CSCI-UA.0101-007: Midterm Exam 2

Duration: 75 minutes

Instructor: Michael Tao
New York University

November 15, 2023

Submit solutions to the following problems as java files on gradescope.

Name: _____

NetID: _____

Do not open this exam until instructed.

Instructions:

- Each problem now specifies which java files you should submit for it, please submit all files to gradescope.
- Make sure to label which problem you are solving in your code.
- Gradescope will confirm that your code compiles, but the final grade will be evaluated manually later on.

Good luck!

```java
// Some useful java functions

// initialize just the outer dimension
int[][] x = new int[4][];
x[3] = new int[]{1,2,3};
int length = x.length;
// grid of values
int[][] y = new int[4][5];


class ArrayList<T> {// same for List
    T at(int index);
    void add(T newValue);
    void remove(int index);
    int length();
}

public class String {
    char charAt(int index);
    int length();
    char[] toCharArray();
}
```

Problem 1: Pascal's Triangle

Submit:

- `PascalTriangle.java`

Pascal's triangle is a generally useful structure in a variety of mathematical areas and in this problem you will implement a class that stores it and uses it to solve some problems. It starts off something like

```
    1     //row 0
   1 1    //row 1
  1 2 1   //row 2
 1 3 3 1  //row 3
1 4 6 4 1 //row 4
...
```

The entries of each row is constructed by taking the sum of the two values "above" it.

```
public class PascalTriangle {
```

(1) Declare a member that can store a number of rows of Pascal's triangle. Note that we will not be changing the size of the triangle after constructing it.

(2) Implement a constructor that takes in the number of rows to be constructed in the triangle. The constructor should fill the held triangle up to the number of rows requested.

```
// PascalTriangle pt = new PascalTriangle(3);
// pt.getValue(2,1) == 2
// pt.getValue(3,0) == -1
```

(3) Implement a function called `getValue` that returns the value of the triangle at the specific row and position. If this value is outside of the range of the triangle stored then this function should return -1.

```
// pt.getValue(2,1) == 2
// pt.getValue(3,0) == -1
```

Problem 2: Albums

Submit:

- `Album.java`
- `Song.java`

Musicians arrange their songs into albums. In this problem you will create some classes to track albums and compare them. We will begin by declaring a class for songs and then the albums that store them.

```
public class Song {
```

(1) Each song has a title, author, and its duration. Declare members to store these values, with the duration stored as an integral number of seconds.

(2) Declare getters for each of these members.

(3) Declare a constructor that specifies each of these members.

```
//Song a = new Song("Chop Suey", "System of a Down", 194);
```

(4) Override the `equals` function to compare if two songs are the same or not. Keep in mind that even though some recordings have different durations they should be treated as the same song.

```
//Song b = new Song("Chop Suey", "System of a Down", 192);
//Song c = new Song("Toxicity", "System of a Down", 146);
// a.equals(b) evaluates to true
// a.equals(c) evaluates to false
```

(5) Define a function such that the following code prints out the following:

```
// System.out.println(a); // prints out:
// Chop Suey - System of a Down [194s]
```

```
}
```

(6) An album has a title and multiple songs and the number of songs does not change. Write members to represent these.

```
class Album {
```

(7) Write a constructor that takes in a title and an array of songs.

```
// Song[] songs = ...;//somehow we figured out which songs are in the album
// Album toxicity = new Album("Toxicity", songs);
```

(8) Write a getter for the title, a `size` function that returns the number of songs, and a getter `getSong` that takes in the index of a song as input.

```
// String title = toxicity.getTitle();
// int numSongs = toxicity.size();
// Song firstSong = toxicity.getSong(0);
```

(9) Write a method to find a single song from its title called `findSong`. It returns `null` if no file exists.

```
// Song chopSuey = toxicity.findSong("Chop Suey");
// not in this album so returned null
// Song highwaySong = toxicity.findSong("Highway Song");
```

(10) Write a overload of `equals` to compare two albums.

Problem 3: Matrix

Submit:

- `Matrix.java`

Matrices are one of the central objects of study in linear algebra and their use is pervasive throughout computing. In fact, graphics cards, Apple's Neural Engine, and Google's Tensor chip are basically dedicated hardware for quickly performing matrix operations. They are typically represented by a two dimensional array of floating point values. In this problem we will explore the implementation of two standard matrix operations.

```
public class Matrix {
```

(1) Specify the member data for matrix.

(2) Specify two constructors: one that takes in the desired resolution of the matrix that populates the matrix with zeros and one that takes that takes as input a two dimensional array.

```
//Matrix A = new Matrix(20,30);
//Matrix B = new Matrix(new double[][]{{1,0},{0,1}});
```

(3) Make a function called `at` that lets users access elements in the array

```
//double value = A.at(5,5);
```

(4) Make a function called `set` that lets users write an element in the array

```
//B.set(1,1,2.0); <- writes at entry 1,1 the value 2.0
```

(5) Make a function called `setIdentity` that sets the current matrix to be an identity matrix. An identity matrix is a matrix with the value 1 for any entry of the form `M[i][i]` for all valid i and all other values are 0. Implement a function that changes the internal data to represent an identity matrix. Feel free to create a new array for this function and don't forget how Java initializes new variable values.

```
// Matrix M = new Matrix(1,1);
// M.setIdentity(2)// M's data is {{1,0},{0,1}}
// M.setIdentity(3)// M's data is {{1,0,0},{0,1,0},{0,0,1}}
public void setIdentity(int size) {
```

(6) Make a function called `transpose`. The transpose of a matrix A is another matrix B such that $A[i][j] == B[j][i]$. Implement a function that returns the transpose of the current matrix. Note that it converts a matrix with R rows and C columns to a matrix with C rows and R columns.

$$transpose\left(\begin{bmatrix}0 & 1\\ 2 & 3\end{bmatrix}\right) = \begin{bmatrix}0 & 2\\ 1 & 3\end{bmatrix}, \quad transpose\left(\begin{bmatrix}0 & 1 & 2\\ 3 & 4 & 5\end{bmatrix}\right) = \begin{bmatrix}0 & 3\\ 1 & 4\\ 2 & 5\end{bmatrix}$$

```
// Matrix A = new Matrix(new double[][]{{0,1},{2,3}});
// Matrix At = A.transpose();// {{0,2},{1,3}}
// Matrix B = new Matrix(new double[][]{{0,1,2},{3,4,5}})
// Matrix Bt = B.transpose(); //{{0,3},{1,4},{2,5}}
public Matrix transpose() {
```

Construct the new Matrix called `T` with the transposed dimensions and then populate `T` with its data.

```
}
```

Problem 4: Expression Representation

Submit:

- `Expression.java` (Just the interface definition found below)
- `Variable.java`
- `Sum.java`
- `Negate.java`

Computers are tasked with representing and evaluating complex sorts of expressions, and these representations are usually done with classes. Expressions are typically defined in terms of other expressions. Say we have an expression A and an expression B. Then their sum $A + B$ is an expression, as is the negation of A, $-A$. In this problem we will implement a simple expression representation for basic arithmetic. The basic object of study will be a `Expression` object that by default just returns 0, but in every class we will write in this problem we will overload this expression.

```
interface Expression {
    public double evaluate();
}
```

This expression system will work by declaring variables, assigning those variables, and then evaluating expressions that use those variables.

```
Variable x = new Variable(3.0);
Variable y = new Variable();
y.setValue(5.0);

Sum sum = new Sum(x,y);// sum.evaluate() is 8.0
Multiplication prod = new Multiplication(x,sum); // prod.evaluate() is 24.0
Negate neg = new Negate(prod); // neg.evaluate() is -24.0
```

(1) We can treat a variable as a sort of expression that stores a value and can be changed. Finish declaring this as an expression and define a single value as a member.

```
class Variable
```

(2) Implement a getter and setter for this function. Note that the only "getter" required is just the `evaluate` function.

(3) This variable class should also have two constructors: one that takes in no argument but sets the default value to 0.0. Implement them.

```
}
```

(4) Negating an expression is yet another expression whose evaluation is the negation of the evaluation of the sub-expression. Implement an expression that negates another one by first finishing the class statement and declaring its member, which is a single expression.

```
class Negate
```

(5) Implement a constructor that takes in the sub-expression.

(6) Implement getters for the member and the `evaluate` function which should call the underlying operation and negate its value.

```
}
```

(7) Adding two expressions is an expression that computes the sum of the evaluation of the two sub-expressions. Start off by declaring two members, which are also expressions.

```
class Sum
```

(8) Implement a constructor that takes in the two sub-expressions.

(9) Implement a getters for the members and the `evaluate` function which should call the underlying operation and negate its value.

```
}
```

Problem 5: Variant

Submit:

- `Variant.java`

Many systems like the Vulkan rendering standard and Google's protobuf serialization library use a sort of number to indicate the type of an object when there are several options. In this problem you will develop one such system within Java. Though this example is generally not necessary due to a feature of Java called reflection, it is a good exercise for a number of things with respect to object orientation.

For this problem we will only keep track of three potential types: `Character,Integer,Double`. To save on your writing you will, for the most part, only have to write one of three versions of various functions. Once you've written a function like `isInteger` you can assume that `isCharacter` and `isDouble` exist as well.

```
// Variant charVar = new Variant(new Character('a'));
// Variant intVar = new Variant(new Integer(3));
// Variant doubleVar = new Variant(new Double(1.5));
public class Variant {
```

(1) The variant class stores two pieces of data: an object and an integral index to indicate which type of object it is. Create these members.

(2) Rather than keep track of which indices are associated with which types, declare some publicly visible members shared between every `Variant` called `CHARACTER_INDEX`, `INTEGER_INDEX`, `DOUBLE_INDEX`. You need to specify the values of these numbers, but just about any numbers will work.

(3) Write a getter for the internally held index.

(4) Write a function called `isInteger` that checks if the internally held index indicates the internally held object is of type `Integer` using the numeric variable defined previously.

(5) Write a function called `isValid` that uses a switch/case to identify if the internally held index is set appropriately (for example it can only be `INTEGER_INDEX` if the object member is an instance of `Integer`).

(6) Write a sort of getter called `getCharacter` that returns the internally held object. This returned object should be properly converted to a `Character`, but only if the held index is set to `CHARACTER_INDEX`. If it isn't set to that this function returns `null`.

```
// Character c = charVar.getCharacter(); <- gets the Character holding 'a'
// Character c2 = intVar.getCharacter(); <- returns null
```

(7) Write two constructors: one that takes in a `Double` and one that takes in a `Character`. Even though these constructors only take in one argument they must set all members properly.

```
// Variant c = new Variant(new Character('c'));
// Variant d = new Variant(new Double(3.0));
```

Scrap paper **Name**: _____

Scrap paper **Name**: _____

Scrap paper **Name**: _____