Honour Code

1. Do not copy the answers from any source.
2. You may not receive or share any part of your code with anyone other than your partner.
3. Cheating students will face the Committee on Discipline.

Assignment Details

In this assignment you will implement a general-purpose min-heap data structure. The heap will be used to help the company Molt Delivery keep track of orders and delivery drivers.

The project consists of three Java classes and some Java Test classes.

1. `MoltOrder.java` represents a single order made by a customer. Each Object stores details about what is being ordered from Molt Delivery, the customer's name, how long it takes to drive to the customer, etc..
2. `MoltDriver.java` represents a Molt Delivery driver. Each Object stores the driver's name, a unique ID (i.e., an employee number), how many orders they have delivered, and the time at which they will be available to perform another delivery (if they are currently delivering an order, then this time will be in the future).
3. `GeneralPurposeHeap.java` is not specific to the Molt Delivery simulation. It is a generic heap capable of storing any Object with one restriction: the heap can only store Comparable objects. A basic understanding of Generics in Java will be necessary. Both `MoltOrder` and `MoltDriver` are Comparable classes since they implement the Comparable interface. This means the GeneralPurposeHeap will be able to store those objects.
4. **The test classes** provided in the test folder are extremely basic. They only check that the mandatory public methods exist in your Java files. You are strongly encouraged to add your own tests to these files! The importance of having tests for your code cannot be overstated, they will help you find bugs/mistakes you didn't think were possible (unfortunately, I'm speaking from experience). While you won't submit your tests to us, we urge you to write tests. Try to write enough tests to cover all the features of the project, and make sure you check edge/corner cases. **However**, you are not required to use tests at all, so long as your solution adheres to the requirements.

Recommended: Create Project in Intellij IDEA (not mandatory)

There are many IDEs that work with Maven, you can write the code in your preferred IDE. Intellij IDEA can be downloaded freely on (almost) any platform, we recommend using it for this homework. The following instructions should get your project ready in IDEA, but the specific details can vary by platform/version. If you Google "Intellij IDEA import existing maven project", you can find guides for your specific operating system.

1. Extract `dshw02.zip` to a folder called `dshw02`.
2. In IDEA, create a new "Project from existing sources", and choose the folder `dshw02`.
3. Select the option "Import project from external model". Select "Maven" from the list.
4. Click "Create".

The Java files you write should be placed in the (currently empty) folder `dshw02/main/java`

Intellij provides you the ability to run the (basic) tests we provided, and can help you write cleaner code, detect bugs, etc.. This is a good opportunity to get familiar with its features!

Class Descriptions

A full list of each class's required public methods can be found at the end of this PDF.
This sections lists constraints/requirements which your code must follow *exactly*.
We intentionally do not provide you with skeleton Java files for these classes because it is important that you gain experience in setting up a project from scratch.

MoltOrder (`MoltOrder implements Comparable<MoltOrder>`)

This class implements the interface `Comparable<MoltOrder>`. This means it is required to have a function `compareTo` which takes another `MoltOrder` as a parameter, and determines which of the two `MoltOrder`s should be handled before the other. The `MoltOrder` whose `priority` value is lower should be handled first, and should thus be considered the smaller one by the `compareTo` function. If you are not familiar with how the return value of `compareTo` represents which object is smaller, you should read into the topic before implementing that part of the code.

MoltDriver (`MoltDriver implements Comparable<MoltDriver>`)

This class implements the interface `Comparable<MoltDriver>`. This means it is required to have a function `compareTo` which takes another `MoltDriver` as a parameter. The same instructions apply to `MoltDriver`'s `compareTo` as the corresponding section of `MoltOrder` above.

To compare two `MoltDrivers`, simply compare their `nextAvailableTimeForDelivery` values (this gets passed to the constructor).

GeneralPurposeHeap

As you can see in the code listing towards the end of this PDF, GeneralPurposeHeap has three constructors for you to implement. They each serve a different purpose to users.

- Constructor taking no parameters. Creates a heap with a sensible default capacity ("sensible" is subjective; use your best judgement).
- Constructor taking a single parameter, an int "initialCapacity" which defines the initial number of elements the heap can store before resizing.
- Constructor taking a single parameter, an array "initialData". If you use "T" as the type variable of `GeneralPurposeHeap` then the type of this parameter should be `T[]`.
  (If this description doesn't make sense to you, consider reading about *Generic Types* in Java).
  Important note: your implementation *must* run in linear time (linear in the size of the resulting heap).

Take particular care when implementing `mergeHeap`. This method takes a second heap as a parameter and "merges" that heap's elements into the current heap. I.e., after merging `otherHeap` and the heap represented by `this`:

- `otherHeap` has not been altered in any way,
- `this` contains the union of the elements of `this` (before the merge) and of `otherHeap`
- `this` still satisfies the heap property

Important note: your implementation *must* run in linear time (linear in the size of the resulting heap).

Insert into a Heap which is full should not cause an error. The Heap should grow in capacity to hold more elements, then insert the element as normal.

deleteMin/findMin on an empty heap should throw an exception.

Misc. Guidelines

- You may need to read about *Generic Types*, the *Comparable* interface, and some other topics of Java. Generics and Interfaces are used in (almost) every non-trivial Java codebase, so it's worth understanding them well.
- In this project, time is represented as an integer. Values should be positive. Values represent a number of minutes since Molt Delivery opened today. E.g., Molt Delivery opens at time 0; time 30 is half an hour after Molt Delivery opened, etc.. If this representation seems unrealistically simplistic, "Unix time" is an interesting topic worth reading when you're bored.
- You are allowed to:
  - Add new private functions to help you (and to keep the code clean and readable).
  - Add new private members/fields to classes if you find it convenient.
- You are not allowed to:
  - Remove or modify the signature of public functions in any class.
  - Rename classes or public members/fields.
  - Import anything unnecessary that might defeat the purpose of implementing a heap, the official solution has no imports.
- You will need to add some fields we do not mention in order to support certain functions of the three classes. This is both necessary and encouraged.

Submission instructions

The assignment may be submitted in pairs, this is not mandatory but it is recommended.

A jar file `submission_check.jar` is included with this assignment. You can tell if your submission (zip file) is properly named and formatted by running the following in a terminal/shell:

```
java -jar submission_check.jar example/path/123456_Jon-Simon.zip
```

Take the time to go over your code and ensure it is understandable, not excessively long, and well formatted. With a little thought, implementations can be kept reasonably short. Deviations from these guidelines will result in a point penalty.

Ensure your code compiles and runs. **Code which does not compile will not be graded.** Submit a single zip file containing only the following files (no additional files, folders, etc.):

- `GeneralPurposeHeap.java`
- `MoltOrder.java`
- `MoltDriver.java`

The zip should be named according to the following format.
If submitting alone: `ID_FIRSTNAME-LASTNAME.zip`
E.g.: `1234_John-Doe.zip`

If submitting as a pair: `ID1_FIRSTNAME1-LASTNAME1_ID2_FIRSTNAME2-LASTNAME2.zip`
E.g.: `1234_John-Doe_5678_Jane-Doe.zip`

Note: If your name consists of more than two parts, you can follow the below examples: `James-T-Kirk`, `Jean-Luc-Picard`, `Albus-Percival-Wulfric-Brian-Dumbledore`, etc.

Deadline

Submit no later than the 3rd of May. You may get an automatic extension up to the 7th of May.

Function Signatures

Listing 1: GeneralPurposeHeap's public methods/constructors

```java
public GeneralPurposeHeap();

public GeneralPurposeHeap(int initialCapacity);

public GeneralPurposeHeap(T[] initialData);

public void insert(T element);

public T findMin();

public int getSize();

public T deleteMin();

public void mergeHeap(GeneralPurposeHeap<T> otherHeap);
```

Listing 2: MoltDriver's public methods/constructors

```java
public MoltDriver(int id, String name, int nextAvailableTimeForDelivery);

public void incrementTotalOrdersDelivered();

public int getNextAvailableTimeForDelivery();

public void setNextAvailableTimeForDelivery(int time);

public String getName();

public String toString();

public int compareTo(MoltDriver otherDriver);
```

Listing 3: MoltOrder's public methods/constructors

```java
public MoltOrder(String name, String orderDescription, int orderReadyTime, int
    timeNeededToDeliver, int priority);

public String toString();

public int getOrderReadyTime();

public int getTimeNeededToDeliver();

public String getName();

public String getOrderDescription();

public int compareTo(MoltOrder otherOrder);
```