|  |  |
| --- | --- |
|  | **University of Maryland College Park** |
| **Dept of Computer Science** |
| ***CMSC132 Summer 2012*** |
| ***Midterm I*** |

*First Name (PRINT):* **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*Last Name (PRINT):* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*University ID:* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Section/TAName:* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*I pledge on my honor that I have not given or received any unauthorized assistance on this examination.*

*Your signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

***Instructions***

* *This exam is a closed-book and closed-notes exam.*
* *Total point value is 100 points.*
* *The exam is a 50 minutes exam.*
* *Please use a pencil to complete the exam.*
* ***WRITE NEATLY****. If we cannot understand your answer, we will not grade it (i.e., 0 credit).*

***Grader Use Only***

|  |  |  |  |
| --- | --- | --- | --- |
| #1 | Java Language Features | (25) |  |
| #2 | Class Implementation | (75) |  |
| **Total** | Total | (100) |  |

**Problem 1 (25 pts) Java Language Features**

1. (2 pts) What is the difference between an interface and an abstract class?
2. (2 pts) What is the difference between a check and an unchecked exception.
3. (2 pts) When is the finalize() method invoked?

1. (3 pts) What is procedural abstraction?
2. (2 pts) Describe two of the components associated with the Model-View-Controller (MVC).
3. (2 pts) For our first project (ClearCellGame) which component (of the Model-View-Controller) did you implement?
4. (3 pts) What is the difference between a static initialization block and a non-static initialization block?
5. (2 pts) **T or F** 🡪 A final method cannot be overridden by a subclass.
6. (2 pts) **T or F** 🡪 The following variable declaration is legal only if the **abstract** class **CPU** has a constructor.

CPU x = new CPU();

1. (2 pts) **T or F** 🡪 A class extending an abstract class will become abstract if abstract method(s) from the super class are not defined in the subclass.
2. (3 pts) The **Ship** interface defines a single method with the following signature: ***public void stop();*** Using an anonymous inner class complete the following assignment where **x** is assigned an object that

 implements the **Ship** interface and the method stop() will print (using System.out.println) the message

 “Stop Ship”.

Ship x =

**Problem 2 (75 pts) Class Implementation**

For this problem you will complete a class named Flight.

public class Flight implements **Comparable<Flight>, Iterable<String>** {

 private int flightNumber;

 private String captain;

 private String[] seats;

 private int numberOfPassengers;

 // Methods you will implement go here

}

For this problem:

1. All the methods are public and non-static.
2. **Constructor** 🡪 It has three parameters (flight number, captain’s name, and maximum number of seats) that are used to initialize the object. The maximum number of seats represents the size of the seats array that will be created by the constructor.
3. **Default Constructor** 🡪 It initializes the object using the values 2, “STAFF”, and 10. You must use “this” to call the previous constructor, otherwise you will not receive any credit.
4. **addPassenger** 🡪 Takes a string representing a passenger’s name. It adds the passenger to the end of the seats array, if there is space. It the parameter provided is null, the method will throw IllegalArgumentException with the message “Invalid name”. The method will return a reference to the current object.
5. **equals** method 🡪 Two flight objects are considered equal if they have the same flight number.
6. **compareTo** method 🡪 The method will allow us to sort flights based on number of passengers. If we called Collection.sort() on an ArrayList of Flight objects, the flights with large number of passengers will be displayed first.
7. **iterator()** 🡪 Returns an iterator that allow us to iterate over the seats array. You do not need to implement the remove method.
8. **toString()** 🡪 It returns a string with the flight number, captain’s name and maximum number of seats. See the sample output below for the output format.
9. The sample driver (and output) below can help you verify the functionality of some of the methods described above.

public class Driver {

 public static void main(String[] args) {

 Flight flight132 = new Flight(132, "Nerdson", 5);

 flight132.addPassenger("Yulu").addPassenger("Justin");

 System.*out*.println(flight132);

 System.*out*.println("Passengers:");

 for (String name : flight132) { System.*out*.println(name); }

 Flight flight106 = new Flight();

 System.*out*.println("Comparison: " + (flight106.compareTo(flight132) > 0));

 }

}

**DRIVER OUTPUT**

Flight Number: 132

Captain: Nerdson

Maximum Number Seats: 5

Passengers:

Yulu

Justin

Comparison: true

**PAGE FOR YOUR ANSWERS**

**PAGE FOR YOUR ANSWERS**