Made By: Vinay Ranjan
Roll no. 18MCA052

JAVA Theory

Assignment #6

Q1. How will you create your own collection type explain it with a suitable programming example?

Ans. Collection is simply an object that holds a collection (or a group) of the objects.
--> Each item in a collection is called element.
--> Collection framework is the environment to represent and manipulate the collections.
Java collections framework consists:
1. Interfaces = [add (), size (), remove ()]
2. Implementation class = [array list, hash map, linked list]
3. Algorithm = Searching, Sorting, Shuffling
To create our own collection type, firstly choose any implementations class like array list, hash map, linked list, tree list etc.
e.g. creating a collection of array list type.

```
Import java.util.ArrayList;
Class Triplet<X,Y,Z>
{
        Public X num1; public Y num2; public Z num3;
        Triplet(X num1, Y num2, Z num3)
        {
                this.num1=num1; this.num2=num2; this.num3=num3;
        }
        @Override
        Public String toString ()
        {
        Return String. Format ("["+num1+"  "+num2+"  "+num3+"]");
        }
}
Class Main
{
        ArrayList<Triplet>  list=new  ArrayList<>();
        Public  void  putData(int  varInt[],  String  varString[],  long  varLong[])
        {
        for  (int  i  =  0;  i  <  4;  i++)  {
```

```
        List.add(new Triplet<Integer,String,Long>(varInt[i], varString[i], varLong[i]));
        }
}

//function to view Data from collection public void getData()
{
System.out.println(list);
}
public static void main(String args[])
{
        String varString[] = {"Write", "Once", "Run", "AnyWhere"};
        int varInt[] = {0, 12, 22, 56};
        long varLong[] = {12345678906L, 98765434567L, 846098763234567898L, 689645 6783456L};
        Main custom = new Main(); custom.putData(varInt, varString, varLong); custom.getData();
        }
}
```

Q2. What do you mean by thread Synchronization? What are the possible ways available in java to handle the Synchronization problem?

Ans. When two or more threads need access to shared resources they need some way to ensure that the resource will be used by only one thread at a time. This process is known as thread synchronization.
--> Synchronized keyword used to refer as critical section.
--> Once a thread enters any synchronized method or an instance, no other thread can enter any other synchronized method on the same instance.

The method to handle synchronization problem:
1. Declaring variable as synch: It is accessible to only one thread at a time.

```
Syntax: Synchronized int x;
```

2. Using synchronized method: While a thread is inside a synchronized method all other threads that try to call on the same instance have to wait.

```
Public synchronized void increment ()
{
        X++;
}
```

3. Synchronized Statement: A synchronized Statement can only be executed once the thread has obtained a lock for the object or the class that has been referred to in the statement. Synchronized statement contains a synchronized block, within which is placed objects and methods that are to be synchronized.

```
Public void add(String str)
{
```

```
                synchronized (this)
                {
                        str1=str;
                        cnt++;
                }
                strlist.add (str);
        }
```

Q3. What is blocking Queue.Explain the concept of thread pooling and its application in the real life enterprise application development?

Ans.
A blocking queue is a queue that blocks when you try to delete element from it and the queue is empty, or if you try to insert items to it and the queue is already full. Blocking Queue does not accept null value. If we try to enqueue null item, then it throws NullPointerException. Blocking Queue interface implementations are thread-safe. Blocking Queue methods are synchronized.
Blocking Queue is an Sub interface.
There Super interfaces are Collection<E>, Iterable<E>, Queue<E>. There implementing Classes are:
ArrayBlockingQueue, DelayQueue, LinkedBlockingDeque, LinkedBlockingQueue, LinkedTransferQueue, PriorityBlockingQueue, SynchronousQueue.
There are two type of Blocking Queue:
1. Unbounded
2. Bounded

1. **Unbounde Blocking Queue:**
The Capacity of blocking queue will be set to Integer.MAX_VALUE. In case of unbounded blocking queue, queue will never block because it could grow to a very large size. When you add elements its size grows.
2. **Bounded Blocking queue:**
In case of bounded queue you can create a queue by passing the capacity of queue in queues.
Methods in Blocking Queue Interface:
add (E e), contains (Object o), drainTo (Collection c), offer (E e), poll (long timeout, TimeUnit unit), put (E e), take (), remove (Object o), remainingCapacity ().

Thread Pooling:
Thread pool is a group of thread which is ready to perform the given tasks. Thus a thread pool may contain 1 or more threads. Threads in a thread pool perform task with good coordination. Every thread pool will restrict the number of threads.

Types of Thread Pools:

Single Thread executor, Cached Thread Pool, Fixed Thread Pool, Schedule threads Pool, single thread schedule thread pool.
In the real life example, suppose that there is MoivestorageServer and multiple user are request for seeing a movies, suppose a user request for a movie to stream then a server create a stream and serve them, but its take time to create a movie stream and serve, in that time user wait for their stream, suppose we have pool where we create a multiple stream and store them when user request the stream then server accept the request and get the stream at pool and provide them to user such that user don't wait for their stream. When server is free at that time server create a m-multiple stream and store at pool.

Q4. Write a simple timer program that can periodically print a timeout message?

Ans.

```
import java.util.Timer; import java.util.TimerTask;
/**
*       Simple demo that uses java.util.Timer to schedule a task
*       to execute once 5 seconds have passed.
*/
Public class Reminder
{
        Timer timer;
        public Reminder(int seconds)
        {
                timer = new Timer();
                timer.schedule(new RemindTask(), seconds*1000);
        }
        Class RemindTask extends TimerTask
        {
                public void run()
                {
                        System.out.println("Time's up!"); timer.cancel(); //Terminate the timer thread
                }
        }
        public static void main(String args[])
        {
                new Reminder(5);
                System.out.println("Task scheduled.");
        }
}
```

Q5. What are the different types of Events used in AWT/SWING programming? Explain various methods available for handling mouse and key board related events?

Ans.

Different types of event in Java AWT:
1. ActionEvent
2. AdjustmentEvent
3. ComponentEvent
4. ContainerEvent
5. FocusEvent
6. InputEvent
7. ItemEvent
8. KeyEvent
9. MouseEvent
10. PaintEvent
11. TextEvent
12. WindowEvent

These are twelve mentioned events are explained as follows:

1. **ActionEvent:** This is the ActionEvent class extends from the AWTEvent class. It indicates the component-defined events occurred i.e. the event generated by the component like Button, Checkboxes etc. The generated event is passed to every EventListener objects that receives such types of events using the addActionListener() method of the object.

2. **AdjustmentEvent:** This is the Adjustment Event class extends from the AWTEvent class. When the Adjustable Value is changed then the event is generated.

3. **ComponentEvent:** ComponentEvent class also extends from the AWTEvent class. This class creates the low-level event which indicates if the object moved, changed and it's states (visibility of the object). This class only performs the notification about the state of the object. The ComponentEvent class performs like root class for other component-level events.

4. **ContainerEvent:** The ContainerEvent class extends from the ComponentEvent class. This is a low-level event which is generated when container's contents changes because of addition or removal of a components.

5. **FocusEvent:** The FocusEvent class also extends from the ComponentEvent class. This class indicates about the focus where the focus has gained or lost by the object. The generated event is passed to every objects that is registered to receive such type of events using the addFocusListener() method of the object.

6. **InputEvent:** The InputEvent class also extends from the ComponentEvent class. This event class handles all the component-level input events. This class acts as a root class for all component-level input events.

7**. ItemEvent:** The ItemEvent class extends from the AWTEvent class. The ItemEvent class handles all the indication about the selection of the object i.e. whether selected or not. The

generated event is passed to every ItemListener objects that is registered to receive such types of event using the addItemListener() method of the object.

8**. KeyEvent:** KeyEvent class extends from the InputEvent class. The KeyEvent class handles all the indication related to the key operation in the application if you press any key for any purposes of the object then the generated event gives the information about the pressed key. This type of events check whether the pressed key left key or right key, 'A' or 'a' etc.

9. **MouseEvent:** MouseEvent class also extends from the InputEvent class. The MouseEvent class handle all events generated during the mouse operation for the object. That contains the information whether mouse is clicked or not if clicked then checks the pressed key is left or right.

10**. PaintEvent:** PaintEvent class also extends from the ComponentEvent class. The PaintEvent class only ensures that the paint() or update() are serialized along with the other events delivered from the event queue.

11. **TextEvent:** TextEvent class extends from the AWTEvent class. TextEvent is generated when the text of the object is changed. The generated events are passed to every TextListener object which is registered to receive such type of events using the addTextListener() method of the object.

12. **WindowEvent** : WindowEvent class extends from the ComponentEvent class. If the window or the frame of your application is changed (Opened, closed, activated, deactivated or any other events are generated), WindowEvent is generated.

Handling mouse events:

--> MouseEvent class also extends from the InputEvent class. The MouseEvent class handles all events generated during the mouse operation for the object. That contains the information whether mouse is clicked or not if clicked then checks the pressed key is left or right.

Mouse events are handled using mouse event listeners. Following are some mouse event listeners:

1. public abstract voidmouseClicked(MouseEvent e);
2. public abstract void mouseEntered(MouseEvent e);
3. public abstract void mouseExited(MouseEvent e);
4. public abstract void mousePressed(MouseEvent e);
5. public abstract void mouseReleased(MouseEvent e);

```
import java.awt.*; import java.awt.event.*;
public class MouseListenerExample extends Frame implements MouseListener
{
        Label l;
```

```
    MouseListenerExample()
    {
        addMouseListener(this);
        l=new Label();
        l.setBounds(20,50,100,20);
        add(l);
        setSize(300,300); setLayout(null); setVisible(true);
    }
    Public void mouseClicked(MouseEvent e)
    {
        l.setText("Mouse Clicked");
    }
    public void mouseEntered(MouseEvent e)
    {
        l.setText("Mouse Entered");
    }
    public void mouseExited(MouseEvent e)
    {
        l.setText("Mouse Exited");
    }
    public void mousePressed(MouseEvent e)
    {
        l.setText("Mouse Pressed");
    }
    public void mouseReleased(MouseEvent e)
    {
        l.setText("Mouse Released");
    }
    public static void main(String[] args)
    {
        new MouseListenerExample();
    }
}
```

Handling Keyboard events:

Keyboard events are handled in java using KeyListener interface. Following are some keyboard event listeners.

1. public abstract void keyPressed(KeyEvent e);
2. public abstract void keyReleased(KeyEvent e);
3. public abstract void keyTyped(KeyEvent e);

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

```
public class KeyListenerExample extends Frame implements KeyListener
{
	Label l;
	TextArea area;
	KeyListenerExample()
	{
		l=new Label();
		l.setBounds(20,50,100,20);
		area=new TextArea();
		area.setBounds(20,80,300, 300);
		area.addKeyListener(this);
		add(l);
		add(area);
		setSize(400,400);
		setLayout(null);
		setVisible(true);
	}
	public void keyPressed(KeyEvent e)
	{
		l.setText("Key Pressed");
	}
	public void keyReleased(KeyEvent e)
	{
		l.setText("Key Released");
	}
	public void keyTyped(KeyEvent e)
	{
		l.setText("Key Typed");
	}
	public static void main(String[] args)
	{
		 new KeyListenerExample();
	}
}
```