Week-6: **Date:**

VISUAL BEANS:

AIM: Create a simple visual bean with a area filled with a color.

The shape of the area depends on the property shape. If it is set to true then the shape of the area is Square and it is Circle, if it is false.

The color of the area should be changed dynamically for every mouse click. The color should also be changed if we change the color in the "property window ".

DESCRIPTION:

A Bean is a JavaBeans component. Beans are independent, reusable software modules. Beans may be visible objects, like AWT components, or invisible objects, like queues and stacks. A builder/integration tool manipulates Beans to create applets and applications.

Beans consist of three things:

- **Events**

An event allows your Beans to communicate when something interesting happens. There are three parts to this communication:

EventObject

Event Listener - (the sink)

An Event Source (the Bean)

The event source defines when and where an event will happen. Classes register themselves as interested in the event, and they receive notification when the event happens. A series of methods patterns represents the registration process:

```
public synchronized void addListenerType(ListenerType l);
public synchronized void removeListenerType(  ListenerType l);
```

- **Properties**

Properties define the characteristics of the Bean. For instance, when examining an AWT TextField for its properties, you will see properties for the caret position, current text, and the echo character, among others. A property is a public attribute of the Bean, usually represented by a non-public instance variable. It can be read-write, read-only, or write-only. There are four different types of properties:

- **Simple -** As the name implies, simple properties represent the simplest of the four. To create a property, define a pair of set/get routines. Whatever name used in the pair of routines, becomes the property name
- **Indexed -** An indexed property is for when a single property can hold an array of values. The design pattern for these properties is:

```
public void setPropertyName (PropertyType[] list)
public void setPropertyName (
PropertyType element, int position)
```

```
public PropertyType[] getPropertyName ()
public PropertyType getPropertyName (int position)
```

- **Bound –** A bean that has the bound property generates an event when the property is changed. The event is of type *propertyChangeEvent* and is sent to objects that previously registered an interest in receiveing such notifications. In order for the notification to happen, you need to maintain a watch list for `PropertyChangeEvent`s via the `PropertyChangeSupport` class. First, you have to create a list of listeners to maintain:

```
private PropertyChangeSupport changes =
        new PropertyChangeSupport (this);
```

And then, you have to maintain the list:

```
public void addPropertyChangeListener (
 PropertyChangeListener p) {
changes.addPropertyChangeListener (p);
}
public void removePropertyChangeListener (
PropertyChangeListener p) {
changes.removePropertyChangeListener (p);
}
```

- **Constrained -** Constrained properties are similar to bound properties. In addition to maintaining a list of PropertyChangeListeners, the Bean maintains a list of `VetoableChangeListener`s. Then, prior to the Bean changing a property value, it asks the VetoableChangeListeners if its okay. If it isn't, the listener throws a `PropertyVetoException`, which you declare the set routine to throw.

➢ **Methods**

Bean methods are available for anyone to call by just making each public. However, you can restrict which methods are visible to the Bean builder/integration tool by providing a getMethodDescriptors method along with your Bean's BeanInfo. Every Bean can provide a supporting BeanInfo class to customize a Bean's appearance to an integration tool.

Procedural Steps to create a Java-Bean:

1) Creating a directory- Create a new directory in C:\beans\demo\sunw\demo with a new folder name colors
2) Create a java source file
3) Compile the java source file
4) Create a manifest file colors.mft in the directory called as C:\beans\demo
5) Create a jar file- to create a jar file type the following command in the command prompt

```
jar cfm ..\jars\colors.jar colors.mft sunw\demo\colors\*.class
```

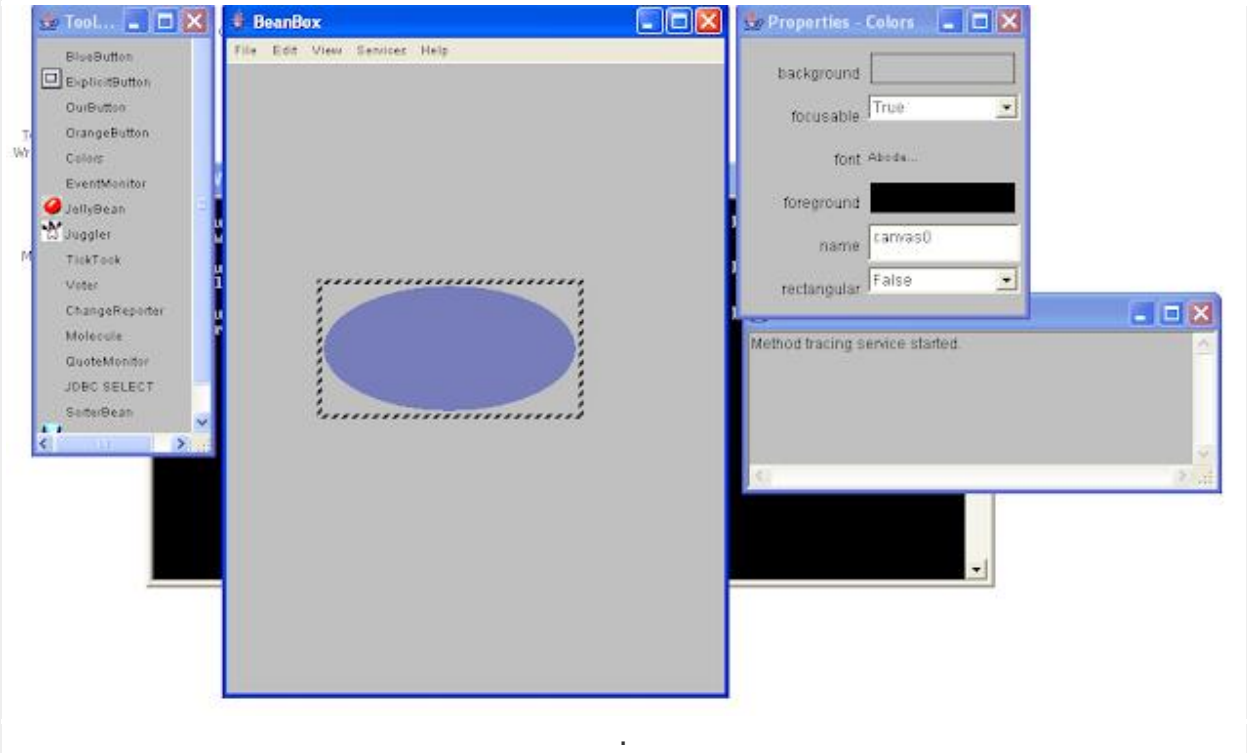
6) Start the BDK
7) Check whether the colors bean is placed in toolbox or not.

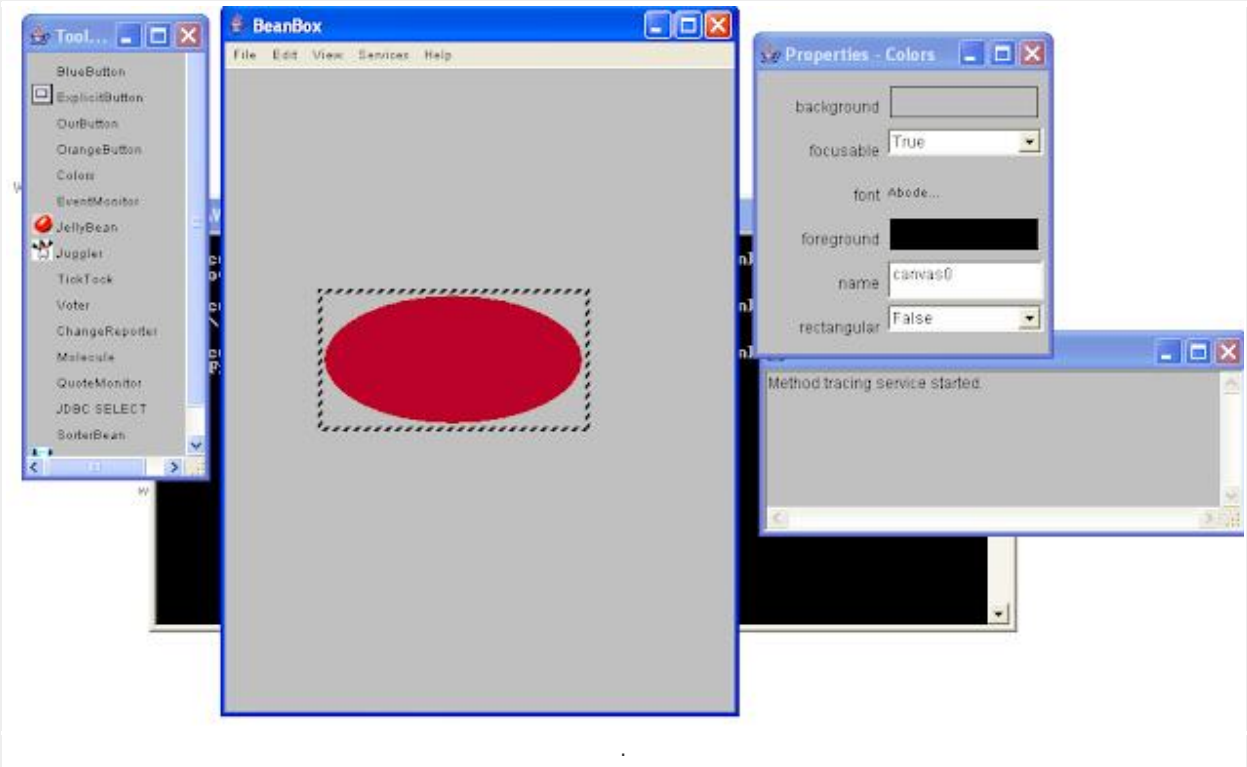
PROGRAM:

```
package sunw.demo.colors;
import java.awt.*;
import java.awt.event.*;
public class Colors extends Canvas
{
 transient private Color color;
private boolean rectangular;
public Colors()
{
 addMouseListener(new MouseAdapter(){
public void mousePressed(MouseEvent me){
change(); }
});
rectangular=false;
setSize(100,100);
change();
}
public boolean getRectangular()
{
 return rectangular;
}
 public void setRectangular(boolean flag)
{
 this.rectangular=flag;
repaint();
}
public void change()
{
 color=randomColor();
repaint();
}
 private Color randomColor()
{
 int r=(int)(255*Math.random());
 int g=(int)(255*Math.random());
 int b=(int)(255*Math.random());
```

```
return new Color(r,g,b);
}
public void paint(Graphics g)
{
Dimension d=getSize();
int h=d.height;
int w=d.width;
g.setColor(color);
if(rectangular)
{
g.fillRect(0,0,w-1,h-1);
}
else
{
g.fillOval(0,0,w-1,h-1);
}
}
}
```

OUTPUT:





RESULT:

Thus the colors bean is created successfully.

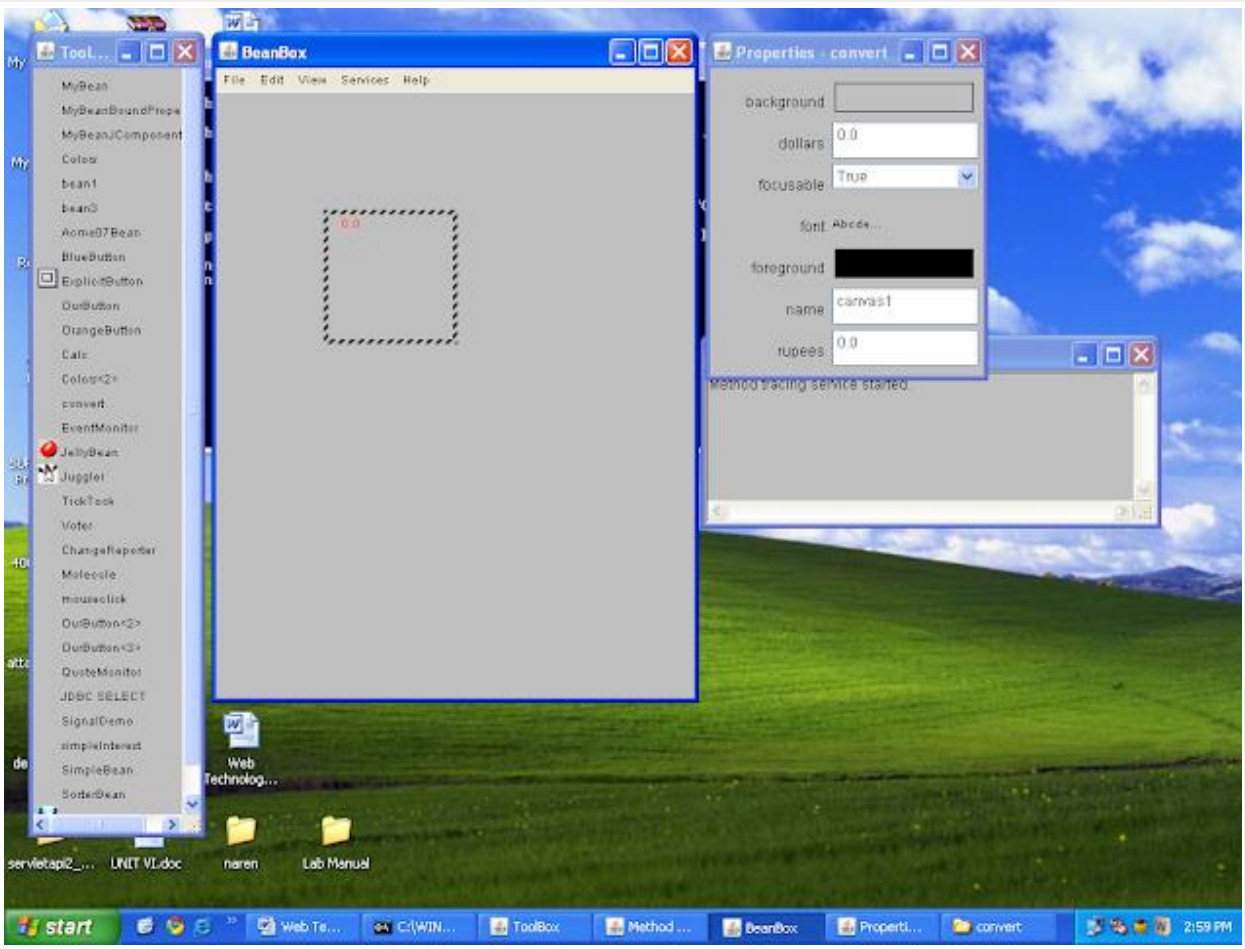
Program 2:

Visual Beans (program 2)
Convert.java

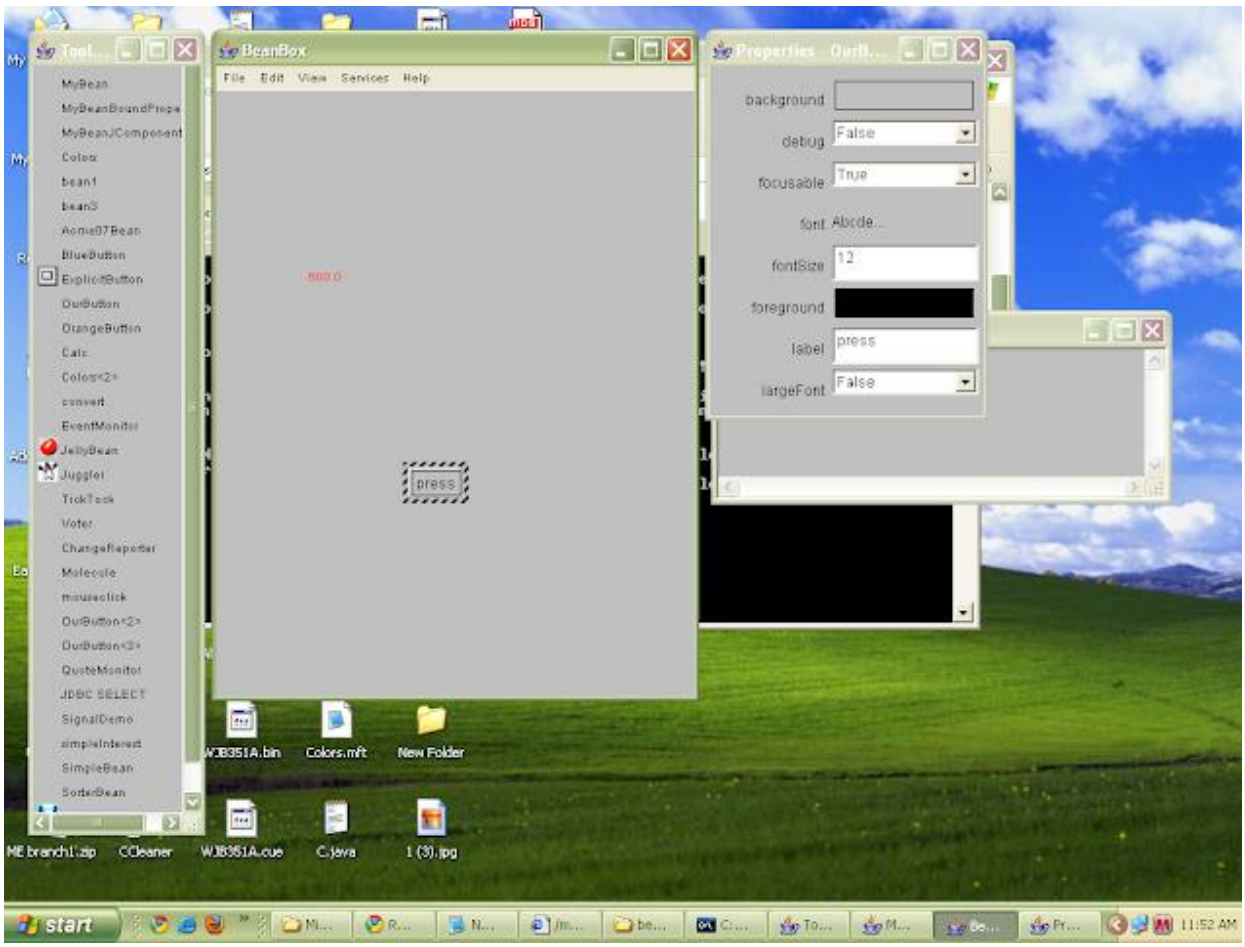
```
package sunw.demo.convert;
import java.awt.*;
import java.awt.event.*;
public class convert extends Canvas
{
        private double dollars=0.0;
        private double rupees=0.0;
        private double dollarvalue=0.0;

        public convert()
        {
        setSize(100,1000);
        }
        public double getDollars()
        {
```

```
        return dollars;
        }
        public void setDollars(double value)
        {
        this.dollars=value;
        }
        public void setRupees(double value)
        {
        this.rupees=value;
        }
        public double getRupees()
        {
        return rupees;
        }
        public void change()
        {
         dollarvalue= value();
        repaint();
        }
        private double value()
        {
        return rupees*dollars;
        }
        public void paint(Graphics g)
        {
        g.setColor(Color.red);
        g.drawString(String.valueOf(dollarvalue),10,10);
        }
}
```

Convert.mf
Name: sunw/demo/convert/convert.class
Java-Bean: True (press Enter)
(Carriage return compulsory)


BeanBox
File Edit View Services Help
Properties - convert
background
dollars
0.0
focusable
True
font
Abcde...
foreground
name
canvas1
rupees
0.0
Method tracing service started
MyBean
bean1
bean3
BlueButton
ExplicitButton
OutButton
OrangeButton
Calc
convert
EventMonitor
JellyBean
Juggler
TickTock
Voter
Molecule
mouseclick
SimpleBean
SorterBean
Web Technolog...
naren
Lab Manual
start
2:59 PM

.



.

Result

Thus the convertion bean is created successfully

Program 3:

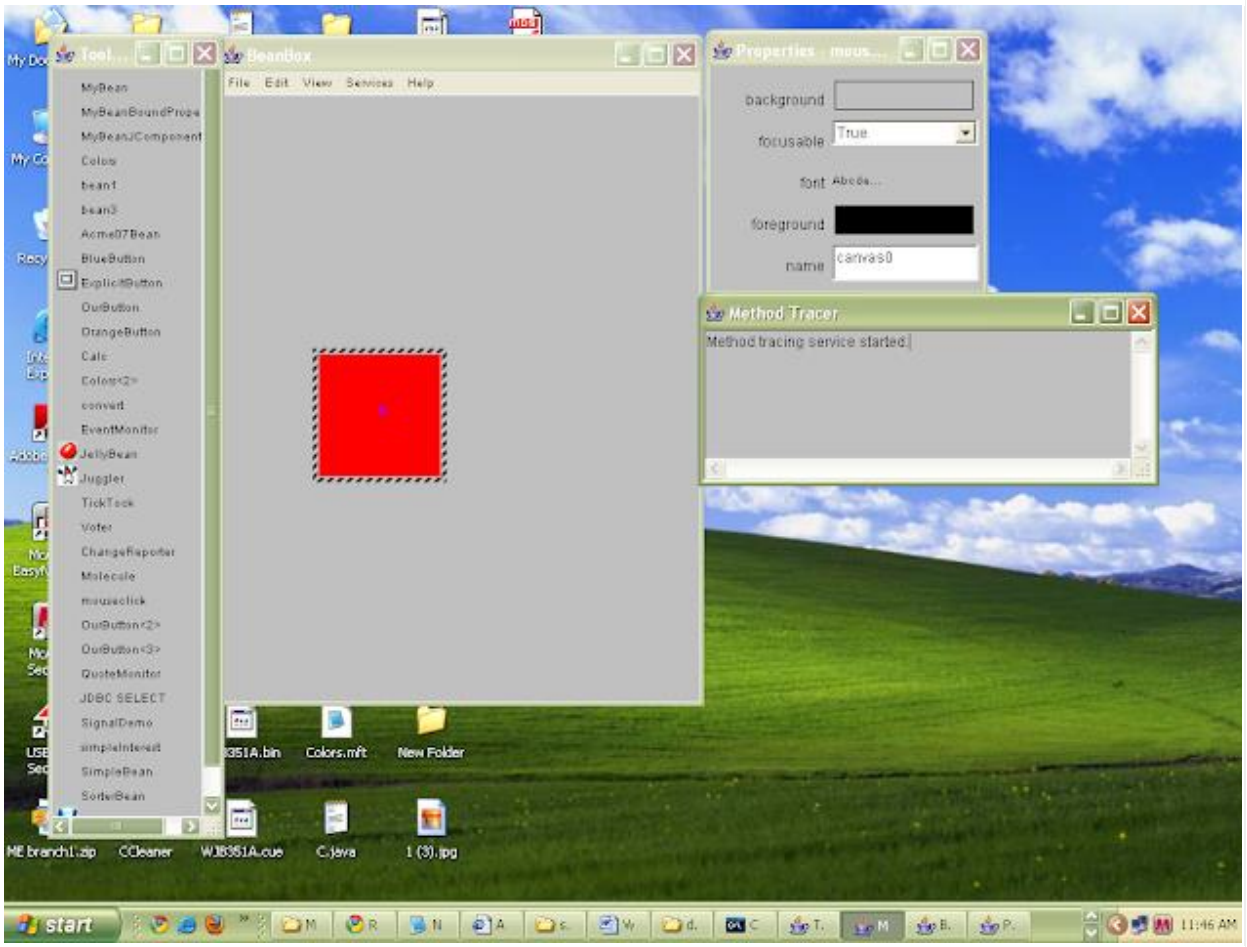
```
package sunw.demo.colors;
import java.awt.*;
import java.awt.event.*;
public class mouseclick extends Canvas {
public int count=0;
public mouseclick() {
addMouseListener(new MouseAdapter() {
public void mousePressed(MouseEvent me) {
change();
}
});
setSize(100,100);
}
public void change() {
```

```
count++;
repaint();
}
public void paint(Graphics g) {
Dimension d = getSize();
int h = d.height;
int w = d.width;
g.setColor(Color.red);
g.fillRect(0,0,100,100);
g.setColor(Color.blue);
g.drawString(String.valueOf(count),50,50);
}
}
```

Mouseclick.mft

Name: sunw/demo/colors/mouseclick.class
Java-Bean: True

Output



Result

Thus the Mouse Clicks bean is created successfully