The class declarations for the LattKey, LattNode, LattList, and Lattice classes are shown here. 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 the corresponding edge probabilities are not fixed. No methods are given here. Part of the exam will ask you to write methods in the Lattice class. You should familiarize yourself with these classes prior to the exam.

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;

}
```