**Advanced Java** refers to a set of concepts and features in the Java programming language that go beyond the basics covered in introductory Java courses. Here are some key components of Advanced Java:

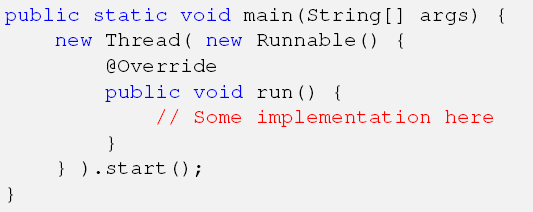
|  |  |
| --- | --- |
| Concurrency  Multithreading  Virtual Thread  Reflections  Dynamic Languages Support | **Java Enterprise Edition (Java EE)**  **Servlets**  **JSP**  **Spring Framework**  **Spring Boot**  **Java Persistence API (JPA)**  **Rest Services** |

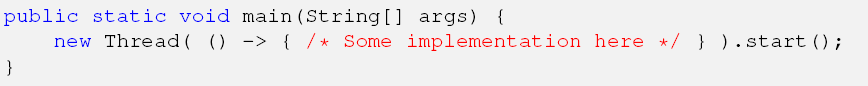
**Threads and Thread Groups**

Threads are the foundational building blocks of concurrent applications in Java.

Threads are sometimes called lightweight processes and they allow multiple execution flows to proceed concurrently.

It is not recommended to directly create and manage threads using the instances of Thread class





**There are two kinds of threads:**

* Platform threads
* Virtual threads

### **Platform Thread**

A *platform thread* is implemented as a thin wrapper around an operating system (OS) thread.

A platform thread runs Java code on its underlying OS thread, and the platform thread captures its OS thread for the platform thread's entire lifetime.

Consequently, the number of available platform threads is limited to the number of OS threads.

#### The Problem with Async Programming : writing asynchronous code is more complex than sequential. Debugging these applications becomes hard

#### Expensive Creation of Threads: [platform](https://theboreddev.com/understanding-java-virtual-threads/) threads is that they are heavy objects which are expensive to create

#### Expensive Context Switching: When there is a context switch, an OS thread switches from one [platform](https://theboreddev.com/understanding-java-virtual-threads/) thread to another

### **Virtual Thread**

Virtual Threads are a lightweight alternative to traditional Java Platform Threads.

Not tied to a specific OS thread, providing more flexibility.

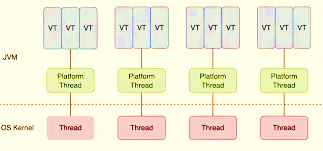
Run code on an OS thread but can be suspended during blocking I/O operations.

Enable the efficient mapping of numerous virtual threads to a limited number of OS threads.

#### Cheap Context Switching: being under the control of the [JVM](https://theboreddev.com/understanding-java-virtual-threads/), the thread stack is stored in the heap memory and not in the stack

#### Scheduling: The traditional [platform](https://theboreddev.com/understanding-java-virtual-threads/) threads are scheduled by the [operating system](https://theboreddev.com/understanding-java-virtual-threads/), whereas **virtual threads are scheduled by the JDK runtime**. The JDK scheduler does not use time slices, in this case is **the virtual thread itself who will yield** and renounce to its carrier thread when it’s waiting for a blocking operation response

It’s crucial to note that Virtual Threads are not designed for long-running CPU-intensive operations.



class VirtualThread

class VirtualPlatformThreadComparison

**Concurrency, Synchronization and Immutability**

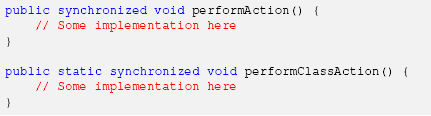
In mostly every Java application, multiple running threads need to communicate with each other and access shared data.

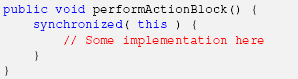
Reading this data is not so much of a problem, however uncoordinated modification of it is a straight road to disaster (so called racing conditions).

That is the point where synchronization kicks in.

Synchronization is a mechanism to ensure that several concurrently running threads will not execute the specifically guarded (synchronized) block of application code at the same time.

The synchronization is Java is re-entrant: it means that the thread can acquire a lock, which it already owns.





**Immutable objects** do not need the synchronization as they are never being updated by more than one threads.

**Futures, Executors and Thread Pools**

Creating new threads in Java is easy, but managing them is really tough.

Java standard library provides extremely useful abstractions in the form of executors and thread pools targeted to simplify threads management.

Essentially, in its simplest implementation, thread pool creates and maintains a list of threads, ready to be used right away.

ExecutorService executor = Executors.newFixedThreadPool( 10 );

class FireAndForgot

class WaitForResponse

**Reflection**

Reflection is the ability of the program to examine or introspect itself at runtime.

Reflection is an extremely useful and powerful feature, which significantly expands the capabilities of the program to perform its own inspections, modifications or transformations during its execution, without a single line of code change.

class ExploreMethods

class ExploreFields

class ExecuteString

class AccessPrivate

**Dynamic languages support**

Since Java 7, the JVM has a direct support of modern dynamic (also often called scripting) languages and the Java 8 release delivered even more enhancements into this space.

Javascript

Ruby

Python

Groovy