**Prep MAP Theoretical Exam**

1. **Basic concepts**

* Checked vs. unchecked exceptions
  + Checked Exception
    - checked at compile time
    - forcing the programmer to handle them explicitly
  + Unchecked Exception
    - checked at runtime
    - do not require explicit handling at compile time
* Reference type vs primitive type
  + Primitive type
    - stores directly the value
    - stored on the stack
    - passed by value
  + Reference type
    - serve as containers for memory addresses or references to objects rather than holding the actual object data
    - stored on the heap
    - passed by reference

1. **Classes**

* Overriding vs overloading
  + Method overriding
    - A derived class can re-implement methods of the base class
    - But preserving the signature from the base class
  + Method Overloading
    - A class may contain multiple methods with the same name
    - But with different signature
* Static vs non static fields/methods
  + Static
    - Belong to the class, and not to a specific instance of the class
    - Static methods can only access static members of the class
    - Static methods called using the class name
    - Static methods cannot be overridden because of early binding
  + Non-static
    - Belong to a specific instance of the class, on which they are called
    - Non-static methods can access both static and non-static members of the class
* What is abstract class?
  + A class which may contain abstract methods (= methods which are declared, but not implemented)
  + It cannot be instantiated
  + If a class contains at least one abstract method then that class must be abstract
* Interface vs class
  + No method implementation
  + No constructors
  + All declared methods are implicitly public.
  + It may not contain any method declaration.
  + It may contain fields which by default are public, static and final
* interfaces vs. abstract classes

A white rectangular chart with black text

Description automatically generated

1. **Generics**

* Complete the given code with generic code for it to work
  + (Seminar examples)
* Generic array in java
  + Why can’t you create it directly?
    - because Java arrays contain information about its component type. So you must know the component type when you create the array.
  + How to create it?
    - use T[] elem=(T[])new Object[dim];
    - (which will produce a warning at compile-time)
* Wildcard
  + We use ? to denote any type (or unknown type)
  + Upper bounded: can have any type that extends from the bound
  + Lower bounded: can have any type that is a superclass of the bound

1. **I/O**

* What is a buffer?
  + Block of memory into which you can write data, which you can then later read again
* Buffer operations?
  + Write data
  + Flip the buffer from writing mode into reading mode
  + Read data
  + Clear buffer to make buffer ready for writing again

1. **Functional programming**

* Stream
  + A stream provides an interface to a sequenced set of values of a specific element type. However, streams don’t store elements, they are computed on demand.
  + Streams consume data from a source such as:
    - Collections
    - arrays
    - I/O resources.
  + Streams support SQL-like operations and common operations from functional programing languages,
  + such as filter, map, reduce, find, match, sorted, and so on.

1. **Synchronization**

* threads vs processes
  + Each process has its self-contained execution environment and its own memory space, whereas threads inside a process share the same memory
  + Threads exist within a process, each process has at least one thread
  + Threads are lightweight processes
  + Communication between threads does not interact with the OS, whereas communication between processes is done through the OS using IPC mechanisms such as pipes and sockets
* What is multithreading
  + Multi-threading is the process of having multiple flows of control when a program is executed.
* Possible states of a thread
  + NEW – between creation and start
  + RUNNABLE – after start, when running or ready
  + BLOCKED – blocked by
    - * + monitor lock
        + a synchronized method/block
  + WAITING – waiting to be notified (no time out set)

wait()

join()

* + TIMED\_WAITING – waiting to be notified (time out set):

sleep(time)

wait(time)

join(time)

* + TERMINATED – execution completed / after stop()
* Synchronization mechanisms:
  1. ExecutorService
     + asynchronous execution mechanism which is capable of executing tasks in the background
     + ThreadPoolExecutor – implementation of ExecutorService
  2. Blocking Queue
     + BlockingQueue is an interface in Java that represents a queue that supports blocking operations for adding and retrieving items.
     + If the queue is empty, the extracting thread is blocked until an item is present. If the queue is full, the inserting thread is blocked until space is available.
  3. Concurrent Collections
     + It only locks the part of the collection that is being written to, internally.
  4. Semaphore
     + The counting semaphore is initialized with a given number of "permits".
     + For each call to acquire() a permit is taken by the calling thread.
     + For each call to release() a permit is returned to the semaphore.
     + Thus, at most N threads can pass the acquire() method without any release() calls, where N is the number of permits the semaphore was initialized with.
  5. ForkJoinPool
     + implements the work-stealing strategy = every time a running thread has to wait for some result, the thread removes the current task from the work queue and executes some other task ready to run.
     + This way the current thread is not blocked and can be used to execute other tasks.
     + Once the result for the originally suspended task has been computed, the task gets executed again and the join() method returns the result.
  6. CountDownLatch
     + Is initialized with a given count.
     + This count is decremented by calls to the countDown() method.
     + Threads waiting for this count to reach zero can call one of the await() methods.
     + Calling await() blocks the thread until the count reaches zero.
  7. CyclicBarrier
     + is a synchronization mechanism that can synchronize threads progressing through some algorithm.
     + it is a barrier that all threads must wait at, until all threads reach it, before any of the threads can continue.
     + The threads wait for each other by calling the await() method on the CyclicBarrier.
     + Once N threads are waiting at the CyclicBarrier, all threads are released and can continue running.
  8. Lock
     + It is a thread synchronization mechanism just like synchronized blocks, but it is more flexible and more sophisticated than a synchronized block.
  9. Read-write lock:
     + allows multiple threads to read a certain resource, but only one to write it, at a time
     + Read Lock: If no threads have locked the ReadWriteLock for writing, and no thread have requested a write lock. Thus, multiple threads can lock the lock for reading.
     + Write Lock: If no threads are reading or writing. Thus, only one thread at a time can lock the lock for writing
  10. Atomic Variables
      + the atomic classes make heavy use of compare-and-swap (CAS), an atomic instruction directly supported by most modern CPUs.

1. **Other**

* What is a monitor model?
  + A monitor is a collection of code (called the critical section) associated with an object (called the lock). At any moment, at most one thread can have its execution point located in the critical section associated with the lock (mutual exclusion).
* What is event driven programming?
  + Event-driven programming is a programming paradigm in which the execution flow of the program is determined by external events or by programmatically generated events.
  + Events sources generate events, which are treated by event listeners, by executing a program module.