

### COP2800C Module 3 Practice Exercise

In this practice exercise we will revise our box factory application from in order to practice new concepts from this module, including the use of arrays, enumerated types, and the **this** reference.

Expected output is as follows (as before, you can use different colors if you choose to do so). Note the additional output at the bottom where we are testing the new mutator and accessor (see below for how these are implemented)

```
Box Factory contents:
First Box is a RED box.
Second Box is a GREEN box.
And the final box is a BLUE box.
Testing mutators and accessors
first box: BLUE
second box: RED
third box: GREEN
```

- We start with the BoxFactory modifications by replacing our individual Box instance variables with an array variable that stores Boxes. We will populate this array in a new overloaded constructor.

```
private Box[] boxes;
```

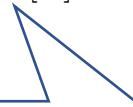
- After the default BoxFactory constructor we add a new overloaded constructor. Overloaded in this case refers to the modified constructor signature which accepts an array parameter of Box objects, whereas the default constructor accepts no parameters. This is our first encounter with the "this" reference. The naming convention used in Java for method and constructor parameters that are modifying instance variables is to use the same name parameter and the instance variable, in this case "boxes". In order to use the same name we need to differentiate between the parameter and the instance variable, so we precede the instance variable name with **this** in the assignment statement; this refers to the current object that we are dealing with, in this case it is our BoxFactory object created in the main method.

```
public BoxFactory(Box[] boxes) {
    this.boxes = boxes;
}
```

- Let's add a toString method for our BoxFactory so we can view its state. Notice that we are printing the Box object states using an indexed array reference preceded by **this**. Even though there are no conflicts in parameter names here, a good practice for consistency is to continue to reference the instance variable with this, but it is not required. Since each one of the array elements is a Box we can just use the Box class's toString method to display the state of each box.

```
// toString
@Override
public String toString() {
    String retString = "Box Factory contents:\n"; // start here

    // append the boxes
    retString += "First Box is a " + this.bboxes[0] + " box.\n";
    retString += "Second Box is a " + this.bboxes[1] + " box.\n";
    retString += "And the final box is a " +
                this.bboxes[2] + " box.";
    return retString;
}
```



These call the Box class's  
toString() method

- Before we make the changes to the main method in the BoxFactory class, let's make the changes to our underlying Box and Color implementations. First, we will delete the Color class from the previous module and replace it with an enumerated type, which is a much simpler, consistent, and type-safe way to represent the color values.

```
// enumerated type to represent color values
enum Color {
    NO_COLOR, RED, GREEN, BLUE;
}
```

- Now down to our Box modifications. Our boxColor instance variable is now no longer an int, it is truly a Color! This demonstrates the type safety characteristic of enumerated types; Color is treated as a data type (even though the underlying representation of enumerated types is int); Java enforces the use of the enumerated type name Color throughout the code.

```
// color is the only attribute
private Color boxColor = Color.NO_COLOR;
```

- Change our overloaded constructor to now use the Color type instead of int, and also use the parameter name "boxColor" since we have the **this** reference available to differentiate between the parameter and the instance variable as described above.

```
// overloaded constructor which accepts a color parameter
public Box(Color boxColor) {
    this.boxColor = boxColor;
}
```

- Add an accessor and mutator for our instance variable. Note the use of **this** in both. It is not strictly required for the accessor, but done for consistency. It is required for the mutator.

```
// accessor
public Color getBoxColor() {
    return this.boxColor;
}

// mutator
public void setBoxColor(Color boxColor) {
    this.boxColor = boxColor;
}
```

- Finally, our toString method in the Box class can now print the name of the color using the name() method, one of the features of enumerated types. If we wanted to print the name in lower case we could chain the name() call with the String's toLowerCase() method as shown in the commented return statement.

```
@Override
public String toString() {
    return this.boxColor.name();
    //return this.boxColor.name().toLowerCase();
}
```

- Now let's finish up by modifying our unit tests (the main method in the BoxFactory class). Declare a local array variable to store three boxes:

```
// create a local array of boxes
Box[] boxArray = new Box[NUMBER_OF_BOXES];
```

- It is critical to note here that this array does not contain any instantiated boxes at this point, we have merely allocated an array with the required number of elements using the NUMBER\_OF\_BOXES size declarator (3). The current value of all of three elements in the boxArray array is null (see table below). If we try to manipulate any of these elements right now we would run into problems that would crash our program with a NullPointerException.

Array element	Current Value in Memory
boxArray[0]	null
boxArray[1]	null
boxArray[2]	null

- Now add boxes to the array:

```
// add boxes to the array
boxArray[0] = new Box(Color.RED);
boxArray[1] = new Box(Color.GREEN);
boxArray[2] = new Box(Color.BLUE);
```

- Here's what the array looks like in memory now:

Array element	Current Value in Memory
boxArray[0]	Red box object
boxArray[1]	Green box object
boxArray[2]	Blue box object

- Now that we have populated our array, create a BoxFactory object and insert the array into that object using our new overloaded constructor:

```
// use the overloaded constructor to create our Box factory
BoxFactory boxFact = new BoxFactory(boxArray);
```

- Using our new toString method for the BoxFactory, print its state:

```
// print the state of the factory
System.out.println(boxFact);
```

- Since we added a mutator and accessor to the Box class, let's change the state of the box objects in the array to test them:

```
// make some changes to test the mutator
System.out.println("Testing mutators and accessors");
boxFact.boxArray[0].setBoxColor(Color.BLUE);
boxFact.boxArray[1].setBoxColor(Color.RED);
boxFact.boxArray[2].setBoxColor(Color.GREEN);

// test the accessor
System.out.println("first box: " + boxFact.boxArray[0].getBoxColor());
System.out.println("second box: " + boxFact.boxArray[1].getBoxColor());
System.out.println("third box: " + boxFact.boxArray[2].getBoxColor());
```