**Threads in Java OOP**

**What are Threads?** Threads in Java are small units of a process that can run concurrently (at the same time). They allow multiple tasks to execute simultaneously, enabling efficient use of CPU resources.

A real-life analogy is **a multi-tasking chef in a kitchen**. Imagine a chef preparing soup, baking a cake, and chopping vegetables at the same time. Each task can be seen as a thread.

**Example 1: Basic Thread Example**

class Task extends Thread {

public void run() {

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

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

try {

Thread.sleep(1000); // Simulates a delay (1 second)

} catch (InterruptedException e) {

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

}

}

}

}

public class ThreadExample {

public static void main(String[] args) {

Task task1 = new Task();

Task task2 = new Task();

task1.start(); // Starts the first thread

task2.start(); // Starts the second thread

}

}

**Output (Threads run concurrently):**

Thread-0 is running: 1

Thread-1 is running: 1

Thread-0 is running: 2

Thread-1 is running: 2

...

Here, each thread (task1 and task2) executes its run() method independently.

**Real-life Example**

* **Video Games:** Threads are used for rendering graphics, playing background music, and handling user input simultaneously.
* **Web Browsers:** Threads load content, manage UI interactions, and execute JavaScript scripts concurrently.

**Example 2: Runnable Interface**

Instead of extending Thread, you can implement the Runnable interface.

class Task implements Runnable {

@Override

public void run() {

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

System.out.println(Thread.currentThread().getName() + " is executing: " + i);

}

}

}

public class RunnableExample {

public static void main(String[] args) {

Thread thread1 = new Thread(new Task());

Thread thread2 = new Thread(new Task());

thread1.start();

thread2.start();

}

}

This approach allows your class to extend other classes since Java does not support multiple inheritance.

**Streams in Java**

**What are Streams?** Streams in Java are a way to process sequences of data (like a stream of water). They allow you to perform operations such as filtering, mapping, and reducing data in a functional style.

A real-life analogy is **an assembly line in a factory**, where items (data) pass through various stages (operations) such as sorting, filtering, and transforming.

**Example 1: Filtering a List**

Suppose you have a list of integers and want to filter only even numbers.

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class StreamExample {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9);

List<Integer> evenNumbers = numbers.stream()

.filter(n -> n % 2 == 0) // Keep only even numbers

.collect(Collectors.toList());

System.out.println("Even numbers: " + evenNumbers);

}

}

**Output:**

Even numbers: [2, 4, 6, 8]

**Example 2: Mapping Values**

Suppose you want to find the squares of all numbers in a list.

public class MappingExample {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

List<Integer> squares = numbers.stream()

.map(n -> n \* n) // Transform each number into its square

.collect(Collectors.toList());

System.out.println("Squares: " + squares);

}

}

**Output:**

Squares: [1, 4, 9, 16, 25]

**Real-life Example**

* **Data Processing Pipelines:** A company might use streams to process customer orders, filter high-value purchases, and generate reports.
* **Stock Market Applications:** Streams can analyze and filter live stock data for trends or alerts.

**Threads and Streams Combined**

Threads and streams can be combined for parallel data processing.

import java.util.Arrays;

import java.util.List;

public class ParallelStreamExample {

public static void main(String[] args) {

List<String> names = Arrays.asList("Alice", "Bob", "Charlie", "David", "Eve");

names.parallelStream()

.forEach(name -> System.out.println(Thread.currentThread().getName() + " processed " + name));

}

}

**Output (Order may vary due to parallel processing):**

ForkJoinPool.commonPool-worker-3 processed Alice

ForkJoinPool.commonPool-worker-1 processed Bob

...

**Comparison Between Threads and Streams**

| **Feature** | **Threads** | **Streams** |
| --- | --- | --- |
| Purpose | Concurrent task execution | Data processing |
| Implementation | Extend Thread or implement Runnable | Use functional pipelines |
| Real-life analogy | Multi-tasking chef | Factory assembly line |
| Use case example | Running tasks like downloading files | Processing large datasets |

**Where These Concepts Are Applied**

1. **Web Applications:**
   * Threads handle multiple user requests simultaneously.
   * Streams process large datasets, such as filtering search results.
2. **Machine Learning Pipelines:**
   * Threads train models in parallel.
   * Streams process input data.
3. **IoT Systems:**
   * Threads manage sensor data collection and device communication.
   * Streams analyze real-time data from sensors.