CS-0401 - Second Exam

Instructor: Paulo Brasko
Term: Spring 2020

Week Day: Monday / Thursday (circle one)
Name: _____

1) Please provide SHORT answers for the following questions. (Also, write using your best handwriting… I cannot grade if I cannot read what you wrote) (30 points)
   a) What is the difference between Java Classes and Java Objects? Give an example.
   Java Classes are like blueprints: they give the instructions on what variables (class fields) should be created in memory, and what are the methods that can be used when an object of such class is created. Java Objects are the actual/concrete object that has values placed in those variables (class fields).

   b) What are the basic components of any given class?
   Basic components: class fields, constructors, getter and setter methods, class methods, toString() and equals() methods.

   c) What are class fields?
   Class Fields are the "variables" of the object, such as price, color, and even other objects, such as a Car Class may have a color, price, and MotorEngine object as class fields.

   d) What are class methods?
   Class Methods are methods that can be used on the object, such as myCar.hasBeenInAnAccident(). These methods have their access mode set to public.

   e) In relation to Data Encapsulation:
      i) What is data encapsulation?
      It is the way we hide class fields from allowing direct access from external applications. Example the following code would give an error, since we cannot change the value of the price directly as shown below:

```
Car myCar = new Car();
myCar.price = 1.00;
```

      ii) Why should we encapsulate fields?
      To avoid external applications to change the values of a given object field to an unrealistic value, such as the price of the car being $1.

      iii) How to make a class field encapsulated?
      A class field can be encapsulated by setting its access type to private or protected as shown below:

```
private double price;
protected boolean hasBeenInAnAccident;
```

iv) How to access values of a given object that have been encapsulated? Create one method to set and another to get the value of a “price” encapsulated field. Note: put logic that a car price shall be in the range of $10000 and $15000.
You access values from encapsulated fields using getter and setter methods (also known as accessors and mutators). Example:

```
public double getPrice() {
   return price;
}

public void setPrice(double price) {
    if (price >= 10000 && price <= 15000) {
      This.price = price;
   }
}
```

f) In relation to Class Constructor:

i) When a constructor is called?
A class constructor is normally called when we use the “new” operator, as shown in the example below:

```
Toyota myCar = new Toyota();
Toyota yourCar = new Toyota(“Black”);
```

ii) What types of constructors can a Class define?
A Class can define multiple constructors: one type is the “default constructor” which takes no input argument and initializes all the class fields with default values, such as minimum price, manufacturer standard color, etc. The other type of constructor is the “non-default” constructor, which can take different input arguments. See examples above on using default and non-default Toyota objects (also known as “instances” of Toyota class)

iii) When should I use one or the other(s)?
Use default constructor to set the class fields to default values. Use non-default constructor when you want some field values to have a specific value as soon as we create the object.

iv) Can I call a constructor from another constructor? If so, when is a good idea for doing it? And what is the Java command to accomplish that? Give an example while answering this question.
Yes, it is possible to call a constructor from another constructor. You can do that using the “this()” command. A common practice is inside a non-default constructor to call the class default constructor to set all the class fields to their default values and then update the ones given in the non-default constructor input arguments. Example:

```
public Toyota() {
    price = 15000;
    color = “Black”;
    hasBeenInAnAccident = false;
}
```

```
public Toyota(String color) {
    this();  // calling the default constructor here
    this.color = color;
}
```

2) Please provide SHORT answers for the following questions. (30 points)
   a) About inheritance:
      i) Give an real-world example of inheritance
      A young girl is a Woman. Therefore the young girl has most of the features that a woman has (such as name, age, Social Security number, etc., i.e., the young girl inherits Woman features.

      ii) What is the advantage of using inheritance when creating applications in Java?
      We avoid code duplication, i.e., we do not need to redefine variables (class fields) and methods in both Woman and YoungGirl classes. Just defined in Woman and make the YoungGirl inherits from it.

      iii) How to correlate two Java classes under inheritance? I.e., What do you need to do (in terms of a Java command or keyword) in a given class A to make it inherit things from a given Class B?

```
class A extends B { // class code here }
```

   b) What is method overload?
   Multiple methods with the same name but different input arguments (i.e., number of input arguments and/or types of input arguments) can be defined in a given class and/or its superclasses. Java Virtual Machine will know what method to call based on the input arguments.

   c) What is method override?
   Multiple methods with the same name AND same input arguments (i.e., number of input arguments and/or types of input arguments) can be defined in a given class AND its superclasses. The method in the subclass overrides the functionality of the method found in the superclass.

   d) What is polymorphism? Give a quick example on how to use polymorphism in a List and as an input argument for a method.
   Polymorphism is when an object of a given class can be treated as an object of a superclass of this class. Example: a YoungGirl can be treated as a Woman or as a Human, as a Being, etc.
   Since lists and arrays accept only one type of object, we cannot create a list to hold different types of humans if we define a List<YoungGirl>. However we can use polymorphism and use List<Human>. Example:

```
YoungGirl guacira = new YoungGirl();
Woman mary = new Woman();
YoungBoy brenno = new YoungBoy();
List<Human> relatives = new ArrayList();
relatives.add(guacira);
relatives.add(brenno);
relatives.add(mary);
```

3) McDonalds menu has the following Meal options: (10 points each item below (a),(b),and (c))

Big Mac Meal	Cheeseburger Meal	Quarter Pounder Meal
Meal name: Big Mac Coca-Cola Medium French Fries 1080 Cal. Ingredients: Ketchup, pickle, onions $5.00	Meal name: Cheeseburger Orange Juice Large French Fries 840 Cal. Ingredients: American Cheese, Mustard $6.00	Meal name: Quarter Pounder Milk Small French Fries 1050 Cal. Ingredients: Seasoning, Onions $4.50

a) Since there are lots of similarities in those meals, we can create a super-class that will contain such similarities, right? Create a superclass called Meal. By "create a superclass Meal" I mean create the whole class: class fields, getters and setters. Don't create class constructors for this class. Let's do the constructors in the subclasses instead. To avoid you spending 30 minutes creating setters and getters methods for all the possible class fields, create setters and getters only for Calories and Ingredients class fields so I know that you know how to create them.

```
public class Meal {
    private String mealName;
    private String drinkType;
    private String friesSize;
    private int calories;
    private List<String> ingredients;

    public void setCalories(int calories) {
        if (calories > 0) {
            this.calories = calories;
        }
    }

    public int getCalories() {
        return calories;
    }

    public void set Ingredients(List<String> ingredients) {
        this.ingredients = ingredients;
    }

    public List<String> getIngredients() {
        return ingredients;
    }
}
```

b) Create the BigMac, CheeseBurger, and QuarterPounder subclasses. (What code must be placed in those classes?)

```
public class BigMac extends Meal {
    public BigMac() {
        this.mealName = "Big Mac";
        this.drinkType = "Coca-cola";
        this.friesSize = "Medium";
        this.calories = 1080;
        ingredients = new List();
        ingredients.add("ketchup");
        ingredients.add("pickle");
        ingredients.add("onions");
        Ingredients.price = 5.00;
    }
}
```

```
public class BigMac extends Meal {
    public BigMac() {
        this.mealName = "Big Mac";
        this.drinkType = "Coca-cola";
        this.friesSize = "Medium";
        this.calories = 1080;
        ingredients = new List();
        ingredients.add("ketchup");
        ingredients.add("pickle");
        ingredients.add("onions");
        Ingredients.price = 5.00;
    }
}
```

```
public class Cheeseburger extends Meal {
    public Cheeseburger() {
        this.mealName = "Cheeseburger";
        this.drinkType = "Juice";
        this.friesSize = "large";
        this.calories = 840;
        ingredients = new List();
        ingredients.add("american cheese");
        ingredients.add("mustard");
        Ingredients.price = 6.00;
    }
}
```

```
public class QuarterPounder extends Meal {
    public QuarterPounder() {
        this.mealName = "Quarter Pounder";
        this.drinkType = "Milk";
        this.friesSize = "small";
        this.calories = 1050;
        ingredients = new List();
        ingredients.add("Onions");
        Ingredients.price = 4.50;
    }
}
```

c) Create a toString() method that uses StringBuilder to construct and return a String containing the information shown on the table above. In which class should you create this toString() method? (Superclass or subclass?)
It is better to place these methods in the superclass, because they will be the same for all the subclasses. By doing so, we avoid code duplication.

```
public String toString() {
    StringBuilder sb = new StringBuilder();
    sb.append("Meal Name: ").append(this.mealName).append("\n");
    sb.append(this.drinkType).append("\n");
    sb.append(this.friesSize).append(" French Fries").append("\n");
    sb.append(this.calories).append(" Cal.").append("\n");
    sb.append("ingredients: \n");
    for (String ingredient : ingredients) {
        sb.append(ingredient).append(", ");
    }
    return sb.toString();
}
```

4) Complete the main application, McDonaldApp, show below, as directed by the comment lines (10 points)

```
public class McDonaldApp {
    // define a list of meals
    List<Meal> listOfMeals = new ArrayList();

    // a car stops in the drive through and order 2 big macs,
    // 1 cheese burger and 1 quarter pounder,
    // so go ahead and create the respective objects for those
    // meals (no Scanner necessary, just hardcode the creation of
    // those meals here
    BigMac bg1 = new BigMac();
    BigMac bg2 = new BigMac();
    Cheeseburger cb = new Cheeseburger();
    QuarterPounder qp = new QuarterPounder();

    // add those meals to the list of meals
    listOfMeals.add(bg1);
    listOfMeals.add(bg2);
    listOfMeals.add(cb);
    listOfMeals.add(qp);

    // loop over the list of meals and print their info
    for (Meal currentMeal : listOfMeals) {
        System.out.println(currentMeal);  // no need to explicitly call
                                          // .toString(). JVM does it for us...
    }
```