you will gain more practice with fixed-length arrays as well as several object-oriented programming concepts: inheritance, abstract classes, and polymorphism.

1 Overview

Inspired by our very own West Hall, which was built by students and faculty in 1827 using stones from Colgate's own quarry, in this lab you will design, implement, and test classes that simulate the construction of a building. Each construction project will have a certain number of square feet which need to be built by a collection of hired workers. Your simulation will determine if a given project's construction can be finished by a specified deadline.

1.1 Simulation Details

Below is an overview of the terminology and mechanics of the construction project, workers, and simulation:

The Construction Project

- Each construction project requires a number of **square feet** to built by a **deadline** (*i.e.* number of working days).
- Any number of **workers** are hired to a project: three different worker types collaborate in the construction.
- If the required square feet are built by the **deadline**, the project is determined to be a **success**; otherwise, it is evaluated as a **failure**.

The Workers

- There are **three** types of workers:
  1. **Wood Workers** harvest *logs*.
  2. **Stone Workers** mine *stones*.
  3. **Construction Workers** use *logs* and *stones* to build your residence hall.
- All workers have:
  - a **name** and **ID number**. The ID numbers are serialized; the serialization is shared between all workers
  - a particular **aptitude** which determines how much they can contribute in their respective role each day
  - a daily **mood** that also factors into their productivity.

The Simulation

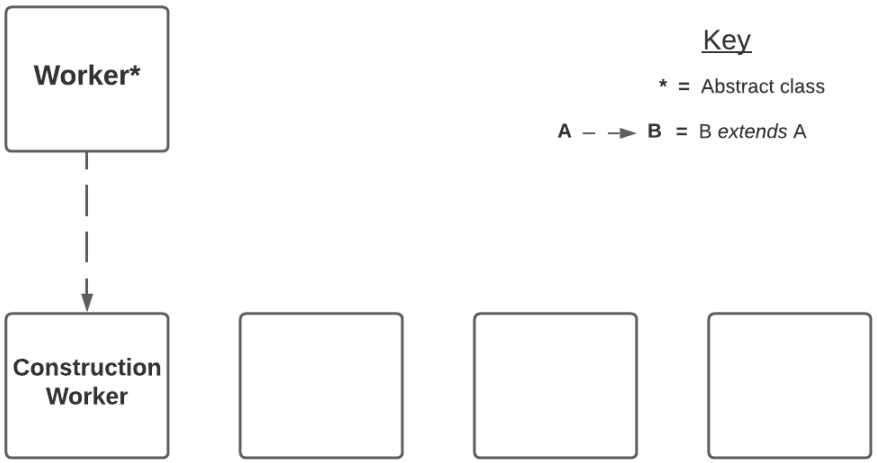
- The simulation executes one day at a time – each day all workers hired to the project contribute based on their role and the resources available. Likewise, each worker's mood gets updated at the start of each day.
- While each day construction workers can cumulatively build a number of square feet of the construction, they require stone and wood to do so, and thus are constrained by the available resources.
- For each **square foot** of the construction, construction workers require and consume **one log** and **one stone**.
- The project maintains a **stash** holding the unused **logs** and **stones**. Each day wood workers and stone workers add their day's work to the stash for the construction workers to use in the following days.
- Construction workers need logs and stones available to be fully productive – their progress depends on materials being available. The stash **only gets updated at the end of the day**, meaning construction workers cannot use resources the same day they are gathered.

2 Your Task

Your task is to design and implement the functionalities of this construction simulation. You are provided with a completed implementation of the `Worker` class; you will write the rest of the classes on your own.

Your program will require **five** classes: **`Worker`**, **`WoodWorker`**, **`StoneWorker`**, **`ConstructionWorker`**, and **`Project`**.

Provided below is an incomplete inheritance diagram of this program – complete it. Fill in the remaining classes in the boxes, draw lines indicate any inheritance, and use asterisks to mark any class(es) which should be abstract.



***** Check your work with an instructor or tutor before continuing *****

2.1 Worker

The `Worker` class contains attributes and functionality common to all types of workers on the project, such as name, ID number and more. The first `Worker` is always assigned ID number **`3000`**, the next assigned **`3001`** (regardless of worker type), and so on.

`Worker` also maintains a `currentMood`, which can be one of three states: *tired*, *normal*, or *pumped*. A workers' mood is updated on each day of the simulation, and one of these three values is randomly chosen.

You must implement the `equals` method in `Worker`. Aside from this, you **cannot modify `Worker`**. Your first step will be to review this class' implementation and methods, thinking about how to utilize them in its children.

2.2 Wood Worker

`WoodWorker` is a particular type of `Worker` who can harvest a certain number of logs in a given day. Each work day a wood worker will harvest a number of logs according to their average and mood.

Specifically, a wood worker:

- harvests **75%** of their average if feeling **tired**
- harvests **100%** of their average if feeling **normal**
- harvests **125%** of their average if feeling **pumped**

Thus, `work()` returns the number of logs the referenced wood worker can log based on their current mood. You must also implement a `toString()` method that returns a wood worker's pertinent information (more on this below).

Implement the `WoodWorker` class one step at a time. Add a `main` method to this class to test each method as you develop it. You should minimally make sure that:

- a `WoodWorker` object is instantiated from the following arguments **in this order**: a name and an average number of logs they can harvest in a day. The minimum average logs per day for a `WoodWorker` is **50**; if an invalid average is provided, 50 should be used instead.
- `work()` returns the number of logs the `WoodWorker` can harvest per their mood.
- `toString()` returns a String with the `WoodWorker`'s job, name, ID, current mood, and average logs/day. Ex:

```
"[Wood Worker] Freya (#3003), currently feeling pumped! (avg logs/day: 200)"
```

2.3 Stone Worker

`StoneWorker` is another type of `Worker`. Similar to wood workers, they have an average number of stones they can mine per day, but their mood affects their production differently – specifically, a stone worker:

- mines **50%** of their average if feeling **tired**
- mines **100%** of their average if feeling **normal**
- mines **150%** of their average if feeling **pumped**

The minimum average stones per day for a `StoneWorker` is **40**. Like `WoodWorker`, your `StoneWorker` class should also implement a `toString`, and you should create a `main` method to test your code as you implement it.

2.4 Construction Worker

`ConstructionWorker` is the last type of `Worker`. Instead of an average square feet they can build per day, construction workers have **three** values: a `low`, `medium`, and `high`. These values directly correspond to the number of square feet a they can build in a day if their mood is *tired*, *normal*, or *pumped*, respectively. A few final notes:

- a `ConstructionWorker` object is instantiated from the following arguments **in this order**: a name, and a low, medium, and high number of square feet they can build in a day. There is no minimum for these attributes, but if **any** of the values are negative or out of order, the low/medium/high are all set to defaults (100/200/300).
- The number of square feet a `ConstructionWorker` builds in a day is constrained by available resources. Think of the number returned from `work()` as the square feet they *could potentially* build, if not restricted by resources.
- `ConstructionWorker` will also implement a `toString()` method, similar to `WoodWorker` and `StoneWorker`.

2.5 Project

The `Project` class represents a construction project, and contains the method to run a simulation to determine the viability of completing a building by the specified deadline. Each project has a number of square feet, a collection of workers, and a duration.

Your `Project` class must have **at least** the following:

- A **constructor** accepting four arguments **in this order**: the project name, required square feet, duration (in days) and the maximum number of workers that can be assigned to it. If any arguments are invalid, assign them to a defaults of your own discretion.
- the method **`assignWorker(Worker w)`** which assigns (*ie. "hires"*) a worker to the project (if possible) and **returns** a boolean indicating whether or not the worker was successfully hired.

  `assignWorker` should return false if a `null` argument is provided. There's two other scenarios where this method returns false – think about it! *(the method you implemented in 2.1 will be handy for one of the scenarios!)*
- the method **`unassignWorker(Worker w)`** which unassigns (*ie. "un-hires"*) the argument worker from the project, **returning a boolean** indicating whether or not the worker was successfully unassigned.

- the method **`simulateConstruction()`** which simulates the project to determine the viability of completing the construction on time. Returns **`true`** if the simulated project completes by the deadline, otherwise **`false`**.

  Re-read **section 1.1** for an overview how the simulation functions, then read the additional details below:

  - Each day, all workers contribute to the project based on their role, attributes, and mood. Each worker's mood is **updated at the start of each day** – the new mood is chosen randomly and may stay the same.
  - Logs and stones generated each day are added to the **stash**, but only **at the end of the day** (and thus are not available until the next day). At the start of the simulation, the stash is empty.
  - Construction workers build however many square feet they are able to based on their mood, attributes, and the available resources. For example, if a construction worker is capable of building 200 square feet on a given day, but the stash only contains 146 logs and 181 stones, they are only able to build **146 square feet**.
  - Again, each square foot requires and thus consumes **one log** and **one stone** from the stash.
  - If the required number of square feet are built, the simulation **ends as soon as the day is over** and is considered a success. If the deadline is reached without the required square footage built, the simulation fails. The simulation does not run past the deadline, and it's ok to go over the target square footage.

  In addition to returning a `boolean`, `simulateConstruction()` prints output of its progress as it is running, which must minimally contain the following:

  - At the start of each day, the current stash totals and square feet remaining to build
  - For each day, each employee's info (role, name, ID, current mood, and attributes) and work performed
  - At the end of each day, a summary of that day's resources gained and square footage built
  - At the end of the simulation, text indicating whether the project was completed on time or not

3 Sample Simulation

Below is an example of output generated from a sample simulation. The project has the following attributes:

- project name of **`"Gladsheim"`**
- **`600`** square feet to build
- **`6`** days to complete the project

Additionally, the following workers were created and assigned to the project:

- Odin, a `WoodWorker`, with an average of **`200`** logs per day
- Freya, a `StoneWorker`, with an average of **`300`** stones per day
- Baldur, a `WoodWorker` with an average of **`150`** logs per day
- Idunn, a `ConstructionWorker`, with a low/medium/high of **`300/350/400`** square feet per day
- Loki, a `ConstructionWorker`, with a low/medium/high of **`150/175/500`** square feet per day

Example output:

```
Beginning "Gladsheim" simulation...

*** Day 1 ***
     --- Starting Totals --- Stash: 0 logs and 0 stone, 600sq feet left to build!
     [Wood Worker] Odin, ID: #3000 is feeling pumped! (avg logs/day: 200)
          ...logged 250 logs!
     [Stone Worker] Freya, ID: #3001 is feeling normal (avg stones/day: 300)
          ...mined 300 stones!
```

```
     [Wood Worker] Baldur, ID: #3002 is feeling tired (avg logs/day: 150)
          ...logged 112 logs!
     [Construction Worker] Idunn, ID: #3003 is feeling tired (sq ft/day: 300/350/400)
          ...built 0 square feet!
     [Construction Worker] Loki, ID: #3004 is feeling tired (sq ft/day: 150/175/500)
          ...built 0 square feet!
     --- Day Totals --- 362 logs and 300 stone gained, 0 square feet built!

*** Day 2 ***
     --- Starting Totals --- Stash: 362 logs and 300 stone, 600sq feet left to build!
     [Wood Worker] Odin, ID: #3000 is feeling normal (avg logs/day: 200)
          ...logged 200 logs!
     [Stone Worker] Freya, ID: #3001 is feeling normal (avg stones/day: 300)
          ...mined 300 stones!
     [Wood Worker] Baldur, ID: #3002 is feeling pumped! (avg logs/day: 150)
          ...logged 187 logs!
     [Construction Worker] Idunn, ID: #3003 is feeling tired (sq ft/day: 300/350/400)
          ...built 300 square feet!
     [Construction Worker] Loki, ID: #3004 is feeling pumped! (sq ft/day: 150/175/500)
          ...built 0 square feet!
     --- Day Totals --- 387 logs and 300 stone gained, 300 square feet built!

*** Day 3 ***
     --- Starting Totals --- Stash: 449 logs and 300 stone, 300sq feet left to build!
     [Wood Worker] Odin, ID: #3000 is feeling pumped! (avg logs/day: 200)
          ...logged 250 logs!
     [Stone Worker] Freya, ID: #3001 is feeling normal (avg stones/day: 300)
          ...mined 300 stones!
     [Wood Worker] Baldur, ID: #3002 is feeling normal (avg logs/day: 150)
          ...logged 150 logs!
     [Construction Worker] Idunn, ID: #3003 is feeling pumped! (sq ft/day: 300/350/400)
          ...built 300 square feet!
     [Construction Worker] Loki, ID: #3004 is feeling pumped! (sq ft/day: 150/175/500)
          ...built 0 square feet!
     --- Day Totals --- 400 logs and 300 stone gained, 300 square feet built!

"Gladsheim" completed successfully after 3 days!
```

4 Submission

Upload your completed code's `.java` files to the **Lab 05** submission link on your lab Moodle or Google Classroom page. You will have **5 `.java`** files in total to upload.

This assignment is due:

- **Tuesday, March 8th** by **10:00PM** for lab sections **A**, **B**, **C**, and **E** (which meet on Wednesday)
- **Wednesday, March 9th** by **10:00PM** for lab section **D** (which meets on Thursday)