Introduction to Java

CS9053

Prof. Dean Christakos

MIDTERM

Friday, March 3rd, 2023

DUE: Wednesday, March 10th, 2023

NO GRACE PERIOD OR EXTENSION

Part I: Written questions

1. What are two uses of the “this” keyword? Explain what it means.
2. If you’re in an interview and someone asks, “Is Java Compiled or Interpreted?” how would you respond?
3. Take a look at this code, which defines two classes::

public class Vehicle {

double cargoSpace;

String color;

int wheels;

public Vehicle(String color, int wheels, double cargoSpace) {

this.color = color;

this.wheels = wheels;

this.cargoSpace = cargoSpace;

}

Public String toString() {

Return “Vehicle [color=” + this.color +”, wheels=” + this.wheels + “, cargoSpace = “+this.cargoSpace+”]”;

}

public class Car extends Vehicle {

int doors;

public Car(String color, int wheels, double cargoSpace, int doors) {

super(color, wheels, cargoSpace);

this.doors = doors;

}

public String toString() {

return “Car [“+super.toString()+”, doors=” + this.doors +”]”;

}

Ignore any syntax errors or wrongly capitalized key words. Assume the code compiles and does “the right thing”.

Now, let’s say I create two objects:

Vehicle v1 = new Vehicle(“black”, 2, 0);

Vehicle v2 = new Car(4, “red”, 4, 20);

1. Why can I assign a Car object to a Vehicle reference variable v2 without any errors but Car c = v2; will give a compile error?
2. v1.toString() will return “Vehicle [color=black, wheels=2, cargoSpace = 0”. However, v2.String() will return “Car [Vehicle [color=”red”, wheels=4, cargoSapce=20], doors=4]”. What accounts for this difference even though v1 and v2 are both Vehicle reference variables?
3. Let’s say you wanted to create an exception specifically for an invalid radius in the Circle object:

public class InvalidRadiusException extends Exception {

public InvalidRadiusException(double radius) {

super(“Invalid Radius: “ + radius);

}

}

public Circle(double radius) throws InvalidRadiusException {

if (radius < 0) {

throw new InvalidRadiusException(radius);

} else {

this.radius = radius;

}

}

The code Circle c = new Circle(5); won’t compile because you need to catch the exception. Why is this? Why don’t you have to catch Exceptions for other methods like the NumberFormatException in Integer.parseInt? How would you change InvalidRadiusException so you stopped getting a compile error when calling the Circle constructor outside of a try/catch block?

1. Take this sample class hierarchy:

A picture containing line chart

Description automatically generated

Looking at the classes:

Person

Customer

Employee

IndividualContributor

Manager

Executive

Create a class hierarchy in the pattern of the Fruit hierarchy above. Justify your decisions.

1. Why would someone use declare an Interface and have classes implement an Interface vs. using a class/subclass structure?
2. Assume GeometricObject is declared as abstract, and it has a method getArea(), also declared as abstract, and thus has no implementation. Why is there always a guarantee than any code that calls getArea() on a GeometricObject will always return a result?
3. Why does an Interface with only a single abstract method defined in it allow us to use lambda functions?
4. Refer back to the Vehicle hierarchy in an earlier problem set. If we wanted to write a method that processed data in an ArrayList of Vehicle objects, why would it be preferable to declare a method like this:

public void processData(ArrayList<? extends Vehicle> objList)

Instead of

public void processData(ArrayList<Vehicle> objList)

?

1. In this code:

Double d1 = 5.5;

Double d2 = 5.5;

String s1 = “Hello”;

String s2 = “Hello”;

Why does (d1 == d2) evaluate to “false” but (s1 == s2) evaluates to “true”?

**Part II: Coding**

Question 1: Inheritance

Create an object hierarchy for people within a university.

* The base class is called **Person**. A person has an ID and an age. The ID should be autogenerated and unique.
* There should be the following subtypes:
  + **Employee**
  + **Student**
  + Affiliate
  + Faculty
  + GraduateStudent
  + Staff
  + Undergraduate
* Your first hint is that an Affiliate is a type of Person
* All classes in bold should not be instantiable—they are purely categories of **Person** classes
* Some classes have specific attributes:
  + Employee objects have a salary
  + Student and Faculty objects have a department
  + Faculty have an attribute indicating if they are tenure track or not
  + Undergraduate objects have an attribute indicating if they are matriculated or not
  + GraduateStudent objects have an attribute indicating if they are PhD students or not
  + The toString method should be implemented with appropriately descriptive output showing all the attributes of the object.
* Provide an equals method for each concrete class.
* In the class Person, there’s a method printPersons. This should take an ArrayList parameterized with Person or subclasses of Person and iterate over the ArrayList and print out the result of toString for each object in the ArrayList

Question 2: Complex Matrix

1. Complex numbers are of the form *a + ib*, where *a* is the real part and *b* represents the imaginary part, multiplied by *i*, . For Java purposes, *a* and *b* are double-precision floating point numbers. The magnitude of a complex number is

To add two complex numbers, add the real parts and then add the imaginary parts. For example, to add (*a0* + *ib0*) to (*a1* + *ib1*), the result would be [(*a0*+*a1*) + i(*b0*+*b1*)]

Multiplying two complex numbers follows algebraic laws of multiplication, such that

(*a0* + *ib0*) \* (*a1* + *ib1*) = (*a0 a1 + ia0b1* + *ib0a1* - *b0b1*) = (*a0 a1 - b0b1) + i(a0b1+ b0a1)*

Dividing two complex numbers requires separating out the real and complex parts, so:

1. Create a class called Complex which implements complex numbers. It should contain the following methods:

Complex() – for which the real and imaginary parts are both zero

Complex(double a) – which sets the real part to a and the imaginary part to 0

Complex(double a, double b) – sets the real part to a and the imaginary part to *b*

getReal() – returns the real part of the complex number

setReal(double a) – sets the real part of the complex number

getImaginary() – returns the imaginary part of the complex number

setImaginary(double b) – sets the imaginary part of the complex number

toString() which returns a String of the format “a + ib”

getMagnitude() – returns the magnitude of the complex number

add(Complex c) – returns a Complex number representing the sum of the object and the Complex number c

subtract(Complex c) - returns a Complex number representing the difference between the object and the Complex number c.

multiply(Complex c) - returns a Complex number representing the product of the object and the Complex number c

divide(Complex c) - returns a Complex number representing the quotient of the object divided by the Complex number c

Complex should also implement the Comparable interface and the results should be based on the magnitude of the two imaginary numbers

1. Now, you are going to implement a class called ComplexMatrix that implements an m by n matrix of Complex numbers you implemented above.

It should have the following methods:

ComplexMatrix(int m, int n) – an m x n matrix of 0-value complex numbers

ComplexMatrix(Complex[][] input) – a matrix corresponding to the two dimensional array contained in input

You will implement the following operations, which will throw a MatrixDimensionException if the operation is not possible due to dimensional incompatibility

public ComplexMatrix add(ComplexMatrix cm) – adds the object to the matrix cm and returns a ComplexMatrix

public ComplexMatrix mult(ComplexMatrix cm) – multiplies the object with the matrix cm and returns a ComplexMatrix

public String toString() – prints out a representation of the matrix where each element of the matrix follows the toString format of Complex

Next, you will notice three files, complexmatrix1.txt and complexmatrix2.txt and complexmatrix3.txt. You should be able to read these in to create a complex matrix object. Each member of the matrix has the format a\_b where *a* is the real part and *b* is the imaginary part. You must now implement this static method:

public static ComplexMatrix read(String filename) – this will read in the files like complexmatrix1.txt and complexmatrix2.txt and complexmatrix3.txt and return a ComplexMatrix object. This will require you to have to parse in data. The String.split() method will come in handy here, as you can pass arguments to it to specify what you want to “split” on. If the data in the file isn’t able to form a proper matrix (say rows have different numbers of columns), throw a IncompatibleMatrixException. I’ve added the file brokenmatrix.txt so you can test this out.

Also, it’s a good practice to make sure you remove any extraneous whitespace that you’ve read in from a file using the String.trim() method.

Similarly, you should implement a non-static method:

public void write(String filename) which takes a filename and writes the ComplexMatrix object out to that file using the format in complexmatrix1.txt and complexmatrix2.txt and complexmatrix3.txt. In fact, whatever you output with write should be able to be read in with read.