

1.00/1.001 Introduction to Computers and Engineering Problem Solving

Quiz 1 October 3, 2003

Name:	
Email Address:	
TA:	
Section:	

You have 90 minutes to complete this exam. For coding questions, you do not need to include comments, and you should assume that all necessary files have already been imported.

Good luck.

<i>Question</i>	<i>Points</i>
Question 1	/ 30
Question 2	/ 30
Question 3	/ 40
Total	/ 100

Problem 1.

Circle TRUE or FALSE for each of the following statements:

1. The size of a `Vector` is less than or equal to the capacity of a `Vector`.

TRUE

FALSE

2. A `do` loop will be executed at least once even if the boolean condition being tested is false.

TRUE

FALSE

3. Both an array and a `Vector` can grow automatically as needed.

TRUE

FALSE

4. Converting a `float` to an `int` requires explicit casting.

TRUE

FALSE

5. `String` is a Java primitive data type.

TRUE

FALSE

6. If two `public` classes are defined in separate files, but belong to the same package, they can access each other's `private` variables.

TRUE

FALSE

7. A `static` method cannot have a reference to an object as an argument.

TRUE

FALSE

8. `double x = (double) 5 / 4` will be evaluated as 1.25 and this value will be stored in `x`

TRUE

FALSE

9. A non-`static` method cannot access `static` variables.

TRUE

FALSE

10. `Static` methods and variables can be used even if no objects of the class have been instantiated with the `new` operator

TRUE

FALSE

11. A `for` loop can always be replaced by an appropriate `while` loop

TRUE

FALSE

12. Suppose `x` is an `int`. `Integer.parseInt(" " + x)` evaluates to the same as `x`

TRUE

FALSE

13. A `float` can be cast to a `double` without losing precision

TRUE

FALSE

14. `Math.pow(4, 1/2)` will return `2.0`

TRUE

FALSE

15. Every class has at least one constructor.

TRUE

FALSE

Problem 2.

Consider the following class `Point`. It is used to represent a Cartesian coordinate:

`Point.java`:

```
public class Point
{
    private int x;
    private int y;

    public Point(int anX, int aY)
    {
        x = anX;
        y = aY;
    }

    public int getX()
    {
        return x;
    }

    public int getY()
    {
        return y;
    }
}
```

Part 1, `add()`: Create a new method for this class named `add()`. Its signature is listed below. As an example of its operation, if `a` and `b` are points that represent (1,2) and (3,3) respectively, then invoking the method `a.add(b)` will return the `Point` representing (4,5). After invoking `add` neither `Point a` or `b` should be modified. Enter the code that completes this method in the box below.

```
public Point add(Point b)
{
    int newX = this.x + b.x;
    int newY = this.y + b.y;
    return new Point(newX, newY);
}
```

Part 2, toString(): Create a new method for this class named `toString()`. If the `Point` represents the Cartesian coordinate (1, 2) then this method should return a `String` equal to "(1, 2)". Enter the code that completes this method in the box below.

```
public String toString()  
{  
    return "(" + x + ", " + y + " )";  
}
```

Part 3, Using your new methods: Having completed your new methods, you're ready to test them out. You create a new class named `PointTest` in the file `PointTest.java` and define a `main()` method inside of it. To test your methods you want to do the following:

1. Create two `Point` objects. One should represent (-1, -5) and the other should represent (3, 4).
2. Print out these two `Points`.
3. Add the two points together and print out the resulting `Point`.

Enter the code that completes these three steps in the box below.

```
public static void main(String[] args)  
{  
    Point one = new Point(-1, -5);  
    Point two = new Point(3, 4);  
    System.out.println(one);  
    System.out.println(two);  
    Point three = one.add(two);  
    System.out.println(three);  
}
```

Problem 3.

Part 1

Complete the method `approximate_PI()` whose signature is shown below. The method should return an approximate value of π using the first 1000 terms of the following infinite series:

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \dots$$

Note: Do **NOT** use `Math.PI`

```
public static double approximate_PI()  
{
```

```
    double result = 0.0;  
  
    // Loop repeated 500 times, each iteration adds  
    // 2 elements to the series  
  
    for (int i =1; i < 2000 ; i+=4)  
        result += 4.0/i - 4.0/(i+2);  
  
    return result;
```

```
}
```

Part 2

Bonifacci, one of Fibonacci's fiercest competitors, decides to invent a new sequence to intimidate his rival. He baptizes his sequence `bonifacci` and defines it as follows: 1, 2, 5, 12, 29, 70, 169... He also sets the starting index to 0, and defines the base elements as `bonifacci(0) = 1` and `bonifacci(1) = 2`. The subsequent elements of the series verify the following:

```
bonifacci(2) = 2×2 +1  
bonifacci(3) = 2×5 +2  
bonifacci(4) = 2×12 +5  
bonifacci(5) = 2×29 +12  
and so on...
```

Based on the pattern above, complete the following method, which is a recursive implementation of `bonifacci`. Assume you correctly receive a positive integer argument $n \geq 0$.

```
public int bonifacci (int n)
```

```
{  
    if (n == 0 || n==1)  
        return n+1;  
  
    return 2*bonifacci(n-1)+bonifacci(n-2);  
}
```