**Name:** Isaiah Hoffer **Select Section:** MW(1:40) TR (1:40) **Spring 2025**

**Worksheet #1: Stacks Worksheet #1 Points: \_\_\_\_\_/17**

Download and import the file **Worksheet1.java.** Use the file to answer the following questions in the worksheet.

Please **write your answers on the worksheet.** Add code to the file to see if it works but do not turn in the code.

1. **Worksheet1.java** contains the generic stack class from assignment 5 used to create stacks of integers & strings. Let’s now create a stack of planes and a stack of semi-trucks. **List any changes**, if any are needed, to the generic class to put planes or semi-trucks objects on the generic stack. Provide a **short reason for your answer**. **(2 pts)**

No change is needed to make a stack for planes and semi-trucks because GenericStackW1 excepts any type E which planes and semi-trucks are an any type E.

1. Note that **Worksheet1.java** also contains a **Plane** and **SemiTruck** class. **(2 pts)**
   1. In main, find the comment “*Worksheet1 Question #2*”
   2. At this point, what code creates a generic stack of planes called **planeStack** and a generic stack of trucks called **truckStack**? Write that code here:

GenericStackW1<PlaneW1> planeStack = new GenericStackW1<>();

GenericStackW1<SemiTruckW1> truckStack = new GenericStackW1<>();

1. What code is needed to add a new plane object to the **planeStack**? **(2 pts)**
2. In main, find the comment “*Worksheet1 Question #3*”
3. At this point, what code will add a plane to the stack? Write that code here:

planeStack.push(plane);

1. What code is needed to print **ONLY** the capacity of each plane in **planeStack**? **(7 pts)**
2. In main, find the comment “*Worksheet1 Question #4*”
3. At this point, write a **while loop** to print the capacity of each plane.
4. Note, the stack will be empty when the code completes; don’t restore it. Write that code here:

//Displays Plane Capacity

while(!planeStack.isEmpty()) {

System.***out***.printf("Capacity: %10.2f\n", planeStack.pop().getCapacity());

}//While

1. Next, obtain the plane on the top of the **planeStack**. **(4 pts)**
   1. In main, find the comment “*Worksheet1 Question #5*”
   2. At this point, add 1 line of code to access the plane on the top of stack, without removing it, and store the plane in a variable. Write that code here:

PlaneW1 topOfStack = planeStack.pop();

* 1. An out of bounds error occurs when you run the code in 5b. Why does this occur?

It gives an out of bounds error because the pop method is trying to take the last element of the stack, but because we removed each element with the while loop in part 4 the stack in now empty and size is 0 and the code is made to take the last element, so size – 1, but 0 – 1 is -1 and that cannot be an index for an array.

**Worksheet #2: Nested Objects Worksheet #2 Points: \_\_\_\_\_/18**

Download and import the file **Worksheet2.java.** Use the file to answer the following questions.

1. Create a queue in main called **queueNotNested**. This queue is only in main and not nested inside a class.

In main, find the comment “*Worksheet2 Question #1*”

* 1. At this point, write the declaration for a queue of integers and place the values 68, 8, 17, 47 into this queue using the ***offer*** method. Write that code here: **(3 pts)**

Queue<Integer> queueNotNested = new LinkedList<>();

queueNotNested.offer(68);

queueNotNested.offer(8);

queueNotNested.offer(17);

queueNotNested.offer(47);

1. Next, place a queue **inside a class** called**IntegerQueue.** Complete the class below by writing on the worksheet the code required for each method. Find *Worksheet2 Question #2*. **(6 pts)**

**class** IntegerQueue {

**private** Queue<Integer> numberQueue = **new** LinkedList<>();

**public** **int** size() {

return numberQueue.size();

}

**public** **void** offer(Integer value) {

numberQueue.offer(value);

}

**public** Integer remove() {

return numberQueue.remove();

}

} // IntegerQueue

1. Finally, test the **IntegerQueue** class using the code in **Worksheet2.java**
   1. Write the declaration for an object of type **IntegerQueue** and place the integer values 68, 8, 17, 47 into the object (that acts like a queue!). Find comment *Worksheet2 Question #3a.* **(3pts)**

**IntegerQueueW2 queueNested = new IntegerQueueW2();**

**//Adding Values**

**queueNested.offer(68);**

**queueNested.offer(8);**

**queueNested.offer(17);**

**queueNested.offer(47);**

* 1. Write the code to print all integers in the **IntegerQueue** object. Do not hardcode the queue’s size. *Worksheet2 Question #3b.* **(6 pts)**

System.***out***.println();

System.***out***.println("Values in a queue nested in a class");

//Size

final int NESTED\_QUEUE\_SIZE = queueNested.size();

//Displaying Values

for(int i = 0; i < NESTED\_QUEUE\_SIZE; i++) {

System.***out***.println("aNestedQueue[" + i + "] = " + queueNested.remove() );

}//For

**Worksheet #3: Complicated Nested Objects Worksheet #3 Points: \_\_\_\_\_/20**

Download and import the file **Worksheet3.java.** Use the file to answer the following questions.

1. Now that you’ve seen how to nest a queue of integers inside a class, let’s nest a queue of **Container** objectsinside the **Plane** class. Complete the **Plane** class below by writing on the worksheet the code required for each method. Find comment *Worksheet3 Question #1* on the Plane class. **(6 pts)**

**class** Container {

**private** String id;

**public** Container(String id) {

**this**.id = id;

}

**public** String getId() {

**return** id;

}

} // Container

**class** Plane {

// Queue of containers for this plane

**private** Queue<Container> containers = **new** LinkedList<>();

**public** **int** getContainersSize() {

return containers.size();

}

**public** **void** addContainer(Container container) {

containers.offer(container);

}

**public** Container removeContainer() {

return containers.remove();

}

} // Plane

1. Assume the **CargoTerminal** class from Assignment 4 and the modified **Plane** class shown above. The tarmac is still an array of plane objects and now the modified plane contains a queue of containers.

**class** CargoTerminal {

**private** **int** numberStands; // Number of stands (parking spots) for planes

**private** Plane[] tarmac; // Array of planes

**public** **void** addCargoPlane(**int** stand, Plane plane) {

tarmac[stand] = plane;

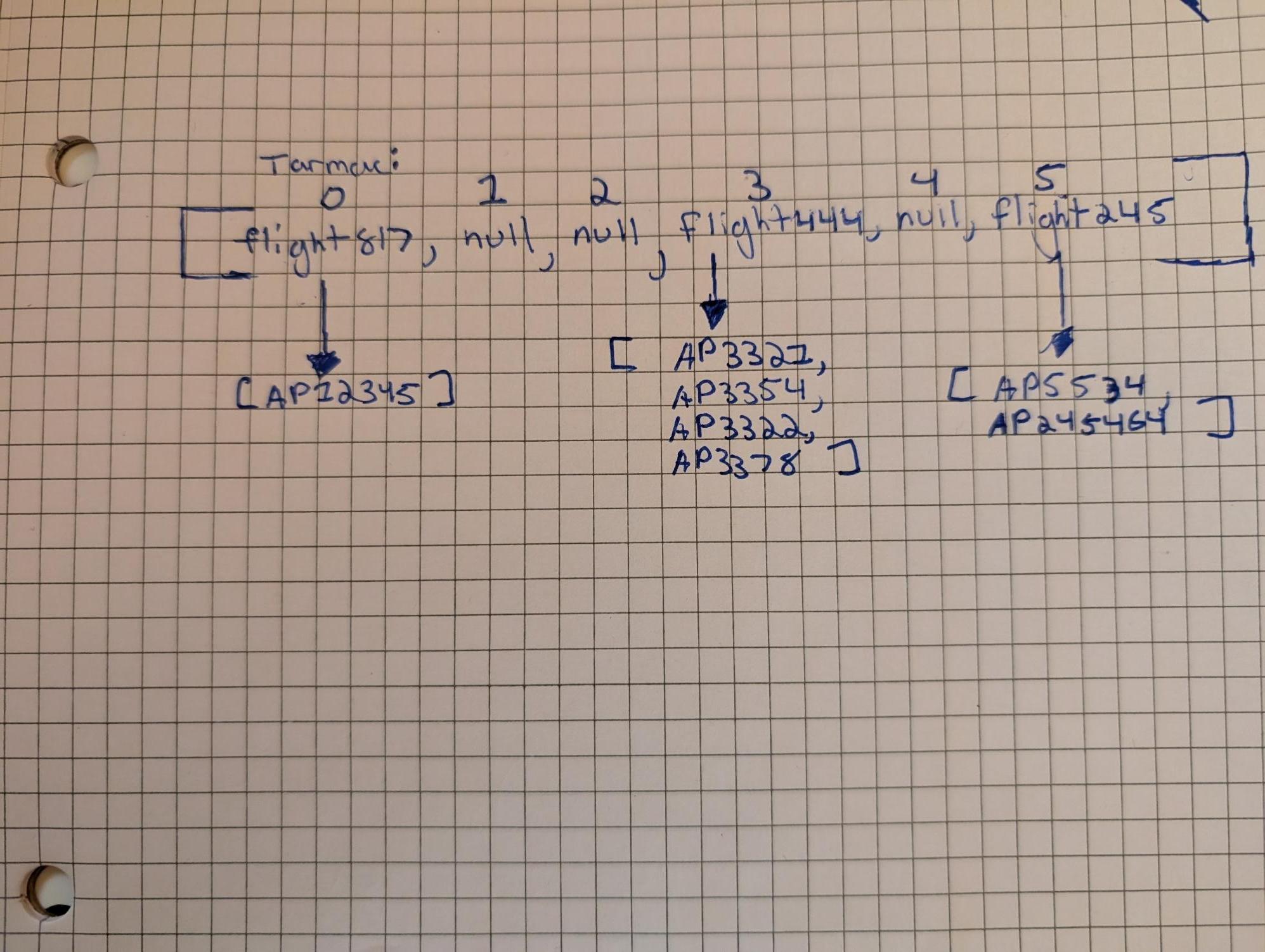
}

} // CargoTerminal

Draw a picture of the **CargoTerminal** object. Show the **array of planes** that represent the tarmac and for each plane in the array, it’s **queue of containers**. Assume it’s a small cargo terminal with a tarmac that contains 6 stands (slots 0-5) and the planes shown below. **(10 pts)**

Show the following in your picture:

1. Draw the entire tarmac array in the **Cargo Terminal** object.
2. Show what is stored in **each location**, include **null** in the locations that **do not** store a plane.
3. Be sure to label each array location (index), the plane or null inside the location, and the containers inside the plane’s containers queue.
   * Flight 817 in stand=0 contains a queue with 1 container (AP12345)
   * Flight 444 in stand=3 contains a queue with 4 containers (AP3321, AP3354, AP3322, AP3378)
   * Flight 245 in stand=5 contains a queue with 2 containers (AP5534, AP245464)



1. In question 2 above to add a plane to the cargo terminal’s tarmac, we used the *addCargoPlane* method. Now, what if we want to add a **container** to a specific plane’s **queue of containers**? This process requires thinking through several layers of nested objects, that is:

* The cargo terminal contains a tarmac array which contains planes and each plane contains a queue of containers. Layers of objects and data structures!

For the **CargoTerminal**, write a method called ***addContainerToPlane*** which takes a container object and a stand number. Use your picture from above to visualize what needs to be done to add **one one container** to the plane’s queue of containers. Find comment *Worksheet3 Question #3* and write that code here: **(4 pts)**

// Add a container to plane in location (stand) in the tarmac

**public** **void** addContainerToPlane(Container container, **int** stand) {

tarmac[stand].addContainer(container);

}

**Worksheet #4: Priority Queues and Comparable Interface Worksheet #4 Points: \_\_\_\_\_/25**

Download and import the file **Worksheet4.java.** Use the file to answer the following questions in the worksheet.

1. When the **remove** method on a priority queue is called, what element is removed first? **(2 pts)**

The remove method for a priority queue removes the index with highest priority, which with default comparable is the smallest number.

1. Write the declaration for a priority queue of integers. Write that code here: **(2 pts)**

PriorityQueue<Integers> integerQueue = new PriorityQueue< >();

1. Let’s do some experimenting in code. Download and import file **Worksheet4.java**. In worksheet 4, you’ll see the **Plane** and **Container** classes from **Worksheet #3: Complicated Nested Objects** with some minor changes to the **Plane** class. After importing the java file, fix the errors by doing the following:
   1. Write the declaration for a priority queue that stores plane objects. Name this priority queue **runway**. Find comment *Worksheet4 Question #3a.* Write that code here. **(2 pts)**

PriorityQueue<PlaneW4> runway = new PriorityQueue< >();

* 1. In the **Plane** class, complete the ***getContainersSize, addContainer, removeContainer*** methods.
     1. In the Plane class, there are 3 comments for “*Worksheet4 Question #3b*”
     2. Copy your answers from **Worksheet #3 Complicated Nested Objects** question #1 to complete these 3 methods.
     3. No need to re-write the code in worksheet for this step.
  2. Now, let’s add planes to the priority queue called **runway**. **(4 pts)**
     1. In main, find the comment “*Worksheet4 Question #3c*”
     2. Write code to add **plane1** and **plane2** to priority queue **runway**.
     3. Write that code here:

runway.offer(plane1);

runway.offer(plane2);

* 1. Run the code. What is the result? Show the exact output. **(2 pts)**

Exception in thread "main" java.lang.ClassCastException: class Assignment6.PlaneW4 cannot be cast to class java.lang.Comparable (Assignment6.PlaneW4 is in module cs1450 of loader 'app'; java.lang.Comparable is in module java.base of loader 'bootstrap')

at java.base/java.util.PriorityQueue.siftUpComparable(PriorityQueue.java:643)

at java.base/java.util.PriorityQueue.siftUp(PriorityQueue.java:639)

at java.base/java.util.PriorityQueue.offer(PriorityQueue.java:330)

at cs1450/Assignment6.Worksheet4\_Spring2025.main(Worksheet4\_Spring2025.java:56)

* 1. The **Plane** class is missing some code that will fix the issue in 3d.

Give a short description of the 2 pieces of code that are missing in the **Plane** class. **(2 pts)**

First code is to implement Comparable to get the compareTo method

Second code is to override the compareTo method in PlaneW4

* 1. Give a short explanation why the 2 pieces of code in 3e are necessary. **(3 pts)**

Since PriorityQueues removes based on highest priority, then the compareTo method it needed to find what is highest priority, and with objects the highest priority is up to coder, in this case could be size, capacity, or flight number.

* 1. Using the *number of containers* for a plane as the comparison factor, add **ALL** necessary code to the **Plane** class in the java file to resolve the issue in 3d. In the Plane class, find comment “*Worksheet4 Question 3g*”. Write that code here: **(8 pts)**

class PlaneW4 implements Comparable {

//Given Code Here

*@Override*

public int compareTo(PlaneW4 otherPlane) {

if(this.getContainersSize() > otherPlane.getContainersSize()) {

return 1;

}//If

else if(this.getContainersSize() < otherPlane.getContainersSize()) {

return -1;

}//Else If

else {

return 0;

}//Else

}//compareTo

}//PlaneW4

This assignment is not for distribution online or by any other means. Copyright UCCS.