Chapter 6 Exercise Solutions

EX 6.1. Write a method called `average` that accepts two integer parameters and returns their average as a floating point value.

```
public double average (int num1, int num2)
{
   return (num1 + num2) / 2.0;
}
```

EX 6.2. Overload the `average` method of Exercise 6.1 such that if three integers are provided as parameters, the method returns the average of all three.

```
public double average (int num1, int num2, int num3)
{
   return (num1 + num2 + num3) / 3.0;
}
```

EX 6.3. Overload the `average` method of Exercise 6.1 to accept four integer parameters and return their average.

```
public double average (int num1, int num2, int num3,
   int num4)
{
   return (num1 + num2 + num3 + num4) / 4.0;
}
```

EX 6.4. Write a method called `multiConcat` that takes a `String` and an integer as parameters. Return a `String` that consists of the string parameter concatenated with itself `count` times, where `count` is the integer parameter. For example, if the parameter values are `"hi"` and 4, the return value is `"hihihihi"`. Return the original string if the integer parameter is less than 2.

```
public String multiConcat (String text, int count)
{
   String result = text;

   if (repeats > 1)
      for (int i=2; i <= count; i++)
         result += text;

   return result;
}
```

EX 6.5. Overload the `multiConcat` method from Exercise 6.4 such that if the integer parameter is not provided, the method returns the string concatenated with itself. For example, if the parameter is `"test"`, the return value is `"testtest"`

```
public String multiConcat (String text)
{
   return text + text;
```

```
}
```

EX 6.6. Write a method called `drawCircle` that draws a circle based on the method's parameters: a `Graphics` object through which to draw the circle, two integer values representing the (x, y) coordinates of the center of the circle, another integer that represents the circle's radius, and a `Color` object that defines the circle's color. The method does not return anything.

```
public void drawCircle (Graphics page, int x, int y,
   int rad, Color color)
{
   page.setColor (color);
   page.drawOval (x–rad, y–rad, rad*2, rad*2);
}
```

EX 6.7. Overload the `drawCircle` method of Exercise 6.6 such that if the `Color` parameter is not provided, the circle's color will default to black.

```
public void drawCircle (Graphics page, int x, int y,
   int rad)
{
   page.setColor (Color.black);
   page.drawOval (x–rad, y–rad, rad*2, rad*2);
}
```

EX 6.8. Overload the `drawCircle` method of Exercise 6.6 such that if the radius is not provided, a random radius in the range 10 to 100 (inclusive) will be used.

```
public void drawCircle (Graphics page, int x, int y,
   Color color)
{
   page.setColor (color);
   Random generator = new Random();
   int rad = generator.nextInt(91) + 10;
   page.drawOval (x–rad, y–rad, rad*2, rad*2);
}
```

EX 6.9. Overload the `drawCircle` method of Exercise 6.6 such that if both the color and the radius of the circle are not provided, the color will default to red and the radius will default to 40.

```
public void drawCircle (Graphics page, int x, int y)
{
   int rad = 40;
   page.setColor (Color.red);
   page.drawOval(x–rad, y–rad, rad*2, rad*2);
}
```

EX 6.10. Discuss the manner in which Java passes parameters to a method. Is this technique consistent between primitive types and objects? Explain.

Java passes all parameters by value*. This means that the current value of the actual parameter is copied into the formal parameter in the method header. This technique is consistent between primitive types and objects because object* references *rather than objects themselves are passed. When an object (actually, an object reference) is passed, the current value of the reference (the object's address) is copied into the corresponding formal parameter in the method header.*

EX 6.11. Explain why a static method cannot refer to an instance variable.

A static method is invoked through a class rather than through an object of the class. No object of the class needs to be instantiated in order to invoke a static method. If no object is instantiated, no instance variable exists. Hence, a static method cannot refer to an instance variable.

EX 6.12. Can a class implement two interfaces that each contains the same method signature? Explain.

Yes. The class which implements an interface provides method implementations for each of the abstract methods defined in the interface. In satisfying the requirements for a method of one interface, it simultaneously satisfies the requirements for a method with the same signature in another interface.

EX 6.13. Create an interface called `Visible` that includes two methods: `makeVisible` and `makeInvisible`. Both methods should take no parameters and should return a `boolean` result. Describe how a class might implement this interface.

```
public interface Visible
{
   public boolean makeVisible();
   public boolean makeInvisible();
}
```

A class implementing `Visible` would include an `implements` clause in the class header, such as:

```
public class ControlPanel implements Visible
```

The class would contain, among other things, two methods with signatures that match those specified in the interface.

EX 6.14. Draw a UML class diagram that shows the relationships among the elements of the previous exercise.

EX 6.15. Create an interface called `VCR` that has methods that represent the standard operations on a video cassette recorder (play, stop, etc.). Define the method signatures any way you desire. Describe how a class might implement this interface.

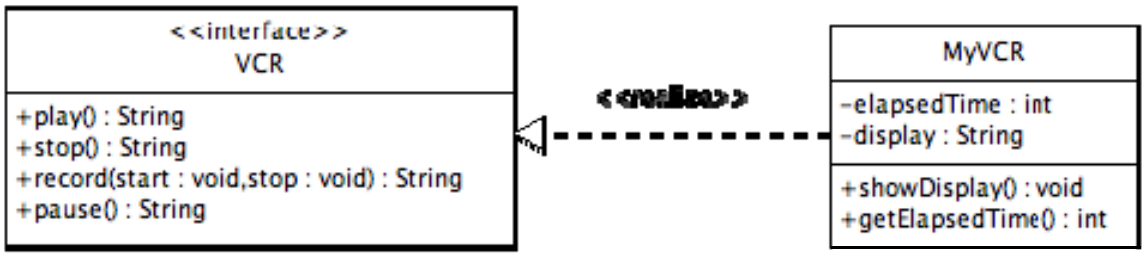
```
public interface VCR
{
   public String play();
   public String stop();
   public String record(int start, int end);
   public String pause();
}
```

A class implementing VCR would include an `implements` clause in the class header, such as:

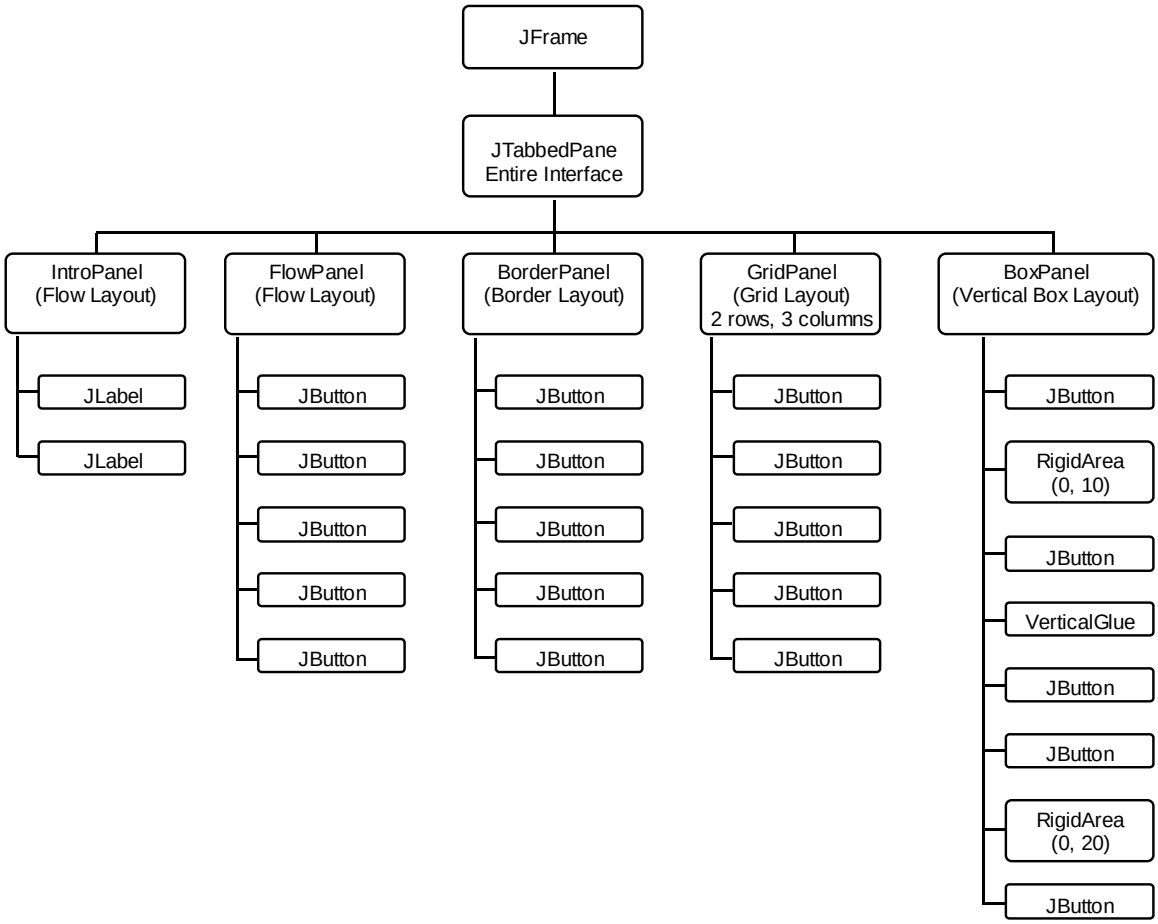
```
public class MyVCR implements VCR
```

The class would contain, among other things, four methods with signatures that match those specified in the interface.

EX 6.16. Draw a UML class diagram that shows the relationships among the elements of the Exercise 6.15.



EX 6.17. Draw the containment hierarchy for the LayoutDemo program.



EX 6.18. What visual effect would result by changing the horizontal and vertical gaps on the border layout used in the `LayoutDemo` program? Make the change to test your answer.

The panel used to display the buttons has a green background, but no green is visible in the display. By default, there are no horizontal or vertical gaps between the areas of a border layout. These gaps can be set with an overloaded constructor or with explicit methods of the `BorderLayout` class. If the gaps are provided, the underlying (green) panel shows through.

EX 6.19. Write the lines of code that will define a compound border using three borders. Use a line border on the inner edge, an etched border on the outer edge, and a raised bevel border in between.

```
//create borders
Border line = BorderFactory.createLineBorder (Color.blue);
Border bevel = BorderFactory.createRaisedBevelBorder();
Border etched = BorderFactory.createEtchedBorder();

// create inner compound border
Border compound = BorderFactory.createCompoundBorder
   (bevel, line);
```

```
// create final compound border
Border final = BorderFactory.createCompoundBorder
   (etched, compound);
```