Computer & Information Science

1. (20) A rational number is a number that can be represented by a pair of integers—a numerator and a denominator. Write a class `Rational` that represents rational numbers. The class should support the following behavior:

   - A constructor that accepts a numerator and a denominator
   - A constructor that accepts a single integer representing a whole number (i.e., with a denominator of 1).
   - A default constructor that initializes the `Rational` to 1 (1/1).
   - (Java) A `toString` method that returns the string representation *num / denom*.
     (C++) A << operator that prints *num / denom*
   - A `multiply` method/member function that returns a new `Rational` consisting of the product of the receiver and (`Rational`) parameter.
   - An `inverse` method that returns a new `Rational` representing the inverse (reciprocal) of the receiver.
   - A `divide` method that returns the quotient of the receiver and (`Rational`) parameter. Remember --to divide two rationals, you multiply the first by the inverse of the second.
   - An `equals` method that returns whether two `Rational`s are the same. Assume 2 / 4 is <u>not</u> equal to 1 / 2 (i.e., you don't have to deal with reducing the rationals to lowest form).

2. Suppose you had the following `Shape` interface:

```
interface Shape {
   int area();
   int perimeter();        // sum of the lengths of all sides
   int numberOfSides();
};
```

and you wanted to code `Rectangle` and `Square` classes, both implementing `Shape`.

Here are some design choices.

i	`class Rectangle implements Shape`
	`class Square implements Shape`
		totally independent

ii	`class Rectangle implements Shape`
	`class Square extends Rectangle`
		Square ***extends*** *Rectangle*

iii	`class Square implements Shape`
	`class Rectangle extends Square`
		Rectangle ***extends*** *Square*

a) Implement each of the above (i.e., for each implementation, code the `Rectangle` and `Square` classes) — try to take advantage of as much OOP as possible. If you find yourself unhappy with one or more of the implementations, all that counts is that the methods work properly

b) Compare the implementations—which was best? worst? Justify your answers.

3.

a. What is the difference between overloading and overriding? Give an example of each.

b. What is the difference between an interface and a class in Java?

c. What does overidding have to do with polymorphism?

d. Explain how the toString method of Java is a classic example of polymorphism.

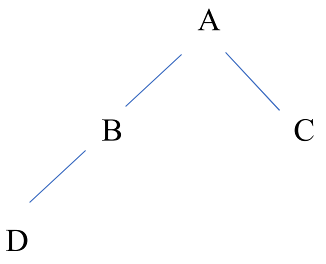
e. Explain why C++ cannot for all practical purposes have a toString method.

f. What are the two things required to make have a function act polymorphically in C++?

4. **This question should be answered using Java.**

   a. What is an upcast? What is a downcast?

   b. Given the hierarchy



   Give a least two examples of an upcast and two examples of a downcast (declare/create any variables/objects you want).

   c. Are upcasts always legal? If not, under what circumstances would the upcast be legal? Illegal?

   d. Are downcasts always legal? If not, under what circumstances would the downcast be legal? Illegal?

   e. Using the above hierarchy, give an example of two classes that can NEVER be cast to each other.

   f. Explain why no cast is required when placing items into a collection object (e.g. a `Vector`, or `Set`) but a cast is usually required when removing the item in order to invoke the object's methods.