Multi-Threading:

1. What is Class Loading process ?

▼ Ans

- Class Loading the .class file into the JVM memory is know as Class loading or Class loading is a process of loading the classes to class area or static pool where static variable, methods and blocks are allocates its memory.
- When we start the execution of java program. First, JVM divides the main memory into stack, Heap and class area. then JVM loads into stack area so once JVM loads into stack area it calls the ClassLoader to load the specified class into class area, while loading the class definition into class area ClassLoader is resposible to allocate the memory for static members of the class. then JVM start searching for main method which as the String array as argument. if it is present main method loaded into stack area then program control switch from JVM to main method then line by line execution will start. if we created any object their it allocates the memory in heap area. if we are calling any method, weather it is static or non-static method that method definition loaded into stack area, then program control switches from calling method to called method. actual execution happen in stack area. this one stack area is a thread. this is the main thread which is created by the JVM.
- When the class definition is not present in the class area, JVM calls the ClassLoader to load specified class into class area

▼ What is stack area ? Why we call execution area as stack area ?

- Stack area is used to store the local variables and order of method execution and it follows LIFO order means (Last In First Out)

▼ What is class area ?

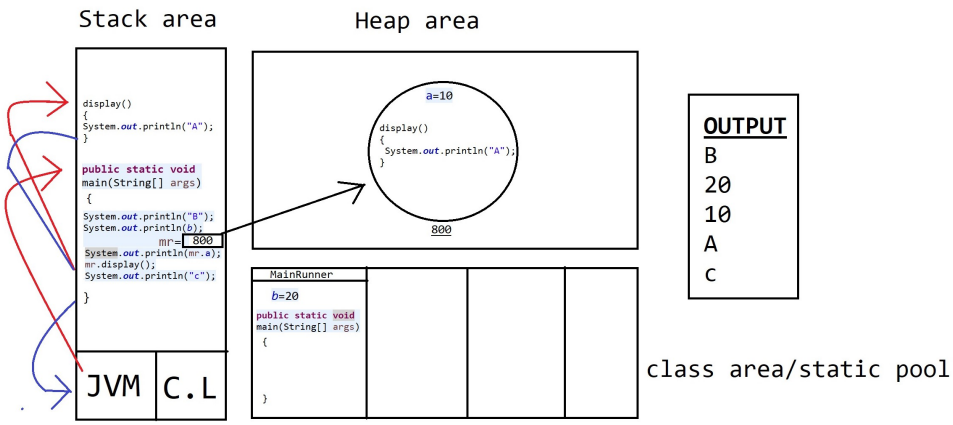
- Class definition is present in class area that is reason we called as class area and also we call it as static pool because static member and variable are allocates memory in this pool

▼ What is Heap area ?

- Heap area is used to store the object and Non-static variable, members will allocates memory in Heap area.

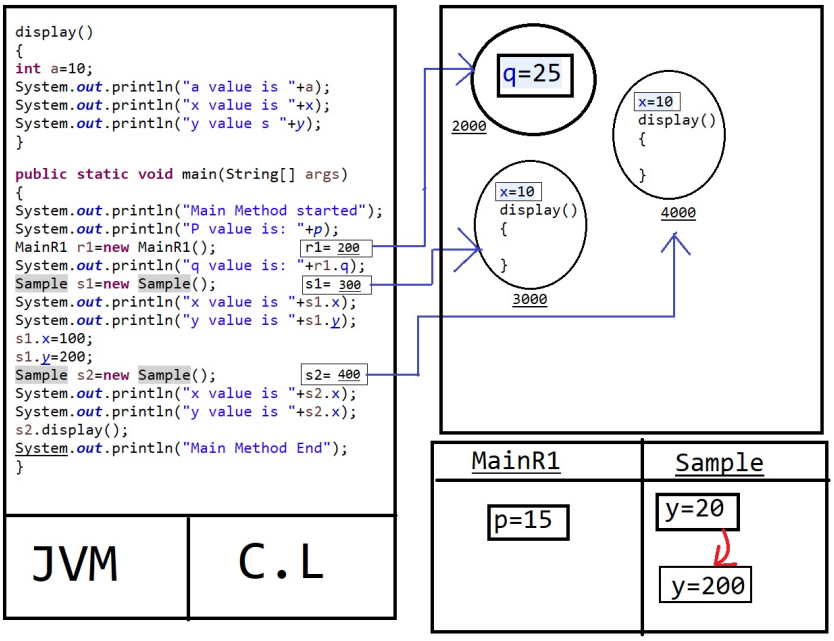
▼ Program with Diagram

```
public class MainRunner
{
  int a=10;
  static int b=20;
  void display()
  {
    System.out.println("A");
  }
  public static void main(String[] args)
  {
    System.out.println("B");
    System.out.println(b);
    MainRunner mr=new MainRunner();
System.out.println(mr.a);
mr.display();
System.out.println("c");
  }
}
------------------OUTPUT-----------------
B
20
10
A
c
```



▼ Program with Diagram

```
public class MainR1
{
  static int p=15;
  int q=25;
  public static void main(String[] args)
  {
    System.out.println("Main Method started");
    System.out.println("P value is: "+p);
    MainR1 r1=new MainR1();
    System.out.println("q value is: "+r1.q);
    Sample s1=new Sample();
    System.out.println("x value is "+s1.x);
    System.out.println("y value is "+s1.y);
    s1.x=100;
    s1.y=200;
    Sample s2=new Sample();
    System.out.println("x value is "+s2.x);
    System.out.println("y value is "+s2.y);
    s2.display();
    System.out.println("Main Method End");
  }
}
public class Sample
{
  int x=10;
  static int y=20;
    void display()
    {
      int a=10;
      System.out.println("a value is "+a);
      System.out.println("x value is "+x);
      System.out.println("y value s "+y);
    }
}
----------OUTPUT----------------------
Main Method started
P value is: 15
q value is: 25
x value is 10
y value is 20
x value is 10
y value is 200
a value is 10
x value is 10
y value s 200
Main Method End
```

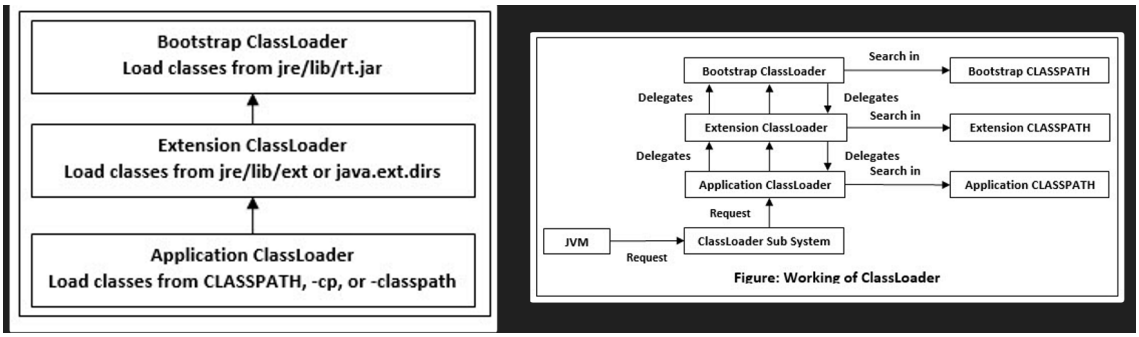


2. What is ClassLoader ?

▼ Ans

- ClassLoader is an abstract class. It present **java.lang** package. It loads classes from different resources.
- When JVM request for a class, it invokes a loadClass() method of the java.lang.ClassLoader class by passing the fully classified name of the class. The loadClass() method calls for findLoadedClass() method to check that the class has been already loaded or not. It is required to avoid loading the class multiple times.
- If the class is already loaded, it delegates the request to parent ClassLoader to load the class. If the ClassLoader is not finding the class, it invokes the findClass() method to look for the classes in the file system. The following diagram shows how ClassLoader loads class in Java using delegation.

▼ ClassLoader images



- Suppose that we have an application specific class Demo.class. The request for loading of this class files transfers to Application ClassLoader. It delegates to its parent Extension ClassLoader. Further, it delegates to Bootstrap ClassLoader. Bootstrap search that class in rt.jar and since that class is not there. Now request transfer to Extension ClassLoader which searches for the directory jre/lib/ext and tries to locate this class there. If the class is found there, Extension ClassLoader loads that class. Application ClassLoader never loads that class. When the extension ClassLoader does not load it, then Application ClaasLoader loads it from CLASSPATH in Java.

3. How many types of Multi-tasking ?

▼ Ans

▼ Programming based Multi-tasking

- Programming based Multi-tasking, We can achieve by Multi-threading.

▼ Simultaneous Multi-tasking

- means at a time it performs only one task.
- Java supports the Simultaneous Multi-tasking.

▼ Parallel Multi-tasking

- means at a time it performs more then one task.
- Python supports the Parallel multi-tasking.

▼ Processed based Multi-tasking

- Processed based Multi-tasking, We can achieve by Multi-processing.

- Multi-processing—> means Multi-processing is a system that has more than one processor. CPU's are added for increasing computing speed of the system.

4. What is the advantages of Simultaneous Multi-tasking ?

▼ Ans

- It prevents the CPU **idle.**
- It utilize the CPU time

5. What is Multi-threading ?

▼ Ans

- The process creating a separate execution path we call it as Multi-threading.
- Execution path or stack are nothing but thread.
- For every task if we create a separate thread then it can be possible switch from one thread to another thread to execute simultaneously.
- If we want perform the multiple task simultaneously, then we should be go for Multi-threading.

▼ Why we want perform the task simultaneously

- For maximum utilization of CPU.

▼ Why we go for Multi-Threading

- If we want perform the multiple task simultaneously, then we should be go for Multi-threading.

6. What is the difference between Multi-threading and Multi-Processing ?

▼ Ans

Multi-Threading	Multi-Processing
1). Multi-Threading is a programming based multi-tasking.	1). Multi-Processing is a process based multi-tasking
2). It creates multiple Threads of a single process to increase computing speed of system.	2). Adds CPU's to increase computing speed of the system.
3). It shares the memory	3). It doesn't share the memory
4). It has only one memory	4). It has separate memory of each and every CPU's
5). Processing time is fast because it has one memory.	5). Processing time is slow compare to multi-Threading because it will take time to switch the CPU
6). It is light weight because multiple Threads shares the same memory so it has only one memory.	6). It is heavy weight because it has more than one memory.
7). No need to provide load and requisites the memory because it has only one memory.	7). Need to provide load and requisites the memory.
8). In single program if we perform multiple tasks that is Thread Based.	8). If we perform multiple tasks in multiple program that is processed.

7. How to create separate execution path or thread execution path ?

▼ Ans

- When we start execute the java program JVM create one thread, that thread name is main i.e. main stack.
- If we wants to executes simultaneously we should create separate thread/stack-area and load our method.
- To create the separate execution-path/tread/stack-area, we need to call start() method.

▼ start()

- This method is present in Thread class and start() is non-static method. (Thread class present in lang package)
- When we call start() method it creates the separate execution path and it loads the run() method to there.

▼ run()

- This method is present in Thread class.
- Anything if we want to execute separate execution path. that code we must write inside run() method of Thread class.
- we should override run() method according to our implementation in sub-class and make that sub-class extends Thread class.

▼ sleep()

- sleep() is a static overloaded method of Thread class.
- When we call sleep() method currently executing thread goes to a sleep state. where we are calling this method, that method goes to a sleep state
- These method declaring with throws, when we are calling this method that statements should surround by try and catch.
- sleep() method declared with InterruptedException. InterruptedException is a checked Exception. Check Exception explicitly we should throws but by default unchecked exception will throws. that is why calling statement should surrounded by try and catch.
- sleep(long) —> When we call this method and specified 3000milisec, currently executing thread goes to sleep state. remaining portion will not get execute util 3000milisec.
- sleep(long , int) —> When we call this method, we should specified mili-seconds, neno seconds.

▼ Program (Local inner class Implementation)

```
public class MainRunner
{
  public static void main(String[] args)
  {
    System.out.println("Main Method Started");
    Sample s1=new Sample();
    class MyThread1 extends Thread
    {
      public void run()
      {                          //Local inner class implementation
        s1.display();
      }
    }
    class MyThread2 extends Thread
    {
      public void run()
      {
        s1.write();              //Local inner class implementation
      }
    }
    Thread t1=new MyThread1();
    Thread t2=new MyThread2();
    t1.start();
```

```
    t2.start();
    System.out.println("Main Method Ends");
  }
}
--------------------------------
public class Sample
{
    void display()
    {
      for(int i=1; i<=10; i++)
      {
        System.out.println("J-Spiers "+i);
        if(i==5)
          try {
            Thread.sleep(8000);
          }
        catch(InterruptedException e)
        {
          System.out.println(e);
        }
      }
    }
    void write()
    {
      for(int i=1; i<=10; i++)
      {
        System.out.println("Q-Spiders "+i);
      }
    }
}
------------------------OUTPUT-----------------------
Main Method Started
Main Method Ends
Q-Spiders 1
Q-Spiders 2
Q-Spiders 3
Q-Spiders 4
Q-Spiders 5
Q-Spiders 6
Q-Spiders 7
J-Spiers 1
J-Spiers 2
J-Spiers 3
J-Spiers 4
J-Spiers 5
Q-Spiders 8
Q-Spiders 9
Q-Spiders 10
J-Spiers 6
J-Spiers 7
J-Spiers 8
J-Spiers 9
J-Spiers 10
```

▼ Program (Separate outer class implementation)

```
public class SmallThread extends Thread
{
  public void run()
  {
    for(char ch='a'; ch<='z'; ch++)
    {
      System.out.println(ch);
      try
      {
        Thread.sleep(1000);
      }
      catch(InterruptedException e)
      {
        System.out.println(e);
      }
    }
  }
}
public class CapitalThread extends Thread
{
  public void run()
  {
    for(char ch='A'; ch<='Z'; ch++)
    {
      System.out.println(ch);
      try
      {
        Thread.sleep(1000);
      }
      catch(InterruptedException e)
      {
        System.out.println(e);
      }
    }
  }
}public class MainRunner
{
  public static void main(String[] args)
  {
    Thread t1=new SmallThread();
    Thread t2=new CapitalThread();
    t1.start();
    t2.start();
  }
}
----------------------OUTPUT--------------
Main Method Ends
A
a
```

```
b
B
c
C
D
d
e
E
F
f
G
g
h
H
I
i
j
J
K
k
L
l
M
m
n
N
O
o
p
P
Q
q
r
R
s
S
T
t
u
U
v
V
w
W
X
x
Y
y
Z
z
```

▼ Program (Anonymous)

```
public class MainRunner
{
  public static void main(String[] args)
  {
    Thread t1=new Thread()
    {
      public void run()
      {
        for(char ch='a'; ch<='z'; ch++)
        {
          System.out.println(ch);
          try
          {
            Thread.sleep(100);
          }
          catch(InterruptedException e)
          {
            System.out.println(e);
          }
        }
      }
    };
    Thread t2=new Thread()
    {
      public void run()
      {
        for(char ch='A'; ch<='Z'; ch++)
        {
          System.out.println(ch);
          try
          {
            Thread.sleep(100);
          }
          catch(InterruptedException e)
          {
            System.out.println(e);
          }
        }
      }
    };
    t1.start();
    t2.start();
  }
}
------------------OUTPUT----------------------------
a
A
b
B
c
C
d
```

```
D
e
E
F
f
G
g
H
h
i
I
J
j
K
k
L
l
m
M
n
N
O
o
p
P
Q
q
r
R
S
s
T
t
U
u
v
V
W
w
X
x
Y
y
Z
z
```

8. What is Thread ?

▼ Ans

- A Thread is a very light-weighted process, or we can say the smallest part of the process that allows a program to operate more efficiently by running multiple tasks simultaneously.
- Thread is a class in java present in lang package and It implements Runnable Interface.
- Thread class has a 9 Constructor. This the example for Constructor Overloading.

9. How many constructors are there in Thread class ?

▼ Ans

▼ It has 9 Constructors (but 4 is more Important)

▼ No argument Constructor (public Thread();)

- When we invoke the no argument constructor, thread object is created and run() method will takes the thread class implementation.
- When we call start() method, it loads the run() method implementation of Thread class.

▼ Example Program

```
public class SmallThread extends Thread
{
  public void run()
  {
    for(char ch='a'; ch<='z'; ch++)
    {
      System.out.println(ch);
      try
      {
        Thread.sleep(1000);
      }
      catch(InterruptedException e)
      {
        System.out.println(e);
      }
    }
  }
}
public class CapitalThread extends Thread
{
  public void run()
  {
```

```
    for(char ch='A'; ch<='Z'; ch++)
    {
      System.out.println(ch);
      try
      {
        Thread.sleep(1000);
      }
      catch(InterruptedException e)
      {
        System.out.println(e);
      }
    }
  }
}public class MainRunner
{
  public static void main(String[] args)
  {
    Thread t1=new SmallThread();
    Thread t2=new CapitalThread();
    t1.start();
    t2.start();
  }
}
```

▼ public Thread(Runnable);

- If we want to invoke this constructor we must pass runnable type object.
- If we want to create a thread object by passing a Runnable type object means, we should provide implementation to Runnable method
- We cannot create a object for Runnable because it is a Interface, but we can create object to their types or class.
- When we invoke the Runnable type constructor by passing Runnable object, thread object is created and run() method will takes the Runnable's child class implementation.
- When we call start() method, it loads the run() method implementation of Runnable Interface.

▼ What is Runnable ?

- Runnable is an interface.
- It has only one abstract method i.e run() which is overridden in Thread class.

- It is a Functional interface. (which interface has only one abstract method that type of interface is called Functional Interface)

▼ Program

```
public class MyRunnable implements Runnable
{
  @Override
  public void run()
  {
    for(int i=1; i<=5; i++)
    {
      System.out.println(i);
    }
  }
}
public class CapitalThread extends Thread
{
  public void run()
  {
    for(char ch='A'; ch<='Z'; ch++)
    {
      System.out.println(ch);
      try
      {
        Thread.sleep(1000);
      }
      catch(InterruptedException e)
      {
        System.out.println(e);
      }
    }
  }
}
public class MainRunner3
{
  public static void main(String[] args)
  {
    Runnable target=new MyRunnable();
    Thread t1=new Thread(target);
    Thread t2=new CapitalThread();
    t1.start();
    t2.start();
  }
}
```

▼ public Thread(String);

- We can set Thread or stack area Name by invoking the String type constructor.

```
public class MainRunner5
{
  public static void main(String[] args)
  {
     Thread t1=new Thread();
    System.out.println(t1.getId());
    System.out.println(t1.getName());
    System.out.println(t1.getPriority());
     Thread t2=new Thread("Java");   //Using String type constructor
    System.out.println(t2.getId());
    System.out.println(t2.getName());
    System.out.println(t2.getPriority());
     Thread t3=new Thread();
      t3.setName("Python"); //Using setter method
      t3.setPriority(8);
      System.out.println(t3.getId());
      System.out.println(t3.getName());
      System.out.println(t3.getPriority());
  }
}
--------------OUTPUT-------------------
11
Thread-0
5
12
Java
5
13
Python
8
```

▼ public Thread(Runnable, String);

10. How many ways can we provide implementation to a Thread ?

▼ Ans

▼ 2ways

1. By defining sub-class of Thread class (through extends Thread)
2. By passing Runnable object (through implements Runnable)

11. What are the different ways of creating a thread ?

▼ Ans

- There are two ways to create a thread: By extending Thread class. By implementing Runnable interface.

12. What are the important Properties Thread ?

▼ Ans

- Thread has 3 important properties that is

  1)Id 2)Name 3)Priority

  - we can set our own Name and Priority to a thread or stack area by invoking no argument constructor with setter method and by invoking String type constructor
  - We can set Priority exactly between 1 to 10. 1 is the MIN priority and 10 is the MAX priority. if it is not between the 1 and 10 like if it is less 1 or more 10, it will give exception called IllegalArgumentException.
  - Id, we cannot set because Id is assigned by JVM and Id is the read only property.
  - Name and Priority are read as well write properties.
  - When we invoke no arguments constructor, it will take default Id, Name, Priority
  - Default Id —> It is assigned by JVM and user thread Id will from 10.
  - Default Name—> Thread 0, Thread 1, ,,,,,etc
  - Defaulrt Priority—> 5

13. How to fetch the Name, ID, Priority of the thread or stack created by JVM ?

▼ Ans

▼ currentThread

- current is the static method Thread class, it returns the instance of the currently executing Thread.
- When we call this method, it returns reference of the current. if we call this in main method, it returns instance of the main method stack i.e main stack because main method loaded into main stack.

▼ Example Progarm

```
public class MainRunner4
{
  public static void main(String[] args)
  {
     Thread t1=Thread.currentThread();
     System.out.println("ID "+t1.getId());   //It gives thread Id
     System.out.println("Name "+t1.getName()); //It gives name of thread
     System.out.println("Priority "+t1.getPriority());//i gives priority value
  }
}
-------------OUTPUT---------------------
ID 1
Name main
Priority 5
```

▼ Example Progarm-2

```
public class MyRunnable implements Runnable
{
  @Override
  public void run()
  {
    for(int i=1; i<=5; i++)
    {
      System.out.println(Thread.currentThread().getName());
    }
  }
}
public class MainRunner3
{
  public static void main(String[] args)
  {
    Runnable target=new MyRunnable();
    Thread t1=new Thread(target);
    Thread t2=new Thread(target);
    t1.start();
    t2.start();
  }
}
--------------OUTPUT-------------
Thread-0
Thread-0
Thread-0
Thread-0
Thread-0
Thread-1
Thread-1
```

```
Thread-1
Thread-1
Thread-1
```

14. How many times we can start a Thread ?

▼ Ans

- only one time we can start.

15. If we start one Thread object more then once what happens ?

▼ Ans

- No, It is not possible. It will throw an IlligelThreadStateException.

16. Difference between start() and run() ?

▼ Ans

- When we call run() method, it creates only one Thread or Main stack area. First it is going to execute main method and in main method we are calling run() method. that run() method also loads to a main stack. it is not going to load to the separate stack. so that simultaneous execution will not happens when we call only run() method.
- If we call start() method, it creates the separate execution path from their it loads the run() method. so simultaneous execution will happens when we start() method.

17. Instead calling start() method, if we call run() method ?

▼ Ans

- Instead of calling start() method, If we call run() method. run() method is going to load into the main stack. it is not going to load to the separate stack. so that simultaneous execution will not happens when we call only run() method.

18. What is join() method ?

▼ Ans

- join() method which allows one thread to wait until another thread completes its execution.
- join() is a non-static overloaded method of Thread class.
- ▼ There are 3 join() methods present in Thread class
  - ▼ join() method with—> no arguments.
    - If we call join() method with no arguments, util current Thread dies, it not allows to start any new Thread.
  - ▼ join() method with—> (long mili-seconds)
    - If we call join() method with mili-second argument, util current Thread finish the mili-seconds of time, it not allows to start any new Thread. Once its finishes the mili-seconds of time then it allows to start any new Thread
  - ▼ join() method with—> (long mili-seconds, int neno-seconds)
    - util current thread finishes mili-secods and neno-seconds, it not allows to start any new Thread.
- ▼ Program

```
public class CapitalThread extends Thread
{
  public void run()
  {
    for(char ch='A'; ch<='Z'; ch++)
    {
      System.out.println(ch);
      try
      {
        Thread.sleep(1000);
      }
```

```
      catch(InterruptedException e)
      {
        System.out.println(e);
      }
    }
  }
}
public class SmallThread extends Thread
{
  public void run()
  {
    for(char ch='a'; ch<='z'; ch++)
    {
      System.out.println(ch);
      try
      {
        Thread.sleep(1000);
      }
      catch(InterruptedException e)
      {
        System.out.println(e);
      }
    }
  }
}
public class DigitThread extends Thread
{
  public void run()
  {
    for(char ch='0'; ch<='9'; ch++)
    {
      System.out.println(ch);
      try
      {
        Thread.sleep(100);
      }
      catch(InterruptedException e)
      {
        System.out.println(e);
      }
    }
  }
}
public class MainRunner6
{
  public static void main(String[] args) throws InterruptedException
  {
    CapitalThread ct=new CapitalThread();
    SmallThread st=new SmallThread();
    DigitThread dt=new DigitThread();
    ct.start();
      ct.join();
     st.start();
        st.join(1700);
```

```
      dt.start();
  }
}
----------------------------OUTPUT---------------------------
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
a
b
0
1
2
c
3
4
5
6
7
8
9
d
e
f
g
h
i
j
k
l
m
n
```

```
o
p
q
r
s
t
u
v
w
x
y
z
```

19. What is suspend() ,stop(), resume() method ?

▼ Ans

▼ suspend() method

- When we call suspended() method, invoking Thread or current Thread goes to blocked state or freeze state.
- We can again get back that Thread by using resume() method.

▼ resume() method

- When we call resume() method, it will get back the Thread from blocked state or freezing state.

▼ stop() method

- When we call stop() method, invoking Thread goes to dead state. when Thread goes to dead state that one we cannot resume.

▼ Program

```
public class DigitThread extends Thread
{
  public void run()
  {
    for(char ch='0'; ch<='9'; ch++)
    {
      System.out.println(ch);
      try
      {
        Thread.sleep(100);
      }
      catch(InterruptedException e)
```

```
      {
        System.out.println(e);
      }
    }
  }
public class MainRunner7
{
  public static void main(String[] args) throws InterruptedException
  {
    DigitThread dt=new DigitThread();
    dt.start();
    Thread.sleep(400);
    dt.suspend();
    Thread.sleep(1000);
    dt.resume();
  }
}
-----------------------OUTPUT----------------------------
0
1
2
3
4
5
6
7
8
9
```

20. What is Thread Interruption Concept ?

▼ Ans

- When one resource is shared with multiple Threads, there chance to occur Thread Interruption , Data Interruption, Consistency is possible to maintain and there is chance to occur dead-lock.
- Note:→ Resource is nothing but a Object in Java.

▼ Program

```
public class Arithmetic
{
  int a;
  int b;
  public Arithmetic(int a, int b)
  {
    super();
    this.a=a;
```

```
    this.b=b;
  }
   void display()
  {
    System.out.println("A value is :"+a);
    System.out.println("B value is :"+b);
    try
    {
      Thread.sleep(3000);
    }
    catch(InterruptedException e)
    {
      System.out.println(e);
    }
    System.out.println("sum is :"+(a+b));
    System.out.println("Difference is :"+(a-b));
    System.out.println("Product is :"+a*b);
    System.out.println("Quoiet is "+a/b);
    System.out.println("Remember is :"+a%b);
  }
   void update()
  {
    this.a=a+100;
    this.b=b+100;
  }
}
public class MainRunner8
{
  public static void main(String[] args)
  {
    Arithmetic ar=new Arithmetic (25,10);
    Thread t1=new Thread()
    {
      public void run()
      {
        ar.display();
      }
    };
    Thread t2=new Thread()
    {
      public void run()
      {
        ar.update();
      }
    };
    t1.start();
    t2.start();
  }
}
--------------------------OUTPUT-------------------------
A value is :125
B value is :110
sum is :235
Difference is :15
```

```
Product is :13750
Quoiet is 1
Remember is :15
```

21. How to avoid Thread Interruption ?

▼ Ans

- To avoid Thread-Interruption/data-Interruption/dead-lock, we should make that resource available to only one thread util it finishes its task or we should make that resource as single treaded.

22. How to make your class as single threaded to avoid data-Interruption or to maintain the consistency or to avoid dead-lock ?

▼ Ans

- By using synchronization.

▼ What is Synchronization ?

- The process of control the access of multiple Threads to any shared resources is called Synchronization. or The process of make the object as a Thread-safe.

▼ How to make Synchronization ?

- Make all behavior of that object prefixed by “synchronized” keyword.

▼ Why we go for Synchronization ?

- To make the object as a single Threaded. When object is single Threaded, it is a Thread safe.

▼ Example for Single-Thread or Synchronized class

- Synchronized class:- means that object can possible to use only one Thread at a time. that type of class objects call it as Synchronized class.

1. Vector class
2. Hashtable class
3. StringBuffer class

▼ Program

```
public class Arithmetic
{
  int a;
  int b;
  public Arithmetic(int a, int b)
  {
    super();
    this.a=a;
    this.b=b;
  }
  synchronized void display()
  {
    System.out.println("A value is :"+a);
    System.out.println("B value is :"+b);
    try
    {
      Thread.sleep(3000);
    }
    catch(InterruptedException e)
    {
      System.out.println(e);
    }
    System.out.println("sum is :"+(a+b));
    System.out.println("Difference is :"+(a-b));
    System.out.println("Product is :"+a*b);
    System.out.println("Quoiet is "+a/b);
    System.out.println("Remember is :"+a%b);
  }
  synchronized void update()
  {
    this.a=a+100;
    this.b=b+100;
  }
}
public class MainRunner8
{
  public static void main(String[] args)
  {
    Arithmetic ar=new Arithmetic (25,10);
    Thread t1=new Thread()
    {
      public void run()
      {
        ar.display();
      }
    };
    Thread t2=new Thread()
    {
      public void run()
      {
```

```
        ar.update();
      }
    };
    t1.start();
    t2.start();
  }
}
--------------------OUTPUT--------------------------
A value is :25
B value is :10
sum is :35
Difference is :15
Product is :250
Quoiet is 2
Remember is :5
```

23. How to make Thread is non-sharable ?

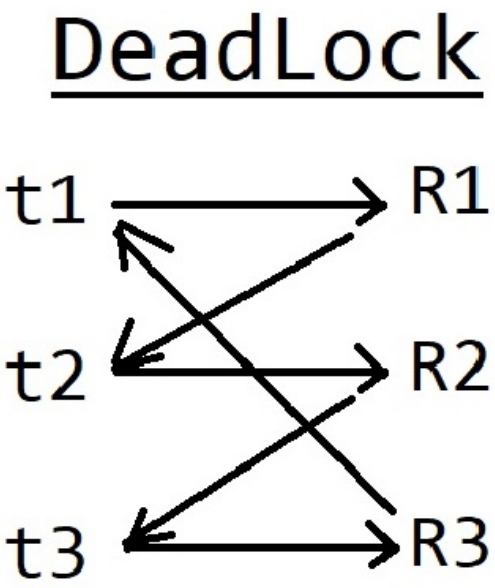
▼ Ans

- By Synchronize the methods of that class then that class object will become a non-sharable or Single Threaded.

24. What is Dead-Lock ?

▼ Ans

- 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, this condition is called deadlock.



25. What are the advantages of Single-Threaded ?

▼ Ans

1. Data-Interruption is not happens.
2. Consistency is maintained.
3. It will not comes under dead-clock situation.

26. What are the Dis-advantages of Single-Threaded ?

▼ Ans

1. Objects becomes slow because only one Thread will perform the task.

Ex:—> StringBuilder is faster then StringBuffer.

27. What is Inter-Thread Commutation ?

▼ Ans

- Inter-Thread Commutation is a way by which synchronized threads can communicate with each other using the methods namely wait(), notify() and notifyAll().
- Inter-Thread Commutation, we can achieve in java by using Object class methods i.e. There are 5 Objects class method to establish the Commutation

between one thread to another thread. 1)wait() overloaded methods 2)notify() method 3)notifyAll() method.

▼ wait() method in Object class

- wait() is non-static final overloaded method of Object class. When wait() method is called, the calling thread stops its execution until notify() or notifyAll() method is invoked by some other Thread.

▼ notify() method in Object class

- When we call notify() method only one Thread comes to a running state from wait state.
- notify() method is used to wake up a single thread.

▼ notifyAll() method in Object class

- When we call notifyAll() method all Threads which are in the wait state will comes to running state.
- notify() method is used to wake up a multiple threads.

▼ Program

```
public class Account
{
  long balance;
  public Account(long balance)
  {
    super();
    this.balance=balance;
  }
  synchronized void withdraw(int amt)
  {
    if(balance<amt)
    {
      System.out.println("insufficient balance..");
    try
    {
      wait();
    }
    catch(InterruptedException e)
    {
      System.out.println(e);
    }
    withdraw(amt);
    return;
```

```
    }
    System.out.println("Withdraw successfull rs. "+amt);
    balance=balance-amt;
    System.out.println("Available balance: "+balance);
  }
  synchronized void deposit(int amt)
  {
    balance=balance+amt;
    System.out.println("Deposited Succesfully...");
    System.out.println("Available balance: "+balance);
    notify();
  }
}
public class Bank
{
  public static void main(String[] args) throws InterruptedException
  {
    Account ac=new Account(500);
    Thread wt=new Thread()
    {
      public void run()
      {
        ac.withdraw(8000);
      }
    };
    Thread dt1=new Thread()
    {
      public void run()
      {
        ac.deposit(3000);
      }
    };
    Thread dt2=new Thread()
    {
      public void run()
      {
        ac.deposit(5000);
      }
    };
    wt.start();
    Thread.sleep(5000);
    dt1.start();
    Thread.sleep(8000);
    dt2.start();
  }
}
--------------------------------OUTPUT-----------------------------
insufficient balance..    //waiting 5k mili-seconds
Deposited Succesfully...
Available balance: 3500
insufficient balance..    //waiting 8k mili-seconds
Deposited Succesfully...
Available balance: 8500
```

```
Withdraw successfull rs. 8000
Available balance: 500
```

28. What is Thread Schedular ?

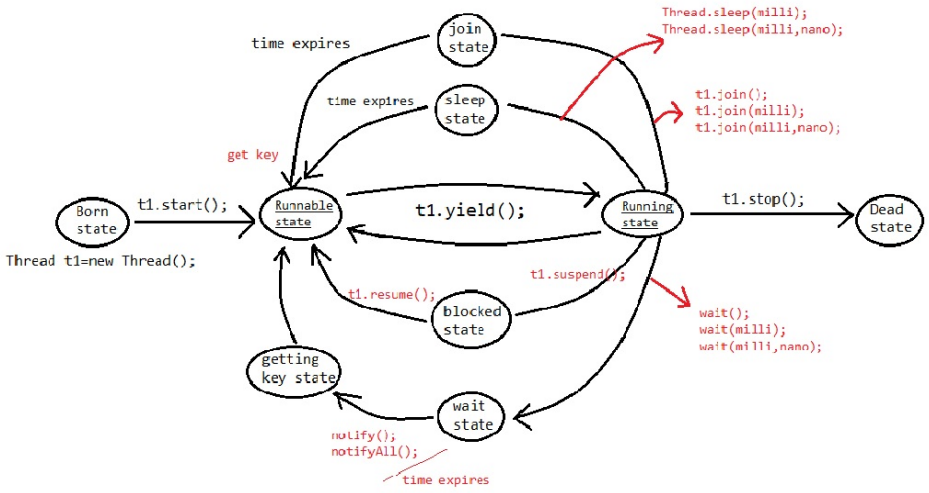
▼ Ans

- Generate key for thread based on the priority of the thread.
- Thread Schedular is part of JVM based on the priority Thread Schedular will schedule the runnable state Threads to running state or execution state.

29. Explain thread life cycle ?

▼ Ans

▼ Diagram



- Each Thread has key
- When we create Thread object Thread is going to born state.

▼ start() method

- When we call t1.start(); method, Thread goes to runnable state. in runnable state may have many Threads according to their priority they are guided by Thread Scheduler. The maximum priority Thread will execute first (1 to 10). (min is 0) and (max is 10).

    - We cannot make multiple Threads at running state because java is by default it supports only simultaneous multitasking.
    - We can make multiple Threads at running state in parallel multitasking. this is possible in Python language.
- Join state, Sleep state, Blocked state are not release the key of particular Thread.
- wait() —> It release the key.

▼ sleep() method

- While running Thread if we call the Thread.sleep(milli) or Thread.sleep(milli , nano), Thread goes to the sleep state for specified time then after remaining code will execute i.e. Thread again comes to runnable state.

▼ yield() method

- When we call t1.yield(); method the Thread is immediately come to runnable state then max priority Thread will execute.

▼ join() method

- While running a thread , if we call join() method currently running Thread will comes to join state. when a Thread is in join state it will not allows to start any new Thread until the invoking Thread dies or until the specified time expires. after that Thread comes to runnable state.

▼ What is the meaning of join state ?

- It will not allows to start any new Thread until the specified time expires or until current Thread dies.

▼ suspend() method

- When we call suspend() method the Thread goes to blocked state, until when release by resume() method then comes to runnable state.

▼ wait() method

- When we call wait(), wait(milli) or wait(milli , nano) method, it goes to wait state until we call notify() method. In wait state key is released after notify() method called and it goes to getting key state and it comes to runnable state after got key from Thread Scheduler.

▼ stop() method

- When we t1.stop() method the Thread is goes to dead state.

▼ Thread has 5 states

1. Born / New state
2. Runnable state
3. Blocked / Non-Runnable state
4. Running state
5. Dead state

- When we create Thread object, it is in born state.
- When we call start() method, it is in runnable state.
- Based on the priority and de-queue order Thread schedular is going to move the runnable state Thread to running state.
- Only one Thread can possible to present in running state. only one thread can execute at a time in Java. multiple Threads cannot execute in Java because is simultaneous process
- More than one Thread can possible to present in runnable state.
- when one Thread is in blocked state another Thread is going Execute.

30. What is Daemon thread ?

▼ Ans

- Daemon thread means if a Thread which executes before termination.
- If we call the Daemon thread that has the least priority value.
- They are created by inheriting the Daemon states of of its parents.
- They are mostly used in garbage collection.

31. How many ways program control will return to JVM ?

▼ Ans

1. By raising the Exception.

2. By Ending the main method.

Note:

- Once control will come back to JVM it terminates the program.
- Once the Exception is raised the program control will go back to JVM, to avoid that we should catch that exception by using try and catch block