**ASSIGNMENT-1**

**1. How to make JAVA objects immutable?**

1. Don't provide "setter" methods — methods that modify fields or objects referred to by fields.
2. Make all fields final and private.
3. Don't allow subclasses to override methods. The simplest way to do this is to declare the class as final. A more sophisticated approach is to make the constructor private and construct instances in factory methods.
4. If the instance fields include references to mutable objects, don't allow those objects to be changed:
   1. Don't provide methods that modify the mutable objects.
   2. Don't share references to the mutable objects. Never store references to external, mutable objects passed to the constructor; if necessary, create copies, and store references to the copies. Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your methods.
5. **Advantages of immutable objects over normal objects.**
6. You can share immutable objects between threads without danger of changes confusing the other thread. You don’t need any locking. Two threads can both work on an immutable object at the same time without any possibility of conflict.
7. Once you check the value, you know it has to **stay** safe. No one can pass you a value, then behind your back swap it to an unsafe one using a background thread. This is particularly important in high security situations where allowing an invalid value to sneak in could compromise system integrity, e.g. a filename. This is probably the main reason that Strings are immutable.
8. You can share duplicates by pointing them to a single instance. You need only one copy, e.g. String interning. This saves RAM (Random Access Memory). With mutable **StringBuilders** that were temporarily identical, you could not throw one away and replace it with a reference to the other, since at any moment their values could diverge again.
9. You can create substrings without copying. You just create a pointer into an existing base String guaranteed never to change. Immutability is the secret behind Java’s very fast substring implementation.
10. Immutable objects are much better suited to be HashTable keys. If you change the value of an object that is used as a hash table key without removing it and re-adding it you lose the mapping.
11. Since String is immutable, inside each String is a char[] exactly the correct length. Unlike a **StringBuilder** there is no need for padding to allow for growth.

**3. Usage of JTree and JTable.**

**JTREE**

The JTree class is used to display the tree structured data or hierarchical data. JTree is a complex component. It has a 'root node' at the top most which is a parent for all nodes in the tree. It inherits JComponent class.

publicclass**JTree**extends**JComponent**implements**Scrollable, Accessible**

import javax.swing.\*;

import javax.swing.tree.DefaultMutableTreeNode;

public class TreeExample {

JFrame f;

TreeExample(){

    f=new JFrame();

    DefaultMutableTreeNode style=new DefaultMutableTreeNode("Style");

    DefaultMutableTreeNode color=new DefaultMutableTreeNode("color");

    DefaultMutableTreeNode font=new DefaultMutableTreeNode("font");

    style.add(color);

    style.add(font);

    DefaultMutableTreeNode red=new DefaultMutableTreeNode("red");

    DefaultMutableTreeNode blue=new DefaultMutableTreeNode("blue");

    DefaultMutableTreeNode black=new DefaultMutableTreeNode("black");

    DefaultMutableTreeNode green=new DefaultMutableTreeNode("green");

    color.add(red); color.add(blue); color.add(black); color.add(green);

    JTree jt=new JTree(style);

    f.add(jt);

    f.setSize(200,200);

    f.setVisible(true);

}

public static void main(String[] args) {

    new TreeExample();

}}

**JTABLE**

The JTable class is used to display data in tabular form. It is composed of rows and columns.

import javax.swing.\*;

public class TableExample {

    JFrame f;

    TableExample(){

    f=new JFrame();

    String data[][]={ {"101","Amit","670000"},

                          {"102","Jai","780000"},

                          {"101","Sachin","700000"}};

    String column[]={"ID","NAME","SALARY"};

    JTable jt=new JTable(data,column);

    jt.setBounds(30,40,200,300);

    JScrollPane sp=new JScrollPane(jt);

    f.add(sp);

    f.setSize(300,400);

    f.setVisible(true);

}

public static void main(String[] args) {

    new TableExample();

}

}

**4. Use of mutability in Jtree**

A mutable tree node is one which can be altered. It is created using DefaultMutableTreenode.

A tree node may have at most one parent and 0 or more children. DefaultMutableTreeNode provides operations for examining and modifying a node's parent and children and also operations for examining the tree that the node is a part of. A node's tree is the set of all nodes that can be reached by starting at the node and following all the possible links to parents and children. A node with no parent is the root of its tree; a node with no 5 children is a leaf. A tree may consist of many subtrees, each node acting as the root for its own subtree.

DefaultMutableTreeNode has three constructors:

public DefaultMutableTreeNode()

public DefaultMutableTreeNode(Object userObject)

public DefaultMutableTreeNode(Object userObject, boolean allowsChildren)

The first constructor creates a node with no associated user object; you can associate one with the node later using the setUserObject method. The other two connect the node to the user object that you supply. The second constructor creates a node to which you can attach children, while the third can be used to specify that child nodes cannot be attached by supplying the third argument as false.

Using DefaultMutableTreeNode, you can create nodes for the root and for all of the data you want to represent in the tree, but how do you link them together? You could use the insert method that we saw above, but it is simpler to use the DefaultMutableTreeNode add method:

public void add(MutableTreeNode child);