1. The first task is what is a thread and what is a process can you to differentiate between a thread and a process  
  
Answer:-

A process is an independent program that is running on your computer. It has its own memory space, system resources, and runs separately from other processes. For example, if you open Google Chrome and Microsoft Word, they are two separate processes. Each process is isolated, so if one crashes, it usually doesn’t affect the other. On the other hand, a thread is a smaller unit of a process. A process can contain multiple threads that share the same memory and resources. For example, in Chrome, each tab or extension might run as a separate thread within the same Chrome process. Threads are more lightweight and faster to create compared to processes. They are useful for doing multiple tasks at the same time inside the same program, like downloading a file while allowing the user to scroll through a page. However, since threads share the same memory, they need careful handling to avoid issues like data conflicts. In short, processes are independent and heavy, while threads are lightweight and run within a process to perform multitasking.

2. What is Thread pool?

Answer:-

A Thread Pool is a group of pre-created worker threads that are kept ready to perform tasks when needed. Instead of creating a new thread every time a task comes in (which takes time and memory), the thread pool reuses existing threads. This makes programs faster and more efficient, especially when there are many short tasks to handle. When a task is submitted, it is placed in a queue, and an available thread from the pool picks it up and executes it. Once the task is done, the thread returns to the pool to wait for the next task. This avoids the overhead of constantly creating and destroying threads. Thread pools are widely used in applications like servers, background jobs, and real-time systems where multiple tasks need to be handled smoothly and quickly. In Java, the ExecutorService and Executors classes are commonly used to create and manage thread pools.

**Task 3.**

class RunnableDemo implements Runnable {

private Thread t;

private String threadName;

RunnableDemo( String name){

threadName = name;

System.out.println("Creating " + threadName );

}

public void run() {

System.out.println("Running " + threadName );

try {

for(int i = 4; i > 0; i--) {

System.out.println("Thread: " + threadName + ", " + i);

// Let the thread sleep for a while.

Thread.sleep(50);

}

} catch (InterruptedException e) {

System.out.println("Thread " + threadName + " interrupted.");

}

System.out.println("Thread " + threadName + " exiting.");

}

public void start ()

{

System.out.println("Starting " + threadName );

if (t == null)

{

t = new Thread (this, threadName);

t.start ();

}

}

}

public class TestThread {

public static void main(String args[]) {

RunnableDemo R1 = new RunnableDemo( "Thread-1");

R1.start();

RunnableDemo R2 = new RunnableDemo( "Thread-2");

R2.start();

}

}

Output:-  
Creating Thread-1

Starting Thread-1

Creating Thread-2

Starting Thread-2

Running Thread-1

Thread: Thread-1, 4

Running Thread-2

Thread: Thread-2, 4

Thread: Thread-2, 3

Thread: Thread-1, 3

Thread: Thread-2, 2

Thread: Thread-1, 2

Thread: Thread-2, 1

Thread: Thread-1, 1

Thread Thread-2 exiting.

Thread Thread-1 exiting.

**Task 4**In the above code … try extending Thread class… and observe the output..

Answer:- **package** samplePackage2;

**class** RunnableDemo **extends** Thread {

**private** String threadName;

RunnableDemo(String name) {

threadName = name;

System.***out***.println("Creating " + threadName);

}

**public** **void** run() {

System.***out***.println("Running " + threadName);

**try** {

**for** (**int** i = 4; i > 0; i--) {

System.***out***.println("Thread: " + threadName + ", " + i);

Thread.*sleep*(50);

}

} **catch** (InterruptedException e) {

System.***out***.println("Thread " + threadName + " interrupted.");

}

System.***out***.println("Thread " + threadName + " exiting.");

}

}

**public** **class** TestThread {

**public** **static** **void** main(String args[]) {

RunnableDemo R1 = **new** RunnableDemo("Thread-1");

R1.start(); // No need for custom start() method

RunnableDemo R2 = **new** RunnableDemo("Thread-2");

R2.start();

}

}

Output:- Creating Thread-1

Creating Thread-2

Running Thread-1

Running Thread-2

Thread: Thread-2, 4

Thread: Thread-1, 4

Thread: Thread-2, 3

Thread: Thread-1, 3

Thread: Thread-2, 2

Thread: Thread-1, 2

Thread: Thread-2, 1

Thread: Thread-1, 1

Thread Thread-2 exiting.

Thread Thread-1 exiting.

**Task 5**class Counter {

    private int count = 0;

    public void increment() {

        count++;

    }

    public int getCount() {

        return count;

    }

}

class ThreadDemo extends Thread {

    Counter counter;

    ThreadDemo(Counter counter) {

        this.counter = counter;

    }

    public void run() {

        for (int i = 0; i < 10; i++) {

            counter.increment();

        }

    }

}

public class Main {

    public static void main(String[] args) {

        Counter counter = new Counter();

        ThreadDemo t1 = new ThreadDemo(counter);

        ThreadDemo t2 = new ThreadDemo(counter);

        t1.start();

        t2.start();

        try {

            t1.join();

            t2.join();

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

        System.out.println("Final count: " + counter.getCount());

    }

}

output:-20  
  
**Task 6**  
  
class Resource {  
    synchronized void method1(Resource r) {  
        System.out.println(Thread.currentThread().getName() + " is executing method1");  
        try { Thread.sleep(100); } catch (InterruptedException e) {}  
        r.method2(this);  
    }  
  
    synchronized void method2(Resource r) {  
        System.out.println(Thread.currentThread().getName() + " is executing method2");  
        try { Thread.sleep(100); } catch (InterruptedException e) {}  
        r.method1(this);  
    }  
}  
  
public class DeadlockExample {  
    public static void main(String[] args) {  
        final Resource r1 = new Resource();  
        final Resource r2 = new Resource();  
  
        Thread t1 = new Thread(() -> r1.method1(r2), "Thread-1");  
        Thread t2 = new Thread(() -> r2.method1(r1), "Thread-2");  
  
        t1.start();  
        t2.start();  
    }  
}  
output:-  
Thread-1 is executing method1

Thread-2 is executing method1

**Task 7**class SharedResource {  
    private boolean ready = false;  
  
    synchronized void produce() {  
        try {  
            while (ready) {  
                wait();  
            }  
            System.out.println("Producing...");  
            ready = true;  
            notify();  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
  
    synchronized void consume() {  
        try {  
            while (!ready) {  
                wait();  
            }  
            System.out.println("Consuming...");  
            ready = false;  
            notify();  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
}  
  
public class InterThreadCommunicationExample {  
    public static void main(String[] args) {  
        SharedResource resource = new SharedResource();  
  
        Thread producer = new Thread(resource::produce);  
        Thread consumer = new Thread(resource::consume);  
  
        producer.start();  
        consumer.start();  
    }  
}

Output:- Producing...

Consuming...  
  
  
**Task 8**class InterruptibleThread extends Thread {  
    public void run() {  
        try {  
            while (!Thread.currentThread().isInterrupted()) {  
                System.out.println("Thread is running");  
                Thread.sleep(100);  
            }  
        } catch (InterruptedException e) {  
            System.out.println("Thread was interrupted");  
        }  
    }  
}  
  
public class InterruptExample {  
    public static void main(String[] args) {  
        InterruptibleThread thread = new InterruptibleThread();  
        thread.start();  
  
        try {  
            Thread.sleep(500);  
            thread.interrupt();  
        } catch (InterruptedException e) {  
            e.printStackTrace();  
        }  
    }  
}

Output:- Thread is running

Thread is running

Thread is running

Thread is running

Thread is running

Thread was interrupted   
  
  
**Task 9  
package** samplePackage2;

**import** java.io.File;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**public** **class** WriteByte {

**public** **static** **void** main(String args[]) {

File f1 = **new** File("FileName01.txt"); // Create new file

FileOutputStream outfile = **null**;

**byte** Text[] = {'I', ' ', 'L', 'O', 'V', 'E', ' ', 'I', 'N', 'D', 'I', 'A'};

**try** {

outfile = **new** FileOutputStream(f1);

outfile.write(Text);

} **catch** (IOException e) {

System.***out***.println(e);

System.*exit*(-1);

} **finally** {

**try** {

**if** (outfile != **null**)

outfile.close(); // Always close the file

} **catch** (IOException e) {

System.***out***.println("Error closing file: " + e);

}

}

System.***out***.println("Write Byte");

System.***out***.println("Thank You...!!!");

}

}

Output:- Write Byte

Thank You...!!!   
  
  
**Task 10**   
  
**package** samplePackage2;

**import** java.io.\*;

**public** **class** ReadingByte {

**public** **static** **void** main(String args[]) {

FileInputStream infile = **null**;

**int** b;

**try** {

infile = **new** FileInputStream("FileName01.txt");

**while** ((b = infile.read()) != -1) {

System.***out***.print((**char**) b); // Use print() to print in same line

}

infile.close();

} **catch** (IOException e) {

System.***out***.println("Sorry..!! Error occurred while reading the file...!!!");

}

}

}

Output:- I LOVE INDIA

**Task 11  
package** samplePackage2;

**import** java.io.\*;

**public** **class** CharacterWrite {

**public** **static** **void** main(String[] args) {

File f1 = **new** File("FileName03.txt");

FileWriter fw = **null**;

**try** {

fw = **new** FileWriter(f1);

fw.write("ahmedabad \n");

fw.write("baroda \n");

fw.close();

} **catch** (FileNotFoundException e) {

System.***out***.println("Sorry..!! File Not Found...!!!");

} **catch** (IOException e) {

System.***out***.println(e.getMessage());

}

System.***out***.println("Write operation done!!");

}

}

Output:- Write operation done!!

**Task 12  
package** samplePackage2;

**import** java.io.\*;

**public** **class** Readchar {

**public** **static** **void** main(String[] args) {

FileReader fr = **null**;

**try** {

fr = **new** FileReader("FileName03.txt");

**int** ch;

**while** ((ch = fr.read()) != -1) {

System.***out***.print((**char**) ch);

}

System.***out***.println("\nReading complete");

fr.close();

} **catch** (FileNotFoundException e) {

System.***out***.println("Sorry..!! File Not Found...!!!");

} **catch** (IOException e) {

System.***out***.println(e.getMessage());

}

}

}

Output:- ahmedabad

baroda

Reading complete

**Task 13  
package** samplePackage2;

**import** java.io.\*;

**class** CopyByte {

**public** **static** **void** main(String[] args) {

**try** {

**int** byteread; // To store byte data from file

FileInputStream infile = **new** FileInputStream("NewFile04.txt");

FileOutputStream outfile = **new** FileOutputStream("NewFile05.txt");

**while** ((byteread = infile.read()) != -1) {

outfile.write(byteread);

}

System.***out***.println("Byte copied from NewFile04.txt to NewFile05.txt");

infile.close();

outfile.close();

} **catch** (FileNotFoundException e) {

System.***out***.println("Sorry..!! File Not Found...!!!");

} **catch** (IOException e) {

System.***out***.println(e.getMessage());

}

}

}

Output:- Sorry..!! File Not Found...!!!

**Task 14**import java.io.\*;

class FileMergeDemo {

public static void main(String args[]) {

try {

FileInputStream file1 = new FileInputStream("File1.txt");

FileInputStream file2 = new FileInputStream("File2.txt");

// Merge file1 and file2 input streams

SequenceInputStream file3 = new SequenceInputStream(file1, file2);

// Wrap the merged stream in BufferedInputStream

BufferedInputStream br1 = new BufferedInputStream(file3);

// Output will be written to the console

BufferedOutputStream br2 = new BufferedOutputStream(System.out);

int ch;

while ((ch = br1.read()) != -1) {

br2.write((char) ch);

}

// Always close streams to avoid memory leaks

br1.close();

br2.close();

file1.close();

file2.close();

System.out.println("\nMerge Two Files Successfully");

} catch (IOException e) {

System.out.println("Sorry..!! File Not Found...!!!");

}

}

}  
output:- Hello World

Merge Two Files Successfully

**Task 15**import java.io.\*;

class FileRenameDemo {

public static void main(String args[]) {

if (args.length != 2) {

System.out.println("Usage: java FileRenameDemo <source\_file> <target\_file>");

return;

}

File f1 = new File(args[0]); // Original file

File f2 = new File(args[1]); // New file name

if (f1.renameTo(f2)) {

System.out.println("Renamed file from " + f1.getName() + " to " + f2.getName() + " successfully.");

} else {

System.out.println("Failed to rename the file.");

}

}

}output:- Usage: java FileRenameDemo <source\_file> <target\_file>

**Task 16  
package** samplePackage2;

**import** java.io.\*;

**public** **class** ReadFileExample {

**public** **static** **void** main(String[] args) {

// Print the current directory so user knows where to place the file

System.***out***.println("Looking for file in directory: " + System.*getProperty*("user.dir"));

// You can change the filename or give a full path if needed

String fileName = "largefile.txt";

**try** (BufferedReader br = **new** BufferedReader(**new** FileReader(fileName))) {

String line;

System.***out***.println("Reading file content:\n");

**while** ((line = br.readLine()) != **null**) {

System.***out***.println(line);

}

} **catch** (FileNotFoundException e) {

System.***out***.println("An error occurred while reading the file.");

System.***out***.println("File not found: " + e.getMessage());

} **catch** (IOException e) {

System.***out***.println("An I/O error occurred: " + e.getMessage());

}

}

}

Output:-  
Looking for file in directory: C:\Users\DELL\eclipse-workspace\SBIproject

Reading file content:

Hello, this is a test file.

Reading from Java works!

Hello srinivas

**Task 17  
package** samplePackage2;

**import** java.io.FileReader;

**import** java.io.IOException;

**public** **class** NonBufferedReaderExample {

**public** **static** **void** main(String[] args) {

**try** (FileReader fr = **new** FileReader("largefile.txt")) {

**int** ch;

**while** ((ch = fr.read()) != -1) {

System.***out***.print((**char**) ch);

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

Output:- Hello, this is a test file.

Reading from Java works!

Hello srinivas  
  
  
**Task 18   
package** samplePackage2;

**import** java.io.BufferedWriter;

**import** java.io.FileWriter;

**import** java.io.IOException;

**public** **class** BufferedWriterDemo {

**public** **static** **void** main(String[] args) {

String filePath = "example.txt";

String content = "Hello, World!\nThis is a BufferedWriter example.";

// Initialize BufferedWriter with a FileWriter

**try** (BufferedWriter writer = **new** BufferedWriter(**new** FileWriter(filePath))) {

// Write content to the file

writer.write(content);

System.***out***.println("Content written to file.");

} **catch** (IOException e) {

System.***err***.println("An error occurred: " + e.getMessage());

}

}

}

Output:- Content written to file.