

ADVANCED PROGRAMMING CONCEPTS
USING JAVA
(CSX-351)
Assignment no 1
COMPUTER SCIENCE AND ENGINEERING



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DR. B R AMBEDKAR NATIONAL INSTITUTE OF TECHNOLOGY
JALANDHAR – 144011, PUNJAB (INDIA)
AUGUST-DECEMBER, 2017

Submitted To:

Dr. Paramveer Singh

Asst. Professor

Department of CSE

Submitted by:

Shailender kumar

15103073

G-3

Assignment – 1

Q1: How to make JAVA objects immutable and what are its advantages over normal objects?

Ans: The following rules define a simple strategy for creating immutable objects. Not all classes documented as "immutable" follow these rules. This does not necessarily mean the creators of these classes were sloppy — they may have good reason for believing that instances of their classes never change after construction. However, such strategies require sophisticated analysis and are not for beginners.

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:
 - Don't provide methods that modify the mutable objects.
 - 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.

Doing so will often restrict the way you can call the class and its methods. It will also make you redesign your software and rethink your algorithms. However, there are many benefits of programming with immutable objects.

1. Immutable objects are thread-safe so you will not have any synchronization issues.
2. Immutable objects are good **Map** keys and **Set** elements, since these typically do not change once created.
3. Immutability makes it easier to write, use and reason about the code (class invariant is established once and then unchanged)
4. Immutability makes it easier to parallelize your program as there are no conflicts among objects.
5. The internal state of your program will be consistent even if you have exceptions.
6. References to immutable objects can be cached as they are not going to change.

Q2: Discuss the usage of jtree and jtable?

Ans: 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.

JTree class declaration

Let's see the declaration for javax.swing.JTree class.

1. **public class JTree extends JComponent implements** Scrollable, Accessible

Commonly used Constructors:

Constructor	Description
JTree()	Creates a JTree with a sample model.
JTree(Object[] value)	Creates a JTree with every element of the specified array as the child of a new root node.
JTree(TreeNode root)	Creates a JTree with the specified TreeNode as its root, which displays the root node.

Java JTree Example

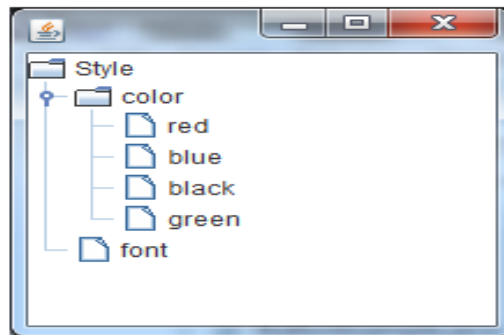
1. **import** javax.swing.*;
2. **import** javax.swing.tree.DefaultMutableTreeNode;
3. **public class** TreeExample {
4. JFrame f;
5. TreeExample(){
6. f=**new** JFrame();
7. DefaultMutableTreeNode style=**new** DefaultMutableTreeNode("Style");
8. DefaultMutableTreeNode color=**new** DefaultMutableTreeNode("color");
9. DefaultMutableTreeNode font=**new** DefaultMutableTreeNode("font");
10. style.add(color);
11. style.add(font);
12. DefaultMutableTreeNode red=**new** DefaultMutableTreeNode("red");
13. DefaultMutableTreeNode blue=**new** DefaultMutableTreeNode("blue");
14. DefaultMutableTreeNode black=**new** DefaultMutableTreeNode("black");
15. DefaultMutableTreeNode green=**new** DefaultMutableTreeNode("green");
16. color.add(red); color.add(blue); color.add(black); color.add(green);
17. JTree jt=**new** JTree(style);

```

18. f.add(jt);
19. f.setSize(200,200);
20. f.setVisible(true);
21. }
22. public static void main(String[] args) {
23.     new TreeExample();
24. }}

```

Output:



Jtable:

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

JTable class declaration

Let's see the declaration for javax.swing.JTable class.

Commonly used Constructors:

Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.

Java JTable Example

```

1. import javax.swing.*;
2. public class TableExample {
3.     JFrame f;
4.     TableExample(){
5.         f=new JFrame();
6.         String data[][]={ {"101","Amit","670000"},
7.                             {"102","Jai","780000"},

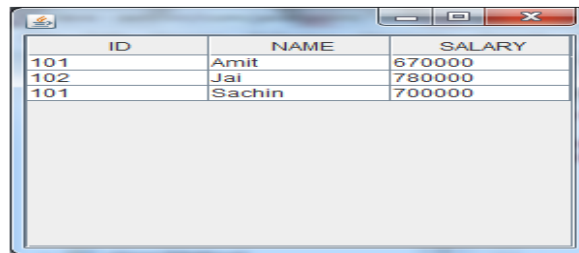
```

```

8.         {"101","Sachin","700000"}));
9.  String column[]={ "ID","NAME","SALARY"};
10.  JTable jt=new JTable(data,column);
11.  jt.setBounds(30,40,200,300);
12.  JScrollPane sp=new JScrollPane(jt);
13.  f.add(sp);
14.  f.setSize(300,400);
15.  f.setVisible(true);
16. }
17. public static void main(String[] args) {
18.     new TableExample();
19. }
20. }

```

Output:



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000

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 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);
```