Homework 03 – New Devices

Authors: Vinayak, Divya, Harry, Helen, Ricky

Topics: abstract classes, overriding, equals, toString

Problem Description

Your group of friends has decided to start a company where you all will develop proprietary devices that will be sold to the public. In doing so, you must figure out how to manage the specifications of each of your devices.

Solution Description

For this assignment, you will create five classes: Task.java, Device.java, CellPhone.java, Laptop.java, and Driver.java.

Note:

1. *All fields should be **properly encapsulated** and must require an instance to be accessed through, unless specified otherwise. Fields in a superclass should have the most restrictive visibility that is possible for encapsulation within the hierarchy.*
2. *All **required** methods should be accessible from everywhere and must require an instance to be accessed through, unless specified otherwise.*
3. *Use constructor and method chaining whenever possible! Ensure that your constructor chaining helps you reduce code reusage!*
4. *None of the **required** fields should have getters and setters, except for one getter in Task.java. It is possible to complete this homework **without the need for ANY more getters and setters** in these classes (except for one). In other words, **DO NOT** write **ANY** getters or setters for the **required** fields listed for each class, unless specified otherwise.*
5. *Reuse code when possible and helper methods are optional.*
    - *If you write helper methods, they should have the most restrictive visibility modifier for the purpose of that method. For instance, if no subclasses need to use a helper method, then don't make it accessible from outside that class. If subclasses will use that method, you can make it protected.*
6. *Make sure to add all Javadoc comments to your methods and classes!*

Task.java

This properly encapsulated class is a representation of a processing task that a Device can handle. Task ***cannot*** be inherited from (i.e. subclassed). Tasks have a name and associated CPU cost.

Variables:

- `name`
    - A `String` that is a description of the task.
    - This variable should be **immutable** after construction of an instance.
- `cpuCost`
    - An `int` representing the cost of processing power for this task.
    - This variable should be **immutable** after construction of an instance.

Constructor(s):

- A constructor that takes in the `name` and `cpuCost`.
- If the name is `null`, the task name should be set to "`GEN_TASK`"
- The value of the CPU cost should always be at least 8. If one attempts to initialize a Task with a `cpuCost` less than 8, set the `cpuCost` to 8.

Methods:

- `equals`
    - Overrides from `Object`.
    - Two tasks are equal if they have the same `name` and `cpuCost`.
- `toString`
    - Overrides from `Object`.
    - Returns the String:

      ```
      "<name> has CPU cost of <cpuCost>"
      ```
- A getter method for `cpuCost`
    - Note: This is the ONLY getter method allowed in this homework assignment.

Device.java

This class is a representation of a device your company has developed. Device should **never** be instantiated, but specific implementations of a Device should inherit functionality from this class. Devices will process Tasks, keeping a list and ensuring that the CPU cost of the tasks are within bounds.

Variables:

- `serialNumber`
    - An `int` that is a unique identifier for the device.
    - This variable should be **immutable** after construction of an instance.
- `cpuCapacity`
    - An `int` representing the total amount of processing power for the Device.
    - This variable should be **immutable** after construction of an instance.
    - This variable should be visible to the subclasses of Device.
- `cpuRemaining`
    - An `int` representing the amount of processing power left on the Device, based on the pending tasks.
    - This variable should be visible to the subclasses of Device.
- `tasks`
    - An array of `Tasks` representing the Tasks that are currently being held by the Device and have not yet been processed.
    - This variable should be visible to subclasses of Device.

Constructor(s):

- A constructor that takes in the `serialNumber, cpuCapacity`, and an int `length` of the `tasks` array. It should create a new empty array for `tasks` with a length equal to the `length` given by the argument in the constructor. `cpuRemaining` should be initialized according to the `cpuCapacity`.

- A constructor that takes in `serialNumber` and the length of the `tasks` array. It should use a default value of 512 for `cpuCapacity`.
- Assume all values passed in are valid.

Methods:

- `canAddTask`
    - Takes in a `Task` object and returns a `boolean` of whether the task can be added to the `tasks` array.
    - This method will **not** have an implementation in the Device class.
      HINT: What keyword allows us to declare a method in a class without providing an implementation?
- `addTask`
    - Takes in a `Task` object. This method will add the task to the `tasks` array.
    - Returns `true` if the task is successfully added to the `tasks` array, and `false` otherwise.
    - This method will **not** have an implementation in the Device class.
      HINT: What keyword allows us to declare a method in a class without providing an implementation?
- `processTask`
    - Takes in a `Task` object and returns a `boolean` of whether the task was processed successfully.
        - If the input task is `null`, return `false`
    - Processing a task means it must be removed from the `tasks` array.
        - Remove the first occurrence of the task in the array that is equal to the one passed in, starting from index 0.
            - If an equal task is found in the `tasks` array, remove it from `tasks` by setting that index to `null`, and return `true`.
        - If an equal task is not found, return `false`.
        - Processing a task frees up the CPU cost of the Task from the CPU remaining on the device. So, before returning, add the cost of the task to CPU remaining and print out the following:

          ```
          "Processed: <task's toString>"
          ```
    - **HINT:** If necessary, this method can be overridden in the child classes of Device.
- `equals`
    - Overrides from `Object`.
    - Two devices are equal if they have the same `serialNumber`, `cpuCapacity`, and `cpuRemaining`.
      HINT: Even though we can't instantiate Device, this method will be useful in the subclasses.
- `toString`
    - Overrides from `Object.`
    - Returns the String:

      ```
      "Device with serial number <serialNumber> has
      <cpuRemaining> of <cpuCapacity> CPU remaining."
      ```

CellPhone.java

This class represents a cell phone created by your company.

Variables:

- `tasksCompleted`
  - An `int` representing the total number of tasks the cell phone has completed.
  - A task is completed when a task is successfully processed and removed from the `tasks` array
    - **HINT:** Think carefully about how this field will need to be modified. It's only visible in ***this*** class, but `processTask` is defined in the parent `Device` class.
  - This variable should always be initialized to 0.

Constructor(s):

- A constructor that takes the `serialNumber`, `cpuCapacity`, and an int `length` of the `tasks` array
- A constructor that takes the `serialNumber`, `cpuCapacity`, and defaults the `tasks` array to an array of length 10
- Assume all values passed in are valid

Method(s):

- `canAddTask`
  - Takes in a `Task` object and returns a `boolean` of whether the task can be added to the `tasks` array.
  - The task can be added to the `tasks` array if both conditions are satisfied:
    - There is a slot in the `tasks` array that is empty (`null`)
    - There is enough CPU remaining to cover the cost of this task
- `addTask`
  - Takes in a `Task` object. This method will add the task to the `tasks` array.
  - Returns `true` if the task is successfully added to the `tasks` array, and `false` otherwise.
    - `canAddTask` must be used to ensure that the cell phone has an empty slot and enough CPU remaining.
    - If the task can be added to the `tasks` array, add the task at the first available empty slot, starting from index 0.
  - Update `cpuRemaining` accordingly.
- `equals`
  - Overrides from `Device`.
  - Two cell phones are equal if they have the same `serialNumber`, `cpuCapacity`, `cpuRemaining`, and `tasksCompleted`.
- `toString`
  - Overrides from `Device`.
  - Returns the String:

```
"Device with serial number <serialNumber> has
<cpuRemaining> of <cpuCapacity> CPU remaining. It has
completed <tasksCompleted> tasks."
```

- ***Reuse code if possible***

Laptop.java

This class represents a laptop computer created by your company.

Variables:

- `overclockable`
  - A boolean representing whether the laptop has the ability to temporarily increase its CPU remaining.

Constructor(s):

- A constructor that takes the `serialNumber`, `cpuCapacity`, the length of the `tasks` array, and whether it is `overclockable`.
- A constructor that takes in the `serialNumber`, `cpuCapacity`, and the length of the `tasks` array, and defaults `overclockable` to `false`.
- Assume all values passed in are valid.

Method(s):

- `bufferSlotsRequired`
  - This method calculates how many slots should be left open in the `tasks` array to ensure that the instance of `Laptop` is able to function optimally while completing tasks.
  - Takes in an `int` representing the amount of CPU remaining that should be used to calculate the number of buffer slots required and returns the calculated value as an `int`.
    - ~ This is dependent on the `tasks` array and input CPU remaining.
    - ~ If the length of the `tasks` array is less than or equal to 4, return 0 buffer slots regardless of the value of the input `cpuRemaining`.
    - ~ If the input `cpuRemaining` is less than 128, then the required number of buffer slots should be 2. For any other value of the input `cpuRemaining`, the required number of should be 1.
- `canAddTask`
  - Takes in a `Task` object and returns a `boolean` of whether the task can be added to the `tasks` array.
  - The task can be added to the `tasks` array if both conditions are satisfied:
    - ~ There is enough CPU remaining to cover the cost of this task, after checking for overclocking
      - ∗ If there is not enough CPU remaining and the laptop's CPU can be overclocked, check if the task can be completed if the `cpuRemaining` is increased by one-fourth of the `cpuCapacity` (use integer division).
    - ~ There are enough empty slots (i.e. elements that contain `null`) in `tasks` ***after*** adding this task to satisfy the minimum number of buffer slots required
      - ∗ i.e. if the number of required buffer slots is two and there are three slots left, only one more task can be added
  - **IMPORTANT:** This method ***should not*** change the state of this object. The state of the object should only be updated once the task is added.
- `addTask`
  - Takes in a `Task` object. This method will add the task to the `tasks` array.

- Returns `true` if the task is successfully added to the `tasks` array, and `false` otherwise.
  - `canAddTask` must be used to ensure that an empty slot exists in the `tasks` array and that there is enough CPU and buffer slots remaining.
  - If the task can be added to the `tasks` array, add the task at the first available empty slot, starting from index 0.
- Update `cpuRemaining`, and `overclockable` accordingly.
  - Once the laptop's CPU has been overclocked, ensure that it cannot be overclocked again

- `equals`
  - Overrides from `Device`.
  - Two laptops are equal if they have the same `serialNumber`, `cpuCapacity`, `cpuRemaining`, and if the value of `overclockable` is the same.
- `toString`
  - Overrides from `Device`.
  - Returns the String:

```
"Device with serial number <serialNumber> has
<cpuRemaining> of <cpuCapacity> CPU remaining. This laptop
<does/does not> have overclocking."
```

- ***Reuse code if possible***

Driver.java

This Java file is a test driver, meaning it will use all of the above classes! Use this class's main method to test your code. Here are some suggestions to help you get started with testing.

Method(s):

- main
  - Create 2 CellPhone objects
  - Use addTask() on at least one of the CellPhone objects
  - Use processTask() on at least one of the CellPhone objects
  - Call toString() on at least one of the CellPhone objects
  - Check to see if the 2 CellPhone objects are equal
  - Create 2 Laptop objects
  - Use addTask() on at least one of the Laptop objects
  - Use processTask() on at least one of the Laptop objects
  - Call toString() on at least one of the Laptop objects
  - Check to see if the 2 Laptop objects are equal
  - These tests and the ones on Gradescope are by no means comprehensive, so be sure to create your own!

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `Task.java`
- `Device.java`
- `CellPhone.java`
- `Laptop.java`
- `Driver.java`

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder test are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work.** Autograder tests are **NOT** guaranteed to be released when the assignment is released, so ***YOU*** are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via our class discussion forum for clarification.
You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your latest submission. **Be sure to submit every file each time you resubmit**.

Checkstyle

You must run Checkstyle on your submission (To learn more about Checkstyle, check out cs1331-style-guide.pdf under CheckStyle Resources in the Modules section of Canvas.) **The Checkstyle deduction cap for this assignment is 20 points.** If you do not have Checkstyle yet, download it from Canvas -> Modules -> CheckStyle Resources -> checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run Checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar yourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the Checkstyle cap mentioned in the Rubric section). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!
Additionally, you must Javadoc your code.

Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

For additional help with Checkstyle see the CS 1331 Style Guide.

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g. non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or Checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.

Burden of Testing

You are responsible for thoroughly testing your submission against the written requirements to ensure you have fulfilled the requirements of this assignment.

Be **very careful** to note the way in which text output is formatted and spelled. Minor discrepancies could result in failed autograder cases.

If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

Allowed Imports

To prevent trivialization of the assignment, you are not allowed to import any classes or packages.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our auto grader. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`
- `System.arraycopy`

Collaboration

Only discussion of the Homework (HW) at a conceptual high level is allowed. You can discuss course concepts and HW assignments broadly, that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files

- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- **Check on Ed Discussion for a note containing all official clarifications and sample outputs**

It is expected that everyone will follow the Student-Faculty Expectations document, and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. **No inappropriate language is to be used, and any assignment, deemed by the professor, to contain inappropriate, offensive language or threats will get a zero**. You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.