CONCURRENCY AND MULTITHREADING

OBJECT ORIENTED PROGRAMMING I

Sercan Külcü | Object Oriented Programming I | 10.01.2023

Contents

Introduction

Concurrency and multithreading are essential concepts in modern programming languages. In today's computing world, where processors are faster and computers have multiple cores, it is important to understand how to design and implement programs that can run efficiently in a concurrent environment. In this chapter, we will explore the concept of concurrency and multithreading in the Java programming language.

Concurrency

Concurrency is the ability of a computer system to perform multiple tasks simultaneously. It allows multiple tasks to execute concurrently, making efficient use of available resources. In Java, concurrency is implemented through threads, which are lightweight units of execution within a program.

Multithreading

Multithreading is the ability of a program to create and manage multiple threads simultaneously. A program with multiple threads can perform multiple tasks concurrently, leading to faster execution times and better resource utilization. In Java, multithreading is implemented through the java.lang.Thread class.

Creating Threads

In Java, there are two ways to create threads: by extending the Thread class or by implementing the Runnable interface. The Thread class provides the basic functionality for creating and managing threads. To create a new thread by extending the Thread class, you need to create a new class that extends Thread and overrides the run() method. The run() method contains the code that is executed when the thread starts.

Implementing the Runnable interface is an alternative way to create threads in Java. This method is preferred over extending the Thread class because it allows for better separation of concerns. To create a new thread by implementing the Runnable interface, you need to create a new class that implements the Runnable interface and overrides the run() method.

Synchronization

In a multithreaded environment, it is important to ensure that multiple threads do not access the same shared resource simultaneously. This can cause unpredictable behavior and lead to race conditions. In Java, synchronization is achieved through the use of the synchronized keyword.

The synchronized keyword can be applied to a block of code or a method. When a thread enters a synchronized block, it acquires a lock on the object that the block is synchronized on. This lock prevents other threads from executing the synchronized block until the lock is released.

Deadlocks

Deadlocks occur when two or more threads are blocked waiting for each other to release a lock. This can happen when two or more threads are trying to acquire locks on the same set of resources in a different order. To avoid deadlocks, it is important to ensure that threads always acquire locks on resources in the same order.

Thread Safety

Thread safety is the property of a program that ensures correct behavior in a multithreaded environment. Thread safety can be achieved through proper synchronization of shared resources and the use of immutable objects. Immutable objects are objects that cannot be modified once they are created. This makes them safe to share between threads without the need for synchronization.

Conclusion

Concurrency and multithreading are important concepts in modern programming languages. In Java, concurrency is implemented through threads, which are lightweight units of execution within a program. Multithreading allows for better resource utilization and faster execution times. Synchronization is used to ensure that multiple threads do not access the same shared resource simultaneously. Deadlocks occur when two or more threads are blocked waiting for each other to release a lock. Thread safety can be achieved through proper synchronization of shared resources and the use of immutable objects.