See the additional notes and examples at the end of this posting.

Implement a threads and a GUI interface using advanced Java Swing classes.

The project will be graded according the criteria for the final project - see below.

Elaboration:

1. Extend Project 2 to use the Swing class JTree effectively to display the contents of the data file.
   * See Liang 8th, Chapter 36.
   * (Optional) Implement a JTable to also show the contents of the data file. There are lots of options here for extending your program.
2. Threads:
   * Implement a thread for each creature representing a task that creature will perform.
   * Use delays to show the creature doing the task.
   * Use a JProgressBar for each creature to show the creature performing the task.
   * Use JButton's on the JProgressBar to allow the task to be started and cancelled.
3. As before, the GUI elements should be distinct from the other classes in the program.

**Deliverables**

1. Java source code files
2. any configuration files used
3. a well-written Word document describing:
   1. your overall design, including a UML class diagram showing the type of the class relationships
   2. description of how to set up your application
   3. your test plan, including test data and results, with screen snapshots of each of your test cases
   4. your approach, lessons learned, design strengths and limitations, and suggestions for future improvement and alternative approaches

**Format**

Documentation format and length. The documentation describing and reflecting on your design and approach should be written using Microsoft Word, and should be no more than five pages in length and no less than two pages. The font size should be 12 point. The page margins should be one inch. The paragraphs should be double spaced. All figures, tables, equations, and references should be properly labeled and formatted using APA style.

* Code format:
  + header comment block
  + appropriate comments within the code
  + appropriate variable and function names
  + correct indentation

**Grading**

This activity is awarded 15 percent of the total grade in the course. In the grade book, the total number of points will be set to 100. The project elements will be assessed as follows:

|  |  |
| --- | --- |
| Attributes | Value |
| Project design | 20 points |
| Project functionality | 40 points |
| Test data | 20 points |
| Approach documentation | 15 points |
| Approach documentation grammar and spelling | 5 points |
| Total | 100 points |

**Data file - add the following line to define a job:**

// Jobs for creatures  
// measure time in seconds  
//    j:<index>:<name>:<creature index>:<time>:[:<required artifact type>:<number>]\*  
j : 50001 : Heal Troll : 20007 : 10 : Wand : 1 : Stone : 2 : Scroll : 1

I have added code to this sample which does the following:

* Reads that additional line type, "j".
* The Creature class now also implements the Runnable interface, implementing the method run ().
* (Yellow) The addJob (Scanner) method:
  + Connects the job to the proper creature
  + Uses the Creature.addJob (long, Scanner) method to process the data line, and pass the duration of the task to the proper Creature
  + Instantiates and starts the Creature thread
* (Green) The run () method in the Creature class:
  + Sets up a JFrame to hold the display, particularly the JProgressBar
  + Puts a label in the JFrame, instantiates and adds the JProgressBar
  + Calculates the start and stop times for this job
  + Runs the job in a while loop, using Thread.sleep to simulate the delay in processing the job.
  + Updates the progress bar inside the while loop to display the progress of the job visually to the user.
* To end the program, kill the JFrame.
* There is still a lot to do to get this working nicely with the rest of the code you have already written.

// File: Cave2.java  
// Date: Feb 17, 2012  
// Author: Nicholas Duchon  
// Semester project  
// - started with Cave.java and added JProgressBar stuff.

import java.util.ArrayList;  
import java.util.Scanner;  
import java.util.HashMap;  
import java.io.File;  
import java.io.FileNotFoundException;  
import javax.swing.JProgressBar;  
import javax.swing.JFrame;  
import javax.swing.JLabel;  
import java.awt.BorderLayout;

public class Cave2 {  
 ArrayList <Party>    parties = new ArrayList <Party>    (); // the parties  
 ArrayList <Creature> loners  = new ArrayList <Creature> (); // creatures not in parties  
 ArrayList <Treasure> drops   = new ArrayList <Treasure> (); // Treasures not carried  
 ArrayList <Artifact> magics  = new ArrayList <Artifact> (); // Artifacts not carried  
   
 // hashmap of everything by index  
 HashMap <Integer, CaveElement> hh = new HashMap <Integer, CaveElement> ();  
   
 public static void main (String args []) {  
   Scanner sf = null;  
   String fileName = "../../dataA.txt";  
   try {  
     sf = new Scanner (new File (fileName));  
   } catch (FileNotFoundException e) {  
     System.out.println ("File not found: " + fileName);  
     System.exit (1);  
   }  
   Cave2 cv = new Cave2 ();  
   cv.readFile (sf);  
     
   System.out.println (cv);  
 } // end main  
   
 public String toString () {  
   String sb = "";  
   sb += toString (parties, "Party List");  
   sb += toString (loners , "Unaligned Creatures");  
   sb += toString (drops  , "Unheld Treasures");  
   sb += toString (magics , "Unheld Artifacts");  
//     sb += toString (hh.values (), "HashMap");  
   return sb;  
 } // end method dump  
   
 String toString (Iterable <? extends CaveElement> it, String label) {  
   String sb = "\n----- DUMP " + label + " -------";  
   for (CaveElement c: it)  
     sb += "\n" + c.toString ();  
   return sb;  
 } // end method dump Iterable String  
   
 void readFile (Scanner sf) {  
   String inline;  
   Scanner line;  
   while (sf.hasNext()) {  
     inline = sf.nextLine().trim();  
//       System.out.printf (">%s<\n", inline);  
     if (inline.length() == 0) continue;  
     line = new Scanner (inline).useDelimiter ("\\s\*:\\s\*"); // compress white space also, else nextInt fails  
     switch (inline.charAt (0)) {  
       case 'p': addParty    (line); break;  
       case 'c': addCreature (line); break;  
       case 't': addTreasure (line); break;  
       case 'a': addArtifact (line); break;  
       case 'j': addJob      (line); break;  
     } // end switch  
   } // end while reading data file  
 } // end readFile  
   
 public void addJob (Scanner sc) {  
   //    j:<index>:<name>:<creature index>:<time>:[:<required artifact type>:<number>]\*  
   sc.next (); // dump first field, j  
   int jobIndex = sc.nextInt ();  
   String jobName = sc.next ();  
   int target = sc.nextInt ();  
   Creature c = (Creature)(hh.get (target));  
   long time = sc.nextInt ();  
   c.addJob (time, sc);  
   Thread t = new Thread (c);  
   t.start (); // starts method Creature.run()  
 } // end method addJob  
   
 public void addParty (Scanner sc){  
   Party pt = new Party (sc);  
   parties.add (pt);  
   hh.put (pt.index, pt);  
 } // end method addParty  
   
 public void addCreature  (Scanner sc) {  
   Creature c = new Creature ();  
   int target = c.makeCreature (sc);  
   Party p = (Party)(hh.get (target));  
   if (p == null)  
     loners.add (c);  
   else {  
     p.addCreature (c);  
     c.party = p;  
   }  
   hh.put (c.index, c);  
 } // end method addToParty  
   
 public void addTreasure (Scanner sc) {  
   Treasure t = new Treasure ();  
   int target = t.makeTreasure (sc);  
   Creature c = (Creature)(hh.get (target));  
   if (c == null)  
     drops.add (t);  
   else {  
     c.addTreasure (t);  
     t.holder = c;  
   }  
   hh.put (t.index, t);  
 } // end method addTreasure  
   
 public void addArtifact (Scanner sc) {  
   Artifact a = new Artifact ();  
   int target = a.makeArtifact (sc);  
   Creature c = (Creature)(hh.get (target));  
   if (a == null)  
     magics.add (a);  
   else {  
     c.addArtifact (a);  
     a.holder = c;  
   }  
   hh.put (a.index, a);  
 } // end method addArtifact

} // end class Cave

class CaveElement {}

//    p:<index>:<name>  
class Party extends CaveElement {  
 ArrayList <Creature> members = new ArrayList <Creature> ();  
 int index;  
 String name;  
   
 public Party (Scanner s) {  
   s.next (); // dump first field, p  
   index = s.nextInt ();  
   name  = s.next();  
 } // String constructor  
   
 public void addCreature (Creature c) {  
   members.add (c);  
 } // end method addCreature  
   
 public String toString () {  
   String sr = String.format ("p:%6d:%s", index, name);  
   for (Creature c: members)  
     sr += "\n   " + c;  
   return sr;  
 } // end toString method  
   
} // end class Party

//    c:<index>:<type>:<name>:<party>:<empathy>:<fear>:<carrying capacity>  
class Creature extends CaveElement implements Runnable {  
 ArrayList <Treasure> treasures = new ArrayList <Treasure>  ();  
 ArrayList <Artifact> artifacts = new ArrayList <Artifact> ();  
 int index;  
 String type, name;  
 Party party = null;  
 double empathy, fear, capacity;  
 long jobTime;  
   
 public int makeCreature (Scanner s) {  
   int partyInd;  
   s.next (); // dump first field, c  
   index    = s.nextInt ();  
   type     = s.next ();  
   name     = s.next ();  
   partyInd = s.nextInt ();  
   empathy  = s.nextDouble ();  
   fear     = s.nextDouble ();  
   capacity = s.nextDouble ();  
   return partyInd;  
 } // Scanner Creature factory method  
   
 public void addTreasure (Treasure t) {treasures.add (t);}  
 public void addArtifact (Artifact a) {artifacts.add (a);}  
   
 public void addJob (long t, Scanner sc) {  
   jobTime = t;  
 } // end addJob  
   
 public void run () {  
   JProgressBar pm = new JProgressBar ();  
   pm.setStringPainted (true);

   JFrame jf = new JFrame ("Demo");  
   jf.add (new JLabel (" Close this window to end program "), BorderLayout.PAGE\_START);  
   jf.add (pm, BorderLayout.CENTER);  
   jf.pack ();  
   jf.setVisible (true);  
   jf.setDefaultCloseOperation (JFrame.EXIT\_ON\_CLOSE);  
   jf.setLocationRelativeTo (null);

   System.out.println ("Starting: " + this);  
   System.out.println ("Job duration: " + jobTime);

   long time = System.currentTimeMillis();  
   long startTime = time;  
   long stopTime = time + 1000 \* jobTime;  
   System.out.println ("Stop time: " + stopTime);  
   double duration = stopTime - time;

   while (time < stopTime) {  
     try {  
       Thread.sleep (100);  
     } catch (InterruptedException e) {}  
//       System.out.printf ("Running, time: %.2f\n", (time - startTime)/1000.0);  
     pm.setValue ((int)(((time - startTime) / duration) \* 100));  
     time = System.currentTimeMillis ();  
   } // end runninig

   pm.setValue (100);

 } // end method run - implements runnable  
   
 public String toString () {  
   String sb = "";  
   if (party == null)  
        sb += String.format ("c:%6d: %s : %s : %6d : %4.1f : %4.1f : %4.1f", index, type, name,           0, empathy, fear, capacity);  
   else sb += String.format ("c:%6d: %s : %s : %6d : %4.1f : %4.1f : %4.1f", index, type, name, party.index, empathy, fear, capacity);  
   for (Treasure t: treasures) sb += "\n        " + t;  
   for (Artifact a: artifacts) sb += "\n        " + a;  
   return sb;  
 } // end toString method  
}

//    t:<index>:<type>:<creature>:<weight>:<value>  
class Treasure extends CaveElement {  
 int index;  
 String type;  
 double weight, value;  
 Creature holder;  
   
 public int makeTreasure (Scanner s) {  
   int partyInd;  
   s.next (); // dump first field, c  
   index    = s.nextInt ();  
   type     = s.next ();  
   partyInd = s.nextInt ();  
   weight   = s.nextDouble ();  
   value    = s.nextDouble ();  
   return partyInd;  
 } // Scanner Treasure factory method  
   
 public String toString () {  
   if (holder == null)  
   return String.format ("t:%6d: %s : %6d : %4.1f : %4.1f", index, type,            0, weight, value);  
   return String.format ("t:%6d: %s : %6d : %4.1f : %4.1f", index, type, holder.index, weight, value);  
 } // end toString method  
} // end class Treasure

//    a:<index>:<type>:<creature>[:<name>]  
class Artifact extends CaveElement {  
 int index;  
 String type, name = "";  
 Creature holder;  
   
 public int makeArtifact (Scanner s) {  
   int partyInd;  
   s.next (); // dump first field, c  
   index    = s.nextInt ();  
   type     = s.next ();  
   partyInd = s.nextInt ();  
   if (s.hasNext()) name     = s.next ();  
   return partyInd;  
 } // Scanner Artifact factory method  
   
 public String toString () {  
   if (holder == null)  
   return String.format ("a:%6d: %s : %6d : %s", index, type,            0, name);  
   return String.format ("a:%6d: %s : %6d : %s", index, type, holder.index, name);  
 } // end toString method  
} // end class Artifact