Concurrent Programming Elements

Posted on October 11, 2019

Concurrent programming is defined as when more than one process collaborates to accomplish a common objective. Methods can also be entitled as ‘tasks’ or ‘threads.’ Numerous torrents of actions are implemented in concurrent programming. It does not require multiprocessors. In this programming, several sequential programs, which are called ‘ordinary’ programs, work together to accomplish their goal.

The necessities for a Concurrent program can be defined in standings of characteristics it should retain. If the features of the concurrent program are articulated correctly, which means mathematically, then it can probably validate formally that the execution has all these characteristics. Many elements of the concurrent program are categorized as either for safety purposes.

Single-Threaded Programs

Every torrent of actions is executed as a continuous program, excluding the fact that this stream can always interconnect and hinder each other. Such an arrangement of commands is known as the thread. This is the reason that these programs are mostly known as Single-Threaded Programs.

Multi-Threaded Programs:

Several threads are enclosed in a volatile run, which is mostly subjected to the restrictions obligatory by unobvious coordination of operations that may be implanted in the code; this happens when a Multi-Threaded Program accomplishes.

Runnable Thread:

There is a Runnable Thread which executes an exceptional operation necessitating coordination that delays until a specific or a particular condition takes place if there are two or more threads that are runnable, if not all but one of the threads may make no progress because none of its operations is being finished.

Java Thread:

While performing a program, a path is followed, which is called a Java thread. Every program in Java has at least one thread. Nowadays, most programs run as a lone thread, which apparently creates complications when more than one event needs to take place at the same time. This can be resolved

by the smooth implementation of more than one fragment of a program at the same time. Threads have no problem permitting us to do this thing. (Goetz et al., 2006)

Java offers a built-in provision for multi-threaded programming.

Class Thread:

It delivers methods and constructors to establish and implement operations on a thread. This class prolongs to the Object class and executes a runnable and interactive interface.

The most used constructors of the thread class are:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r, String name)

Our class cannot be preserved as a thread object if we do not prolong the class thread. Therefore, it is necessary to establish an overtly thread class object.

Java Exception

One of the most effective and influential appliances to grip the runtime errors so that the usual stream of presentations can be sustained is called the Java exception.

The exception is an occasion or a signal in Java that disturbs the usual stream of the program. It is an entity that occurs at runtime. Exceptions must be detected and sorted out so that the program can progress if at all possible.

For Example,

We have 15 statements in our program, and an exception shows up at statement number 8; at this point, the rest of the code cannot be implemented, i.e., statements 7 to 15 will not be able to run. Now, if we perform exception handling, the rest of the code will be executed successfully. Therefore, this is the reason for using exception handling in Java.

If we want to implement method overriding with exception handling. The following rules are performed:

If The Superclass Method Cannot Pronounce An Exception

If the superclass method is not declaring an exception, the subclass overridden method can also not proclaim the checked exception, but it can report the un-tested exception.

If The Superclass Method Pronounces An Exception

If the superclass method is declaring an exception, the subclass overridden method can also do the same, with no exception or subclass exception, but it will not pronounce the parent exception.

Try Catch Structure In Stop Watch GUI

If an exception takes place, the program immediately jumps to the catch structure, skipping the rest of the code in the try block.

Try is mainly used to watch out for a chunk of code in which there is a possibility of occurrence of an exception. This chunk of code is called a guarded region. A catch statement assists in the declaration of the kind of exception we are trying to catch. Now, if the exception appears in a protected code, following the try is checked. If the catch block has enlisted the type of exception performed, it is further supplied to the catch block, which ultimately grips it. (Herlihy & Shavit, 2012)

In a situation where all of the exceptions are in a similar class series, then we should catch the base exception type. A single catch block deals with two or more exceptions. If we are supposed to catch each exception, it should be done individually in their catch blocks.

Multiple catch blocks flow a try block. After a solo catch block, we can have many catch blocks. The exception is delivered to the first catch enlisted if it occurs in the code, which is guarded. The exception is wedged if it matches the first catch block, and if not, then it is delivered to the next catch block. This cycle continues until and unless the exception is wedged or drops overall wedges.

The Stop Watch GUI needed a catching structure because the exception took place during the execution of the program.

References

Goetz, B., Peierls, T., Bloch, J., Bowbeer, J., Holmes, D., & Lea, D. (2006). Java concurrency in practice. Addison-Wesley. https://jcip.net/

Herlihy, M., & Shavit, N. (2012). The art of multiprocessor programming (Rev. 1st ed.). Morgan Kaufmann. https://www.elsevier.com/books/the-art-of-multiprocessor-programming/herlihy/978-0-12-397337-5