COMP 131 Practice Midterm Exam II Solution
135 points

name: KEY

1. (5 points) Assume that `x` and `y` are variables of type `int`. Can the following snippet of code ever print `not a root vegetable` to the terminal window? Why or why not?

```
if (x == y) {
    System.out.println("carrot");
} else if (x < y) {
    System.out.println("potato");
} else if (x > y) {
    System.out.println("rutabaga");
} else {
    System.out.println("not a root vegetable");
}
```

No. `x` *is either equal to* `y`*, less than* `y` *or greater than* `y` *- there are no other possibilities.*

2. (10 points) Suppose that the following method is defined in a class that has three `int` fields named `x`, `y` and `z`.

```
public int foo() {
    if (x < y) {
        if (y < z) {
            return 1;
        } else {
            return 2;
        }
    } else if (y < z) {
        return 3;
    } else {
        return 4;
    }
}
```

   (a) How many test cases are required to achieve statement coverage for this method?
   4

   (b) What values should `x`, `y` and `z` be set to for each test case? (Many different combinations of values for the three fields would work for each test case - you need only give one of the possibilities for each.)

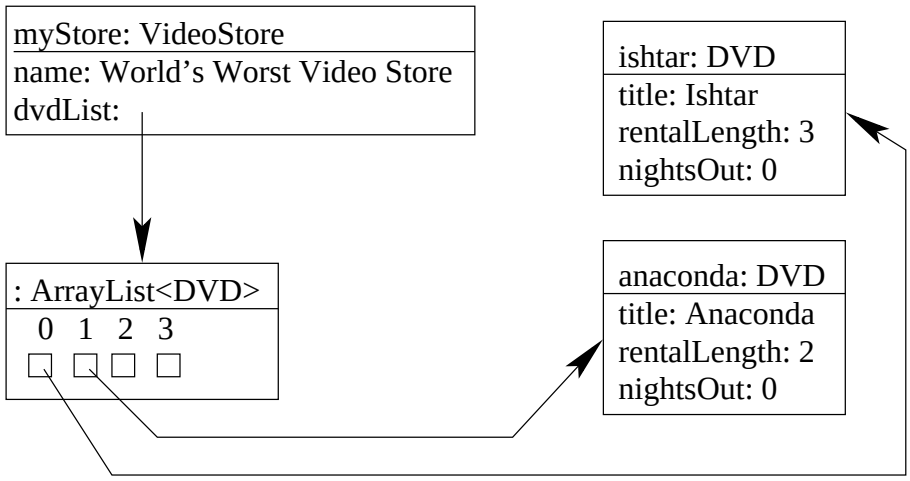
case 1: `x`*: 3* `y`*: 5* `z`*: 7 (*`x` $<$ `y` *and* `y` $<$ `z`*)*
case 2: `x`*: 3* `y`*: 5* `z`*: 1 (*`x` $<$ `y` *and* `y` $\geq$ `z`*)*
case 3: `x`*: 5* `y`*: 3* `z`*: 4 (*`x` $\geq$ `y` *and* `y` $<$ `z`*)*
case 4: `x`*: 5* `y`*: 4* `z`*: 3 (*`x` $\geq$ `y` *and* `y` $\geq$ `z`*)*

3. (10 points) Recall the `DVD` and `VideoStore` classes from the homeworks. Some of the aspects of these classes that are relevant to this problem are:

   - the fields of the `VideoStore` class are `storeName` (a `String`) and `dvdList` (an `ArrayList`)
   - the fields of the `DVD` class are `title` (a `String`), `rentalLength` and `nightsOut` (both of type `int`)
   - the constructor for class `DVD` sets `rentalLength` using the value of its second parameter, and sets `nightsOut` to 0

   Draw an object diagram for the object referred to by variable `myStore` after the following snippet of code has been executed.

```
VideoStore myStore = new VideoStore("World's Worst Video Store");
DVD glitter = new DVD("Glitter", 1);
DVD ishtar = new DVD("Ishtar", 3);
DVD batEarth = new DVD("Battlefield Earth", 4);
DVD anaconda = new DVD("Anaconda", 2);
DVD gili = new DVD("Gili", 7);
myStore.addDVD(glitter);
myStore.addDVD(ishtar);
myStore.addDVD(batEarth);
myStore.addDVD(anaconda);
myStore.addDVD(gili);
myStore.removeDVD(4);
myStore.removeDVD(0);
myStore.removeDVD(1);
```



4. (40 points) Consider the following definition of class `CrewMember`, which represents a member of an airline flight crew. This class will be used by the `FlightCrew` class, which represents an entire flight crew consisting of a pilot, copilot, and navigator.

```
public class CrewMember {
    private String name;
    private double flightHours; // number of flight hours this crew member has logged

    /** create a new crew member with the specified name and no flight hours
      * @param initName the name of the crew member */
    public CrewMember(String initName) {
        name = initName;
        flightHours = 0.0;
    }

    /** return the name of the crew member */
    public String getName() {
        return name;
    }

    /** return the number of flight hours logged for the crew member */
    public double getFlightHours() {
        return flightHours;
    }

    /** increase the number of flight hours for the crew member
      * @param increaseBy the number of hours to increase the flight time by */
    public void increaseFlightHours(double increaseBy) {
        flightHours = flightHours + increaseBy;
    }
}
```

Fill in each of the following constructor and method definitions for class `FlightCrew` according to the given specifications.

```
public class FlightCrew {
    private CrewMember pilot;
    private CrewMember coPilot;
    private CrewMember navigator;
    /** create a new flight crew from three existing crew members */
    public FlightCrew(CrewMember initPilot, CrewMember initCoPilot,
                      CrewMember initNavigator) {

        pilot = initPilot;
        coPilot = initCoPilot;
        navigator = initNavigator;

    }
```

(continued on next page)

```
    /** return the total number of flight hours logged by the three crew members */
    public double totalFlightHours() {

        return pilot.getFlightHours() + coPilot.getFlightHours() + navigator.getFlightHours();

    }

    /** return the crew member with the most flight hours.  If there is a tie, any of
      * the crew members with the most flight hours could be returned */
    public CrewMember getMostExperiencedMember() {

        int p = pilot.getFlightHours();
        int cp = coPilot.getFlightHours();
        int n = navigator.getFlightHours();
        if (p >= cp && p >= n) {
            return pilot;
        } else if (cp >= p && cp >= n) {
            return coPilot;
        } else {
            return navigator;
        }

    }

    /** print the name and number of flight hours for the most experienced member of
     * of the flight crew.  For full credit, do NOT repeat the code that you wrote
     * for the getMostExperiencedMember method in this method. */
    public void printMostExperiencedMember() {

        CrewMember most = this.getMostExperiencedMember();
        System.out.println(most.getName();
        System.out.println(most.getFlightHours();

    }
}
```

5. (15 points) Consider the following definition of class `CarDealership`, which tracks the inventory of cars for a car dealer. The constructor is correct, but each method in the class contains at least one error that would prevent the class from compiling, or that could cause the program to crash when it was run. Find one such error in each method, and explain why it is an error. Write your answer for each method next to that method. Assume that the `Car` class is correctly implemented and is defined in the same BlueJ project as the `CarDealership` class. Javadoc comments have intentionally been omitted to save space, so do not consider the lack of these comments to be an error.

```
import java.util.ArrayList;

public class CarDealership {
    private String dealerName;
    private ArrayList<Car> carLot; // all cars in the dealership's inventory

    public CarDealership(String name) {
        dealerName = name;
        carLot = new ArrayList<Car>();
    }

    public Car getCar(int index) {
        if (index >= 0 && index < carLot.size()) {
            Car theCar = carLot.get(index);
            return theCar;
        } else {
            System.out.println("Error: invalid Car index!");
        }
    }
```

Even if the index is invalid, this method must still return an appropriate value - most likely `null` *in this case.*

```
    public Car getFirstCar() {
        Car firstCar = (Car) carLot.get(0);
        return firstCar;
    }
```

This will cause an IndexOutOfBoundsException if the ArrayList is empty.

```
    public void printCar(int position) {
        if (position >= 0 && position <= carLot.size()) {
            System.out.println("The car at position " + position + " is:");
            // assume that Car has a toString method
            System.out.println(carLot.get(position));
        } else {
            System.out.println("Error: invalid Car position!");
        }
    }
```

This will cause an IndexOutOfBoundsException if position is equal to the size of the ArrayList.

```
}
```

6. (35 points) Consider the following `Bank` class that maintains a collection of `Account` objects (as defined in class). Complete the definition of this class by filling in the fields, constructor and method bodies. Recall that the `Account` class has a `getBalance()` method that takes no parameters and returns the account balance (in cents).

```
import java.util.ArrayList;

public class Bank {

    private ArrayList<Account> accts;

    /** create a new Bank containing no Accounts */
    public Bank() {

        accts = new ArrayList<Account>();

    }

    /** add the specified Account to the Bank
      * @param acct the Account to add */
    public void addAccount(Account acct) {

        accts.add(acct);

    }

    /** find the total balance of all Accounts in the Bank (0 if there are no Accounts)
      * @return the total balance of all Accounts */
    public int totalBalance() {

        int total = 0;
        for (int i = 0; i < accts.size(); i++) {
            total = total + accts.get(i).getBalance();
        }
        return total;

    }

    /** Find and return the account with the largest balance.  If there is a tie,
        return the account that was most recently added to the bank.  If the bank has
        no accounts, return null. */
    public Account largestBalance() {

        if (accts.size() == 0) {
            return null;
        } else {
            int max = 0;
            for (int i = 1; i < accts.size(); i++) {
                if (accts.get(i).getBalance() >= accts.get(max).getBalance()) {
                    max = i;
                }
            }
            return accts.get(max);
        }


    }
```

```
    /** create a new Bank containing all Accounts that have a balance over 900 cents
      * @return the new Bank */
    public Bank bigAccounts() {

        Bank big = new Bank();
        for (int i = 0; i < accts.size(); i++) {
            if (accts.get(i).getBalance() > 900) {
                big.addAccount(accts.get(i));
            }
        }
        return big;

    }
}
```

7. (5 points) What output (if any) would be produced by the following snippet of code?

```
for (int i = 3; i < 10; i = i + 2) {
    System.out.println(i);
}
```

3
5
7
9

8. (5 points) Assume that `x` is a variable of type `int`. List or describe all values of `x` that will make the following boolean expression false: `(x <= 2) || (x != 5)`

5

9. (5 points) Assume that `x` is a variable of type `int`. List or describe all values of `x` that will make the following boolean expression true: `!(x < 2) && (x < 7)`

2, 3, 4, 5, 6

10. (5 points) Give an example of how we have used abstraction in writing programs in this class.

When implementing the BankCustomer class, we used the Account class (specifically the withdraw and deposit methods) without worrying about how it was implemented.