Problem Set 3, Part I

Please start your answer to each problem on a new page, as we have done below!

Problem 1: Our Rectangle class revisited

1-1)

type of method: mutator
header: public void rotate()

1-2)

type of method: accessor
header: public boolean largerThan(Rectangle other)

1-3)

problems in code: The code cannot compile since appropriate encapsulation makes the fields of the Rectangle class private and makes clients outside of the class unable to directly access the fields. So, when the client code tries to directly retrieve the fields of r1 (i.e. r1.width; r1.width) and even make changes to them, the code cannot compile due to the inaccessibility of these fields without calling getter methods.

rewritten version:

```
Rectangle r1 = new Rectangle(60, 80);
System.out.println("r1's height is: " + r1.getHeight());
r1.setWidth(r1.getWidth() + 20));
System.out.println(r1);     // print the new dimensions
```

Problem 2: A class that needs your help

2-1) *Revise the code found below:*

```
public class ValuePair {
    private int a;
    private double b;

    public static double product() {
        return this.a * this.b;
    }

    // add the new methods here

     public int getA() {
          return this.a;
     }

     public double getB() {
          return this.b;
     }

     public void setA(int newA) {
          if (newA % 2 == 0) {    // test if newA is an odd number
               throw new IllegalArgumentException();
          }
          this.a = newA;
     }

     public void setB(double newB) {
          if (newB <= 0.0) {    // test if newB is greater than 0.0
               throw new IllegalArgumentException();
          }
          this.b = newB;
     }

     public ValuePair(int valueA, double valueB) {
          this.setA(valueA);
          this.setB(valueB);
     }

}
```

2-2)

```
a)   ValuePair vp1 = new ValuePair(7, 15.5);
b)   vp1.setA(25);
c)   double b1 = vp1.getB();
d)   vp2.setA(vp2.getA() + 2);
```

Problem 3: Static vs. non-static

3-1)

type and name of the variable	**static or non-static?**	**purpose of the variable, and why it needs to be static or non-static**
double rawScore	non-static	stores the raw score associated with a given Grade object; needs to be non-static so every Grade object will have its own instance of this variable
int latePenalty	non-static	stores the late penalty associated with a given Grade object; needs to be non-static so every Grade object will have its own instance of this variable
String typeGrade	non-static	stores the category associated with a given Grade object, which is one from “assignment”, “quiz”, or “exam”; needs to be non-static so every Grade object will have its own instance of this variable
int numAssignment	static	keeps track of the total number of Grade objects created for the “assignment” category; needs to be static so there is only one numAssignment for all instances of the class.
int numQuiz	static	keeps track of the total number of Grade objects created for the “quiz” category; needs to be static so there is only one numQuiz for all instances of the class.
int numExam	static	keeps track of the total number of Grade objects created for the “exam” category; needs to be static so there is only one numExam for all instances of the class.

3-2)

a) static or non-static?: non-static
explanation: This method is to access and alter typeGrade — an internal of a Grade object — which means that only when there is a valid instance of the Grade class does this method work.

b) changes it would need to make: first, the category of the Grade object needs to be changed, so typeGrade will be changed from “quiz” to “exam”; second, the total number of Grade objects created for the “quiz” category decreases by 1 (i.e. numQuiz--); third, the total number of Grade objects created for the “exam” category increases by 1 (i.e. numExam++).

3-3)

a) static or non-static?: non-static

explanation: This method is to access and alter latePenalty — an internal of a Grade object — which means that only when there is a valid instance of the Grade class does this method work.

b) example of calling it: g.waiveLatePenalty()

3-4)

a) static or non-static?: static

explanation: This method is to only make use of the two parameters and do some calculations, and there is no need for it to access the internals of any Grade object.

b) example of calling it: Grade.computeRaw(90.0, 100)