UNIT IV MULTITHREADING AND GENERIC PROGRAMMING

Differences between multi-threading and multitasking, thread life cycle, creating threads, synchronizing threads, Inter-thread communication, daemon threads, thread groups. Generic Programming – Generic classes – generic methods – Bounded Types – Restrictions and Limitations.

Question: 1 a) What is a thread? Explain multithreading and multitasking in detail.(8)
Or
1 b) Define thread. Write about multithreading and multitasking in detail.(8)

Answer:

A thread is a single sequential flow of control within a program. Sometimes it is called an execution context or light weight process.

Multithreading

- Multithreading in java is a process of executing multiple threads simultaneously.
- Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.
- But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.
- Java Multithreading is mostly used in games, animation etc.

Advantages of Java Multithreading

- It doesn't block the user because threads are independent and you can perform multiple operations at same time.
- You can perform many operations together so it saves time.
- Threads are independent so it doesn't affect other threads if exception occur in a single thread.

Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:

- Process-based Multitasking(Multiprocessing)
- Thread-based Multitasking(Multithreading)

1) Process-based Multitasking (Multiprocessing)

- Each process have its own address in memory i.e. each process allocates separate memory area.
- Process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

2) Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- Thread is lightweight.
- Cost of communication between the thread is low

Question: 2 a) What are the difference between multithreading and multitasking.(4)
Or
2 b) Differentiate multithreading and multitasking in detail.(4)

Answer:

BASIS FOR COMPARISON	MULTITASKING	MULTITHREADING
Basic	Multithreading is to execute multiple threads in a process concurrently.	Multitasking is to run multiple processes on a computer concurrently
Switching	In Multithreading, the CPU switches between multiple threads in the same process.	In Multitasking, the CPU switches between multiple processes to complete the execution.
Memory and Resource	In multitasking system has to allocate separate memory and resources to each program that CPU is executing.	In multithreading system has to allocate memory to a process, multiple threads of that process shares the same memory and resources allocated to the process.

Complexity	Multithreading is light-weight and easy to create	Multitasking is heavy-weight and harder to create

Question: 3 a) Explain in detail about different states of a thread.(13)
Or
3 b) Label the different states of a thread and explain it. (13)
Or
3 c) What is a thread? Describe the complete lifecycle of thread.(13)

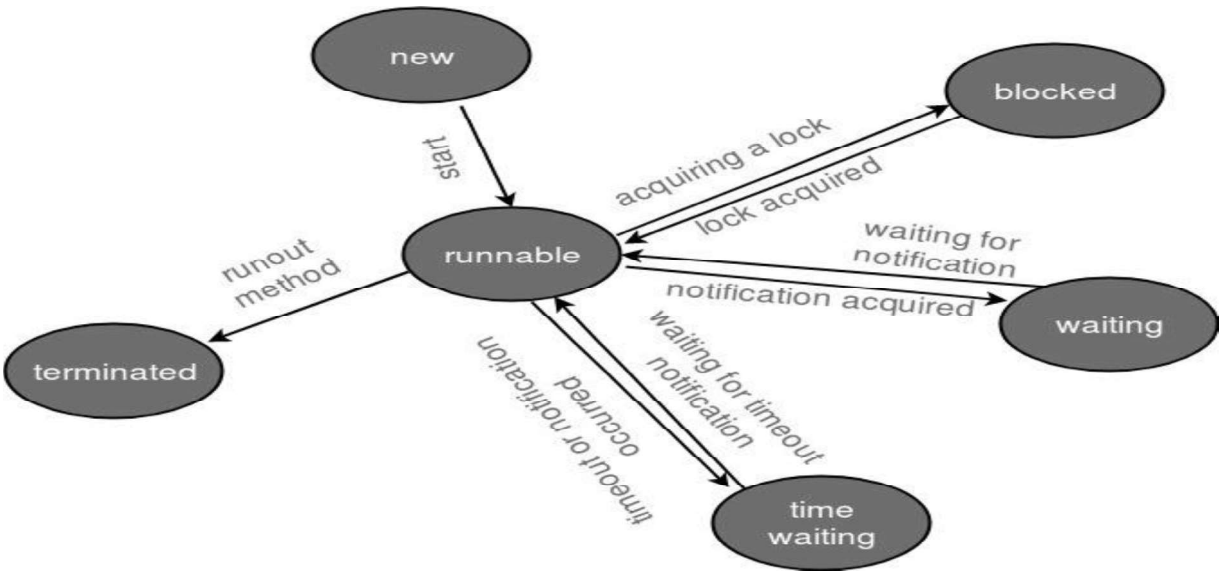
Answer:

A thread is a single sequential (separate) flow of control within program. Sometimes, it is called an execution context or light weight process.

A thread can be in one of the five states. According to sun, there is only 4 states in thread life cycle in java new, runnable, non-runnable and terminated. There is no running state.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated



Following are the stages of the life cycle −

- New − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
- Runnable − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- Waiting − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- Timed Waiting − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- Terminated (Dead) − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Question: 4 a) List the two ways of implementing threads with an example program. (13)
Or
4 b) What are the two ways to implement thread?(13)
Or
4 c) How threads are created in java.(13)

Answer:

Creating a thread

Java defines two ways by which a thread can be created.

- By implementing the Runnable interface.
- By extending the Thread class.

Implementing the Runnable Interface

The easiest way to create a thread is to create a class that implements the runnable interface. After implementing runnable interface, the class needs to implement the run() method, which is of form,

public void run()

- run() method introduces a concurrent thread into your program. This thread will end when run() method terminates.
- specify the code that your thread will execute inside run() method.

- run() method can call other methods, can use other classes and declare variables just like any other normal method.

```
class MyThread implements Runnable
{
        public void run()
        {
                System.out.println("concurrent thread started running..");
        }
}
class MyThreadDemo
{
        public static void main( String args[] )
        {
                MyThread mt = new MyThread();
                Thread t = new Thread(mt);
                t.start();
        }
}
```

Output

Concurrent thread started running..

To call the run() method, start() method is used. On calling start(), a new stack is provided to the thread and run() method is called to introduce the new thread into the program.

If you are implementing Runnable interface in your class, then you need to explicitly create a Thread class object and need to pass the Runnable interface implemented class object as a parameter in its constructor.

Extending Thread class

This is another way to create a thread by a new class that extends Thread class and create an instance of that class. The extending class must override run() method which is the entry point of new thread.

```
class MyThread extends Thread
{
```

```
        public void run()
        {
                System.out.println("concurrent thread started running..");
        }
}
classMyThreadDemo
{
        public static void main( String args[] )
        {
                MyThread mt = new  MyThread();
                mt.start();
        }
}
```

Output

Concurrent thread started running..

In this case also, we must override the run() and then use the start() method to run the thread. Also, when you create MyThread class object, Thread class constructor will also be invoked, as it is the super class, hence MyThread class object acts as Thread class object.

Question: 5 a) Describe the creation of a multiple thread with an example.(6)
Or
5 b) Explain multiple thread with an example.(6)

Answer:

Many threads can be created. Multiple threads can be created by extending thread class and by implementing the Runnable interface.

Example(Print even and odd numbers using Threads)

```
class Even extends Thread               //Thread1-even
{
        public void run()
        {
        for(int i=0;i<=10;i+=2)
```

```
        {
                System.out.println("Even number="+i);
        }
        }
}
class Odd extends Thread                //Thread2-odd
{
        public void run()
        {
        for(int i=0;i<=10;i+=2)
        {
                System.out.println("Odd number="+i);
        }
        }
}
class EvenOdd
{
public static void main(String args[])
{
        Even e=new Even();
        Odd o=new Odd();
        e.start();
        o.start();
}
}
```

Output:

```
Even number=0
Odd number=1
Even number=2
Odd number=3
Even number=4
```

Odd number=5

Even number=6

Odd number=7

Even number=8

Odd number=9

Even number=10

Explanation Steps:

During program execution, at first even thread starts its execution

Along with that, odd thread also executes

Both threads will get executed alternatively and finally terminated

Question: 6 a) Discuss about thread priority.(6)
Or
6 b) Explain thread priority with an example.(6)

Answer:

Priority of a Thread (Thread Priority):

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defined in Thread class:

1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

Example of priority of a Thread:

```
class TestMultiPriority1 extends Thread
{
   public void run()
   {
```

```
        System.out.println("running thread name is:"+Thread.currentThread().getName());
        System.out.println("running thread priority is:"+Thread.currentThread().getPriority());
    }
  public static void main(String args[])
  {
    TestMultiPriority1 m1=new TestMultiPriority1();
     TestMultiPriority1 m2=new TestMultiPriority1();
     m1.setPriority(Thread.MIN_PRIORITY);
    m2.setPriority(Thread.MAX_PRIORITY);
    m1.start();
    m2.start();
  }
  }
```

Output:

```
running thread name is:Thread-0
running thread priority is:10
running thread name is:Thread-1
running thread priority is:1
```

Question: 7 a) What is Synchronization? Explain the different types of synchronization in java(13)

Or

7 b) Summarize briefly about thread synchronization with an example. (13)

Or

7 c) Explain in detail about thread synchronization in java.(13)

Answer:

Synchronization in java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Why use Synchronization

The synchronization is mainly used to

1. To prevent thread interference.

2. To prevent consistency problem.

Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Thread Synchronization

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

1. Mutual Exclusive
2. Synchronized method.

Understanding the problem without Synchronization

In this example, there is no synchronization, so output is inconsistent.

```
class Table
{
        void printTable(int n)
          {//method not synchronized
           for(int i=1;i<=5;i++)
           {
                System.out.println(n*i);
                try
                {
                        Thread.sleep(400);
                 }
                catch(Exception e)
                {
                        System.out.println(e);}
                 }
           }
}
class MyThread1 extends Thread
{
```

```
        Table t;
        MyThread1(Table t)
        {
            this.t=t;
        }
        public void run()
        {
            t.printTable(5);
        }
 }
class MyThread2 extends Thread
{
        Table t;
        MyThread2(Table t)
        {
            this.t=t;
        }
        public void run()
        {
            t.printTable(100);
         }
}
 class TestSynchronization1
 {
 public static void main(String args[])
 {
        Table obj = new Table();//only one object
        MyThread1 t1=new MyThread1(obj);
        MyThread2 t2=new MyThread2(obj);
        t1.start();
        t2.start();
```

```
}
}
```

Output: 5

```
100
10
200
15
300
20
400
25
500
```

Java synchronized method

If you declare any method as synchronized, it is known as synchronized method. Synchronized method is used to lock an object for any shared resource. When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

```
//example of java synchronized method
class Table
{
        synchronized void printTable(int n)
        {//synchronized method
        for(int i=1;i<=5;i++)
        {
                System.out.println(n*i);
         try
        {
                Thread.sleep(400);
        }
        catch(Exception e)
        {
```

```
                System.out.println(e);
            }
            }
  }
}
class MyThread1 extends Thread
{
        Table t;
        MyThread1(Table t)
        {
                this.t=t;
        }
        public void run()
        {
                t.printTable(5);
        }
  }
 class MyThread2 extends Thread
 {
 Table t;
        MyThread2(Table t)
        {
                this.t=t;
        }
        public void run()
        {
                t.printTable(100);
        }
   }
    public class TestSynchronization2
    {
```

```
public static void main(String args[])
{
    Table obj = new Table();//only one object
    MyThread1 t1=new MyThread1(obj);
    MyThread2 t2=new MyThread2(obj);
    t1.start();
    t2.start();
}
}
```

Output : 5
10
15
20
25
100
200
300
400
500

Synchronized block in java

Synchronized block can be used to perform synchronization on any specific resource of the method. Suppose you have 50 lines of code in your method, but you want to synchronize only 5 lines, you can use synchronized block.

If you put all the codes of the method in the synchronized block, it will work same as the synchronized method.

- Synchronized block is used to lock an object for any shared resource.
- Scope of synchronized block is smaller than the method.

Syntax to use synchronized block

```
synchronized (object reference expression)
{
        //code block
```

```
    }
```

Program of synchronized block

```
class Table
{
    void printTable(int n)
    {
        synchronized(this)
        {//synchronized block
        for(int i=1;i<=5;i++)
        {
            System.out.println(n*i);
            try
            {
                Thread.sleep(400);
            }
            catch(Exception e)
            {
                System.out.println(e);
            }
        }
        }
    }//end of the method
}
class MyThread1 extends Thread
{
    Table t;
    MyThread1(Table t)
    {
        this.t=t;
    }
    public void run()
```

```
        {
                t.printTable(5);
        }
}
class MyThread2 extends Thread
{
        Table t;
        MyThread2(Table t)
        {
                this.t=t;
        }
        public void run()
        {
                t.printTable(100);
        }
}

public class TestSynchronizedBlock1
{
        public static void main(String args[])
        {
                Table obj = new Table();//only one object
                MyThread1 t1=new MyThread1(obj);
                MyThread2 t2=new MyThread2(obj);
                t1.start();
                t2.start();
        }
}
```

Output:

5
10
15
20
25
100
200
300
400
500

Static synchronization

If you make any static method as synchronized, the lock will be on the class not on object.

Problem without static synchronization

Suppose there are two objects of a shared class(e.g. Table) named object1 and object2.In case of synchronized method and synchronized block there cannot be interference between t1 and t2 or t3 and t4 because t1 and t2 both refers to a common object that have a single lock.But there can be interference between t1 and t3 or t2 and t4 because t1 acquires another lock and t3 acquires another lock.I want no interference between t1 and t3 or t2 and t4.Static synchronization solves this problem.

Example of static synchronization

In this example we are applying synchronized keyword on the static method to perform static synchronization.

```
class Table
{
        synchronized static void printTable(int n)
        {
        for(int i=1;i<=10;i++)
        {
           System.out.println(n*i);
```

```
                try
                {
                        Thread.sleep(400);
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
                }
              }
}
class MyThread1 extends Thread
{
    public void run()
    {
         Table.printTable(1);
   }
}
class MyThread2 extends Thread
{
         public void run()
        {
              Table.printTable(10);
       }
}
class MyThread3 extends Thread
{
              public void run()
              {
              Table.printTable(100);
              }
```

```
}
class MyThread4 extends Thread
{
        public void run()
        {
        Table.printTable(1000);
        }
}

public class TestSynchronization4
{
        public static void main(String t[])
        {
           MyThread1 t1=new MyThread1();
           MyThread2 t2=new MyThread2();
           MyThread3 t3=new MyThread3();
           MyThread4 t4=new MyThread4();
           t1.start();
           t2.start();
           t3.start();
           t4.start();
        }
}
```

Output:

1
2
3
4
5
6
7

8
9
10
10
20
30
40
50
60
70
80
90
100
100
200
300
400
500
600
700
800
900
1000
1000
2000
3000
4000
5000
6000
7000
8000

9000

10000

Question: 8 a) Discuss about Deadlock in java.(13)
Or
8 b) Explain Deadlock with an example.(13)

Answer:

Deadlock in java

Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.

Example of Deadlock in java

```
public class TestDeadlockExample1
{
    public static void main(String[] args)
    {
        final String resource1 = "ratan jaiswal";
        final String resource2 = "vimal jaiswal";
        // t1 tries to lock resource1 then resource2
        Thread t1 = new Thread()
        {
        public void run()
        {
            synchronized (resource1)
            {
                System.out.println("Thread 1: locked resource 1");
                try
                {
                    Thread.sleep(100);
                }
```

```
                    catch (Exception e)
                    {
                        System.out.println(e);
                    }
                    synchronized (resource2)
                    {
                        System.out.println("Thread 1: locked resource 2");
                    }
                }
            }
        };

    // t2 tries to lock resource2 then resource1
        Thread t2 = new Thread()
        {
            public void run()
            {
            synchronized (resource2)
            {
                System.out.println("Thread 2: locked resource 2");
                try
                {
                    Thread.sleep(100);
                }
                catch (Exception e)
                {
                    System.out.println(e);
                }
             synchronized (resource1)
            {
                System.out.println("Thread 2: locked resource 1");
```

```
                    }
                }
    }
    };
    t1.start();
    t2.start();
  }
}
```

Output: Thread 1: locked resource 1

Thread 2: locked resource 2

Question: 9 a) Examine Inter thread Communication. Why this feature is required and how it is achieved? (13)

Or

9 b) Demonstrate Inter thread Communication.(13)

Or

9 c) Using an example, explain inter thread communication in detail.(13)

Answer:

Inter-Thread Communication

Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other. Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of Object class:

- wait()
- notify()
- notifyAll()

1) wait() method

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

Method	Description
public final void wait()throws InterruptedException	waits until object is notified.
public final void wait(long timeout)throws InterruptedException	waits for the specified amount of time.

2) notify() method

Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.

Syntax:

public final void notify()

3) notifyAll() method

Wakes up all threads that are waiting on this object's monitor.

Syntax:

public final void notifyAll()

Example of inter thread communication in java

```
class Customer
{
  int amount=10000;
  synchronized void withdraw(int wamount)
  {
        System.out.println("going to withdraw...");
        if(amount<wamount)
        {
              System.out.println("Less balance; waiting for deposit...");
         try
```

```
                {
                        wait();
                }
                catch(Exception e)
                {
                        System.out.println(e);
                }
               }
      amount=amount-wamount;
      System.out.println("withdraw completed...");
      }
 synchronized void deposit(int damount)
 {
        System.out.println("going to deposit...");
        amount=amount+damount;
        System.out.println("deposit completed... ");
        notify();
}
}
class Test
{
public static void main(String args[])
{
        Customer c=new Customer();
        new Thread()
        {
                public void run()
                {
                        c.withdraw(15000);
                }
        }.start();
```

```
        new Thread()
        {
                public void run()
                {
                        c.deposit(10000);
                }
        }.start();
        }
}
```

Output:

```
going to withdraw...
Less balance; waiting for deposit...
going to deposit...
deposit completed...
withdraw completed
```

Question: 10 a) What is daemon thread explain with an example?(8)

Or

10 b) Discuss about daemon thread in java.(8)

Answer:

Daemon Threads

Daemon thread in java is a service provider thread that provides services to the user thread for background supporting tasks. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically. It is a low priority thread.

There are many java daemon threads running automatically e.g. gc, finalizer etc. The jconsole tool provides information about the loaded classes, memory usage, running threads etc.

Why JVM terminates the daemon thread if there is no user thread?

The sole purpose of the daemon thread is that it provides services to user thread for background supporting task. If there is no user thread, JVM terminates the daemon thread if there is no user thread.

Methods for Java Daemon thread by Thread class

The java.lang.Thread class provides two methods for java daemon thread.

S.No.	Method	Description
1)	Public void setDaemon(boolean status)	is used to mark the current thread as daemon thread or user thread.
2)	public boolean isDaemon()	is used to check that current thread is daemon.

Simple example of Daemon thread in java

File: MyThread.java

```
public class TestDaemonThread1 extends Thread
{
    public void run()
    {
        if(Thread.currentThread().isDaemon()) // checking for daemon thread
        {
            System.out.println("daemon thread work");
        }
        else
        {
            System.out.println("user thread work");
        }
    }
    public static void main(String[] args)
    {
        TestDaemonThread1 t1=new TestDaemonThread1();//creating thread
        TestDaemonThread1 t2=new TestDaemonThread1();
        TestDaemonThread1 t3=new TestDaemonThread1();
        t1.setDaemon(true);//now t1 is daemon thread
        t1.start();//starting threads
```

```
        t2.start();
        t3.start();
    }
    }
```

Output

daemon thread work
user thread work
user thread work

Question: 11 a) Explain thread group with an example?(8)
Or
11 b) Discuss about thread group in java.(8)

Answer:

ThreadGroup

Java provides a convenient way to group multiple threads in a single object. All threads can be started or suspended within a group with a single method call.

Java thread group is implemented by *java.lang.ThreadGroup* class.

Constructors of ThreadGroup class

There are only two constructors of ThreadGroup class.

S.No.	Constructor	Description
1)	ThreadGroup(String name)	creates a thread group with given name.
2)	ThreadGroup(ThreadGroup parent, String name)	creates a thread group with given parent group and name.

S.No.	Method	Description

1)	int activeCount()	returns number of threads running in current group.
2)	int activeGroupCount()	returns a number of active group in this thread group.
3)	void destroy()	destroys this thread group and all its sub groups.
4)	String getName()	returns the name of this group.
5)	ThreadGroup getParent()	returns the parent of this group.
6)	void interrupt()	interrupts all threads of this group.
7)	void list()	prints information of this group to standard console.

Methods of ThreadGroup class

There are many methods in ThreadGroup class. A list of important methods are given below.

Code to group multiple threads.

```
ThreadGroup tg1 = new ThreadGroup("Group A");
Thread t1 = new Thread(tg1,new MyRunnable(),"one");
Thread t2 = new Thread(tg1,new MyRunnable(),"two");
Thread t3 = new Thread(tg1,new MyRunnable(),"three");
```

Now all 3 threads belong to one group. Here, tg1 is the thread group name, MyRunnable is the class that implements Runnable interface and "one", "two" and "three" are the thread names.

Now we can interrupt all threads by a single line of code only.

```
Thread.currentThread().getThreadGroup().interrupt();
```

ThreadGroup Example

File: ThreadGroupDemo.java

```
public class ThreadGroupDemo implements Runnable
{
```

```
public void run()
{
        System.out.println(Thread.currentThread().getName());
 }
public static void main(String[] args)
{
         ThreadGroupDemo runnable = new ThreadGroupDemo();
         ThreadGroup tg1 = new ThreadGroup("Parent ThreadGroup");
         Thread t1 = new Thread(tg1, runnable,"one");
         t1.start();
         Thread t2 = new Thread(tg1, runnable,"two");
         t2.start();
        Thread t3 = new Thread(tg1, runnable,"three");
         t3.start();
         System.out.println("Thread Group Name: "+tg1.getName());
         tg1.list();
   }
   }
```

Output:

```
one
two
three
Thread Group Name: Parent ThreadGroup
java.lang.ThreadGroup[name=Parent ThreadGroup,maxpri=10]
        Thread[one,5,Parent ThreadGroup]
        Thread[two,5,Parent ThreadGroup]
        Thread[three,5,Parent ThreadGroup]
```

Question: 12 a) What is generic programming? Explain with an example.(8)
Or
12 b) Discuss about generic programming.(8)

Answer:

Generic programming enables the programmer to create classes, interfaces and methods that automatically works with all types of data(Integer, String, Float etc). It has expanded the ability to reuse the code safely and easily..

Before generics, we can store any type of objects in collection i.e. non-generic. Now generics, forces the java programmer to store specific type of objects.

Advantages of Java Generics

There are mainly 3 advantages of generics. They are as follows:

1) Type-safety : We can hold only a single type of objects in generics. It doesn't allow to store other objects.

2) Type casting is not required: There is no need to typecast the object.

Before Generics, we need to type cast.

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);//typecasting
```

After Generics, we don't need to typecast the object.

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);
```

3) Compile-Time Checking: It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

```
List<String> list = new ArrayList<String>();
list.add("hello");
list.add(32);//Compile Time Error
```

Syntax to use generic collection

Class Or Interface<Type>

Example to use Generics in java

ArrayList<String>

Example of Generics in Java

Here, we are using the ArrayList class, but you can use any collection class such as ArrayList, LinkedList, HashSet, TreeSet, HashMap, Comparator etc.

```
import java.util.*;
class TestGenerics1
{
	public static void main(String args[])
	{
		ArrayList<String> list=new ArrayList<String>();
		list.add("rahul");
		list.add("jai");
		//list.add(32);//compile time error
		String s=list.get(1);//type casting is not required
		System.out.println("element is: "+s);
		Iterator<String> itr=list.iterator();
		while(itr.hasNext())
		{
			System.out.println(itr.next());
		}
	}
}
```

Output:element is: jai

rahul

jai

Question: 13 a) What is generic class? Explain with an example.(8)
Or
13 b) Discuss about generic class in Java.(8)

Answer:

Generic Class

A class that can refer to any type is known as generic class. Here, we are using T type parameter to create the generic class of specific type.

Simple example to create and use the generic class.

```
public class Pair<T>
{
        public Pair()
        {
                first=null;
                second=null;
        }
        public Pair(T first, T second)
        {
                this.first=first;
                this.second=second;
        }
        public T getFirst()
        {
                return first;
        }
        public T getSecond()
        {
                return second;
        }
class Test
{
public static void main(String args[])
{
        Pair<Integer> p1=new Pair(1,2);
        Pair<Float> p2=new Pair(1.1,2.2);
        Pair<String> p3=new Pair("SUNIL", "KUMAR");
        System.out.println("p1:"+p1.getFirst()=" "+p1.getSecond());
```

```
        System.out.println("p2:"+p2.getFirst()=" "+p2.getSecond());
        System.out.println("p3:"+p3.getFirst()=" "+p3.getSecond());
}
}
```

Output:

P1: 1 2

P2: 1.1 2.2

P3: SUNIL KUMAR

The T type indicates that it can refer to any type (like String, Integer, Employee etc.). The type you specify for the class, will be used to store and retrieve the data.

Question: 14 a) What is generic method? Explain with an example.(8)
Or
14 b) Discuss about generic method in Java.(8)

Answer:

Generic Method

Like generic class, we can create generic method that can accept any type of argument. Single method can be defined with type parameters. This method is defined inside ordinary class.

Syntax:

```
[access specifier]<type_variable1, type_variable2…> returntype method_name(parameters)
{
        Action Block;
}
```

Example:

```
public class GenericDemo
{
  public static < T > void print_val (T val)
 {
        System.out.println("Value:"+val );
 }
}
```

```
class Maindemo
{
public static void main( String args[] )
 {
        GenericDemo g=new GenericDemo();
        System.out.println("Integer");
        g.print_val(10);
        System.out.println("\nFloating point");
        g.print_val(5.5f);
        System.out.println("\nString");
        g.print_val("welcome");
 }
 }
```

Output:

```
Integer
Value:10
Floating point
Value:5.5f
String
welcome
```

Question: 15 a) What is Wildcard in Java Generics? Explain with an example.(8)
Or
15 b) Discuss about Wildcard in Java Generics.(8)

Answer:

Wildcard in Java Generics

The ? (question mark) symbol represents wildcard element. It represents an unknown type. The wildcard can be used in a variety of situations such as the type of a parameter, field, or local variable; sometimes as a return type. If we write <? extends Number>, it means any child class of Number e.g. Integer, Float, double etc. Now we can call the method of Number class through any child class object.

```
import java.util.*;
abstract class Shape
{
    abstract void draw();
}
class Rectangle extends Shape
{
    void draw()
    {
            System.out.println("drawing rectangle");
    }
}
class Circle extends Shape
{
    void draw()
    {
            System.out.println("drawing circle");
    }
}
class GenericTest
{
//creating a method that accepts only child class of Shape
public static void drawShapes(List<? extends Shape> lists)
{
for(Shape s:lists)
{
    s.draw();//calling method of Shape class by child class instance
}
}
public static void main(String args[])
{
```

```
List<Rectangle> list1=new ArrayList<Rectangle>();
list1.add(new Rectangle());
List<Circle> list2=new ArrayList<Circle>();
list2.add(new Circle());
list2.add(new Circle());
drawShapes(list1);
drawShapes(list2);
}
}
```

Output

```
drawing rectangle
drawing circle
drawing circle
```

Question: 16 a) What is bounded types in java? Explain with an example.(13)
Or
16 b) Discuss about bounded types in java.(13)

Answer:

Bounded Type Parameters

- A class or a method needs to place restrictions on type variables.
- While creating objects to generic classes we can pass any derived type as type parameters
- To limit the types that can be passed to type parameters. For that purpose, bounded types are introduced in generics.
- For example, if we want a generic class that works only with numbers(like int, double, float, long….) then declare type parameter of that class as a bounded type to number class.
- Then while creating objects to that class you have to pass only number types or it's subclass type as type parameters.
- For example in a method that compares two objects and we want to make sure that the accepted objects are Comparables.

- The invocation of these methods is similar to unbounded method except that if we will try to use any class that is not Comparable, it will throw compile time error.

Declare a bounded type parameter

1. List the type parameter's name.
2. Along by the extends keyword
3. And by its upper bound

Syntax

<T extends superClassName>

This specifies that T can only be replaced by superClassName, or subclasses of superClassName.

Thus, superclass defines an inclusive, upper limit.

Example:

```
class Test<T extends Number>
{
        T t;
        public Test(T t)
        {
                this.t=t;
        }
        public T get()
        {
                return t;
        }
}
public class BoundedTypeDemo
{
public static void main(String args[])
{
        //creating obj by passing integer as a type parameter
        Test<Integer> obj=new Test<Integer>(123);
        System.out.println("The integer is:"+obj.getT());
        //creating obj by passing double as a type parameter
        Test<Double> obj1=new Test<Double>(12.3);
```

```
        System.out.println("The Double is:"+obj1.getT());
        //creating obj by passing String as a type parameter
        //It gives compile time error
        //Test<String> obj2=new Test<String>("Hai");
        //System.out.println("The String is:"+obj2.getT());
}
}
```

Output:

```
        The Integer is: 123
        The Double is : 12.3
```

- Java provides bounded types when specifying a type parameter, an upper bound must be created that declares the super class from which all type arguments must be derived.
- Type parameter can be specified using an extends clause.
- Bounding of T prevents non-numeric objects from being created. In this example, if the comments from the lines are removed, it shows compile time error. Because String is not a subclass of Number.
- In addition to class type, interface type can also be used as a bound.

Example:

```
class Gen<T extends MyClass & MyInterface>
{
    //
}
```

T is bounded by a class called MyClass and an interface called MyInterface.

Any argument passed to T must be a subclass of MyClass and implement MyInterface.

Multiple Bounds

Bounded type parameters can be used with methods as well as classes and interfaces. Java Generics supports multiple bounds also, i.e . In this case A can be an interface or class. If A is class then B and C should be interfaces. We can't have more than one class in multiple bounds.

Syntax

<T extends superClassName & Interface>

Example:

```
class Bound<T extends A & B>
{
    private T objRef;
    public Bound(T obj)
  {
     this.objRef = obj;
  }
 public void doRunTest()
{
   this.objRef.displayClass();
}
}
 interface B
{
   public void displayClass();
}
 class A implements B
{
   public void displayClass()
   {
      System.out.println("Inside super class A");
   }
}
public class BoundedClass
{
   public static void main(String a[])
   {
      //Creating object of sub class A and
      //passing it to Bound as a type parameter.
      Bound<A> bea = new Bound<A>(new A());
```

```
        bea.doRunTest();
    }
}
```

Output :

Inside super class A

Question: 17 a) What are the restrictions and limitations to be considered in generic programming?(8)
Or
17 b) Discuss the various restrictions and limitations to be considered in generic programming.(8)

A number of restrictions need to consider when working with Java generics.

i)Type Parameters cannot be Instantiated with primitive Types

It is not possible to create an instance of a type parameter.

Example:

```
//can't create an instance of T
class Gen<T>
{
        T ob;
        Gen()
        {
                ob=new T();    //Illegal
        }
}
```

ii) Restrictions on Static Members:

No static member can use a type parameter declared by the enclosing class.

Example:

All of the static members of this class are Illegal.

```
class Wrong<T>
{
        //wrong, no static variables of Type T
        static T ob;
        //wrong, no static method can use T
        static T getob()
        {
                return ob;
        }
        //wrong, no static method can access object of type T
        static void showob()
        {
                System.out.println(ob);
        }
```

```
}
```

iii) Arrays of Parameterized Types are not legal

Cannot declare arrays of parameterized type such as

Example:

```
Pair<String>[] table=new Pair<String>[10]; //Error
```

In the above line Pair[] can convert it to object[]

```
Object[] objarray=table;
Objarray[0]= "Hello";          //error component type is Pair
```

iv) Can't Instantiate Type Variables:

Type Variables cannot be used in an expression such as new T(…)

Example:

```
public Pair()
{
        first=new T();          //Error
        second=new T();         //Error
}
```

v) Generic Array Restrictions:

There are two important generic restrictions that apply to arrays

i) Cannot Instantiate an array whose base type is a type parameter
ii) Cannot create an array of type-specific generic references.

Example:

1) Vals=new T[10]; //Can't create an array of T
2) Gen<Integer> gen[]=new Gen<Integer>[10]; //Can't declare an array of type-specific generic references

vi) Generic Exception Restriction:

A generic class cannot extend throwable (ie) Generic exception classes cannot be created.

vii) Cannot Throw or catch Instances of a Generic Class

Generic class can neither throw nor catch objects.

Example:

```
public static<T extends throwable>
void doWork(class <T> t)
{
        try
        {
                -----
        }
        catch(T e)      //Error, can't catch type Variable
        {
                --------
        }
}
```