CMP-168 Fall 2021
2 Part Exam (Blackboard & ZyBook) 120 points

TWO PART EXAM: The score will be the sum of the points from the two portions [up to 120 points]

Blackboard Portion of Final Exam (50 points)
Start time: **Friday 12/10/2021 12:00 AM EST**
Deadline for last gradable submission: **Thursday 12/16/2021 11:59 PM EST**
Instructions: Take the exam on blackboard with a maximum of 3 attempts.
Each attempt is limited to 75 minutes.

ZyBook Portion of Final Exam (70 points)
Start time: **Friday 12/10/2021 12:00 AM EST**
Deadline for last gradable submission: **Thursday 12/16/2021 11:59 PM EST**
Unlimited attempts prior to the deadline. Start early, submit, edit, resubmit as often as needed.

ZYBOOK PORTION INSTRUCTIONS:
Use the instructions below to write the class and methods required for the Final Exam.

Q1. Use the UML and Notes in the pages below to create the following classes and interfaces:

a. `( 4 pts)` class Person
b. `( 1 pt )` class InvalidDriverException extends Exception
c. `(20 pts)` class Vehicle
d. `( 1 pt )` interface Announcements
e. `(12 pts)` class Car extends Vehicle implements Comparable<Car>, Announcements
f. `(12 pts)` class Bus extends Car
g. `(10 pts)` class Bicycle extends Vehicle implements Comparable<Bicycle>

Q1 a) (4 pts) Create the following class **Person**

class Person

- name: String
- hasDriverLicense: boolean
- age: int //years
- height: int //inches

+ Person(String name, boolean hasDriverLicense, int age, int height)
+ getName(): String
+ hasDriverLicense(): boolean
+ getAge(): int
+ getHeight(): int
+ clone() : Person //returns a copy of the Person with all the same values without revealing the original memory address
+ equals(Object o) : boolean //2 Person objects are equal if all their variables are equal
+ toString(): String
  //"Person [name= %10s | age= %02d | height= %02d | has license/no license]", name, age, height, hasDriverLicense

Q1 b) (4 pts) Create the following class **InvalidDriverException**

class InvalidDriverException extends Exception

+ InvalidDriverException()
+ InvalidDriverException(String message)

Q1 c) (20 pts) Create the following class **Vehicle**

abstract class Vehicle

#	personsOnBoard: Person [ ][ ]	
#	numberOfRows: int	
#	maxSeatsPerRow: int	
#	numSeatsPerRow: int [ ]	
+	Vehicle(int numRows, int numSeatsPerRow)	//SEE NOTE 1
+	Vehicle(int [ ] numSeatsPerRow)	//SEE NOTE 2
+	Vehicle(Person driver, int [ ] numSeatsPerRow)	//SEE NOTE 3
+	abstract loadPassenger(Person p): boolean	//SEE NOTE 4
+	abstract loadPassengers(Person [] peeps): int	//SEE NOTE 5
+	setDriver(Person p): void throws InvalidDriverException	//SEE NOTE 6
+	getDriver(): Person	
+	hasDriver() : boolean	
+	getNumberOfAvailableSeats(): int	//SEE NOTE 7
+	getNumberOfAvailableSeatsInRow(int row): int	//SEE NOTE 8
+	getNumberOfPeopleOnBoard(): int	//SEE NOTE 9
+	getNumberOfPeopleInRow(int row): int	
+	getPersonInSeat(int row, int col): Person	
+	getLocationOfPersonInVehicle(Person p): int [ ]	//SEE NOTE 10
+	getPeopleInRow(int row): Person[ ]	//SEE NOTE 11
+	getPeopleOnBoard(): Person[ ][ ]	//SEE NOTE 12
+	isPersonInVehicle(Person p): boolean	
+	isPersonDriver(Person p): boolean	

Notes for Class Vehicle:

1. This constructor is used when all the rows have the exact same number of seats.
2. This constructor is used when all the rows do not have the same configuration. The number of seats in each row is provided by the array **numSeatsPerRow**, the number of rows is derived from the size of the array.
3. This constructor does everything specified by (2) above plus it sets the driver of the vehicle. Please note that the specified driver **must** have a driver's license, otherwise they are not qualified to be assigned to the driver position which is **personsOnBoard[0][0]**.
4. This abstract method will be implemented by derived classes to load the specified person into the vehicle at the first available seat. Searching for an available seat must start in the first row, please note that a person below the age of 5 or has a height less than 36 is not allowed to sit in the first row. This method returns **true** if there is room for the specified person, otherwise the method returns **false**.
5. This method attempts to load as many of the persons specified in the **peeps** array as possible. This method returns the number of people that were loaded into the Vehicle.
6. This method sets the driver of the Vehicle if the specified person has a driver's license. If the person does not have a driver's licence, this method will throw the **InvalidDriverException**.
7. This method returns the number of available/empty seats in the Vehicle.
8. This method returns the number of available/empty seats in the specified row. If the specified row is invalid, the method returns -1.

9. This method counts the number of people currently assigned to seats in the vehicle, including the driver.
10. This method returns the location of the specified person in the vehicle by returning an int array containing [**row**, **seatColumn**]. Please note that if the specified person is not seated in the vehicle, the method will return [-1, -1].
11. This method will return a Person array containing a **clone** of all the persons in the specified row. The populated Person array will not contain empty index locations. If the specified row has no one, or the specified row is invalid, this method returns a **null**.
12. This method returns a full clone of the **personsOnBoard** array with clones of the Person objects contained within. Please note that personsOnBoard.clone() is not sufficient to clone a 2-dimensional array.

Q1 d) (1 pt) Create the following interface **Announcements**

interface Announcements

+	departure(): String	//SEE NOTE 1
+	arrival(): String	//SEE NOTE 2
+	doNotDisturbTheDriver(): String	//SEE NOTE 3

Notes for Interface Announcements:

1. The departure() method of a Car should return a line stating “All Aboard\n”, while a Bus should return a String containing 2 lines “All Aboard\nThe Bus\n”. The Bus implementation should make use of super.departure().
2. The arrival() method of Car should return a line stating “Everyone Out\n”, while a Bus should return a String containing 2 lines “Everyone Out\nOf The Bus\n”. The Bus implementation should make use of super.arrival().
3. The doNotDisturbTheDriver() method of Car should return a line stating “No Backseat Driving\n”, while a Bus should return a String containing 2 lines “No Backseat Driving\nOn The Bus\n”. The Bus implementation should make use of super.doNotDisturbTheDriver().

Q1 e) (12 pts) Create the following class **Car**

class Car extends Vehicle implements Comparable<Car>, Announcements

-	numDoors: int	
-	numWindows: int	
+	Car(int numDoors, int numWindows)	//SEE NOTE 1
+	Car(int numDoors, int numWindows, int numSeatsPerRow)	//SEE NOTE 2
+	Car(int numDoors, int numWindows, int [ ] numSeatsPerRow)	//SEE NOTE 3
+	Car(int numDoors, int numWindows, Person driver, int [ ] numSeatsPerRow)	//SEE NOTE 4
+	canOpenDoor(Person p): boolean	//SEE NOTE 5
+	canOpenWindow(Person p): boolean	//SEE NOTE 6
+	getNumDoors(): int	
+	getNumWindows(): int	
+	equals(Object o): boolean	//SEE NOTE 7
+	toString(): String	//SEE NOTE 8
+	compareTo(Car c): int	//SEE NOTE 9

Notes for Class Car:

1. This constructor calls the parent class's constructor, passing in 2 for the **numRows** and 2 for the **numSeatsPerRow**, before assigning the **numDoors** and **numWindows** values.
2. This constructor calls the parent class's constructor passing in 2 for the **numRows** and the **numSeatsPerRow** argument, before assigning the **numDoors** and **numWindows** values.
3. This constructor calls the parent class's constructor passing in the **numSeatsPerRow[]** array as an argument, before assigning the **numDoors** and **numWindows** values.
4. This constructor calls the parent class's constructor passing in the **driver** and the **numSeatsPerRow[]** array, before assigning values to **numDoors** and **numWindows**.
5. This method returns true if the **Person** is seated in either of the exterior seats of a row that has a door (column index 0 or last index of that row) and is over **age** 5. It returns false otherwise.
   If the number of doors is less than 2 * numberOfRows, the row/s past numDoors / 2 has/have no doors.
6. This method returns true if the Person is seated in either of the exterior seats of a row that has a window (column index 0 or last index of that row) and is over **age** 2. It returns false otherwise. If the number of windows is less than 2 * numberOfRows, the row/s past numWindows / 2 has/have no windows.
7. Two Car objects are considered equal if they have the same **numDoors**, same **numWindows**, same **numberOfRows**, same **maxSeatsPerRow,** and the same seat configuration (numSeatsPerRow[row]) at every row.
8. Return a string containing the Car's details formatted as follows:
   "Car: number of doors= %02d | number of windows = %02d | number of rows= %02d | seats per row= %s | names of people on board= %s\n"
   The seats per row will be from the array of numSeatsPerRow as [val1,val2,val3, valn]
   The names of people on board will be separated by commas with no trailing comma.
9. This method returns -1 if the calling object's total number of seats is less than the passed in object's total number of seats, 1 if the calling object's total number of seats is greater than the passed in object's total number of seats, 0 if they have the same total number of seats.
10. Be sure to implement the loadPassenger, and loadPassengers methods in the Car class.

Q1 e) (12 pts) Create the following class **Bus**

class Bus extends Car

+	Bus(int[] numSeatsPerRow)	//SEE NOTE 1
+	Bus(Person driver, int[] numSeatsPerRow)	//SEE NOTE 2
+	canOpenDoor(Person p): boolean	//SEE NOTE 3
+	canOpenWindow(Person p): boolean	//SEE NOTE 4
+	toString(): String	//SEE NOTE 5

Notes for Bus class:

1. This constructor calls the parent class's constructor, passing in **2** for the **numDoors**, **numWindows** equal to (**(2 * length of numSeatsPerRow array) -1**) since the first row only has 1 window, and as the last argument, **numSeatsPerRow[]** where the first row (index 0) always has 1 seat, while the rest of the array has **numSeatsPerRow** in each index.
2. This constructor does everything specified by (1) above, plus passes in the **Person**.
3. This method returns true if the **Person** is the driver, or seated in the last populated row. The Person must also be above the age of 5, and taller than 40 inches. It returns false otherwise.
4. This method returns true if it's parent class's **canOpenWindow** method returns true, and the **Person** is above the age of 5.
5. This method prepends **"Bus is an extension of "** to the string returned from the Car class's toString() method before returning the entire string.
6. Be sure to implement the loadPassenger, and loadPassengers methods in the Bus class.

Q1 e) (10 pts) Create the following class **Bicycle**

class Bicycle extends Vehicle implements Comparable<Bicycle>

- weight: double
+ Bicycle() //SEE NOTE 1
+ Bicycle(Person driver) //SEE NOTE 2
+ Bicycle(Person driver, double weight) //SEE NOTE 3
+ equals(Object o): boolean //same weight. ACCURACY_RANGE = 0.5 //SEE NOTE 4
+ getWeight(): double
+ setWeight(double w): void
+ setDriver(Person p): void throws InvalidDriverException //SEE NOTE 5
+ toString(): String //"Bicycle [rider= " + getDriver().getName() + " | weight= " + weight + "]"
+ compareTo(Bicycle b) //ACCURACY_RANGE = 0.5 //SEE NOTE 6

Notes for Bicycle class:

1. This constructor calls the parent class's constructor, passing in 1 for the **numRows** and 1 for the **numSeatsPerRow** before initializing **weight** to 0.
2. This constructor calls the parent class's constructor, passing in the specified **driver** and an array specifying that there is only one row with one seat,before initializing **weight** to 0.
3. This constructor calls the parent class's constructor, passing in the specified **driver** and an array specifying that there is only one row with one seat, before setting the **weight**. (Negative **weight** not permitted, set to 0 if that occurs).
4. Two bicycle objects are considered equal if their weights are within 0.5 of each other regardless of who the driver is.
5. Override the inherited setDriver method so that any Person age 3 or older can be the driver of the Bicycle. If the driver is younger than 3, throw an InvalidDriverException
6. If the calling object's weight is less than the passed in object's weight by more than the ACCURACY_RANGE return -1. If the calling object's weight is greater than the passed in object's weight by more than the ACCURACY_RANGE return 1. Otherwise return 0;
7. Be sure to implement the loadPassenger, and loadPassengers methods in the Bicycle class. Hint: A Bicycle cannot have any passengers. (A driver is not a passenger)

Q2) (10 pts) Create a class named RecursionQuestion

In this class you will be using the Car class defined above, and you will be performing a binary search on an array of Car objects.

Your class will have a method with the following signature.

public static int binarySearch(Car[] cars, Car c)

This method will call a recursive helper method to perform the binary search for the specified Car c.

In the examples below, s is the start index, e is the end index, and mid is the mid index at each phase of the recursive method calls.

When the car being searched for is found, your code will produce output similar to the following:

```
Looking for Car: number of doors = 02 | number of windows = 04 | number of rows
= 03 | seats per row = [2,3,3] | names of people on board = []

s=0, e=9, mid=4
go left
s=0, e=3, mid=1
go right
s=2, e=3, mid=2
go right
s=3, e=3, mid=3
FOUND at 3
```

When the car being searched for is not found, your code will produce output similar to the following:

```
Looking for Car: number of doors = 02 | number of windows = 04 | number of rows
= 02 | seats per row = [1,3] | names of people on board = []

s=0, e=9, mid=4
go left
s=0, e=3, mid=1
go left
s=0, e=0, mid=0
go right
s=1, e=0
Not Found
```

Please note that the line beginning with: `Looking for Car:` is actually “Looking for Car: “ + c.toString()