Experiment No.11

Title: Implementing the concept of Multithreading using Thread class
Aim: To study what is multithreading and how to handle multiple tasks simultaneously.

Theory:

A ***thread*** is the flow of execution of a single set of program statements. ***Multithreading*** consists of multiple sets of statements which can be run in parallel. With a single processor only one thread can run at a time but strategies are used to make it appear as if the threads are running simultaneously. Depending on the operating system ***scheduling method***, either timeslicing or interrupt methods will move the processing from one thread to another.

Serialization is the process of writing the state of an object to a byte stream. It can be used to save state variables or to communicate through network connections.

The Thread Class

The ***Thread Class*** allows ***multitasking*** (ie running several tasks at the same time) by instantiating (ie creating) many threaded objects, each with their own run time characteristics. Tasks that slow the processor can be isolated to prevent apparent loss of GUI response. One way to create threads is to ***extend*** the Thread class and override the ***run()*** method such as:

```
class HelloThread extends Thread
{
  public void run()
  {
    for      int      x=0;x<100;      ++x)
      System.out.print(" Hello ");
  }
}
```

Multithreading:

Multithreading is a technique that allows a program or a process to execute many tasks concurrently (at the same time and parallel). It allows a process to run its tasks in parallel mode on a single processor system

In the multithreading concept, several multiple lightweight processes are run in a single process/ task or program by a single processor. For Example, When you use a **word processor** you perform a many different tasks such as **printing, spell checking** and so on. Multithreaded software treats each process as a separate program.

Advantages of multithreading over multitasking:

- Reduces the computation time.
- Improves performance of an application.

- Threads share the same address space so it saves the memory.
- Context switching between threads is usually less expensive than between processes.
- Cost of communication between threads is relatively low.

Statement:

Write a program that bounces a blue ball inside a JPanel. The ball should begin moving with a mousePressed event. When the ball hits the edge of the JPanel, it should bounce off the edge and continue in the opposite direction. The ball should be updated using a Runnable.

Program:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Ball extends JApplet implements Runnable
{
private Thread blueBall;
private boolean xUp, yUp, bouncing;
private int x, y, xDx, yDy;
private final int MAX_X = 200, MAX_Y = 200;

public void init()
{
//initialize values
xUp  = false;
yUp  = false;
xDx = 1;
yDy = 1;
bouncing = false;

//let ball applet be its own MouseListener
addMouseListener(

new MouseListener()
{
public void mousePressed( MouseEvent event )
{
createBall( event );
//delegate call to ball starter
}
public void mouseExited( MouseEvent event ) {}
public void mouseClicked( MouseEvent event ) {}
public void mouseReleased( MouseEvent event ) {}
```

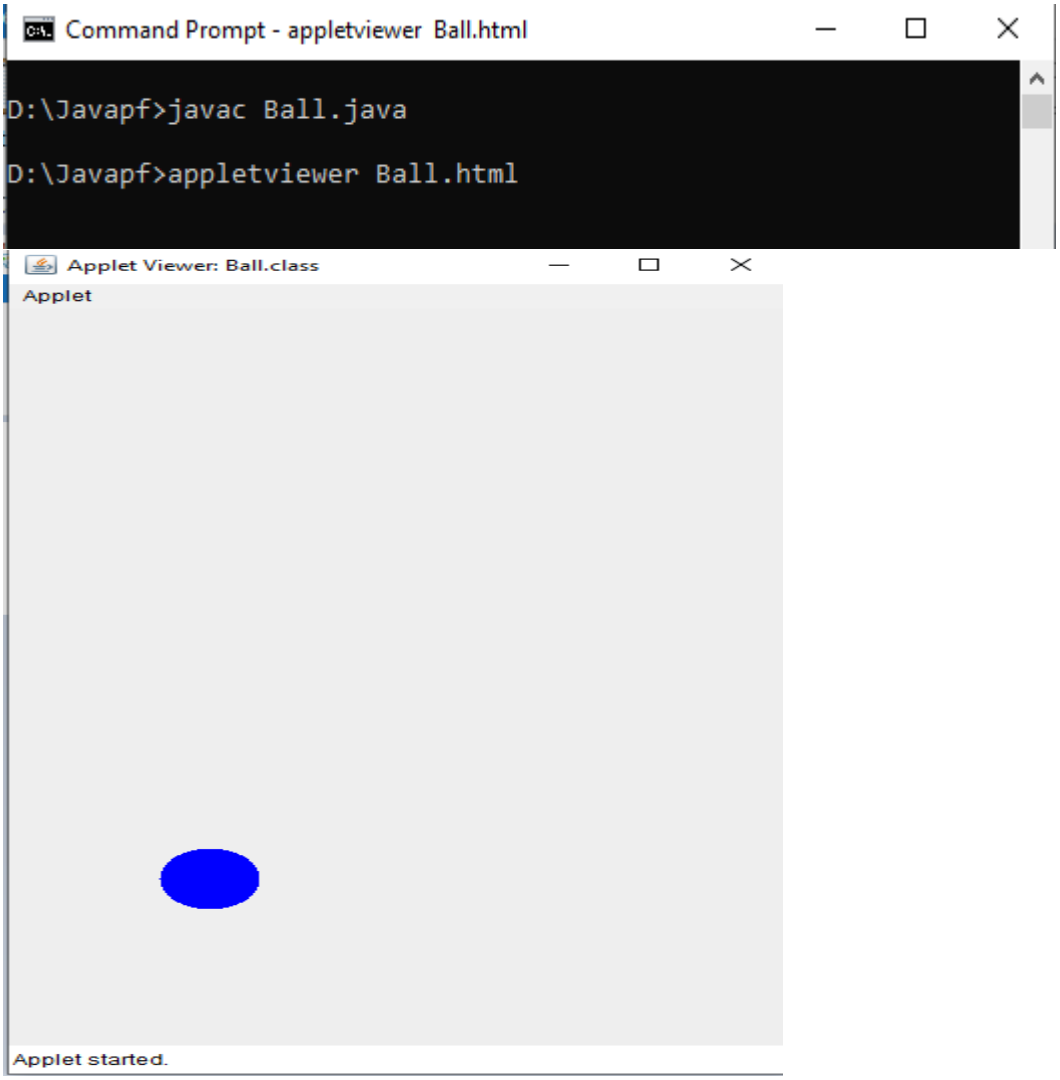
```
public void mouseEntered( MouseEvent event ) {}
}
);
setSize( MAX_X, MAX_Y ); // set size of Applet
}

//creates a ball and sets it in motion if
//no ball exists
private void createBall( MouseEvent event )
{
if ( blueBall == null )
{
x = event.getX();
y = event.getY();
blueBall = new Thread( this );

bouncing = true; // start ball's bouncing
blueBall.start();
}
}
//called if applet is closed. by setting blueBall to null, threads will be ended.
public void stop()
{
blueBall = null;
}

// draws ball at current position
public void paint( Graphics g )
{
super.paint( g );

if ( bouncing )
{
g.setColor( Color.blue );
g.fillOval( x, y, 50, 50 );
}
}
// action to perform on execution, bounces ball perpetually until applet is closed.
public void run()
{

while ( true )
{
//sleep for a random interval
try
{
```

```
blueBall.sleep( 2000 );
}

//process InterrupedException during sleep
catch( InterruptedException exception )
{
System.err.println( exception.toString() );
}

// determine new x position
if ( xUp == true )
x += xDx;
else
x -= xDx;

//determine new y position
if ( yUp == true )
y += yDy;
else
y -= yDy;

// randomize variables for creating next move
if ( y <= 0 )
{
yUp = true;
yDy = ( int ) ( Math.random() * 5 + 2 );
yUp = false;
}
if ( x <= 0 )
{
xUp = true;
xDx = ( int ) ( Math.random() * 5 + 2 );
}
else if ( x >= MAX_X - 10 )
{
xUp = false;
xDx = ( int ) ( Math.random() * 5 + 2 );
}
repaint();
}
}
}
```

Applet code:-

```
<html>
<body>
<applet code="Ball.class" width="300" height="300">

</applet>
</body>
</html>
```

Output:-



Conclusion: Thus we studied implementing the concept of Multithreading using Thread class.