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.

4.1 THREADS

- One of the exciting features of windows operating system is that – It allows the user to handle multiple task together. This facility in windows operating system is called multitasking.
- Thread is a small program running continuously.
- A thread is also called as light weight process.
- We can create multiple threads and execute them simultaneously.
- A thread based multitasking environment allows you to execute more than one part of the single program simultaneously/concurrently. For example with in single program, one thread can used to display live time in the top-right corner screen, and the other thread can display animation in the whole screen.

4.2 DIFFERENCE BETWEEN MULTITHREADING AND MULTITASKING

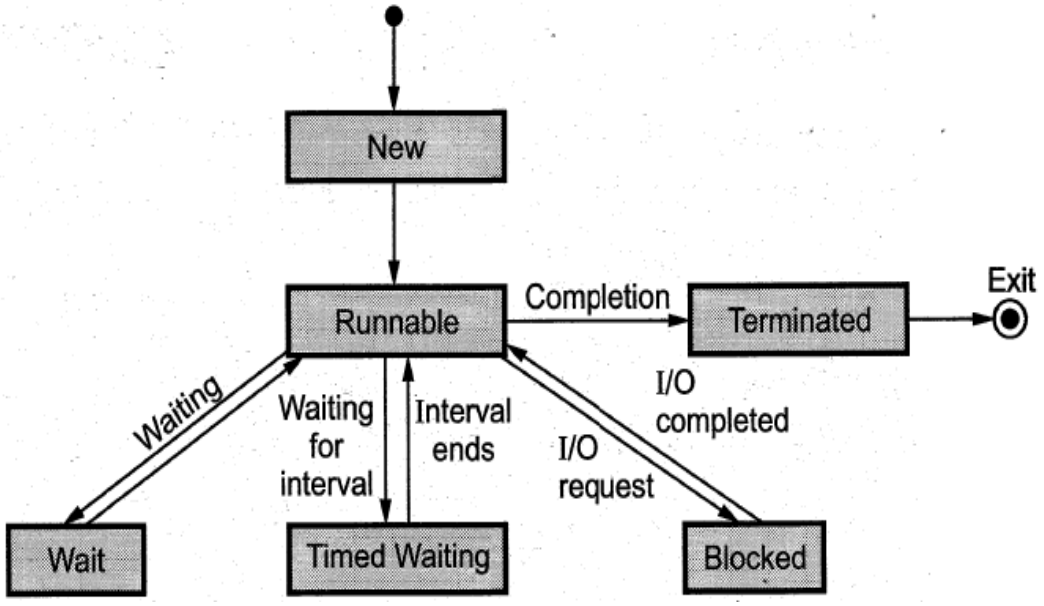
Sl.No	Multithreading	Multitasking
1	Thread is a fundamental unit of multithreading.	Program or process is a fundamental unit of multiprocessing environment
2	Multiple parts of a single program gets executed in multithreading environment	Multiple program get executed in multiprocessing environment.
3	During multithreading the processing switches between multiple threads of the program	During multiprocessing the processor switches between multiple program or processes
4	It is cost effective because CPU can be shared among multiple thread at a time.	It is expensive because when a particular process uses CPU other processes has to wait.
5	It is highly efficient	It is less effiecient
6	It helps in developing efficient application programs	It helps in developing efficient operating system programs

4.3 THREAD LIFE CYCLE

Thread States

Thread always exits in any one of the following states

- New or create state
- Runnable state
- Waiting State
- Timed waiting state
- Blocked State
- Terminated state



- **New State:** 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 State:** After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting State:** Sometime one thread has to undergo in waiting state because another thread starts executing.
- **Timed waiting state** This allows to execute high prioritized threads first. After the timing gets over, the thread in waiting state enters in runnable state.
- **Blocked state:** A thread can enter in this state because of waiting for the resources that are hold by another thread.
- **Terminated State** After successful completion of the execution the thread in runnable state enters the terminated state.

4.4 CREATION OF THREAD

In java threads can be created in two ways such as

1. Using **Thread** class
2. Using **Runnable** interface

4.4.1 Predefined Methods in Thread Class

public void start()
public void run()
public final void setName(String name) - Changes the name of the Thread object.
getName() - method for retrieving the name.
public final void setPriority(int priority)
public final void setDaemon(boolean on)
public final void join(long millisec)
public void interrupt()
public final boolean isAlive()
public static void sleep(long millisec)
public static boolean holdsLock(Object x) - Returns true if the current thread holds the lock on the given Object.
public static Thread currentThread() - Returns a reference to the currently running thread,
public static void dumpStack() - Prints the stack trace for the currently running thread

- run()
- start()
- sleep()
- stop()
- wait()
- suspend()
- resume()
- getName()

4.4.2 Creation of Thread Using Thread Class (or) Multi Thread

*//Example program to create a Multi Thread using the **Thread class**:*

```
import java.io.*;
import java.lang.*;

public class A extends Thread
{
void run()
{
System.out.println("This is from A");
}
}
class B extends Thread
{
void run()
{
System.out.println("This is from B");
}
}
class ThreadDemo1
{
```

Output:
This is from A
This is from B

```
public static void main(String srgs[])
{
A obj1= new A();  //object creation
B obj2= new B();
obj1.start();       //thread starts its execution
obj2.start();
}
}
```

4.4.3 Creation of Thread Using Runnable Interface

*//Example program to create a Multi Thread using the **Runnable Interface**:*

```
class A implements Runnable
{
void run()
{
System.out.println("This is from A");
}
}
class B implements Runnable
{
void run()
{
System.out.println("This is from B");
}
}

class ThreadDemo1
{
public static void main(String srgs[])
{
A obj1= new A();  //object creation
Thread ob1= new Thread(obj1);


Ob1.start();                  //thread starts its execution
B obj2= new B();
Thread ob2= new Thread(obj2);
Ob2.start();}
}
```

Output:

This is from A

This is from B

4.5 SYNCHRONIZATION IN JAVA

When we start two or more threads within a program, there may be a situation when multiple threads try to access the same resource and finally they can produce unexpected result due to concurrency issues. For example, if multiple threads try to write within a same file then they may corrupt the data because one of the threads can override data or while one thread is opening the same file at the same time another thread might be closing the same file. So there is a need to synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time.

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

```
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
{
```

Output:

```
5
10
15
20
25
100
200
300
400
500
```

```
   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();
   }
   }
```

Create an application that executes two threads. First thread displays the alphabet A to Z at every one second. The second thread will display the alphabet Z to A at every two seconds. Both the threads need to synchronize with each other for printing alphabets. The second thread has to wait until the first thread finishes its execution. The application waits for all the threads to finish the execution (10) (A/M 15)

```
Class Test
{

Synchronized void display1 ()
{
for(char i='A';i<='Z';i++)
{
System.out.println(i);

try
{
Thread.sleep(1000);
}
catch(Exception e)
{
  System.out.println(e);
}
}

Synchronized void display2 ()
{
for(char i='Z';i>='A';i--)
{
System.out.println(i);
try
{
Thread.sleep(1000);
}
catch(Exception e)
{
  System.out.println(e);
}
}

}

Class Mythread1 extends Thread
{
Test t;
Mythread1(Test t)
{
this.t=t;
}
public void run()
{
t.display1();
}
}

Class Mythread2 extends Thread
{
Test t;
Mythread2(Test t)
{
This.t=t;
}
public void run()
{
t.display2();
}
}
public class TwoThreads
{
public static void main(String arhs[])
{
Test obj =  new Test();
MyThread1 t1 = new Mythread1(obj);
MyThread1 t2 = new Mythread2(obj);
t1.start();
t2.start();
}
}
```

4.6 INTER THREAD COMMUNCATION

- Two or more threads communicate with each other by exchanging the messages. This mechanism is called inter thread communication.
- Polling is a mechanism generally implemented in a loop in which certain condition is repeatedly checked.
- To better understand the concept of polling, consider producer-consumer problem in which producer thread produces and the consumer thread consumes whatever is produced.
- Both must work in co-ordination to avoid wastage of CPU cycles.
- But there are situations in which the producer has to wait for the consumer to finish consuming of data. Similarly the consumer may need to wait for the producer to produce the data.
- In Polling system either consumer will waste many CPU cycle when waiting for producer to produce or the producer will waste CPU cycles when waiting for the consumer to consume the data.
- In order to avoid polling there are three in-built methods that take part in inter-thread communication
    - notify() – If a particular thread is in the sleep mode than that thread can be resumed using the notify call.
    - notifyall() – This method resumes all the threads that are in suspended stae.
    - wait() – The calling thread can be send into a sleep mode.

Example Program:

- Following is a simple Java program in which two threads are created one for producer and another is for consumer.
- The producer thread produces(write) the numbers from 0 to 9 and the consumer thread consumes(reads) these numbers.
- The wait and notify methods are used to send particular thread to sllep or to resume the thread from sleep mode respectively.

```
Class Myclass
{
int val;
boolean flag=false;
synchronized int get()
{
if(!flag)
try
{
wait();
}
catch(InterruptedException e)
{
System.out.println(“InterruptedException!”);
}
System.out.println(“Consumer consuming:” + val);
flag=false;
notify();
return val;
}
synchronized void put(int val)
{
if(flag)
try
{
```

```
wait();
}
catch(InterruptedException e)
{
System.out.println("InterruptedException");
}
this.val=val;
flag=true;
System.out.println("Producer producing" +val);
notify();
}
}
class Producer extends Thread
{
MyClass th1;
Producer(MyClass t)
{
th1=t;
}
public void run()
{
For(int i=0;i<10;i++)
{
th1.put(i);
}
}
}

Class Consumer extends Thread
{
MyClass th2;
Conumer(MyClass t)
{
th2=t;
}
public void run()
{
For(int i=0;i<10;i++)
{
th2.get();
}
}
}

class InterThread
{
public static void main(String[] arg)
{
MyClass TObj = new MyClass();
Producer pthread =new POroducer(TObj);
Consumer cthread = new Consumer(TObj);
pthread.start();
cthread.start();
}
}
```

```
Producer producing 0
Consumer consuming: 0
Producer producing 1
Consumer consuming: 1
Producer producing 2
Consumer consuming: 2
Producer producing 3
Consumer consuming: 3
Producer producing 4
Consumer consuming: 4
Producer producing 5
Consumer consuming: 5
Producer producing 6
Consumer consuming: 6
Producer producing 7
Consumer consuming: 7
Producer producing 8
Consumer consuming: 8
Producer producing 9
Consumer consuming: 9
```

4.7 SETTING THE PRIORITY OF A THREAD

We can assign different priorities to the threads in a program by using the setPriorities() method, we have to pass a priority parameters to it. The priority parameters always takes a value between 0 and 10. The parameters are MAX-PRIORITY, MIN_PRIORITY, NORM_PRIORITY is used to set the priority of a thread. MAX-PRIORITY represents 10, MIN_PRIORITY represent 1 and NORM_PRIORITY represent 5 points.

```
Class ThreadE extends Thread
{
public void run()
{
for(int m=1; m<=5;m++)
System.out.println("Callling from ThreadE");
System.out.println("Quitting ThreadE");
}
}

Class ThreadF extends Thread
{
public void run()
{
for(int m=1; m<=5;m++)
System.out.println("Callling from ThreadF");
System.out.println("Quitting ThreadF");
}
}

Class Test
{
public static void main(String args[])
{
ThreadE obj1 = new ThreadE();
obj.setPriority(Thread.MIN_PRIORITY+1);
obj.start();
ThreadF obj2 = new ThreadF();
obj.setPriority(Thread.MAX_PRIORITY-1);
obj2.start();
}
```

4.8 DAEMON THREADS

- Daemon thread is a low priority thread which runs in the back ground.
- For example the thread doing the garbage collection operation for the java runtime system is a daemon thread
- A daemon is simply a thread that has no other role in life than to serve others.
- t.setDaemon(true); method is used to create a daemon thread.
- The daemon thread are also called service provider thread.
- The run() method is an infinite loop for the daemon thread in which it waits for servicing.

```
class First extends Thread
{
public void run()
{
System.out.println("Begin First thread");
for(int i=1;i<5;i++)
System.out.println("i="+i);
System.out.println("Over first thread");
}
}
class DaemonTest
{
public static void main(String args[])
{
First f=new First();
f.setDaemon(true);
if(f.isDaemon())
System.out.println("This is a Daemon thread");
else
System.out.println("This is not a Daemon thread");
f.start();
}
}
```

OUTPUT

This is a Daemon thread
Begin First thread
i=1
i=2
i=3
i=4
Over first thread

4.9 THREAD GROUP (Handlers for Uncaught Exceptions)

Java provides a convenient way to group multiple threads in a single object. In such way, we can suspend, resume or interrupt group of threads by a single method call. Java thread group is implemented by *java.lang.ThreadGroup* class.

Constructors of ThreadGroup class

- `ThreadGroup(String name)` - creates a thread group with given name.
- `ThreadGroup(ThreadGroup parent, String name)` - creates a thread group with given parent group and name.

Important methods of ThreadGroup class

- `int activeCount()` - returns no. of threads running in current group.
- `int activeGroupCount()` - returns a no. of active group in this thread group.
- `void destroy()` - destroys this thread group and all its sub groups.
- `String getName()` - returns the name of this group.
- `ThreadGroup getParent()` - returns the parent of this group.
- `void interrupt()` - interrupts all threads of this group.
- `void list()` - prints information of this group to standard console.

Let's see a 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

```
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]
```

4.10 GENERIC PROGRAMMING

Generic Programming is a programming method that provides common code to many different types of objects.

- Provides reusability of codes.
- Compact code can be created
- It saves the programmer burden of creating separate methods for handling data belonging to different data types.

4.11 GENERIC CLASSES

- A class, interface, or method that operates on a parameterized type is called **generic**, as in **generic class** or **generic method**.
- Provides common code for different types of objects.
- It contains one or more type variables defined between < > and separated by commas.
- These variables are used to declare fields and used to specify return types of methods inside the generic class.

• General form:

```
[access specifier] class generic_classname <type var1,type var2,..>
{
 Field declarations;
 Methods definition;
}
```

• General form of generic instantiation:

```
classname <data type >  object=new classname(parameters);
```

```
import java.io.*;
class pair<T>
{
T first;
T second;

pair()
{
first=null;
second=null;
}

pair( T f,T s)
{
first=f;
second=s;
}

public T getfirst()
{
return first;
}

public T getsecond()
{
return second;
```

```
}
}

class main1
{
public static void main(String args[])
{
 pair<Integer> p1=new pair<Integer>(11,12);
 pair<String> p2=new pair<String>("sunil","kumar");
 System.out.println(p1.getfirst());
 System.out.println(p1.getsecond());
 System.out.println(p2.getfirst());
 System.out.println(p2.getsecond());
}
}
```

OUTPUT
11
12
Sunil
Kumar

4.12 GENERIC METHODS

- A special type of method that can be used for different types of data.
- A generic methods can be defined both inside ordinary classes and inside generic classes. generic methods are defined with one or more variables.

General form of generic method definition:

```
[access specifier] <type var1,type var2,..> return _type method_name(parameter){}
```

General form of generic method calling:

```
Class_name/obj_name.method_name(parameters);
   or
Class_name/obj_name.[<type>]method_name(parameters);
```

PROGRAM

```
import java.io.*;
class Print
{
static <T> void print(T val)
{
System.out.println("Value:"+val);
}
}
class maint
{
 public static void main(String args[])
 {
  System.out.println("Integer");
  Print.print(10);
  System.out.println("Float");
  Print.print(1.23);
  System.out.println("String");
  Print.print("java");
 }
}
```

OUTPUT
Integer
Value: 10
Float
Value: 1.23
String
Value: java

PROGRAM

```
import java.io.*;
class g<T>
{
T a;
public void set(T a)
{
 this.a=a;
}
public T get()
{
 return a;
}
}
class maint
public static void main(String args[])
{
g<Integer> d=new g<Integer>();
g<String> d1=new g<String>();
d.set(new Integer(11));
d1.set(new String("sachin"));
System.out.println(d.get());
System.out.println(d1.get());
}}
```

OUTPUT
11
sachin

4.13 BOUNDED TYPES

- While creating objects to generic classes we can pass any derived types as type parameters.
- Many times it will be useful to limit the types that can be passed to type parameters. For that purpose, bounded types are introduced in generics.
- Using bounded types, we can make the objects of generic class to have data of specific derived types.
- 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 types as type parameter.
- The syntax for declaring bounded type parameters is
  <T extend SuperClass>
- This specifies that T can only be replaced by 'SuperClass' or it's sub classes.

For Example:

```
Class Test<T extends Number>
{
T t;
public Test(T t)
{
this.t=t;
}
public T getT()
{
return t;
```

```
}
}
public class BoundedTypeDemo
{
public static void main(String args[])
{
// Creating object by passing Number as a type parameter
Test<Number> obj1 = new Test<Number>(123);
System.out.println("The integer is:"+obj1.getT());
//While Creating object by passing String as a type paramenter, it gives compiler time error
Test<String> obj2 = new Test<String>("I am string");
System.out.println("The string is:" +obj2.getT());
}
}
```

4.14 RESTRICTIONS AND LIMITATIONS

1. In Java, generic types are compile time entities. The runtime execution is possible only if it is used along with raw type.
2. Primitive type parameters are not allowed for generic programming. For example: Stack<int> is not allowed.
3. For the instances of generic class throw and catch instance are not allowed.

   For example:

   public class Test<T> extends Exception
   {
   //code //Error can't extend the Exception class
   }
4. Instantiation of genric parameter T is not allowed

   For example:

   new T() //Error
   new T[10]
5. Arrays of parameterized types are not allowed

   For example:

   New Stack<String>[10]; //Error
6. Static field and static methods with type parameters are not allowed.

PART-A

1. **What is Generic Programming?**

   Generic is a mechanism for creating a general model in which generic methods and generic classes enable programmers to specify a single method (or a set of related methods) and single class (or a set of related classes) for performing the desired task.

2. **List out motivation needed in generic programming.**
   - Suppose we want to create a stack and it we create a stack of integers then it will store only the integer elements, if we try to push any string or any double type elements then a compile time error will occur. If we want to push any string then we need to create a separate class and separate methods for handling the string elements. Same is true for the elements of any other data type. This results in complex code building. To avoid such complexity, a concept of generic programming is introduced.
   - It saves the programmers burden of creating separate methods for handling data belonging to different data types.
   - It allows the code reusability
   - Compact code can be created.

3. **Why generic programming is needed?**
   - It saves the programmers burden of creating separate methods for handling data belonging to different data types.
   - It allows the code reusability
   - Compact code can be created.

4. **State any two challenge of generic programming in virtual machine.**
   - The virtual machine does not work with generic classes or generic methods. Instead it makes uses raw types in which the raw types are replaced with ordinary java types. Each type variable is replaced with its bound or with object, if it is not bounded. This technique is called type ensure.
   - In order to handle the type erasure methods the compiler has to generate bridge methods in corresponding class.

5. **List any two advantages of type parameters.**

   Due to the use of type parameter it saves the programmers burden of creating separate methods for handling data belonging to different data types.

   Due to type parameters the early error detection at compile time occurs. This avoids crashing of the code(due to type incompatibility) at run time.

6. **What do you mean by threads in java?**

   Thread is tiny program running continuously. It is light weight process in Java.

7. **Give the difference between process and thread.**

Sl.No	Thread	Process
1	Thread is a light weight process	Process is heavy weight process
2	Threads do not require separate address space for its execution. It runs in the address space of the process to which it belongs	Each process requires separate address space to execute.

8. **What are the different stages in thread?**

   1. New State 2. Runnable state 3. Waiting state 4. Time waiting state

5. Blocked state 6. Terminate state

9. What do you mean by synchronization?

When two or more threads need to access shared memory, then there is some way to ensure that the access to the resource will be by only one thread at a time. The process of ensuring one access at a time by one thread is called synchronization. **(OR)**

With respect to multithreading, synchronization is the capability to control the access of multiple threads to shared resources. Without synchronization, it is possible for one thread to modify a shared object while another thread is in the process of using or updating that object's value. This often leads to significant errors

10. Why would you use a synchronized block vs. synchronized method?

Synchronized blocks place locks for shorter periods than synchronized methods.

11. What are the three ways by which the thread can enter in waiting stage?

Waiting state: Sometime one thread has to undergo in waiting state because another thread starts executing. This can be achieved by using wait() state.

Timed waiting state: There is a provision to keep a particular threading waiting for some time interval. This allows to execute high prioritized tread first. After the timing gets over, the thread in waiting state enters in runnable state. This can be achieved by using the sleep() command.

Blocked State : When a particular thread issues an input/output request then operating system sends it in blocked state until the I/O operation gets completed. After the I/O completion the thread is sent back to the runnable state. This can be achieved by using suspended() method.

12. What is multithreading?

Multithreading is an environment in which multiple threads are created and they can execute simultaneously. The multiple threads can be created either by extending the thread class or by implementing the runnable interface.

13. What are the two ways of creating a thread?

- Thread class
- Runnable Interface

14. Why do we need run() and start() method both? Can we achieve it with only run() method?

We use start() method to start a thread instead of calling run() method because the run() method is not a regular class method. It should only be called by the JVM. If we call the run() method directly then it will simply invoke the caller's thread instead of its own thread. Hence the start() method is called which schedule the thread with the JVM.

15. What's the difference between the methods sleep() and wait()

The code sleep(1000); puts thread aside for exactly one second. The code wait(1000), causes a wait of up to one second. A thread could stop waiting earlier if it receives the notify() or notifyAll() call. The method wait() is defined in the class Object and the method sleep() is defined in the class Thread.

16. What is the need for thread?

In Java thread are used to handle multiple tasks together. This kind of programming is called concurrent programming.

17. Name any four thread constructor.

- Thread()
- Thread(String name)
- Thread(Runnable target)
- Thread(Runnable target, String name)

18. When thread is initiated and created what is its initial stage?

A thread is in "Ready" state after it has been created and started. This state signifies that the thread is ready for execution. From here, it can be in the running state.

19. What is daemon thread and which method is used to create the daemon thread?

Daemon thread is a low priority thread which runs intermittently in the back ground doing the garbage collection operation for the java runtime system. setDaemon method is used to create a daemon thread.

20. What is thread priority?

Thread Priority is an integer value that identifies the relative order in which it should be executed with respect to others. The thread priority values ranging from 1- 10 and the default value is 5. But if a thread have higher priority doesn't means that it will execute first. The thread scheduling depends on the OS.

21. What is a thread group?

A thread group is a data structure that controls the state of collection of thread as a whole managed by the particular runtime environment.

UNIT IV PART-A

1. What is concurrent programming? A/M 2018
2. What are threads? A/M 2011
3. What do you mean by Threads in Java? N/D 2011
4. What is the need for thread? A/M 2016 A/M 2013
5. Write the properties of thread. N/D 2014
6. State the properties of thread. N/D 2015
7. What are the different states in thread? N/D 2010
8. What is thread state? N/D 2012
9. Write the life cycle of thread. N/D 2013
10. *Name any four thread constructor. A/M 2016* A/M 2013
11. *Point out the parent thread in any java program. A/M 2018*
12. What is multithreading N/D 2011 A/M 2012
13. What is multithreaded programming? A/M 2014
14. What is multithreaded programming? N/D 2014
15. What do you mean Synchronization? N/D 2010
16. Write note on Synchronization. A/M 2012
17. What are the parts of synchronization that are often needed?
18. Why is synchronization required in thread? N/D 2015

19. Who wants to be generic programmer? A/M 2011
20. What is the need of generic programming A/M 2013
21. Define generic programming. A/M 2014
22. What is meant by generic programming? N/D 2015 A/M 2016
23. *What is virtual machine in generic programming? A/M 2015*

24. Mention some of the exceptions serious occurred in generic programming. N/D 2014
25. Mention the elements of stack trace in generic programming. N/D 2015 A/M 2016
26. Write about simple generic class with an example N/D 2011
27. Describe the generic classes. N/D 2012
28. Define generic classes A/M 2015
29. What is meant by generic classes.? Give example. N/D 2013

30. List out some limitations of intrinsic locks and conditions. A/M 2015
31. List out the methods of executors. N/D 2012
32. Define assertion. A/M 2011
33. Define assertion. N/D 2014
34. Define blocking queue. A/M 2011
35. Define barrier and exchanger. A/M 2015

PART-B

1. Explain how threads are created in Java. (8) **N/D 2011**
2. What is a thread? Explain its states and methods. [Marks 8] **N/D 2010**
3. Explain the various states of thread. (8) **A/M 2011**
4. Write about various threads states in Java. (8) **N/D 2011**
5. Discuss briefly about thread state. (8) **A/M 2014**
6. State the thread with neat diagram (10) **A/M 2012**
7. Define threads. Describe in detail about thread life cycle. (8) **A/M 2013**
8. What is thread? Explain the life cycle of threads. (8) **A/M 2015**
9. Define threads. Describe in detail about thread life cycle. (8) **A/M 2016**
10. Define thread. Explain the life cycle of thread. (4) **A/M 2018**
11. How to extend the thread class? Give an example. (8) A/M 2016
12. How to extend the thread class? Give an example. (8) A/M 2013
13. How to implement runnable interface for creating and starting a threads? (8) A/M 2013 A/M 2016
14. Explain thread properties. [Marks 8] **N/D 2010**
15. Explain thread priorities (6) **A/M 2012**
16. Explain the properties of thread in detail. (8) **A/M 2015**
17. Explain the methods of interrupting threads in java. [Marks 8] N/D 2010
18. How to interrupt the thread? (4) A/M 2015
19. Explain two types of thread implementation supported by JAVA. Give examples. Write note on interrupting thread. (16) N/D 2015
20. Write a Java program to illustrate multithreaded programming. (10) **N/D 2011**
21. Write the java application that illustrate the use of multithreading. Explain the same with sample input. (16) **A/M 2012**
22. Write in detail about multithread programming with example (16) **N/D 2012**
23. Explain in detail about multithread programming with example (8) **A/M 2014**
24. Describe the functions of executors in multithreading. (8) **A/M 2011**
25. Explain the process of synchronization in detail with suitable example. (16) A/M 2011
26. Explain synchronization in detail. (6) N/D 2011
27. Explain about thread synchronization with an example. (8) A/M 2013
28. Discuss the concept of synchronization of thread and develop a Java code for reader/writer problem. N/D 2013
29. Write short notes about synchronization. (4) A/M 2015
30. Explain about thread synchronization with an example. (8) A/M 2016
31. Write short notes on synchronization (4) A/M 2018
32. Explain thread synchronization (4) A/M 2018
33. Define thread. Explain the state of thread briefly. State the reasons foe synchronization in thread. Write a simple concurrent programming to create, sleep, and delete the threads. (16) **N/D 2014**
34. Write brief notes on Concurrent programming. (8) **A/M 2014**
35. What is meant by concurrent programming? Define thread. Discuss the two ways of implementing thread using eg. (16) **N/D 2013**
36. Design a java program to print the odd and even numbers list up to 20 using threading concept. (8) A/M 2018

37. ***What is thread? Discuss the types of threads. Write a program for reader and writer problem using thread in JAVA. (16) N/D 2015***
38. ***Discuss thread-safe collection briefly. Write a simple multithreaded program for reader and writer problem. (16) N/D 14***
39. ***Explain in detail about thread safe collection. (16) N/D 2012***
40. ***Write short notes on thread safe collection. (8) A/M 2018***
41. ***Explain about Executors, Thread Pool, semaphore, countdown hatches. (8) A/M 2015***
42. ***Explain briefly about executors. (8) A/M 2014***
43. ***Write short notes on executors (4) A/M 2018***
44. Explain about reflection and generics. Give an example. (8) **A/M 2013**
45. Discuss briefly about the motivation for generic programming. (8) **A/M 2014**
46. State the motivation of generic programming. Explain the generic classes and method with examples. (16) **N/D 2014**
47. Explain the generic classes and generic methods with example. [Marks 8] **N/D 2010**
48. Explain generic class and generic method (16) **A/M 2012 N/D 2012**
49. Explain in detail about generic classes and methods. (8) **A/M 2015**
50. State the motivations of generic programming. Explain the generic class and methods with example. (16) **N/D 2015**
51. Explain about generic classes and their methods in detail. (8) **A/M 2018**
52. ***Explain the concept of Generic Type Information in Virtual Machine. (8) N/D 2011***
53. ***Explain about generic code and virtual machine. (8) A/M 2013***
54. State the inheritance rule for generic type. (8) **A/M 2011**
55. Write about inheritance rules for generic types with example. (8) **A/M 2015**
56. Explain in detail about generic inheritance and generic interface. Discuss exploring the impact of inheritance in generic class with example. (16) **N/D 2014**
57. Discuss about translating generic expressions and calling legacy code. (8) **N/D 2011**
58. ***Mention the motivation of generic programming. What is an assertion in java? Why use assertion? Discuss the type of assertion. Give example. (16) N/D 2013 A/M 2016***
59. ***Explain briefly about assertion. (8) A/M 2014***