Theory: Threads as objects

7 minutes 0 / 4 problems solved

§1. Threads in Java

Java was originally designed with built-in multithreading support. Threads are supported at the level of JVM, at the level of language by special keywords and at the level of the standard library. Every Java program has at least one thread, that is called **main**, created automatically by the JVM process to execute statements inside the `main` method. Any Java program has some other default threads as well (for example, a separate thread for the garbage collector).

Throughout all stages of development of the Java language, the approach to multithreading has changed from the using of low-level threads to some high-level abstractions. However, understanding the fundamental base remains very important for a good developer.

Current topic:

Threads as objects

Topic depends on:

✓ Objects Stage 3

✗ Concurrency and parallelism

✗ Processes and threads

Topic is required for:

Custom threads

§2. A class for threads

Each thread is represented by an object that is an instance of the `java.lang.Thread` class (or its subclass). This class has a static method named `currentThread` to obtain a reference to the currently executing thread object:

```
Thread thread = Thread.currentThread(); // the current thread
```

Any thread has a name, an identifier (`long`), a priority, and some other characteristics that can be obtained through its methods.

Table of contents:

↑ Threads as objects

§1. Threads in Java

§2. A class for threads

§3. The information about the main thread

Feedback & Comments

§3. The information about the main thread

The example below demonstrates how to obtain the characteristics of the **main** thread by making reference to it through an object of the `Thread` class.

```
public class MainThreadDemo {
    public static void main(String[] args) {
        Thread t = Thread.currentThread(); // main thread

        System.out.println("Name: " + t.getName());
        System.out.println("ID: " + t.getId());
        System.out.println("Alive: " + t.isAlive());
        System.out.println("Priority: " + t.getPriority());
        System.out.println("Daemon: " + t.isDaemon());



        t.setName("my-thread");

        System.out.println("New name: " + t.getName());

    }

}
```

All statements in this program are executed by the **main** thread.

The invocation `t.isAlive()` returns whether the thread has been started and hasn't died yet. Every thread has a **priority**, and the `getPriority()` method returns the priority of a given thread. Threads with a higher priority are executed in preference to threads with lower priorities. The invocation `t.isDaemon()` checks whether the thread is a **daemon**. A daemon thread (comes from UNIX terminology) is a low priority thread that runs in the background to perform tasks such as garbage collection and so on. JVM does not wait for daemon threads before exiting while it waits for non-daemon threads.

The output of the program will look like:

```
Name: main
ID: 1
Alive: true
Priority: 5
Daemon: false
New name: my-thread
```

The same code can be applied to any current thread, not just **main**.

Report a typo

234 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Comments (2) Hints (0) Useful links (0) Show discussion