COP 3330 Sample Exam #2: Classes, Inheritance, Interfaces, Abstract Classes Solutions

A Complex Number is of the form a + bi, where both a and b are real and i = $\sqrt{-1}$. An incomplete version of this class will be given and several questions will consist of adding methods to this class. These numbers can be added, subtracted, multiplied and divided. They can be raised to the power of a non-negative integer. (They can also be raised to other powers, but that's complicated and beyond the scope of standard knowledge.) In addition, one question will consist of utilizing Complex Number objects. **Complex Number objects will be immutable. Once one is created, neither of its instance variables will be changed.**

Here is an incomplete version of the class:

```
public class ComplexNumber {

    private double a;
    private double b;

    public ComplexNumber(double real) {
        a = real;
        b = 0;
    }

    public ComplexNumber(double real, double img) {
        a = real;
        b = img;
    }

    // Other methods not shown.
}
```

1) (4 pts) Complete the method below to add the ComplexNumber other to this ComplexNumber and return the result. Your code should be one line long.

```
public ComplexNumber add(ComplexNumber other) {
    return new ComplexNumber(a+other.a, b+other.b);
}
```

Grading: 1 pt return, 1 pt new, 1 pt ComplexNumber, 1 pt both parameters correct

2) (4 pts) Complete the method below to subtract the ComplexNumber other from this ComplexNumber and return the result. Your code should be one line long. (Note: (4 + 3i) – (2 – 2i) equals 2 + 5i, where this = 4+3i and other = 2 – 2i.)

```
public ComplexNumber subtract(ComplexNumber other) {
    return new ComplexNumber(a-other.a, b-other.b);
}
```

Grading: 1 pt return, 1 pt new, 1 pt ComplexNumber, 1 pt both parameters correct

3) (6 pts) Complete the method below to multiply the ComplexNumber other from this ComplexNumber and return the result.

```
public ComplexNumber multiply(ComplexNumber other) {
    double real = a*other.a - b*other.b;
    double img = a*other.b + b*other.a;
    return new ComplexNumber(real, img);
}
```

Grading: 2 pts computation real part, 2 pts computation imaginary part, 2 pts rest

4) (2 pts) Complete the method below to return the complex conjugate of this. Note: The complex conjugate of a + bi is a – bi.

```
public ComplexNumber conjugate() {
    return new ComplexNumber(a, -b);
}
```

Grading: 1 pt for both return and new, 1 pt for ComplexNumber(a, -b), has to be completely correct to get 2 pts.

5) (8 pts) Complete the method below to divide the ComplexNumber this by the ComplexNumber other and return the result. You may assume that other isn't equal to 0. For full credit, you **MUST CALL** both the multiply method and the conjugate method in your solution.

```
public ComplexNumber divide(ComplexNumber other) {
    ComplexNumber term = other.conjugate();
    ComplexNumber num = this.multiply(term);
    double den = other.a*other.a + other.b*other.b;
    return new ComplexNumber( num.a/den, num.b/den);
}
```

Grading: 2 pts for each line above

6) (8 pts) Complete the method below to return the result of taking this ComplexNumber to the power, exp. You may assume that exp is non-negative.

```
public ComplexNumber pow(int exp) {

    ComplexNumber res = new ComplexNumber(1, 0);
    for (int i=0; i<exp; i++)
        res = res.multiply(this);
    return res;
}
```

Grading: 2 pts initialize res to 1+0i.
1 pt for loop.
4 pts multiply correctly with this (1 pt LHS, 1 pt res, 1 pt .multiply, 1 pt this).
1 pt return.

7) (21 pts) What is the output of the code handout? (It's a separate piece of paper, front and back.) There will be 17 lines of output. The first 13 lines are worth 1 point each, the last four lines are worth 2 points each.

```
def cons A

cons A int

cons B 2 int

def cons A

def cons B

def cons C

def cons A

def cons B

cons C int

A 5

B 8 4

C 5 2 4

C 5 2 7

A 6

B 8 11

C 5 4 7

C 5 11 7
```

Grading: 1 pt for the first 13 lines (must be completely correct to get the point), 2 pts for the last four. For the last four, give 1 pt if you think it's close but isn't completely correct.

8) (7 pts) Consider implementing the following interface:

```
public interface ResearchPaper {
    String getNthAuthor(int n);
}
```

Here is an incomplete version of the class MathArticle, which implements this interface:

```
public class MathArticle implements ResearchPaper {

    private int numAuthors;
    private String[] authorList;
    // other instance variables.

}
```

Write an implementation of the getNthAuthor method for the MathArticle class which works as follows: if n = 1, then the String in index 0 of authorList is returned. More generally, if n is less than or equal to numAuthors, then the String in index n – 1 or authorList is returned. If n is less than 1 or greater than numAuthors, then null should be returned.

```
public String getNthAuthor(int n) {

    if (n<1 || n>numAuthors) return null;

    return authorList[n-1];
}
```

Grading: 4 pts for returning in null case (1 pt if, 1 pt n<1, 1 pt n>numAuthors, 1 pt return null), 3 pts regular return (1 pt return, 1 pt authorList, 1 pt n-1)

9) (12 pts) Consider a class called EloChange which has two instance variables:

```
private int oldEloScore;
private int newEloScore;
```

For this question you'll write code so that this class implements the Comparable<EloChange> interface. Specifically, write the compareTo method so that if this object's change in elo score (newEloScore – oldEloScore) is greater than other's change in elo score, then a negative integer is returned. If this difference is equal, return 0. Otherwise return a positive integer. With this implementation, if the following array of EloChange: [(1500, 1600), (1300, 1350), (2000, 2200), (1750, 1900), (1800, 1700)] were sorted the sorted array would look like this after the sort: [(2000, 2200), (1750, 1900), (1500, 1600), (1300, 1350), (1800, 1700)].

Put your implementation of compareTo below:

```
public int compareTo(EloChange other) {

    return (oldEloScore-newEloScore) - (other.oldEloScore-other.newEloScore);
}
```

Grading: If they write a one liner like above, give the following # of points:

12 pts if fully correct
8 pts if they return the opposite of what they are supposed to
4 pts if they have the right four terms in their expression but are returning neither the correct expression nor the negative of the correct expression
< 4 otherwise

If they break into cases, give 4 pts per case, again 8 total points if they swap the non-zero cases.

Give other amounts of partial as you see fit staying consistent with this framework.

Note that there are several ways to write what is above that are mathematically equivalent.

10) (3 pts) What two primary tastes does one experience when eating Sweet and Sour Chicken?

Sweet, Sour **(Grading: Give to all)**