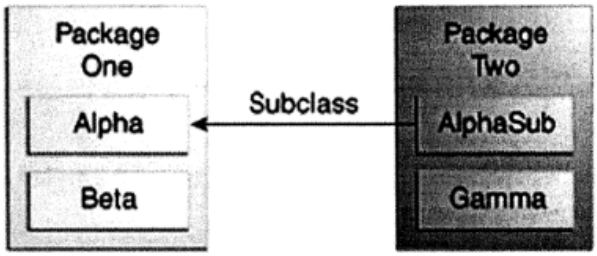
Question 1. Classes Alpha, AlphaSub, Beta, and Gamma are defined in two packages as shown the diagram below. AlphaSub is a subclass of Alpha. Answer the following questions.

[25 marks]



a) If the method called 'alphaMethod()' in the class Alpha is declared as protected, to which of the classes Beta, AlphaSub and Gamma will it be visible?

[4 marks]

b) If the method called 'alphaMethod()' in the class Alpha is declared as default (i.e., no modifier), to which of the classes Beta, AlphaSub and Gamma will it be visible?

[4 marks]

c) Consider the source code for class Alpha. Will the code be able to compile? Why?

```
public class Alpha {

    private final String name;
    private int volume;

    public Alpha(String name, int volume) {
        this.name = name;
        this.volume = volume;
    }

    public String getName() {
        return name;
    }

    public void setName(String newName) {
        this.name = newName;
    }

    public int getVolume() {
        return volume;
    }

    public void setVolume(int newVolume) {
```

```
        this.volume = newVolume;
    }

    public int compareTo(Alpha a) {
        //implement your code here.
    }
}
```

[5 marks]

d) The interface Comparable has a method '`public int compareTo(Object o)`' that allows comparison between objects. The method returns "0" if two instances are equal. It returns a positive number if the current instance is greater than the given "Object o" and a negative number otherwise. For the Alpha class above, assume that the criteria for the comparison is based on the field volume. Implement the compareTo() method for the class Alpha.

[6 marks]

e) What are the differences between an abstract data type and an abstract class?

[4 marks]

f) What are the characteristics of static inner classes?

[2 marks]

Question 2. Consider the BinaryTree class and its inner class TreeNode below. Answer the following questions.

[25 marks]

```
public class BinaryTree<X extends Comparable<X>> {
    private TreeNode root;

    public BinaryTree(X v) {
        root = new TreeNode(null, v, null);
    }

    public BinaryTree(TreeNode l, X v, TreeNode r) {
        root = new TreeNode(l, v, r);
    }

    public int height() {
        return (root == null) ? 0 : root.height();
    }

    private class TreeNode {
        private X value;
        private TreeNode leftSubTree;
        private TreeNode rightSubTree;

        public TreeNode(TreeNode l, X v, TreeNode r) {
            value = v;
            leftSubTree = l;
            rightSubTree = r;
        }

        public void setLeftTree(X l) {
            leftSubTree = new TreeNode(null, l, null);
        }

        public void setRightTree(X r) {
            rightSubTree = new TreeNode(null, r, null);
        }

        private int height() {
            int leftHeight, rightHeight;
            leftHeight = (leftSubTree == null) ? 0 : leftSubTree.height();
            rightHeight    =    (rightSubTree    ==    null)    ?    0    :
rightSubTree.height();
            return Math.max(leftHeight, rightHeight) + 1;
        }
    }
```

```
    public static void main(String args[]) {
        BinaryTree<Integer>  tree  =  new  BinaryTree<Integer>(new
Integer(7));
        tree.root.setLeftTree(new Integer(6));
        tree.root.setRightTree(new Integer(8));
        tree.root.leftSubTree.setRightTree(new Integer(3));
        System.out.println(tree.height());
    }
}
```

a) The implementation above uses generics. Which is the formal parameter and which is the actual parameter?

[4 marks]

b) What is meant by class invariant?

[2 marks]

c) How to check if a property is a class invariant?

[4 marks]

d) Suppose a class invariant for the BinaryTree class is: (1) each value in the left subtree is strictly less than the internal label; and (2) each value in the right subtree is strictly greater than the internal label. Does the implementation preserve the class invariant? Justify your answer with an exampe.

[7 marks]

e) Describe the method call stack for the last two statements in the main() method.

[8 marks]

Question 3. The BACK button functionality of a Web browser can be implemented using a stack, which only stores the most recently visited pages. Consider the implementation below and answer the following questions.

[25 marks]

```
public class BackStack {

    private int[] items = new int[10];
    private int top = -1;

    public boolean isEmpty() {
        return top < 0;
    }

    public int pop(){
       return items[top--];
    }

    public void push(int i) {
       //stack full, make room
       if (top == items.length - 1) {
           for (int j = 0; j < items.length - 1; j++) {
               items[j] = items[j + 1];
           }
           items[top] = i;
       } else {
           items[++top] = i;
       }
    }

    public static void main(String[] args) {
       BackStack s = new BackStack();
       for (int i = 0; i < 20; i++) {
           s.push(i);
       }
       for (int i = 0; i < s.items.length; i++) {
          System.out.println(s.pop());
       }
    }
}
```

a) Briefly describe the differences between 'checked' and 'unchecked' exceptions.

[4 marks]

b) Write a checked exception called StackEmptyException for the class BackStack if the stack is empty.

[6 marks]

c) Modify the pop() method so that it can throw an StackEmptyException when the stack is empty.

[4 marks]

d) What other changes are necessary for the above implementation?

[6 marks]

e) In java programming, what is the preferred place to use the try-catch to handle exceptions? Why?

[5 marks]

Question 4. Class Group consists of a vector of its 'members'. A MemberThread instance can become a member by calling the join() method and cancel its membership by calling leave() method. The numFriends field of an existing member in the group is equal to the total number of other members. When a new MemberThread instance becomes a member, the join() method increments the numFriends field of all existing members; similarly, the leave() method decrements these fields.

[25 marks]

```
class MemberThread extends Thread {
   // number of Group friends
   int numFriends = 0;

   public void run() {
   // join and then leave the Group
      Group.join(this);
      try {
          Thread.sleep(10000);
      } catch (InterruptedException (ie) {
      }
      Group.leave(this);
      }
   }

   class Group {
      // list of Group members
      static Vector<MemberThread> members
             = new Vector<MemberThread>();

      static void join(MemberThread f) {
      // add a new friend to all existing members
         int size = members.size();
          for (int i = 0; i < size; i++) {
              members.elementAt(i).numFriends++;
          }
          f.numFriends = size; // new member's friends
          members.add(f); // add to list of members }


      static void leave(MemberThread f) {
          members.remove(f); // remove from list
          int size = members.size();
          for (int i = 0; i < size; i++) {
              members.elementAt(i).numFriends--;
          }
      }
```

```
    public static void main() {
        for (int n = 0; n < 100; n++) {
            new MemberThread().start();
        }
    }
}
```

a) In the context of Java multithreading, what is meant by interference?

[4 marks]

b) Are there interference problems in the Group class? Justify your answer.

[5 marks]

c) How can the problem in b) be solved?

[6 marks]

d) Threads may move between different states under the control of the Java interpreter's time-slicing mechanism, for example, 'ready' and 'running' are two of the possible states. Briefly describe what other states can threads be in, and how a running thread can be changed to those states?

[6 marks]

e) Briefly describe the purpose of critical sections in Java multithreading programs.

[4 marks]

Question 5. Answer the following questions.

[25 Marks]

a) Define a generic interface called 'AtomicFunction'. The interface should have two methods. The first method is called compute(), which accepts a parameter of type X, and returns a value of type Y. The second method is called getSuperclassName(), which returns the name of the superclass of an implementing class.

[5 marks]

b) Define an abstract class called 'AbstractFunction' by implementing the AtomicFunction interface. The class should define the compute() method as abstract and implements the getSuperclassName() method.

[5 marks]

c) Define a class called 'LengthFunction' by extending the AbstractFunction class. The compute() method should take a String as input and return the length of the String as an Integer.

[5 marks]

d) Define a class called 'DivBy5Function' by extending the AbstractFunction class. The compute() method for this class should take an Integer as input and return a Boolean value indicating if it is divisible by 5.

[5 marks]

e) Consider the class CompositeFunction below. The compose() method is defined by using anonymous class. What is the output after the main() method is executed? Briefly explain how the answer is obtained.

[5 marks]

```
public class CompositeFunction {

    public AtomicFunction compose(final AtomicFunction<String, Integer>
f1, final AtomicFunction<Integer, boolean> f2){
        return new AtomicFunction<String, boolean>(){
            public boolean compute(String s){
                return f2.compute(f1.compute(s));
            }
        };
    }

    public static void main(String args[]){
        CompositeFunction f = new CompositeFunction();
        AtomicFunction<String,    boolean>    mf    =    f.compose(new
LengthFunction(), new DivBy5Function());
        System.out.println(mf.compute("ObjectOrientation"));
```

```
        }
    }
```

END OF EXAM PAPER