Advance Problem Solving using Java Programming

Theory Assignment

Submitted To : Dr. Khalid Raza

By-

Name	:	Sandeep
RollNo.	:	18MCA040
Course	:	MCA III SEM (2019)
Subject	:	Java

Question 1: How will you create your own collection type explain it with a suitable programming example.

Answer:

>>Example : Here I have created a **MyArray** class which is implementing the **Collection** interface which is something similar to ArrayList in java.

>>Here few method of Collection interface has been implemented and other methods are not

>>Defined but included in the class as empty . because this class has to implement all the methods available in Collection because MyArray class implements Collection interface.

>>MyArray class is Generic class initially it set length =10;

>>size – tells the no. of data present in array a ;

>>length – capacity of the array;

>>if size == length then we reallocate the array with double the size all copy all previous data into it set to array a;

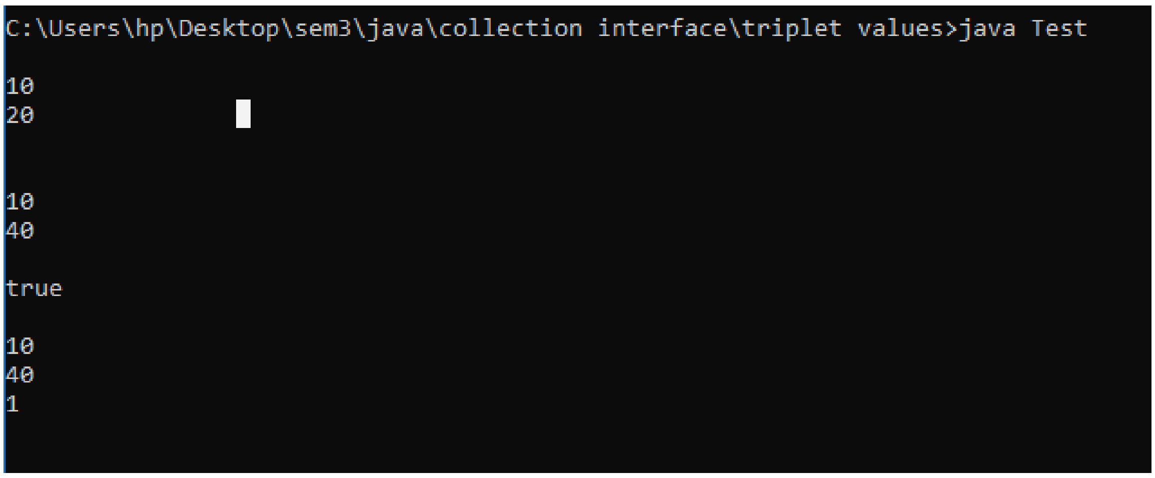
Program Code:

```
//class implements Collection

import java.util.*;

class MyArray<T> implements Collection{

        private Object a[];

        private int size;  //no. of elemetns present in a

        private int length;  //capacity of a


        MyArray(){

                this.size=0;

                this.length=10;    //initial capacity

                T array[]=(T[])new Object[10];  //allocating

                this.a=array;
        }


        public boolean add(Object obj){

                if(size == length){

                        increaseSizeAndReallocate();   //reallocate if size array is full

                }

                a[this.size]=obj;

                this.size++;

                return true;

        }

//method to increase capacity of the array;

        private void increaseSizeAndReallocate(){

                length=2*length;  //increase the size by double

                Object newData[]=new Object[length];

                for(int i=0; i<size; i++){

                        newData[i]=a[i];

                }

                this.a=newData;

        }

//return the element present at index in the array

        public Object get(int index) throws Exception{

                if(index > this.size-1 || index <0){

                        throw new Exception("ArrayIndexOutOfBoundsException");

                }


                return this.a[index];
        }
//remove element from array at index
        public void remove(int index) throws Exception{
                if(index > this.size-1 || index < 0){
                        throw new Exception("ArrayIndexOutOfBound");
                }
                for(int i=index; i<size-1; i++){
                        a[i]=a[i+1];
                }
                this.size--;
//if size is less than half of the array length then reduce the size to half;
                if(size < length/2){
                        reduceArraylength();
                }
        }
        private void reduceArraylength(){
                length = length/2;
                Object newData[]=new Object[length];
                for(int i=0; i<size; i++){
                        newData[i]=a[i];
                }
                this.a=newData;
        }



        public int size(){
```

```
			return size;

		}

//remove all elements from the list

		public void clear(){

			size=0;

			this.length=10;

			Object newData[]=new Object[length];

			this.a=newData;

		}

		public boolean contains(Object element){

			for(int i=0; i<size; i++){

				if(a[i] == element ){

					return true;

				}

			}

			return false;

		}

		public String toString(){

			StringBuilder sb=new StringBuilder();

			sb.append("\n");

			for(int i=0; i<size; i++){

				sb.append(String.valueOf(a[i]));

				sb.append("\n");

			}

			return sb.toString();

		}
```

```
        public boolean isEmpty(){

                if(size==0){

                        return true;

                }

                return false;

        }


//Other methods of Collection Interface

        public boolean addAll(Collection c){return true;}

        public boolean remove(Object element){return true;}

        public boolean removeAll(Collection c){return true;}

        //default boolean removeIf(Predicate<? super Triplet> filter){}

        public boolean retainAll(Collection c){return true;}

        public boolean containsAll(Collection c){return true;}

        public Iterator iterator(){return null;}

        public Object[] toArray(Object [] o){return this.a;}

        public Object[] toArray(){return this.a;}


        //default Stream parallelStream(){}

        //default Stream stream(){}

        //default Spliterator<Object> spliterator(){}

        public boolean equals(Object element){return true;}

        public int hashCode(){return 0;}
```

```
}


class Test{
	public static void main(String args[]) throws Exception{
		MyArray<Integer> A=new MyArray<Integer>();
		A.add(10);
		A.add(20);
		System.out.println(A);
		A.remove(1);
		A.add(40);
		System.out.println(A);
		System.out.println(A.contains(40));
		A.add(1);
		System.out.println(A);
		A.clear();
		System.out.println(A);
	}
}
```



```
C:\Users\hp\Desktop\sem3\java\collection interface\triplet values>java Test

10
20


10
40

true

10
40
1
```

Question 2: What do you mean by thread Synchronization.what are the possible ways available in java to handle the Synchronization problem.

Answer:

Thread Synchronization :

>>Thread synchronization is the concurrent execution of two or more threads that share critical resources.

>> Threads should be synchronized to avoid critical resource use conflicts. Otherwise, conflicts may arise when parallel-running threads attempt to modify a common variable at the same time.

>>There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
   1. Synchronized method.
   2. Synchronized block.
   3. static synchronization.
2. Cooperation (Inter-thread communication in java)

Ways to handle Synchronization in java :-

Synchronization in Java is possible by using Java keywords ***"synchronized"*** and ***"volatile"***.

- Synchronized Method: Includes the **synchronized** keyword in its declaration. When a thread invokes a synchronized method, synchronized method automatically acquires the intrinsic lock for that method's object and releases it when the method returns, even if that return was caused by an uncaught exception.

- Synchronized Statement: Declares a block of code to be synchronized. Unlike synchronized methods, synchronized statements should specify the objects that provide the intrinsic lock. These statements are useful for improving concurrency with fine-grained synchronization, as they enable the avoidance of unnecessary blocking.

Example :

Synchronized methond in java:

```
Class Counter{

        static  int   count=0;

        static synchronized int getCount(){

                return count;

        }

        Synchronized addCount(){

                this.count ++;

        }

}
```

Question 3: What is blocking Queue, explain the concept of thread pooling and its application in the real life enterprise applicaton development.

Answer:

Blocking Queue :

>> A blocking queue is a queue that blocks when you try to dequeue from it and the queue is empty, or if you try to enqueue items to it and the queue is already full.

>> A thread trying to dequeue from an empty queue is blocked until some other thread inserts an item into the queue.

>> A thread trying to enqueue an item in a full queue is blocked until some other thread makes space in the queue, either by dequeuing one or more items or clearing the queue completely.

Thread Pooling:

>>A *thread pool* is a collection of worker threads that efficiently execute asynchronous callbacks on behalf of the application.

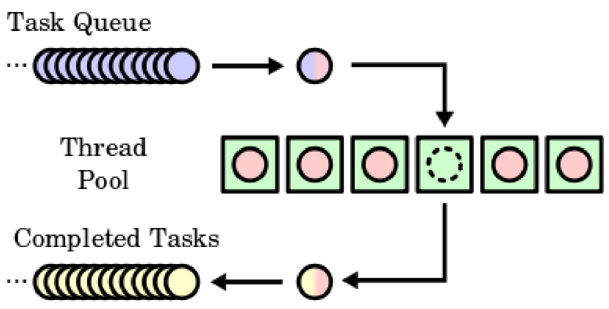
>> The thread pool is primarily used to reduce the number of application threads and provide management of the worker threads. Applications can queue work items, associate work with waitable handles, automatically queue based on a timer, and bind with I/O.

>>A thread pool reuses previously created threads to execute current tasks and offers a solution to the problem of thread cycle overhead and resource thrashing.

In simple words:

>>A thread pool is a group of pre-instantiated, idle threads which stand ready to be given work.

>>These are preferred over instantiating new threads for each task when there is a large number of short tasks to be done rather than a small number of long ones. This prevents having to incur the overhead of creating a thread a large number of times.



A sample thread pool with waiting tasks (blue) and completed tasks(yellow)

The thread pool architecture consists of the following:

- Worker threads that execute the callback functions
- Waiter threads that wait on multiple wait handles
- A work queue
- A default thread pool for each process
- A worker factory that manages the worker threads

Applications of Thread pooling (Enterprise Development) :

Real Life Example;

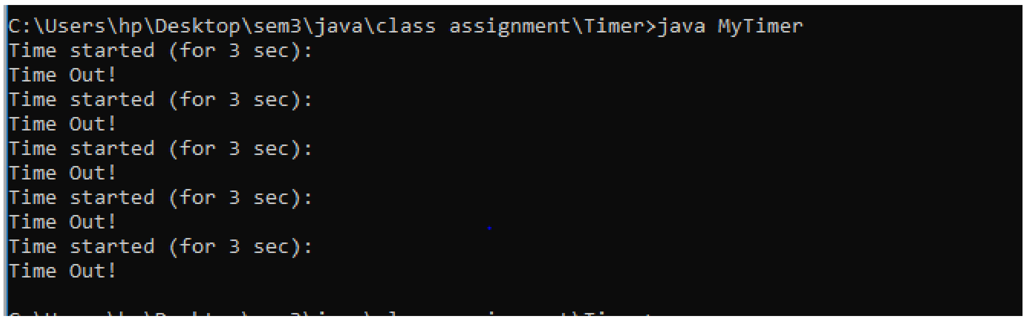
1. Facility: Operating system
2. Sections: Applications
3. People: Threads

Question 4: Write a simple timer program that can periodically print a timeout message.

Answer:

```
import java.util.*;

class MyTimer extends TimerTask{

        public void run(){

                try{

                        System.out.println("Time started (for 3 sec):");

                        Thread.sleep(3000);

                        System.out.println("Time Out!");

                }catch(Exception e){

                        System.out.println(e);

                }

        }

        public static void main(String args[]) throws Exception{

                Scanner sc=new Scanner(System.in);

                TimerTask timerTask = new MyTimer();


                Timer timer = new Timer(true);

                timer.scheduleAtFixedRate(timerTask, 0, 3*1000);

                Thread.sleep(15000);

                timer.cancel();

                System.out.println("Time Out!");

        }
```

}



```
C:\Users\hp\Desktop\sem3\java\class assignment\Timer>java MyTimer
Time started (for 3 sec):
Time Out!
Time started (for 3 sec):
Time Out!
Time started (for 3 sec):
Time Out!
Time started (for 3 sec):
Time Out!
Time started (for 3 sec):
Time Out!
```

Question 5: What are the different types of Events used in AWT/SWING programming. Explain various methods available for handling mouse and key board related events.

Answer:

Events broadly classified into two categories:

Foreground Events – These events require direct interaction of the user. They are generated as consequences of a person interacting with the graphical components in the Graphical User Interface.

For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page, etc.

Background Events – These events require the interaction of the end user. Operating system interrupts, hardware or software failure, timer expiration, and operation completion are some examples of background events.

Types of Events

Here are some of the most common types of events in Java:

>>***ActionEvent***: Represents a graphical element is clicked, such as a button or item in a list. Related listener: *ActionListener.*

>>***ContainerEvent***: Represents an event that occurs to the GUI's container itself, for example, if a user adds or removes an object from the interface. Related listener: *ContainerListener.*

>>***KeyEvent***: Represents an event in which the user presses, types or releases a key. Related listener: *KeyListener.*

>>***WindowEvent***: Represents an event relating to a window, for example, when a window is closed, activated or deactivated. Related listener: *WindowListener.*

>>***MouseEvent***: Represents any event related to a mouse, such as when a mouse is clicked or pressed. Related listener: *MouseListener.*

MouseEvent methods:

>>**MouseListener** and **MouseMotionListener** is an interface in java.awt.event package .

>>MouseListener handles the events when the mouse is not in motion.

>>MouseMotionListener handles the events when mouse is in motion.

MouseListener :

There are five abstract functions that represent these five events. **The abstract functions are :**

1. **void mouseReleased(MouseEvent e)** : Mouse key is released
2. **void mouseClicked(MouseEvent e)** : Mouse key is pressed/released
3. **void mouseExited(MouseEvent e)** : Mouse exited the component
4. **void mouseEntered(MouseEvent e)** : Mouse entered the component
5. **void mousepressed(MouseEvent e)** : Mouse key is pressed

There are two types of events that MouseMotionListener can generate.

1. **void mouseDragged(MouseEvent e)** : Invoked when a mouse button is pressed in the component and dragged. Events are passed until the user releases the mouse button.
2. **void mouseMoved(MouseEvent e)** : invoked when the mouse cursor is moved from one point to another within the component, without pressing any mouse buttons.

KeyBoardEvent methods :

Java **KeyListener** Interface:

>>The Java KeyListener is notified whenever you change the state of key. It is notified against KeyEvent.

>>The KeyListener interface is found in java.awt.event package. It has three methods.

Methods of KeyListener interface:

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