46-935

Java for Computational Finance

Midterm Exam Solution

Name: _____Solution_____

Location: _____

Question	1	2	3	4	5	6	Total
Points	15	15	15	15	20	20	100
Score							

Question 1 [15 points]

In the following program segment, what is being printed?
Write your answer in the program.

```
class Xplain
{
      void print() {System.out.println("A");}
}

class Xtend extends Xplain
{
      void print() {System.out.println("B");}
}

public class Question1
{
      public static main(String args[])
      {
            Xplain x1 = new Xplain();
            Xtend  x2 = new Xtend();
            Xtend  x3;

            x1.print();
                  // your answer _________A_____________

            x2.print();
                  // your answer _________B___________

            x1 = x2;
            x1.print();
                  // your answer _________B_____________

            x3 = (Xtend) x1;
            x3.print();
                  // your answer _________B___________

            ((Xtend) x1).print();
                  // your answer _________B_____________
      }

}
```

Question 2 [15 points]

This question will explore the tradeoffs between using an abstract class and an interface.

- What arguments are there for using an abstract class over an interface?

1) An abstract class allows partial implementations (some abstract methods and some defined methods).
2) An abstract is more flexible with data fields (fields in interfaces must be static and final – which makes them constants).
3) An abstract class is more flexible with access modifiers (everything in an interface is publicly accessible).

- What arguments are there for using an interface over an abstract class?

1) Java does not allow multiple inheritance – a class may only extend one superclass. However, a class may implement many interfaces. Thus, if you need to inherit from more than one source, you must use interfaces.
2) The conventional uses of interfaces and abstract classes are different. Interfaces tend to be used to provide information about a class (*"I provide this set of methods!"*). However, abstract classes are usually used to create a hierarchical structure of related classes.

Question 3 [15 points]

One of the early criticisms of Java was that it was not fit for programs that involved heavy computation because it was an interpreted language.

- What does it mean to say that Java is interpreted?

Java technology uses both compilation and interpretation. Compilation is the translation of code from one language to another and storing the results of the translation for later use. Java source is compiled into byte-codes which contain architecture-independent instructions for the Java Virtual Machine. Interpretation also involves translating code from one language to another, except you directly execute the translation instead of storing it. In Java Virtual Machines, interpretation picks up where compilation left off. Java Virtual Machine instructions that were compiled from source code are interpreted by the virtual machine – converted on the fly to native machine code, which is then executed rather than stored. This interpretation process is obviously slower than directly executing native machine code.

- The current versions of Java are now distributed with a just-in-time compiler (JIT). How does this address the criticism listed above?

A JIT compiler translates Java Virtual Machine instructions into native machine code at run time on the local platform. Once the virtual machine implementation has this translation, it can run the native code at speeds comparable to other compiled languages. Once Java programs are running at speeds comparable to other compiled languages, they will be fit for executing programs that involve heavy computation.

Question 4 [15 points]

This question will test your understanding of exceptions. Suppose we have the simple class shown below that implements a stack as an array with a set size.

```
//A stack that has a three item limit
class ThreeDeep
{
       //fields
       static final int STACK_SIZE = 3;
       private int[] stack_store = new int[STACK_SIZE];
       private int   numElements = 0;

}
```

- Write the class declarations for two custom exceptions named – StackOverFlow and StackUnderFlow. StackOverFlow will occur when an attempt is made to push an element onto a full stack. StackUnderFlow will occur when an attempt is made to pop an element off of an empty stack.

```
public class StackOverFlow extends Exception
{
    public StackOverFlow()         {super();}
    public StackOverFlow(String s) {super(s);}

}
public class StackUnderFlow extends Exception
{
    public StackUnderFlow()         {super();}
    public StackUnderFlow(String s) {super(s);}

}
```

- Write the push and pop methods for the ThreeDeep class. Make sure to raise the above exceptions in the methods if appropriate. The push method is of type void, and takes an int parameter (the integer to push onto the stack). The pop method is of type int, and takes no parameter.

```
public void push(int i) throws StackOverFlow
{
    if (numElements < STACK_SIZE)
    {
        stack_store[numElements++] = i;
    }
    else
    {
        throw new StackOverFlow("Stack Full");
    }
}

public void pop(int i) throws StackUnderFlow
{
    if (numElements > 0)
    {
        return stack_store[--numElements++];
    }
    else
    {
        throw new StackUnderFlow("Stack Empty");
        return -1;
    }
}
```

Question 5 [20 points]

The class declarations for the LattKey, LattNode, LattList, and Lattice classes are shown on the last few pages of the exam. These classes provide a data structure for holding a lattice that is similar to the data structure used in the AbstractOption class you saw in homework. This lattice data structure, however, is more flexible because it tracks multiple assets and multiple auxiliary processes. In addition, the number of successors of each node and corresponding edge the probabilities are not fixed.

- Write a forward equation that describes the expected discount rate at a node in terms of the value of the expected discount rate at its predecessors.

$$EDR(v) = \sum_{v' \in pred(v)} EDR(v') \left(\frac{1}{1 + r(v')} \right) P(v', v)$$

$EDR(\text{root}) = 1$

- Write a method for the Lattice class (called CalcExpectedDiscount()) using forward induction that will efficiently evaluate the forward equation at each node in the lattice and store the results in the LattKey data field named quest5_value.

```
void CalcExpectedDiscount()
{
    int i;
    int j;
    double sum;
    LattNode x;

    for (i=0; i< timeHorizon; i++)
    {
        x = nodes[i].head;

        while (x != null)
        {
          if (i==0)
          {
              x.key.quest5_value = 1.0;
          }
          else
          {
              sum = 0.0;

              for (j=0; j<x.numPred; j++)
              {
                  sum = sum + ((x.predecessors[j].key.quest5_value *
                                x.incoming_edge_probs[j])/
                               (x.predecessors[j].key.short_rate));

              }

              x.key.quest5_value = sum;
          }
          x = x.next;

        }
    }
}
```

Question 6 [20 points]

Refer to the description of the lattice classes in Question 5.

Assume that the method you have written in Question 5 has already been executed on the lattice.

- Write a backward equation that describes the expected discounted price of the first asset at the time horizon. Recall that a backward equation is an equation that is defined in terms of its value at its successors.

$$EDP(v) = \frac{\left[\sum_{v' \in succ(v)} EDP(v')P(v', v)\right]}{EDR(v)}$$

$EDR(\text{leaf}) = \text{leaf.asset\_prices}[0]$

- Write a method for the Lattice class (called CalcExpectedPrice()) using backward induction that will efficiently evaluate the backward equation at each node in the lattice and store the results in the LattKey data field named quest6_value.

```
    void CalcExpectedPrice()
    {
        int i;
        int j;
        double edp;
        LattNode x;

        x = nodes[timeHorizon].head;

        while(x != null)
        {
            x.key.quest6_value = x.key.asset_prices[0];
            x = x.next;
        }

        for (i = (timeHorizon - 1); i >= 0; i--)
        {
            x = nodes[i].head;

            while (x != null)
            {
                edp = 0.0;

                for (j = 0; j < x.numSucc; j++)
                {
                    edp = edp + (x.successors[j].key.quest6_value *
                          x.outgoing_edge_probs[j]);
                }

                x.quest6_value = edp/x.key.quest5_value;
                x = x.next;

            }
        }
}
```

LattKey.java

```
package ExamLattice;

public class LattKey implements LattValues
{
    /**
     * The (int) time value
     */
    public int      time;

    /**
     * The (double) short rate
     */
    public double   short_rate;

    /**
     * The (double) array of asset prices
     */
    public double   asset_prices[] = new double[NUM_ASSETS];

    /**
     * The (double) array of aux process values
     */
    public double   aux_processes[] = new double[NUM_AUX_PROCESSES];

    /**
     * The (double) holder variable (refer to Question 5)
     */
    public double   quest5_value;

    /**
     * The (double) holder variable (refer to Question 6)
     */
    public double   quest6_value;

    /**
     * Constructs a Key object, and sets the initial time value.
     *
     * @param   aTime   the (int) time value
     */
    public LattKey(int aTime)
    {
        //details removed
    }

    //other methods removed
}
```

LattNode.java

```
package ExamLattice;

public class LattNode implements LattValues
{
    /**
     * The (LattNode) reference to the previous node with this time value
     */
    public LattNode prev;

    /**
     * The (LattNode) reference to the next node with this time value
     */
    public LattNode next;

    /**
     * The (LattKey) data value
     */
    public LattKey  key;

    /**
     * The (int) number of predecessors
     */
    public int numPred;

    /**
     * The (int) number of successors
     */
    public int numSucc;

    /**
     * The (LattNode) reference to the predecessor node
     */
    public LattNode predecessors[];

    /**
     * The (LattNode) array of references to successors
     */
    public LattNode successors[];

    /**
     * The (double) arrays of incoming and outgoing edge
     * probabilities (the ith probability corresponds to
     * the ith edge in the pred/succ array)
     */
    public double incoming_edge_probs[];
    public double outgoing_edge_probs[];

    /**
     * Constructs a LattNode object
     */
    public LattNode(LattKey k, int succ, int pred)
    {
      //details removed
    }
}
```

LattList.java

```
package ExamLattice;

public class LattList
{
    /**
     * The (LattNode) head of the list
     */
    public LattNode head;

    /**
     * Constructs a LinkList object, initially the
     * list will be empty (head == null)
     */
    public LattList()
    {
        head = null;
    }

    //other methods removed …
}
```

Lattice.java

```
package ExamLattice;

public class Lattice implements LattValues
{
  /**
   * The (integer) time horizon.
   */
  public int timeHorizon;

  /**
   * The array of LattList objects that represents the PDAG.
   * nodes[i] is a linked list of lattice nodes with time value i
   */
  public LattList nodes[];

  /**
   * Constructs an empty lattice
   */
  public Lattice(int T)
  {
      //details removed
  }

  /**
   * Generates the lattice to the time horizon
   */
  void GenerateLattice()
  {
     //details removed
  }
}
```

LattValues.java

```
package ExamLattice;

public interface LattValues
{
    /**
     *  The (int) number of assets modeled by the lattice
     */
   public final static int  NUM_ASSETS = 3;

    /**
     *  The (int) number of aux_processes
     */
   public final static int  NUM_AUX_PROCESSES = 4;

}
```