Your Name: _____ **Student Number:** ___

Lab 4a: to be done in class with a partner – due at the end of the lab

Create a class named **RoyalBankAccount** that has the following data members:

Bank motto is "You're poorer than you think" for all accounts; it cannot change.

Prime interest rate is 0.025 for all accounts, although it may change.

Each account also has a personal interest rate, a balance in USD, a PIN, account holder first and last names, and a boolean to represent whether there is overdraft available (also store the amount of overdraft that is available).

Each account also has a personal account number that cannot change.

Remember: constants must be in UPPERCASE.

Create accessors and mutators for the data members as appropriate, and a constructor that takes appropriate parameters too. Mutators throw IllegalArgumentExceptions if bad data is passed to them (see below).

Furthermore, add withdraw and deposit methods that ensure that the following rules are met:

- You cannot withdraw zero or negative amounts, nor can you withdraw more than your balance (unless you have overdraft available...in that case, you may withdraw up to that amount in the negative)
- You cannot deposit a negative or zero amount. In addition, if more than 10000USD are being deposited, System.out.println a message saying, "The manager will be contacting you. Thank you for your deposit".

Make sure there are no magic numbers in your code! Use constants instead of numbers.

Also, include a static method to get the prime interest rate and to get the bank's motto.

Submission (10 marks)

This in-class lab is due at the end of this class. Do not upload your lab to BCIT's servers. When you are finished, show your instructor so he can sign your paper.

Checked by: _____

NOTE: keep this paper for your instructor to verify your grades later in the course.

NOTE: EVERY SINGLE STUDENT MUST SHOW THIS LAB AND GET HIS OR HER PAPER SIGNED....

Lab 4b: to be done at home alone – submit to D2L for marking

For this lab you will create a class called Model. Include a Javadoc comment above the class including a description, @author tag, and @version tag. Here are the attributes of a Model that we are concerned with for the purpose of this project:

- first name
- last name
- height (inches – use int)
- weight (pounds – use double)
- can travel (true or false)
- smokes (true or false)
- occupation (static; value is "modeling")

Choose appropriate data types and descriptive names for the fields (instance variables), and declare them. Be sure to specify that they are private.

Write **two** constructors for this class (please see the NOTE FOR CONSTRUCTORS on the next page first). The first constructor will expect *the first six fields* to be passed in as parameters. Choose descriptive names for the parameters but remember that they cannot be exactly the same as the field names. Use the parameters to initialize the fields, but only if they follow these rules (i.e., do not store parameter values which fail to meet the following criteria):

- first name and last name must be 3 to 20 characters long or else they won't be stored
- height must be 24 to 84 inches or it won't be stored
- weight must be 80 to 280 pounds or it won't be stored

The second constructor takes parameters for only the first name, last name, height in inches, and weight in pounds; this constructor will set canTravel to **true** and smokes to **false**. Include a Javadoc comment with @param tags for each of the parameters above the two constructors.

NOTE FOR CONSTRUCTORS:

Make the mutator methods **final**, as follows:

```
public final void setCanTravel(boolean willTravel)
{
        canTravel = willTravel;
}
public final void setFirstName(String first)
{
        if((first != null) && (first.length() >= 3) && (first.length() <= 20))
        {
                firstName = first;
        }
}
```

In both of the non-default constructors, **call the mutator method for each field instead of using an assignment statement**, as in this example:

```
public Model(String theFirstName, String theLastName, int theHeightInches,
        double theWeightKg, boolean traveler, boolean smoker)

{
        setFirstName(theFirstName);
        setCanTravel(traveler);                    // or setCanTravel(true);
```

```
        etc...
    }
```

Write an accessor ("get") method **for each instance variable**. Include Javadoc comments with @return tags above each method. Write a mutator ("set") method **for each field**. Include Javadoc comments with @param tags above each method, and only store the parameter if it meets the respective criteria listed above.

Also add the following accessor methods, which do exactly what they say:

```
public String getHeightInFeetAndInches()   // e.g. "5 feet", or
                                            // e.g. "5 feet 1 inch", or
                                            // e.g. "5 feet 6 inches", or
public long getWeightKg();                  // e.g. 103 (Math.round() the number)

public static String getOccupation()
```

Also add the following mutator methods:

public void setWeight(long kilograms)

public void setWeight(double pounds)

public void setHeight(int feet, int inches)

public void setHeight(int inches)

Also add the following method: **public void printDetails():**

Which prints in the following format, exactly:

Name: Susan Smith
Height: 70 inches
Weight: 120 pounds

Does not travel
Does smoke

or

Name: Tiger Woods
Height: 72 inches
Weight: 190 pounds
Does travel
Does not smoke

etc...

NOTE: this method *must* call your own object's accessor methods; do not access the instance variables directly. For example:

```
public void printDetails()
{
    System.out.println("Name: " + getFirstName() + " " + getLastName());
    System.out.println("Height: " + getHeightInches() + " inches");
    System.out.println(" Weight: "  + Math.round(getWeightPounds()) + " pounds");
    if(canTravel){
        System.out.println("Does travel ");
    }else{
        System.out.println("Does not travel ");
    }
    if(smokes){
        System.out.println("Does smoke ");
    }else{
        System.out.println("Does not smoke ");
    }
}
```

Test your project by compiling your class and then creating a Model object. Use the BlueJ inspector to check the contents of the fields (right-click the red-box object at the

bottom of the screen, in the object bench). Call each mutator method to be sure it is storing (or ignoring, if the criteria is not met) the correct value. Call each accessor method to be sure it is returning the correct value. **Also right-click the CLASS and call the static getOccupation() method.**

Submission (15 marks)

Test your code and submit it to the Lab 3 folder on D2L (Activities -> Assignments -> Lab4) by Friday, Oct. 5, 2018 at midnight. The submission must include the **Model.java** file from your BlueJ project.

Mark Breakdown:

- 0 points the Model class does not compile
- (3 points) Valid JavaDoc (class and constructor), good names and data types
  - 1 point taken off for each violation
- (12 points) Meets the specifications in the description