**:- Threads : -**

1. **What is the basic structure of creating and running a thread in Java?**

class MyThread extends Thread {

@Override

public void run() {

System.out.println("Thread is running...");

}

public static void main(String[] args) {

MyThread thread = new MyThread();

thread.start(); // Start the thread

}

}

1. **What is the difference between extending Thread and implementing Runnable interface?**

class MyRunnable implements Runnable {

@Override

public void run() {

System.out.println("Thread is running using Runnable.");

}

public static void main(String[] args) {

MyRunnable runnable = new MyRunnable();

Thread thread = new Thread(runnable);

thread.start();

}

}

1. **How do you run multiple threads in parallel?**

class MyRunnable implements Runnable {

private String message;

MyRunnable(String message) {

this.message = message;

}

@Override

public void run() {

System.out.println(message);

}

1. **What is the purpose of Thread.sleep()? How does it work?**

class MyRunnable implements Runnable {

@Override

public void run() {

try {

System.out.println("Thread is going to sleep for 2 seconds.");

Thread.sleep(2000); // Sleep for 2 seconds

System.out.println("Thread woke up.");

} catch (InterruptedException e) {

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

}

}

public static void main(String[] args) {

Thread thread = new Thread(new MyRunnable());

thread.start();

}

}

public static void main(String[] args) {

Thread thread1 = new Thread(new MyRunnable("Thread 1 is running"));

Thread thread2 = new Thread(new MyRunnable("Thread 2 is running"));

thread1.start();

thread2.start();

}

}

1. **How do you set the priority of a thread in Java?**

class MyRunnable implements Runnable {

@Override

public void run() {

System.out.println(Thread.currentThread().getName() + " is running.");

}

public static void main(String[] args) {

Thread thread1 = new Thread(new MyRunnable());

Thread thread2 = new Thread(new MyRunnable());

thread1.setPriority(Thread.MIN\_PRIORITY);

thread2.setPriority(Thread.MAX\_PRIORITY);

thread1.start();

thread2.start();

}

}

1. **What is a daemon thread, and how do you create one?**

class MyRunnable implements Runnable {

@Override

public void run() {

while (true) {

System.out.println(Thread.currentThread().getName() + " is running.");

}

}

public static void main(String[] args) {

Thread thread = new Thread(new MyRunnable());

thread.setDaemon(true); // Set thread as daemon

thread.start();

System.out.println("Main thread is finished.");

}

}

1. **What is thread synchronization, and why is it important?**

class Counter {

private int count = 0;

public synchronized void increment() {

count++;

}

public synchronized int getCount() {

return count;

}

}

class MyRunnable implements Runnable {

private Counter counter;

MyRunnable(Counter counter) {

this.counter = counter;

}

@Override

public void run() {

for (int i = 0; i < 1000; i++) {

counter.increment();

}

}

public static void main(String[] args) {

Counter counter = new Counter();

Thread thread1 = new Thread(new MyRunnable(counter));

Thread thread2 = new Thread(new MyRunnable(counter));

thread1.start();

thread2.start();

}

}

1. **How do you make one thread wait for another thread to complete?**

class MyRunnable implements Runnable {

@Override

public void run() {

try {

Thread.sleep(2000); // Simulate work

System.out.println(Thread.currentThread().getName() + " has finished work.");

} catch (InterruptedException e) {

e.printStackTrace();

}

}

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

Thread thread1 = new Thread(new MyRunnable());

Thread thread2 = new Thread(new MyRunnable());

thread1.start();

thread2.start();

thread1.join(); // Main thread will wait for thread1 to finish

thread2.join(); // Main thread will wait for thread2 to finish

System.out.println("Both threads have finished.");

}

}

1. Producer-Consumer Problem (Using Wait and Notify)

class SharedResource {

private int data = 0;

public synchronized void produce() throws InterruptedException {

while (data != 0) {

wait(); // Wait if data is already produced

}

data = 1;

System.out.println("Produced data");

notify(); // Notify consumer

}

public synchronized void consume() throws InterruptedException {

while (data == 0) {

wait(); // Wait if no data is produced

}

data = 0;

System.out.println("Consumed data");

notify(); // Notify producer

}

}

class Producer implements Runnable {

private SharedResource resource;

Producer(SharedResource resource) {

this.resource = resource;

}

@Override

public void run() {

try {

while (true) {

resource.produce();

Thread.sleep(1000);

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

class Consumer implements Runnable {

private SharedResource resource;

Consumer(SharedResource resource) {

this.resource = resource;

}

@Override

public void run() {

try {

while (true) {

resource.consume();

Thread.sleep(1000);

}

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public class ProducerConsumer {

public static void main(String[] args) {

SharedResource resource = new SharedResource();

Thread producer = new Thread(new Producer(resource));

Thread consumer = new Thread(new Consumer(resource));

producer.start();

consumer.start();

}

}

1. **What is ThreadLocal in Java, and how does it work?**

class MyRunnable implements Runnable {

private ThreadLocal<Integer> threadLocal = ThreadLocal.withInitial(() -> 0);

@Override

public void run() {

threadLocal.set(threadLocal.get() + 1);

System.out.println(Thread.currentThread().getName() + " value: " + threadLocal.get());

}

public static void main(String[] args) {

MyRunnable runnable = new MyRunnable();

Thread thread1 = new Thread(runnable);

Thread thread2 = new Thread(runnable);

thread1.start();

thread2.start();

**-: Exceptions :-**

**1.Handling ArithmeticException**

public class ArithmeticExceptionExample {

public static void main(String[] args) {

try {

int result = 10 / 0; // This will throw ArithmeticException

} catch (ArithmeticException e) {

System.out.println("Error: Cannot divide by zero.");

} finally {

System.out.println("Execution completed.");

}

}

}

**2. Handling NullPointerException**

public class NullPointerExceptionExample {

public static void main(String[] args) {

String str = null;

try {

System.out.println(str.length()); // This will throw NullPointerException

} catch (NullPointerException e) {

System.out.println("Error: Null object reference.");

} finally {

System.out.println("End of program.");

}

}

}

**3.finally block in Java exception handling**

public void readFile() {

FileReader reader = null;

try {

reader = new FileReader("file.txt");

// Read the file

} catch (IOException e) {

System.out.println("An error occurred.");

} finally {

if (reader != null) {

try {

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

} catch (IOException e) {

System.out.println("Error closing file.");

}

}

}

}

**4. Throwable class in Java, and how is it related to Exception and Error.**

public class Example {

public static void main(String[] args) {

try {

throw new Exception("A checked exception");

} catch (Exception e) {

System.out.println("Caught: " + e.getMessage());

}

try {

throw new Error("A serious error");

} catch (Error e) {

System.out.println("Caught error: " + e.getMessage());

}

}

}

**5. ArrayIndexOutOfBoundsException**

public class ArrayIndexOutOfBoundsExceptionExample {

public static void main(String[] args) {

int[] numbers = {1, 2, 3, 4};

try {

System.out.println(numbers[5]); // This will throw ArrayIndexOutOfBoundsException

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Error: Index is out of bounds.");

}

}

}