**Saint Petersburg Electrotechnical University LETI**

**Faculty of Computer Science and Technology, Department of Computer Science and Engineering**

**Master of Computer Science and Knowledge discovery**

**Parallel Computing**

HOME WORK 3 TEORICAL & PRACTICAL

Professor : Natalya V. Razmochaeva

Student : Azzazene Mohammed Elmehdi

# Concurrency

Concurrency is the ability of different parts or units of a program, algorithm, or problem to be executed out-of-order or in partial order, without affecting the final outcome. This allows for parallel execution of the concurrent units, which can significantly improve overall speed of the execution in multi-processor and multi-core systems. In more technical terms, concurrency refers to the decomposability property of a program, algorithm, or problem into order-independent or partially-ordered components or units.

# Processes and Threads

**Processes** are what actually execute the program. Each process is able to run concurrent subtasks called threads.

**Threads** are sub-tasks of processes and if synchronized correctly can give the illusion that your application is performing everything at once. Without threads you would have to write one program per task, run them as processes and synchronize them through the operating system.

# Thread Objects

This class represents an activity that is run in a separate thread of control. There are two ways to specify the activity: by passing a callable object to the constructor, or by overriding the run() method in a subclass.

No other methods (except for the constructor) should be overridden in a subclass.

In other words, only override the \_\_init\_\_() and run() methods of this class.

# Defining and Starting a Thread

To create and start an instantiation of thread object we should provide a code which we want to be executed by the thread. Generally speaking there are two ways to do that:

* Runnable object
* Thread subclass

# Pausing Execution with Sleep

There is a possibility to stop or suspend a thread in multithreading in JAVA programming language. The Sleep function is able to do such an operation in concurrent programming with a parameter for the sleeping time (In Milli seconds usually).

# Interrupts

The interrupt() method of thread class is used to interrupt the thread. If any thread is in sleeping or waiting state, then using the interrupt() method, we can interrupt the thread execution by throwing InterruptedException.

# Joins

It will put the current thread on wait until the thread on which it is called is dead. If thread is interrupted then it will throw InterruptedException.

# Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive

* Synchronized method.
* Synchronized block.
* static synchronization.

1. Cooperation (Inter-thread communication in java)

# Thread Interference

When multiple threads share the same memory, there is a chance that two or more different threads performing different operations on the same data interleave with each other and create inconsistent data in the memory. Threads interleave when they are performing operations with multiple steps and their sequence of steps overlap. This is called thread interference.

# Memory Consistency Errors

In multithreading, there can be possibilities that the changes made by one thread might not be visible to the other threads and they all have inconsistent views of the same shared data. This is known as memory consistency error.

# Synchronized Methods

There are two different synchronization types in java programming language:

* synchronized methods:
  + To make a method synchronized we just add synchronized keywords to the method’s declaration easily.
  + Making a method synchronized has two effects in concurrent programming:
    - It brings the improbability of accessing one shared memory location or shared resource by two invocations at the same time.
    - when a synchronized method exits, it automatically establishes a happens-before relationship with any subsequent invocation of a synchronized method for the same object. This guarantees that changes to the state of the object are visible to all threads.
* synchronized statements

# Intrinsic Locks and Synchronization

Intrinsic lock or monitor lock plays an important role in different aspects of synchronization :

* It enforces exclusive access to the object's state .
* It establishes a Happens-before relationship that is essential for visibility .

# Reentrant Synchronization

Synchronized blocks in Java are reentrant. This means, that if a Java thread enters a synchronized block of code, and thereby takes the lock on the monitor object the block is synchronized on, the thread can enter other Java code blocks synchronized on the same monitor object.

# Atomic Access

In concurrent programming atomic operations means those kinds of operations which can be done completely or they don't happen at all and no middle action such as reading from or writing to a memory variable. And there is no side effect of doing the action until the end of the operation.

* Reads and writes are atomic for reference variables and for most primitive variables (all types except long and double).
* Reads and writes are atomic for all variables declared volatile (including long and double variables).

# Liveness

A liveness property asserts that program execution eventually reaches some desirable state. While termination has been studied extensively, many other liveness properties are important for concurrent programs: deadlock,starvation and livelock.

## Deadlock

Deadlocks happen when two or more threads aren’t able to make any progress because the resource required by the first thread is held by the second and the resource required by the second thread is held by the first.

## Starvation and Livelock

### Starvation

Other than a deadlock, an application thread can also experience starvation, where it never gets CPU time or access to shared resources because other “greedy” threads hog the resources.

### Livelock

A livelock happens when two threads keep taking actions in response to the other thread instead of making any progress. The best analogy is to think of two persons trying to cross each other in a hallway. John moves to the left to let Arun pass, and Arun moves to his right to let John pass. Both block each other now. John sees he’s now blocking Arun and moves to his right and Arun moves to his left seeing he’s blocking John. They never cross each other and keep blocking each other. This scenario is an example of a livelock.

# Guarded Blocks

Guarded block is a mechanism of coordinating the execution of multiple threads in a multithreaded environment. Guarded block keeps checking for a particular condition to become true and only in that case the actual execution of the thread resumes.

# Immutable Objects

An immutable object is an object whose state cannot be changed after it’s creation or construction (Constructor calling).

# High Level Concurrency Objects

## Lock Objects

If a thread wants to execute a synchronized method on the given object. First, it has to get the lock of that object. Once thread gets the lock then it is allowed to execute any synchronized method on that object. Once method execution completes automatically thread releases the lock.

## Executors

Making threads by Runnable Objects and Thread Objects are discussed previously and they work well for small applications. But for larger applications the thread making and management should be separated to have better efficiency which is implemented by encapsulation concept in executors.

### Thread Pools

In a multithreadedWhen the thread pool is created, it will either instantiate a certain number of threads to make available or create new ones as needed depending on the needs of the implementation. application, thread pool is a "pool of available threads" that can be used by your application.Thread pool is a collection of managed threads usually organized in a queue, which execute the tasks in the task queue.

### Fork/Join

fork/join framework helps you to get benefit from multiprocessing applications through the ExecutorService interface. It is useful if you want to break down the application into a number of pisces recursively. The first step of using the join fork framework is to write the part of the code which is going to be executed as a single segment of the whole application.

## Concurrent Collections

All the classes present in Concurrent Collections are synchronized in nature. Therefore In Concurrent classes, we don't have to take care about Thread-safety.

There are a number of Concurrent Collections provided by java in java.util.concurrent package.

* BlockingQueue
* ConcurrentMap
* ConcurrentNavigableMap
* ConcurrentSkipListMap

# Atomic Variables

They are a kind of classes provided by java to work with atomic operations on some variables like volatile variables and they have provided get and set like read and write.

# Concurrent Random Numbers

It is useful for concurrent applications which want to generate random numbers for multiple threads. And it has been implemented in java.util.concurrent package.

**Stepik**

# Task 1

ConcurrentHashMap

ConcurrentHashMap

# Task 2

setDaemon(true)

# Task 3

currentThread()

# Task 4

synchronized

synchronized

synchronized

# Task 5

shutdown()

# Task 6

join()

# Task 7

start()

# Task 8

ReadWriteLock

# Task 9

atomic.AtomicBoolean

AtomicBoolean

AtomicBoolean

# Task 10

try

finally

unlock

# Task 11

ReadWriteLock

ReentrantReadWriteLock

ReadWriteLock

ReentrantReadWriteLock

lock()

unlock()

lock()

unlock()

lock()

unlock()

# Task 12

Runtime.getRuntime().availableProcessors()

# Task 13

CyclicBarrier

CyclicBarrier

CyclicBarrier

await()

# Task 14

CountDownLatch

CountDownLatch

CountDownLatch

countDown()

await()

# Task 15

isInterrupted()

interrupt()

# Task 17

isInterrupted()

interrupt()

# Task 19

CountDownLatch

# Task 21

Number of available cores / BC

# Task 22

Condition

Object wait/notify/notifyAll

# Task 23

Number of available cores