University of Maryland College Park
Department of Computer Science
CMSC132 Fall 2021
Exam #2

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):

STUDENT ID (e.g., 123456789):

Instructions

- Please print your answers and use a pencil.
- This exam is a closed-book, closed-notes exam with a duration of 50 minutes and 100 total points.
- **Do not remove the exam's staple.** Removing it will interfere with the scanning process (even if you staple the exam again).
- Write your directory id (e.g., terps1, not UID) at the bottom of pages with **DirectoryId**.
- Provide answers in the rectangular areas.
- Do not remove any exam pages. Even if you don't use the extra pages for scratch work, return them with the rest of the exam.
- Your code must be efficient and as short as possible.
- If you continue a problem on the extra page(s) provided, make a note on the particular problem.
- You don't need to use meaningful variable names; however, we expect good indentation.
- **You must write your name and id at this point (we will not wait for you after time is up).**
- You must stop writing once time is up.

Grader Use Only

#1	Part #1 (Algorithmic Complexity)	13	
#2	Part #2 (Nested Types)	11	
#3	Part #3 (Generics)	31	
#4	Problem #4 (Miscellaneous)	5	
#5	Part #5 (Class Implementation)	40	
Total	Total	100	

Part #1 (Algorithmic Complexity)

1. (3 pts) List the following Big O expressions in order of asymptotic complexity (lowest complexity first).

   O(**n**log(**n**)) O(**n**) O(log(**n**)) O($\mathbf{n}^2$) O(1)

   O(1), O(log(**n**)), O(**n**), O(**n**log(**n**)), O($\mathbf{n}^2$)

2. (3 pts) Assume both algorithm *A* and algorithm *B* are in O(**1**). How is it possible that when you run the algorithms on the same machine, algorithm B always takes twice as long as A to complete? **Explain** using your knowledge of what Big-O notation tells you about an algorithm. No more than 2 sentences.

   Big -O is just telling you that both algorithms run in constant time. B could have twice as many instructions as A and therefore it will take twice the amount of time.

3. (3 pts) Assume all you know is that algorithm *A* is in O($\mathbf{n}^3$). You consider any algorithm that is not in O($\mathbf{n}^2$) to be slow. Can you say for sure that algorithm *A* is slow? **Explain** using your knowledge of what Big-O notation tells you about an algorithm. No more than 2 sentences.

   No, you cannot say A is slow as it could also be in O(n^2) since it is a subset of O($\mathbf{n}^3$). Big-O is just an upper bound and you need a lower bound to tell you it is slow, so it could be that O(n^3) was a deceptively chosen upper bound

4. (2 pts) Indicate the algorithm complexity of the following expression using the best bound: 7**n** + 2nlog(**n**) + 57**n**

   nlog(**n**)

5. (2 pts) **True or False:** If an algorithm is in O(**n**), it is also **always** in O($\mathbf{n}^2$)?

   true

Part #2 (Nested Types)

6. (8 pts) The **`Exam2Interface`** interface is defined below.

```
public interface Exam2Interface {

    public int increase(int x);

}
```

Given the code below, in the same package as the interface:

```
    public class P2 {
            public static void main(String[] args) {

                    Exam2Interface anonymousVersion = //answer to part a

                    demo(anonymousVersion, 10); //output is 12

                    demo(/*answer to part b*/, 10);  //output is 15

            }

            public static void demo(Exam2Interface e, int num) {
                    System.out.println(e.increase(num));
            }
    }
```

a. Using **anonymous** class syntax, show the code for the assignment to the `anonymousVersion` variable where the implementation simply adds 2 to the argument and returns it.

```
Exam2Interface anonymousVersion =
                                        new Exam2Interface() {
                                              public int increase(int x) {
                                                    return x + 2; } };
```

b. Using a **lambda** expression, show the 1st argument to the second `demo` call *without using an explicit return statement*. The implementation simply adds 5 to the argument and returns it.

```
demo(              x->x+5                                              , 10);
```

7. (3 pts) What is a functional interface? **Explain** in no more than 2 sentences.

An interface with just one abstract method that can be the target of a lambda expression.

Part #3 (Generics)

8. (15 pts) Given the code below (assume the needed import statement are added):

```
public class Utilities {
        /* code for  minimum is the answer to this question – returns the smallest element in the ArrayList*/
        public static void main(String[] args) {
                ArrayList<String> s = new ArrayList<String>();
                s.add("cat"); s.add("rat"); s.add("bat");
                ArrayList<Integer> i = new ArrayList<Integer>();
                i.add(5); i.add(4); i.add(7);
                System.out.println(minimum(s)); // returns bat
                System.out.println(minimum(i)); // returns 4 }  }
```

Write the code for the public static generic method `minimum` with type variable `T` bounded to be a `Comparable<T>`. **Hint: Your code will have no wildcards (i.e. <?>) in it.**

```
public static <T extends Comparable<T>> T minimum(ArrayList<T> myList) {
            T min = myList.get(0);

            for (T e : myList) {
                  if (e.compareTo(min) < 0)
                        min = e;
            }

            return min;

      }
```

Directory ID:

9. (16 pts @ 2 each) Assume the class `CSstudent` extends `Student` which extends `Person`. Further, assume that each of the 3 classes has a default constructor that takes no argument. Which of the following are valid (i.e. **<u>will compile</u>**)? Circle your answer. If there are more than one line of code, as long as there is one line that prevents compilation, circle INVALID.

a.	`ArrayList <Object> q1 = new ArrayList<Person>();`	VALID / [✓ INVALID]
b.	`ArrayList <Person> q2 =  new ArrayList<Student>();`	VALID / [✓ INVALID]
c.	`ArrayList <? extends Person> q3 = new ArrayList<Student>();`	[✓ VALID] / INVALID

d.	`ArrayList <? extends Student> q4 = new ArrayList<Student>();`
	`q4.add(new Student());`
		VALID / [✓ INVALID]

e.	`ArrayList <Student> list = new ArrayList<Student>();`
	`list.add(new Student());`
	`ArrayList <? extends Student> q5 = list;`
	`Person p = q5.get(0);`
		[✓ VALID] / INVALID

f.	`ArrayList <? super CSstudent> q6 = new ArrayList<Person>();`
	`q6.add(new CSstudent());`
		[✓ VALID] / INVALID

g.	`Person arr [] = new CSstudent[2];`
	`arr[0] =new Person();`
		[✓ VALID] / INVALID

h.	`CSstudent arr1 [] = new Person[2];`
	`arr1[0] =new CSstudent();`
		VALID / [✓ INVALID]

<u>Part #4 (Miscellaneous)</u>

10. (3 pts) Using no more than 4 sentences, explain the Mark-and-Sweep Algorithm.

In this Garbage collection algorithm, all objects are marked as dead. The algorithm checks to see which object are reachable by reference in the stack and the heap. If it is reachable, it is marked alive. Any object still remaining dead becomes a candidate for garbage collection.

11. (2 pts) Complete the code below by simply adding the static initialization block that assigns 3.14 to `pi`.

```
public class StaticExample {

        static double pi;

        static { pi=3.14;}                              //add your code here


        public static void main(String[] args) {
                System.out.println(pi);  //prints 3.14

        }

}
```

Part #5 (Class Implementation)

Given the code below in the same package and with correct import statements , simply write the code for the missing iterator method.

```java
public class Player {
 private int playerID;
 private int teamID;

 public Player(int playerID, int teamID) {
       this.playerID = playerID;
       this.teamID = teamID;
 }

 public int getTeamID() {
       return teamID;
 }

 @Override
 public String toString() {
       return "[playerID=" + playerID + "]";
 }


}
```

```java
public class PlayerRoster implements Iterable<Player> {

 private ArrayList<Player> roster = new ArrayList<Player>();
 private int teamNumber;


 public PlayerRoster (int teamNumber) {
       this.teamNumber = teamNumber;
 }

 public PlayerRoster add(Player player) {
       roster.add(player);
       return this;

 }

 public void setTeamNumber(int teamNumber) {
       this.teamNumber = teamNumber;
 }

 public void print() {
       System.out.println(roster);
 }


 /* Relying on anonymous inner class */
 public Iterator<Player> iterator() {
       //code you will write
 }
}
```

Driver

```java
public class Driver {
       public static void main(String[] args) {
              PlayerRoster roster = new PlayerRoster(17);

              roster.add(new Player(5, 12)).add(new Player(10, 17)).add(new Player(15, 12));
              roster.add(new Player(20, 12)).add(new Player(25, 17)).add(new Player(30, 12));
              roster.add(new Player(35,17)).add(new Player(40, 12)).add(new Player(45, 12));

              Iterator<Player> it = roster.iterator();
              while (it.hasNext()) {
                     System.out.println(it.next());  //only prints players in team 17
              }

              System.out.println("---------");

              it = roster.iterator(); //new iterator object

              System.out.println(it.next());  //will return next without calling hasNext
              System.out.println(it.hasNext());  //true, player 25 is the next one
              System.out.println(it.next());
              System.out.println(it.next());

              try {
                     System.out.println(it.next()); } //bad idea, nothing left
              catch (NoSuchElementException e){
                     System.out.println(e.getMessage()); }
              System.out.println(it.hasNext());  //false

              System.out.println("---------");

              roster.setTeamNumber(12); //change to team 12

               it = roster.iterator();

              while (it.hasNext()) {
                     System.out.println(it.next());  //only prints players in team 12
              }
              System.out.println("---------");
              System.out.println("Original data is never modified");
              roster.print();  //prints all players

       }
}
```

Directory ID:

Driver Output

```
[playerID=10]
[playerID=25]
[playerID=35]
---------
[playerID=10]
true
[playerID=25]
[playerID=35]
no element left
false
---------
[playerID=5]
[playerID=15]
[playerID=20]
[playerID=30]
[playerID=40]
[playerID=45]
---------
Original data is never modified
[[playerID=5], [playerID=10], [playerID=15], [playerID=20], [playerID=25], [playerID=30], [playerID=35], [playerID=40], [playerID=45]]
```

You **must use the anonymous class syntax** to create the returned `Iterator<Player>` object. Although you are creating an object to be returned, you are also at the same time defining the class. Therefore you can have fields, private helper methods, and use initialization blocks. The iteration is defined to iterate over players with `teamID` fields that are equal to the `PlayerRoster's teamNumber` field. You can assume that the `teamNumber` used for the iteration must be set before creating an Iterator, and changes to the `teamNumber` after the Iterator's creation will not change the behavior of the iteration. For example, if the `teamNumber` is set to 12 and the iteration process has already started, changing the team number to 17 will not change the Iterator's behavior of iterating over players in team 12. In your implementation, you are not allowed to modify the original data (i.e. `PlayerRoster's roster` field). Notice in the sample driver, the last line shows all players are still in the roster. Finally, if you decide you want to use an ArrayList as one of your fields, you are free to use the ArrayList methods shown below:

`boolean add(E e)`	Appends the specified element to the end of this list.
`E get(int index)`	Returns the element at the specified position in this list.
`E remove(int index)`	Removes the element at the specified position in this list.
`int size()`	Returns the number of elements in this list.

You **MUST** implement the following 2 methods

`public boolean hasNext()`	The method should return true if the iteration has more elements (i.e. if there is a Player in the `PlayerRoster's roster` with a `teamID` equivalent to `PlayerRoster's teamNumber`). Otherwise, it returns false.
`public Player next()`	Throws the unchecked `NoSuchElementException` if the iteration has no more elements. The message used in the exception is `no element left`.

Finally, although it is convention to call `hasNext` before calling `next` (as seen in the first iteration of the driver), your implementation should support being able to call just `next` and getting the next element in the iteration **without** having to call `hasNext`. Of course, if a call is made to `next` and there is nothing left, the `NoSuchElementException` is thrown. See the second iteration in the driver for details.

```
public Iterator<Player> iterator() {

return new Iterator<Player>() {

 private ArrayList<Integer> indexList = new ArrayList<Integer>();  //list of index of Players in teamNumber

 private int removedIndex;

 { getIndices(); } //initialization block since can not have constructor

 private void getIndices() {

        for (int i =0; i <roster.size(); i++)  //just a linear search to find the indices
          {
               if(roster.get(i).getTeamID() ==teamNumber)
               {
                       indexList.add(i);

               }

            }

}

 public boolean hasNext() {

        if (indexList.size()!=0)  //if an index left in the list, there must be another
               return true;
        return false;
}

public Player next() {

        if (indexList.size()!=0) {
               removedIndex = indexList.get(0);
               Player temp = roster.get(removedIndex);
               indexList.remove(0); //remove processed index

               return temp;
        }
               throw new NoSuchElementException ("no element left");

}
};
```

Directory ID:

EXTRA PAGE IN CASE YOU NEED IT

LAST PAGE