Please Read

These are some sample questions which hopefully give you an idea of the flavor of the questions you may expect in the exam.

Question 1

Consider the following items

- Country
- Germany
- Europe
- Asia
- Continent
- Asian country
- United States
- City
- New York
- European country
- China

1a

Which items above most naturally belong to class category and which ones most naturally belong to object category?

1b

Give two examples of objects along with classes they instantiate.

1c (7 points)

Give at least two plausible examples of Is–a relationship among classes in these items.
Give at least two examples of Has–a relationship among classes in the items above.

Question 2

A multi–set is a collection of items much like a set, in that order of items is not important. However it differs from a set in that an item can be in the multi–set many times. An example of a multi–set is

\{1, 2, 5, 6, 7, 7\}

We like to extended the HashSet class of Java Collections API to accommodate multi–sets.

The following is a skeleton of the code for multi–set implementation. First we define an Item to be an Object with a count (or multiplicity) attached to it. Then we define multi–sets.
public Class Item{
    
    // the item that is part of the multiset
    private Object item;

    // the multiplicity of the item
    private int count;

    public int getCount(){
        return count;
    }

    // increase count by one
    public void incCount(){
        ++count;
    }

    // decrease count by one
    public void decCount(){
        --count;
    }
}

public Class MultiSet extends HashSet{
    public Boolean add(. . .){
        
    }

    public Boolean remove(. . .){
        
    }
}

We need a new form of add method. The new add method should add a new item to the multi-set if it does not already exist. Otherwise it should increase the count of the item in the multi-set. Similarly we need a new form of remove method. The new remove will subtract one from the count of item in the multi-set if there is more than one copy of it in there. Otherwise it simply removes it from the multi-set.
2a (5 points)
Which is more appropriate when you are writing your new add method for the multiSet class: overloading the add method, or overwriting it. Justify your answer.

2b (10 points)
Based on your answer on part 2a fill in the parameter definition of the add method for the multiSet class and fill in the missing code as indicated (where there is "--------").

```java
public boolean add(--------){
    //if there are some of this item already in the multiset
    if(--------){
        //increase count of the item
        -------------------;
    }
    //otherwise add it to the multiset using HashSet
    // (parent class) add method
    else{
        -------------------;
    }
} //end of if
} //end of add
```

2c (5 points)
Which is more appropriate when you are writing your new remove method for the multiSet class: Overloading the remove method, or overwriting it. Justify your answer.
Based on your answer on part 2c fill in the parameter definition of the remove method for the multi-set class and fill in the missing code as indicated (where there is $\ldots\ldots\ldots$).

```java
public Boolean remove($\ldots\ldots\ldots$) {
    // if there are some of this item already in the multiSet
    if ($\ldots\ldots\ldots$) {
        // decrease count of the item
        $\ldots\ldots\ldots$
        // otherwise remove it from the multiset
        // using HashSet (parent class) remove method
        else
            $\ldots\ldots\ldots$
    } // end of if
} // end of remove
```
We like to define a web page class hierarchy. This hierarchy is composed of following classes:

- **WebPage**: The most general web page
- **ActiveWebpage**: A kind of web page that interacts with user
- **PassiveWebPage**: A kind of web page that just shows content without interacting with user
- **TextArea**: containing text
- **GraphicArea**: A file containing any kind of graphics
- **GifGraphicFile**: a graphic file in Gif format
- **JpegGraphicFile**: a graphic file in jpeg format
- **SoundFile**: A general sound file
- **WaveSoundFile**: A special Sound file in Wave format
- **MidiSoundFile**: A graphic sound file in Midi format

Our web page system must satisfy the following rules:
1. Web pages that we are considering are either active web pages or passive web pages.
2. Sound files are either in the Wave format or in Midi format.
3. Graphic files are either in Gif format or Jpeg format.
4. Any web page may contain zero or more graphics and zero or more sound files but only one text area
5. Active web pages cannot share graphics or sound files with other web pages (active or passive).
6. Passive web pages may share graphics files with other passive web pages. But cannot share sound files with any other web page.

4a. Draw the UML class diagrams for the classes mentioned above. Make sure to incorporate all the information provided that is possible. For each class you only need to write its name and need not worry about the methods and other fields.

4b. How can this set of classes fit into the composite pattern?

Question 4 (15 points)

In following table we have listed some concepts in object oriented programming (OOP) in one column and some concepts in Java in another column. By drawing a clear line match an OOP concept to a Java concept if the Java concept is a mechanism to realize the OOP concept. For instance the ability to extend a class is one way in Java to realize inheritance. Note that an OOP concept may have several ways of being realized in Java. Similarly a Java concept may realize several OOP concepts. Indicate all the matches that are appropriate. If you wish you may provide explanations.
<table>
<thead>
<tr>
<th>Object Oriented Programming Concepts</th>
<th>Java Facilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Abstraction</td>
<td>• Class</td>
</tr>
<tr>
<td>• Inheritance</td>
<td>• Interface</td>
</tr>
<tr>
<td>• Multiple inheritance</td>
<td>• Abstract class</td>
</tr>
<tr>
<td>• Composition/aggregation</td>
<td>• Private/Protected fields and methods</td>
</tr>
<tr>
<td>• Encapsulation</td>
<td>• Overriding methods</td>
</tr>
<tr>
<td>• Polymorphism</td>
<td>• Overloading methods</td>
</tr>
<tr>
<td>• Modularity</td>
<td>• Extending classes</td>
</tr>
<tr>
<td>• Software reuse</td>
<td>• Implementing interfaces</td>
</tr>
<tr>
<td></td>
<td>• Nested classes</td>
</tr>
<tr>
<td></td>
<td>• Packages</td>
</tr>
</tbody>
</table>
Consider the following class definition:

```java
public Class Student{
    public String Name;
    public int classification;
    public static int ID;

    public Student(String Name, int classification, int ID){
        this.name=name;
        this.classification = classification;
        Student.ID++;
    }

    // . . . . .
```

5a (5 points)

Draw the detailed UML class diagram of this class

5b (5 points)

In what way does this piece of code violates the principle of encapsulation.
5c (5 points) Draw the detailed class diagram for an alternative definition of this class that adheres to encapsulation rule, possibly by adding any necessary methods.

6 Consider a JButton object with the following properties:
  • when the mouse pointer goes over the button Background becomes red
  • when the mouse pointer leaves the button the original color (gray) is restored
  • When the button is pushed the the font of the label turns into bold
  • when the button is released the original font (plain) is restored

Draw the state diagram of the object (Notice the sequence of events, there is a subtlety here!)

7 For the homework project 3 & 4 draw the collaboration diagram for the status of an order from the moment it is initiated by the customer to the moment that it is shipped back to him/her.

8 Variation on a Singleton pattern: Suppose you wish to create at most two instances of an object in a class and no more. How would you modify the Singleton pattern to give you at most two object and no more?

9 For a software system of a bank, a class AccountHandler is responsible for creating Account objects for a new customer. The Account class is an abstract class with CheckingAccount and SavingAccount as its concrete subclasses. Thus AccountHandler creates CheckingAccount or SavingAccount objects; this decision is made at run time. We wish to write AccountHandler class in such a way that it remains independent of types Account (in case new types of Accounts are created in the future, we wish AccountHandler to be able to cope with them without any changes.)

9a What pattern is appropriate for solving this problem? Draw a class diagram depicting the relationship between AccountHandler, Account and its subclasses and any additional interface or class necessary.

9b For each additional class or interface, show any relevant method or field.
Consider the following situation which seems to require multiple inheritance:

We have the following set of classes:

class staff {
    private String name;
    public String getName() {
        return name;
    }
}

class Manager extends staff {
    public String getName() {
        return super.getName() + ", V.P."
    }
}

class Scientist extends staff {
    public String getName() {
        return super.getName() + ", Ph.D"
    }
}

Now we like to have a class that represents objects that are both Manager and Scientist. Since there is no provision in Java for multiple inheritance use delegation pattern to design the class ManagerScientist. In particular, how would the ManagerScientist class implements its getName method? (Such a method should output write the name, followed by ", Ph.D" Followed by ", V.P."