Name _____ Period _____

Box Maker

Your Tasks (Mark these off as you go)

- ☐ Define key vocabulary
- ☐ Write your own *Box* class
- ☐ Add getter and setter methods to your *Box* class
- ☐ Add additional public methods to your *Box* class
- ☐ Add a toString method to your *Box* class
- ☐ Write the *BoxMaker* driver class
- ☐ Receive credit for this lab guide

☐ Define key vocabulary

Instance variable

Encapsulation (visibility)

Constructor

Constructor parameters

Private class members

Public class members

Accessor (getter) methods

Mutator (setter) methods

void method

instantiate (as it applies to java)

□ Write your own box class

Classes in java allow us to model other data types that are not built into the programming language and create as many subsequent objects of the class as needed. Consider, for example, how you might model an Element on the periodic table. An element has the following characteristics: atomic number, mass, and a symbol. Elements can also be defined as metals or nonmetals.

A class that could be used to model an element is model characteristics illustrated below,

```
public class Element{

    private int atomicNumber;
    private double atomicMass;
    private String symbol;
    private boolean isMetal;

    public Element(int atomicNumber, double atomicMass, String symbol, boolean isMe
tal){
        this.atomicNumber = atomicNumber;
        this.atomicMass = atomicMass;
        this.symbol = symbol;
        this.isMetal = isMetal;
    }

}
```

instance variables all declared a private

parameters used to initialize instance variables or attributes

constructor

"this" refers to the instance variables

In the above example the following are referred to as *instance variables*. Notice the instance variables are (1) not declared inside a method and (2) are declared as private.

```
    private int atomicNumber;
    private double atomicMass;
    private String symbol;
    private boolean isMetal;
```

The next block of code is referred to as the *constructor.* Notice the constructor takes the same name as the class. In our example the constructor also contains the parameters: atomicNumber, atomicMass, symbol, and isMetal. These parameters can be used to initialize the instance variables declared above. The *this* notation is used to reference the instance variables.

```
public Element(int atomicNumber, double atomicMass, String symbol, boolean isMeta
l){
        this.atomicNumber = atomicNumber;
        this.atomicMass = atomicMass;
        this.symbol = symbol;
        this.isMetal = isMetal;
    }
```

Consider a class that can be used to model a Box. Your Box class will need variables to store the dimensions (*width, height, and length*) of the box. All of these variables will be of type *private double.* Your Box class will also need a variable to keep track of whether or not the box is full (*isFull*). This will be of type *private boolean*. The Box constructor should accept the width, height, and length of the box as parameters and initialize the width, height, and length instance variables. Each newly created box will be empty. This means that you will need to initialize *isFull* to false in your constructor.

Write the Box class that meets the above requirements below.

☐ Add getter and setter methods to your Box class

In the Element class defined above, all the instance variables are private – that is, they are *encapsulated*. To access the values of these variables requires getter (or accessor) methods. An example of how a getter method could be used to access atomicMass is illustrated below,

```
public double getAtomicMass(){
    return atomicMass;
}
```

the datatype returned

Name of getter method

allows other classes access

value returned

Setter methods, also referred to as mutators, allow us to change the values of the instance values. An example of a setter method that could be used to modify the value of the atomicMass is illustrated below,

```
public void setAtomicMass(double atomicMass){
    this.atomicMass = atomicMass;
}
```

used to indicate no value is returned

allows other classes access

parameter used to set the value of the instance variable

"this" refers to the instance variable

The box class you wrote above has the following instance variables: *width*, *height*, *length*, and *isFull*. In the space below write getter and setter methods for each of these instance variables.

□ Add additional public methods to your Box class

In addition to getter and setter methods, you can include additional methods that provide more information about the datatype you are modeling. As an example, let's return to our Element class. In this class we have the following instance variables: *atomicNumber, atomicMass, symbol*, and *isMetal*. An additional method that calculates and returns the number of neutrons is illustrated below,

```
    public int numNeutrons(){
        return (int)Math.round(atomicMass) - atomicNumber;
    }
```

allows other classes access
data type returned
returns the value
calculates the neutrons

In the space below,

- Write a *public double volume()* method that will calculate and return the volume of the box based on the instance data values.
- Write a *public double surfaceArea()* method that will calculate and return the surface area of the box mased on the instance data values.

□ Write a toString method to your Box class

A *toString* method is a method that returns a String representation of an object. Using our Element class as an example, we could use a *toString* method to return a summary of our element,

```
public String toString(){

        String result = symbol + "has an atomic mass of " + atomicMass + ".\n";
        result += symbol + " has " + atomicNumber + " protons and " + numNeutro
ns() + " neutrons";

        return result;
    }
```

Write a *toString* method for your Box class. Your *toString* method should return a one-line description of the box. The description should provide the dimensions of the box and whether or not the box is full.

☐ Write the BoxMaker driver class

The driver class is a separate class that contains the main method used to "drive" your program. In the driver class, you can instantiate new objects of the data types you create using the *new* keyword. How to do this using our Element class is illustrated below,

```
Element carbon = new Element(6, 12.011, "C", false);
```

data type — new keyword used to instantiate data type

name — parameters

Once a data type has been instantiated, it can access any of the public methods you have created. You can access these methods using the "dot" notation. As an example, we could change the mass of the carbon by calling the setAtomicMass setter method,

```
Element carbon = new Element(6, 12.011, "C", false);

carbon.setAtomicMass(13);
```

sets the atomic mass of carbon to 13

Likewise, we can access the atomic mass by calling the getAtomicMass method,

```
double mass = carbon.getAtomicMass();
System.out.println(mass);//prints 13
```

Write the *BoxMaker* driver class. In the main method of this class write code to do the following,

- Create a *Box* variable called *smallBox*, and instantiate it with width 4, height 5, and depth 2.
- Print the one-line description of *smallBox* using its toString() method.
- Confirm that *smallBox*'s width, height, and depth getter methods are returning the correct values by printing the results of the appropriate method calls.
- Confirm that *smallBox*'s *volume()* and *surfaceArea()* methods are returning the correct values by printing the results of the appropriate method calls.
- Use *smallBox*'s setter methods to change it from ''empty'' to ''full'' and to change its dimension values to width 2, height 3, and depth 1.
- Print the one line description of *smallBox* using *toString()*, again, and confirm that all changed values are reflected.

□ Receive credit for this lab guide

Submit this portion of the lab to Pluska to receive credit for the lab guide. Once received, your completed code challenges will also be graded and will count towards your final lab grade.