**Name:** Joshua White **Select Section:** MW(1:40) TR (1:40) **Fall 2023**

**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 and strings. Let’s now create a stack of book objects. **List** any changes that are needed to the generic class to put book objects on the generic stack, if any are needed. Provide a **short reason for your answer**. **(2pts)**

There are no changes needed for the generic class to put book objects onto the stack. The reason is because the GenericStack takes in objects of type E, which means any object can be put onto the stack, including book objects.

1. Note that **Worksheet1.java** also contains a **Book** class? **(2 pts)**
   1. In main, find the comment “*Worksheet1 Question #2*”
   2. At this point, what code creates a stack of books called **bookStack**? Write that code here:

GenericStackW1<BookW1> bookStack = new GenericStackW1<>();

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

bookStack.push(book);

1. What code is needed to print **ONLY** the name of each book in the **bookStack**? **(7 pts)**
2. In main, find the comment “*Worksheet1 Question #4*”
3. At this point, write a **while loop** to print the title of each book.
4. Note, the stack is empty when the code completes. Write that code here:

while (!bookStack.isEmpty()) {

System.***out***.println(bookStack.peek().getTitle());

bookStack.pop();

}

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

BookW1 topBook = bookStack.peek();

1. An out of bounds error occurs when you run the code in 5b. Give the reason why this occurred.

This error occurred because the list is currently empty, and so has no top. When we try to access the top and there’s no index to find, it gives an error that we are outside of the boundaries of the stack.

**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 books and place the books that have been created for you into this queue using the ***offer*** method. Write that code here: **(3 pts)**

Queue<BookW2> queueNotNested = new LinkedList<>();

queueNotNested.offer(javaBook);

queueNotNested.offer(cBook);

queueNotNested.offer(pythonBook);

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

class BookQueueW2 {

private Queue<BookW2> queue = new LinkedList<>();

public int size() {

return queue.size();

}

public void offer(BookW2 book) {

queue.add(book);

}

public BookW2 remove() {

return queue.remove();

}

} // BookQueue

1. Finally, test the **BookQueue** class using the code in **Worksheet2.java**
   1. Write the declaration for an object of type **BookQueue** and place the 3 books that were created for you into this object (which acts like a queue). Find comment *Worksheet2 Question #3a.* **(3 pts)**

BookQueueW2 nestedQueue = new BookQueueW2();

nestedQueue.offer(javaBook);

nestedQueue.offer(cBook);

nestedQueue.offer(pythonBook);

* 1. Write the code to print the names of the books in the **BookQueue** object. Do not hardcode the queue’s size. *Worksheet2 Question #3b.* **(6 pts)**

int nestedQueueSize = nestedQueue.size();

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

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

for (int i = 0; i < nestedQueueSize; i++) {

System.***out***.println("aQueue = " + nestedQueue.remove().getTitle());

}

**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 books inside a class, let’s nest a queue of **Score** objects inside a **Player** class. Complete the **Player** class below by writing on the worksheet the code required for each method. Find comment *Worksheet3 Question #1.* **(6 pts)**

class PlayerW3 {

private String name;

private Queue<ScoreW3> scores = new LinkedList<>(); // Queue of scores this player obtained

public PlayerW3 (String name) {

this.name = name;

}

public String getName() {

return name;

}

public int getScoresSize() {

return scores.size();

}

public void addScore(ScoreW3 score) {

scores.offer(score);

}

public ScoreW3 removeScore() {

return scores.remove();

}

} // Player

1. Assume the **Team** class from Assignment 4 and the modified **Player** class as shown above.

**class** Team {

**private** Player[] roster; // Array of players on the team

**public** **void** addPlayerToRoster(**int** slot, Player player) {

roster[slot] = player;

}

} // Team

Draw a picture of a **Team** object. Show the **roster** (1D array) and for each player in the array show the **queue of scores**. Assume it’s a small team that contains 5 available slots on the roster (0-4) and the players shown below. **(10 pts)**

**A diagram of a team

Description automatically generated**

Show the following in your picture:

1. Draw the entire 1D roster array in the Team object.
2. Show what is stored in **each location**, including **null** in the locations that **do not** store a player
3. Be sure to label each array location, the player or null inside the location, the scores inside the queue.
   * Slot 0: Garrett who has a queue with 4 scores (80, 67, 95, 70)
   * Slot 3: Selene who has a queue with 2 scores (42, 63)
   * Slot 4: Paul who has a queue with 3 scores (90, 85, 68)
4. In question 2 above, to add a player to the team, we use the *addPlayer* method. Now, what if we want to add a **score** to a specific player’s **queue of scores**? This process requires thinking through several layers of nested objects, that is:

* Team contains a roster 1D array which contains players, and each player contains a queue of scores. Layers of objects and data structures!

For the team, add the method ***addScoreToPlayer*** which takes a score object and a slot number.

Use the picture from above to visualize what needs to be done to add **one score** to the player’s queue of scores. Find comment *Worksheet3 Question #3* and write that code here: **(4 pts)**

public void addScoreToPlayer(ScoreW3 score, int spot) {

roster[spot].addScore(score);

}

**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 is called on a priority queue, how does it decide which element to remove? **(2 pts)**

It uses compareTo to find the highest priority element, and removes that one

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

PriorityQueue<Integer> intPriQue = new PriorityQueue<>();

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

PriorityQueue<PlayerW4> results = new PriorityQueue<>();

* 1. In the **Player** class, complete the ***getScoresSize, addScores,*** and ***removeScore*** methods.
     1. In Player class, there are 3 comments for “*Worksheet4 Question #3b*”
     2. Copy the answers from **Worksheet #3 Complicated Nested Objects** question #1 to complete these 3 methods.
     3. No need to re-write that code in the worksheet for this step.
  2. Now, add players to the priority queue **results**. **(4 pts)**
     1. In main, find the comment “*Worksheet4 Question #3c*”
     2. At this point, write code to add **player1** & **player2** to the priority queue called **results**. Write that code here:

results.offer(player1);

results.offer(player2);

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

Exception in thread "main" java.lang.ClassCastException: class PlayerW4 cannot be cast to class java.lang.Comparable (PlayerW4 is in unnamed module 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 Worksheet4\_Fall2023.main(Worksheet4\_Fall2023.java:49)

* 1. The **Player** 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 **Player** class? **(2 pts)**

The two pieces of code that are missing are an implementation of Comparable<> from the PlayerW4 class, and an Override of compareTo within PlayerW4.

* 1. Give a short explanation why the code in 3e is necessary. **(3 pts)**

Because a PriorityQueue needs to be able to compare the objects in the Queue, and if you put objects of a class that does not provide Comparable into a PriorityQueue, then you need to implement it yourself.

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

*@Override*

public int compareTo(PlayerW4 otherPlayer) {

if (this.scores.size() < otherPlayer.scores.size()) {

return -1;

}

else if (this.scores.size() > otherPlayer.scores.size()) {

return 1;

}

return 0;

}

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