Threads as objects

Theory | Practice | 25% completed, 0 problems solved

Theory

5 minutes reading

Skip this topic | Start practicing

§1. Threads in Java

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

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

§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:

```
1  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.

§3. The information about the main thread

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

```
1  public class MainThreadDemo {
2      public static void main(String[] args) {
3          Thread t = Thread.currentThread(); // main thread
4
5          System.out.println("Name: " + t.getName());
6          System.out.println("ID: " + t.getId());
7          System.out.println("Alive: " + t.isAlive());
8          System.out.println("Priority: " + t.getPriority());
9          System.out.println("Daemon: " + t.isDaemon());
1
0
1
1          t.setName("my-thread");
1
2          System.out.println("New name: " + t.getName());
1
3      }
1
4  }
```

2 required topics

- Objects
- Processes and threads

1 dependent topic

- Custom threads

Table of contents:

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 (which 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 whereas it waits for non-daemon threads.

The output of the program will look like this:

```
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

446 users liked this piece of theory. **4** didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing Skip this topic

Comments (5) Useful links (0) Show discussion