**Experiment No.12**

| |
|---|
| **Title:** Implementing the concept of Multithreading using Runnable interface |
| **Aim:** To study what is multithreading and how to handle multiple tasks simultaneously. |

**Theory:**

**Implementing java.lang.Runnable :**

Implementing the Runnable interface gives you a way to extend from any class you like, but still define behavior that will be run by a separate thread. It looks like this:

```
class MyRunnable implements Runnable
  {
        public void run()
         {
              System.out.println("Important job running in MyRunnable");
        }
}
```

Regardless of which mechanism you choose, you've now got yourself some code that can be run by a thread of execution. So now let's take a look at *instantiating* your thread-capable class, and then we'll figure out how to actually get the thing *running*.

However if you need to inherit from another class as well, you can ***implement*** a Runnable interface instead and write the required ***run()*** method.

Thread object methods are used on instantiated thread objects to control the thread appropriately. These methods include ***currentThread(), getName(), getPriority(), sAlive(), join(), run(), setName(string), setPriority(int), sleep(longInt)*** and ***start()***.

**Synchronization**

Thread synchronization is required when two or more threads need to share a resource. A ***monitor*** (aka semaphore) is an object that provides a muually exclusive lock (mutex). Java provides the ***synchronized*** keyword as the key that locks/unlocks an object. It can be used as a class or method modifier or as a statement (very localized). Any long running method should not be synchonized as it would become a traffic bottleneck. To guarantee that a variable is threadsafe (ie. not shared between threads) it can be marked as ***volatile***.

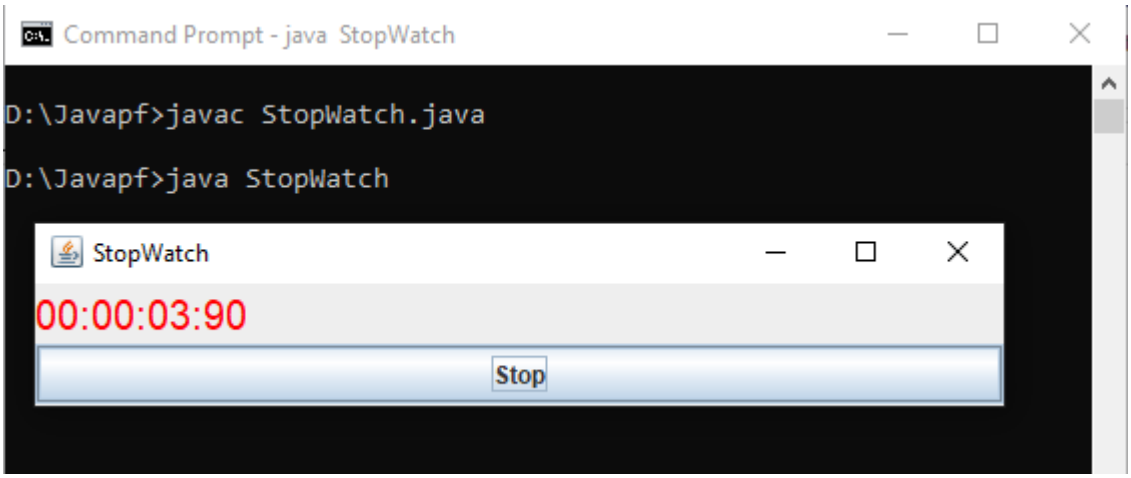
**Statement:** Create Stop Watch with Swing GUI and Multithreading.

**Program:**

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.text.DecimalFormat;
import java.text.NumberFormat;

public class StopWatch extends JFrame implements ActionListener,Runnable
{
JLabel disp; JButton btn; boolean stop=false; int i,j,k,l;
public StopWatch()
{
disp=new JLabel();
btn=new JButton("Start");
disp.setFont(new Font("Helvetica",Font.PLAIN,20));
disp.setBackground(Color.cyan);
disp.setForeground(Color.red);
Container c=getContentPane();
c.setLayout(new GridLayout(2,1));
c.add(disp);
c.add(btn);
btn.addActionListener(this);
}
public void run()
{
for(i=0;;i++)
{
for(j=0;j< 60;j++)
{
for(k=0;k< 60;k++)
{
for(l=0;l< 100;l++)
{

if(stop)
{
break;
}
```

```
NumberFormat nf = new DecimalFormat("00");
disp.setText(nf.format(i)+":"+nf.format(j)+":"+nf.format(k)+":"+nf.format(l));
try
{
Thread.sleep(10);
}
catch(Exception e){}
}          }
}
}
}
public void actionPerformed(ActionEvent ae)
{
Thread t=new Thread(this);
if(ae.getActionCommand().equals("Start"))
{
t.start(); btn.setText("Stop");


}
else
{
stop=true;
}
}


public static void main(String[] args)
{
StopWatch s=new StopWatch();
s.setSize(500,100);
s.setVisible(true);
s.setTitle("StopWatch");
s.setDefaultCloseOperation(EXIT_ON_CLOSE);
}
}
```

**Output:-**




**Conclusion:** Thus we implementing the concept of Multithreading using Runnable interface.