CSE213, Object Oriented Programming
Independent University, Bangladesh

Java Practice Problems
(Instructor: S. K Dey)

CAUTION: These are practice problems only. There is NO GUARANTEE that your Mid/Final exam questions will have resemblance with these questions

---

1. Write **COMPLETE java program** (Test.java) for the following incomplete code: **[Only ONE JAVA FILE]**

```
class Complex {
    int real, img;
    //add constructor and other necessary methods for input, output
    //represents complex number in the form of "real +/- img i" e.g: 2+3i
}
class ComplexList {
    ArrayList<Complex> cList;
    //write constructors, toString, getters & setters
    //write populate, show & augment methods
}

//rewrite MainClass class with full functioning code
public class MainClass {
  public static void main(String[] args){
    ComplexList list;
    //….
    do-while loop: as long as user wants [choice: 1 to add, 2 to exit]
    {
       list.add(new Complex().setComplex());

    }
    int n = s.nextInt(); //no of additional Complex numbers
    Complex[] cArr = new Complex[n];
    list.augment(cArr);
    //augment method will instantiate cArr with n Complex instances,
    //set their values, and finally merge cArr with cList of
    //ComplexList instance list

    int lower, upper;
    //get values of lower & upper from user

    List.display(lower,higher);
    //display all the Complex numbers from cList of list
    //whose real<=lower & img>=upper, using showCopmlex()
    //method of Complex class;
  }
}
```

P.T.O

2. Write **COMPLETE java program** for the following incomplete code (Test.java): **[Only ONE JAVA FILE]**

```
class MyArray {
   int[] intData;
   // add necessary methods
}

class Matrix {
   MyArray[ ][ ] oneDObjects;
   // add necessary methods ( see main() )
}

public class TestClass {
  public static void main(){
   Matrix m = new Matrix();

   Loop: as long as user wants to continue
   {
      m.populateAndAugment();
      // get values of # of rows & cols from user and instantiate
      // oneDObjects. Now ask size of each MyArray instances of
      // oneDObjects and populate them with random integers.

      // If oneDObjects is already populated, then ask for additional
      row
      // & col size and do the above for additional MyArray instances
      // to fill the new cells (additional elements of oneDObjects) of
      the matrix

      m.display();
      // display the Matrix of MyArray instances

   }//end loop
  }//end main()
}//end class
```

P.T.O

3. Create a NetBeans java **CONSOLE** project with two packages: myarrays and mainpkg

- Classes of your projects are: myarrays.***OneDArray***, myarrays.**Matrix** and mainpkg.**MainClass**
- Class ***OneDArray*** has following **private** fields: **int[] values, float average**
  - Methods: a) **void getArray();** b) **void showArray();**
  - If necessary, add other methods to ensure that your main method works
- Class **Matrix** has following **private** fields: **OneDArray[][] arrays**
  - According to given RUN, you need to add appropriate methods in Matrix class

MainClass has the following main method:

```
public static void main(String[] args){
            Matrix m1, m2, m3;
            r = no of rows for Matrix class instance. r is a user input
            c = no of columns for Matrix class instance. c is a user input

            m1 = new Matrix(r, c); // m1 will have r rows & c cols
      //stores OneDArray instances in arrays[i][j] inside m1 Matrix instance
      //ask user for length and values for each OneDArray
      //average value of each MyOneDArray is also calculated

      Sout("First Matrix:"); m1.showMatrix(); //see RUN

            m2 = new Matrix(r, c, 2, 10); // m2 will have r rows & c cols
      //3rd parameter is the length of first OneDArray in m2,
      //which gets incremented by 1 for subsequent OneDArray instances in m2
      //4th parameter is the upper limit of random values to populate m2
      //average value of each OneDArray is also calculated

    Sout("Second Matrix:"); m2.showMatrix();//see RUN

    m3 = m1.merge(m2);

    Sout("Merged Matrix:"); m3.showMatrix();//see RUN
}
```

RUN:

```
How many rows? 2               | First Matrix:
How many columns? 2            | {1,3} Avg: 2            {4,8,6} Avg: 6
                               | {7,2} Avg: 4.5          {3,7,9,1}  Avg: 5
How many numbers: 2            |
Enter values: 1 3              | Second Matrix:
How many numbers: 3            | {5,1}  Avg: 3            {3,1,7} Avg: 3.67
Enter values: 4 8 6            | {9,1,5,4} Avg: 4.75      {2,7,1,8,5} Avg: 4.6
How many numbers: 2            |
Enter values: 7 2              | Merged Matrix:
How many numbers: 4            | {1,3,5,1} Avg: 2.5        {4,8,6,3,1,7} Avg: 4.83
Enter values: 3 7 9 1          | {7,2,9,1,5,4} Avg: 4.67   {3,7,9,1,2,7,1,8,5} Avg: 4.78
```

Now,

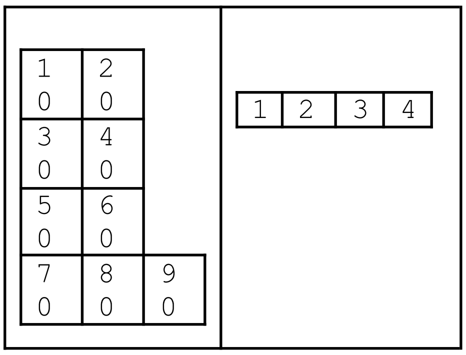
- Implement the above project without changing main(), fields of the classes and given RUN

4. Create a NetBeans java **FXML project** for the same problem described in **Q-3**

P.T.O

5. Write **COMPLETE java program** (MainClass.java) for the following: **[Only ONE JAVA FILE]**

Declare and populate a 3D Java array called **arr3** as per the following diagram and print it:



---

6. Create a NetBeans java **CONSOLE** project to implement the following:

Your console application package called **mypkg** has the following:

class **`Student`** with fields (id, name, cgpa, dept, major), constructors, toString, setter & getter methods

public class **`MainClass`** having:

- private field: an ArrayList of **Student** objects called **studArray**
- private field: an array of ArrayList<Float> **cgpaTable**
- Private method **populateStudArray** which reads Student information from user for n students (n is a user input and ensures that n is >0 and <=45) and stores them in studArray. Now, if the same ID is given by the user for two students, you must not proceed (show custom message) until the user gives a unique ID.
- private method **scanStudArray** to go through the already loaded **studArray** and copy student cgpa values into an array of 3 ArrayList (s) of floats named **cgpaTable [SIMULATING 2D ARRAY]**, where
  - all the cgpa (s) < 2.0 to be added to cgpaTable[0]
  - all the cgpa (s) >= 2.0 and <3.0 to be added to cgpaTable[1]
  - all the cgpa (s) >= 3.0 to be added to cgpaTable[2]
- private method **displayCgpaTable** to print 3 rows of cgpas from **cgpaTable** to the console separated by comma
- public static void main(…){

  //menu based do-while loop to call **populateStudArray, scanStudArray and displayCgpaTable**

  }

  WRITE Student class with appropriate fields & methods
  WRITE MainClass class with appropriate field declaration
  WRITE populateStudArray method of MainClass class
  WRITE scanStudArray method of MainClass class
  WRITE displayCgpaTable method of MainClass class
  WRITE main method of MainClass class

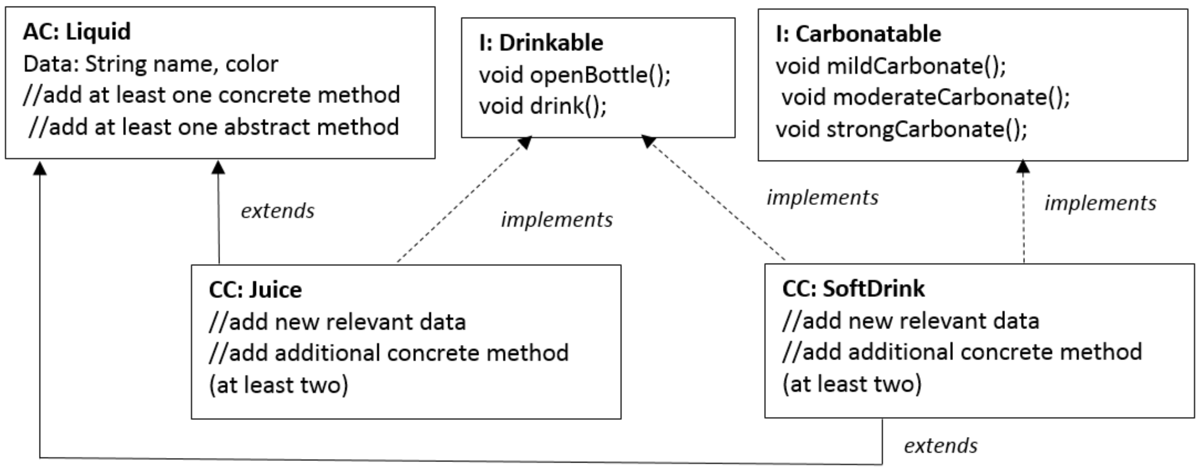
7. Create a NetBeans java **FXML project** for the same problem described in **Q-6**

P.T.O

8. Implement necessary classes for **ExerciseFXML.fxml** given in the project called **"Complete File Handling (read, write, append)"** (uploaded in your classroom) and make sure that it works.

9. Implement a netBeans **CONSOLE** project to implement the following class-diagram. Your main() method should have a menu-based option to perform operations to various types of objects.

- **First, if relevant, modify the following diagram** by adding "**associations**" and **multiplicities**



```
CC: Test
main() {
  /*add menu based option to
  perform   various operations on
  concrete classes*/
}
```

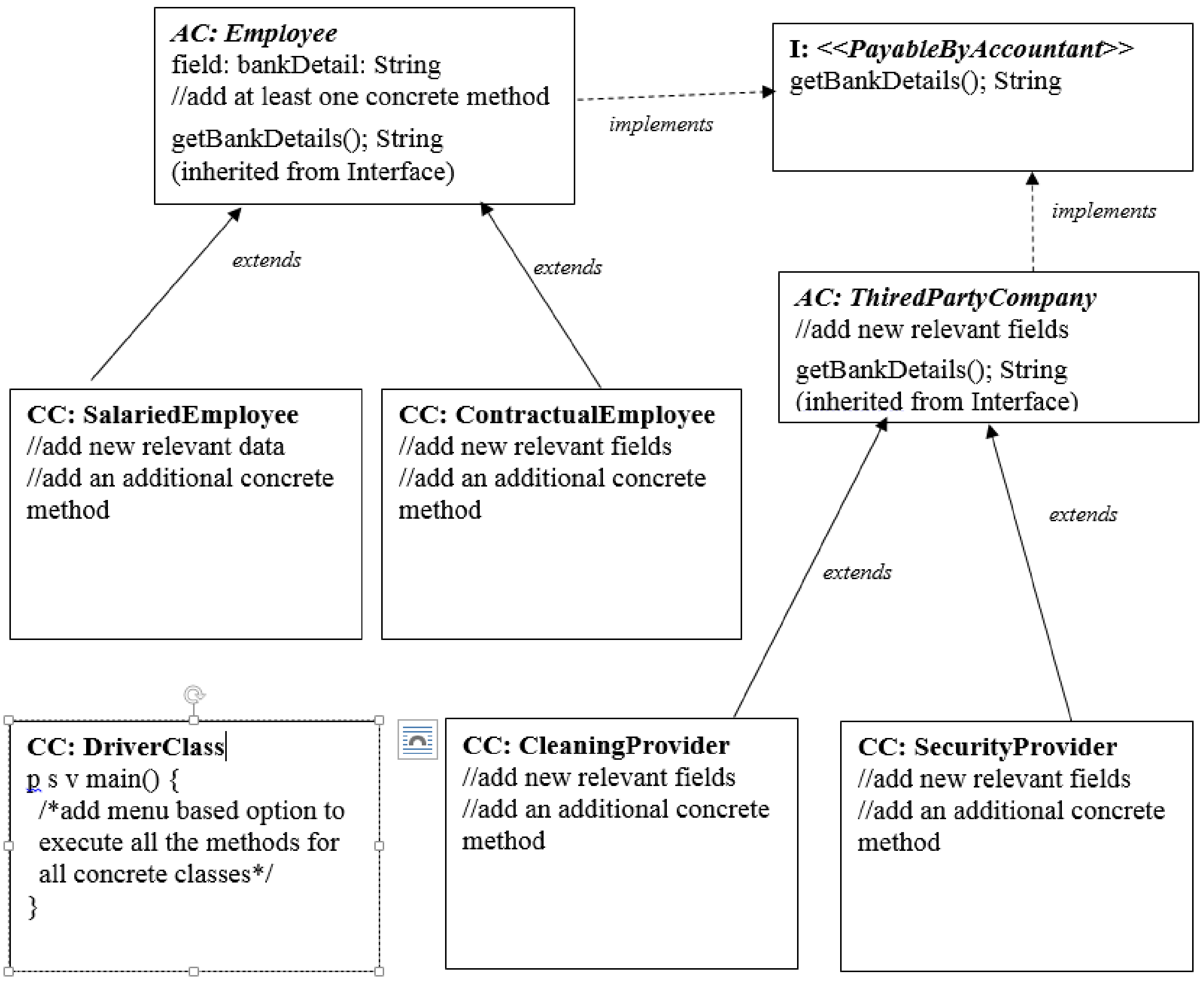
10. Create a NetBeans java **FXML project** for the same problem described in **Q-9**

P.T.O

11. Implement a NetBeans **CONSOLE** project for the following class-diagram. Your main() method should have a menu-based option to perform operations to various types of objects.

Note:

- **First, if relevant, modify the following diagram** by adding "**normal association**" and **multiplicities**
- Interface and Abstract classes belongs to "**mypackage**" package
- All concrete classes belongs to "**testpackage**" package


AC: Employee
field: bankDetail: String
//add at least one concrete method
getBankDetails(); String
(inherited from Interface)
implements
I: <<PayableByAccountant>>
getBankDetails(); String
implements
extends
extends
AC: ThiredPartyCompany
//add new relevant fields
getBankDetails(); String
(inherited from Interface)
CC: SalariedEmployee
//add new relevant data
//add an additional concrete method
CC: ContractualEmployee
//add new relevant fields
//add an additional concrete method
extends
extends
CC: DriverClass
p s v main() {
/*add menu based option to execute all the methods for all concrete classes*/
}
CC: CleaningProvider
//add new relevant fields
//add an additional concrete method
CC: SecurityProvider
//add new relevant fields
//add an additional concrete method

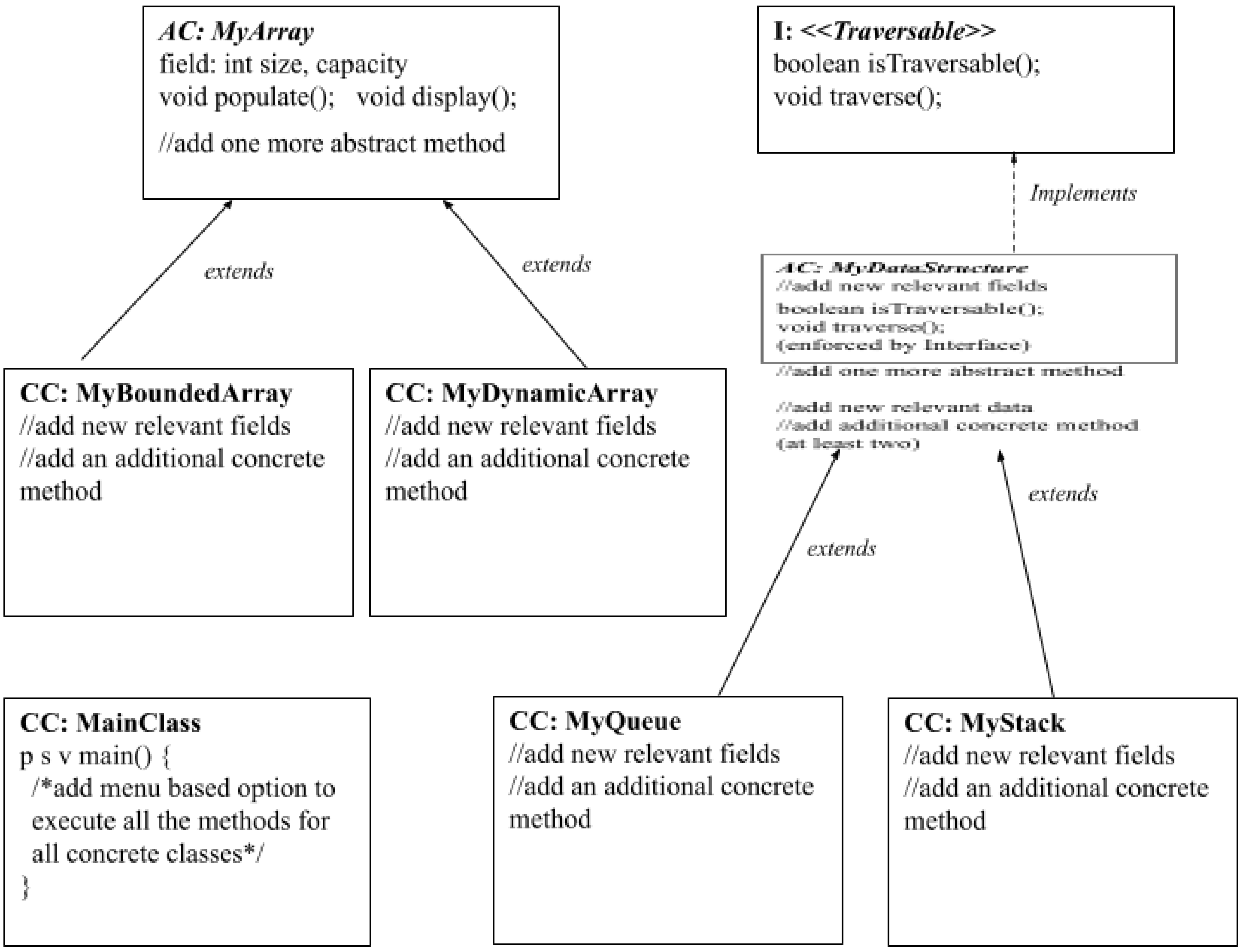
12. Create a NetBeans java **FXML project** for the same problem described in **Q-11**

P.T.O

13. **Implement a netbeans CONSOLE project for the following class-diagram. Your main() method should have a menu-based option to perform operations to various types of objects.**

Note:

- **First, if relevant, modify the following diagram** by adding "**normal association**" and **multiplicities**
- Interface and Abstract classes belongs to "**mypackage**" package
- All concrete classes belongs to "**testpackage**" package



AC: MyArray
field: int size, capacity
void populate(); void display();
//add one more abstract method
I: <<*Traversable*>>
boolean isTraversable();
void traverse();
Implements
AC: MyDataStructure
//add new relevant fields
boolean isTraversable();
void traverse();
(enforced by Interface)
//add one more abstract method
//add new relevant data
//add additional concrete method
(at least two)
extends
extends
CC: MyBoundedArray
//add new relevant fields
//add an additional concrete method
CC: MyDynamicArray
//add new relevant fields
//add an additional concrete method
extends
extends
CC: MainClass
p s v main() {
/*add menu based option to execute all the methods for all concrete classes*/
}
CC: MyQueue
//add new relevant fields
//add an additional concrete method
CC: MyStack
//add new relevant fields
//add an additional concrete method

14. Create a NetBeans java **FXML project** for the same problem described in **Q-13**

P.T.O

15. Design a UML class diagram to simulate IRAS.

Your design MUST have multiple inheritance (using classes and Interfaces), and:

a. There must be at least 2 abstract classes
   i. With some meaningful abstract methods
   ii. One of the abstract classes must have at least one concrete method
b. There must be at least 3 concrete classes
c. There must be composition, where
   i. at least one of the classes (abstract/concrete) must have an object-handle of another class as its field
d. There must be aggregation, where
   i. at least one of the classes (abstract/concrete) must have an object-handle of another class as its field
e. In each concrete (non-abstract) class, there MUST be setter and getter methods for input/output
f. There must be a MainClass to test all operations of your project
g. Show all necessary associations and multiplicities

NOTE: class/interface and their fields & methods MUST have meaningful and logically acceptable names. And, they MUST make sense in terms of "is-a" / "has-a" relationship.

16. Implement a NetBeans CONSOLE application for the following:

Following are the private fields of your public class called **Matrix**:

i) Int row; // gets no of rows from user (<= 10000)
ii) Int col; // gets no of columns from user (<= 10000)
iii) Int upperLimit; // gets upper limit of matrix element's value from user

main() method of your mainClass will have the following local Strings

String fileName1, fileName2, fileName3
Matrix handles as necessary

Now, there will menu options available for user as follows:

Menu option - 1: Generate Matrix files

Selecting this option will proceed to get two text file names from the user in fileName1 & fileName2 for storing two randomly generated matrices.

- At this point, you need to create the first two text files representing randomly populated matrices. First line of those files will have two integers separated by space (no of rows & no of columns). Subsequent lines will represent rows of those matrix values (separated by space).
- Upon successfully writing each file, a notification message will be displayed.

Menu option - 2: Generate multiplied Matrix file

clicking this option will do the following:

- If fileName1 & fileName2 are null, show some error message.
- Else If (noOfColsOfFile1 != noOfRowsOfFile2), then also show some error message.
- Otherwise, ask the third text file name from the user in fileName3 to store the multiplied matrix.
- At this point, you need to load the source matrices from the first two files, multiply them and write the multiplied matrix into the third file. For that purpose you need to use some Matrix handles.

17. Create a NetBeans java **FXML project** for the same problem described in **Q-17**