### **Creating a Thread using Runnable Interface**

### **Explanation**: A more flexible and commonly used approach is implementing the Runnable interface. This approach allows you to create a thread by separating the task (business logic) from the thread itself, making the code more reusable. It’s useful when the class needs to extend another class, as Java does not support multiple inheritance.

### **Example Code**

### ****class MyRunnable implements Runnable {

### public void run() {

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

### }

### }

### 

### public class TestRunnable {

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

### MyRunnable myRunnable = new MyRunnable();

### Thread t1 = new Thread(myRunnable);

### t1.start(); // Starts the thread, calling the run() method from Runnable

### }

### }

###  Output

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



### **Advantages of Runnable Interface**:

### Enables a class to extend another class, as you can implement Runnable and still inherit from a parent class.

### Promotes better object-oriented design by separating the task (logic) from the thread behaviour.

### 

### 

### 

### **Important Thread Class Methods**

### **sleep() - Taking a Break**

### **Real World Analogy:** Like telling a worker to take a short break.

### **Explanation**: The sleep() method pauses the execution of the current thread for a specified period of time (in milliseconds).

### Example:

### class SleepExample extends Thread {

### public void run() {

### for (int i = 1; i <= 5; i++) {

### try {

### Thread.sleep(1000); // Sleeps for 1 second

### } catch (InterruptedException e) {

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

### }

### System.out.println(i);

### }

### }

### }

### public class TestSleep {

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

### SleepExample t1 = new SleepExample();

### t1.start(); // Thread sleeps for 1 second between each iteration

### }

### }

### 

### 

### **join() - Waiting for Others**

### **Real World Analogy:** Like waiting for one worker to finish before starting another task.

### **Explanation:** The join() method waits for a thread to finish its execution before allowing the current thread to continue. It ensures that the calling thread completes its work before proceeding.

### 

### 

### Example

### ****class JoinExample extends Thread {

### public void run() {

### for (int i = 1; i <= 5; i++) {

### try {

### Thread.sleep(500);

### } catch (InterruptedException e) {

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

### }

### System.out.println(Thread.currentThread().getName() + " - " + i);

### }

### }

### }

### 

### public class TestJoin {

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

### JoinExample t1 = new JoinExample();

### JoinExample t2 = new JoinExample();

### 

### t1.start();

### try {

### t1.join(); // Main thread waits until t1 finishes

### } catch (InterruptedException e) {

### e.printStackTrace();

### }

### 

### t2.start(); // t2 starts only after t1 has completed

### }

### }

### 

### **interrupt() Method - Stopping a Task**

### Think of interrupt() like a school bell that signals break time:

### It doesn't force students to stop immediately

### It just tells them they should stop what they're doing

### Students can choose how to handle the interruption

### 

### 

### **Real-World Analogy:**

### Like when your mom calls you for dinner while you're playing a video game:

### You might want to save your game first

### Then properly exit the game

### Finally respond to the dinner call

### **Explanation**: The interrupt() method signals a thread that it should stop what it’s doing and handle the interruption, especially if it's sleeping or waiting. It doesn’t stop the thread immediately, but it raises an InterruptedException that the thread can handle.

### Example

### class InterruptExample extends Thread {

### public void run() {

### try {

### Thread.sleep(10000); // Sleep for 10 seconds

### } catch (InterruptedException e) {

### System.out.println("Thread interrupted: " + e);

### }

### }

### }

### 

### public class TestInterrupt {

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

### InterruptExample t1 = new InterruptExample();

### t1.start();

### t1.interrupt(); // Interrupts the thread during sleep

### }

### }

### 

### **Simple Example: Interrupting a long task**

### ****class LongTask extends Thread {

### public void run() {

### try {

### System.out.println("Starting a long task...");

### // Simulate long work with sleep

### Thread.sleep(10000); // 10 seconds

### System.out.println("Task completed!");

### } catch (InterruptedException e) {

### System.out.println("Sorry, I was interrupted!");

### System.out.println("Cleaning up before stopping...");

### }

### }

### }

### 

### // Using it:

### public class InterruptExample {

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

### LongTask task = new LongTask();

### task.start();

### 

### // Wait for 3 seconds then interrupt

### try {

### Thread.sleep(3000);

### } catch (InterruptedException e) {}

### 

### task.interrupt();

### }

### }

### ****

### **Bigger Example: Download Manager**

### ****class FileDownloader extends Thread {

### private boolean isDownloading = true;

### private String fileName;

### 

### public FileDownloader(String fileName) {

### this.fileName = fileName;

### }

### 

### public void run() {

### int progress = 0;

### while (isDownloading && progress < 100) {

### try {

### System.out.println("Downloading " + fileName + ": " + progress + "%");

### Thread.sleep(1000); // Simulate download time

### progress += 10;

### 

### if (Thread.interrupted()) {

### System.out.println("\nDownload interrupted!");

### System.out.println("Saving partial download...");

### isDownloading = false;

### return;

### }

### } catch (InterruptedException e) {

### System.out.println("\nDownload interrupted!");

### System.out.println("Saving partial download...");

### isDownloading = false;

### return;

### }

### }

### System.out.println("\nDownload complete!");

### }

### }

### 

### **Using the download manager:**

### **public class DownloadManager {**

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

### **FileDownloader downloader = new FileDownloader("movie.mp4");**

### **downloader.start();**

### 

### **// Simulate user cancelling download after 5 seconds**

### **try {**

### **Thread.sleep(5000);**

### **} catch (InterruptedException e) {}**

### 

### **downloader.interrupt();**

### **}**

### **}**

### ****

### 

### **yield() Method - Being Polite**

### Think of yield() like being polite in a queue:

### When someone needs to go first, you let them pass

### But they might say "no, you go ahead"

### It's just a suggestion, not a rule

### 

### **Real-World Example: Walking in a Corridor**

### Like when you're walking in a corridor and see someone in a hurry:

### You step aside to let them pass

### But they might still wait for you

### It's just being courteous

### **Explanation:** The yield() method pauses the currently executing thread to give a chance for other threads of the same or higher priority to execute. However, there’s no guarantee that it will actually yield to other threads, as it depends on the thread scheduler.

### While yield() suggests that the current thread should give way to others, it’s entirely up to the thread scheduler to decide whether other threads will execute or the current one will continue.

### **Simple Example: Two polite threads**

### class PoliteThread extends Thread {

### private String name;

### 

### public PoliteThread(String name) {

### this.name = name;

### }

### 

### public void run() {

### for (int i = 1; i <= 5; i++) {

### System.out.println(name + " counting: " + i);

### Thread.yield(); // Being polite, letting others run

### }

### }

### }

### // Using it:

### public class YieldExample {

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

### PoliteThread thread1 = new PoliteThread("Thread-1");

### PoliteThread thread2 = new PoliteThread("Thread-2");

### thread1.start();

### thread2.start();

### }

### }

### 

### 

### **Bigger Example: Printer Queue**

### class PrintJob extends Thread {

### private String document;

### private int pages;

### 

### public PrintJob(String document, int pages) {

### this.document = document;

### this.pages = pages;

### }

### 

### public void run() {

### for (int page = 1; page <= pages; page++) {

### System.out.println("Printing " + document + " page " + page);

### 

### // If it's a long document, let others print too

### if (pages > 5) {

### System.out.println(document + " yielding printer...");

### Thread.yield();

### }

### 

### // Simulate printing time

### try {

### Thread.sleep(1000);

### } catch (InterruptedException e) {

### System.out.println("Printing interrupted!");

### return;

### }

### }

### System.out.println(document + " finished printing!");

### }

### }

### 

### // Using the printer queue:

### public class PrinterQueue {

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

### PrintJob longDoc = new PrintJob("Book", 8);

### PrintJob shortDoc = new PrintJob("Letter", 2);

### 

### longDoc.start();

### shortDoc.start();

### }

### }

