

Chapter 19 Multithreading

1. Multithreading can make your program more responsive and interactive, and enhance the performance. Multithreading is needed in many situations, such as animation and client/server computing. Because most of time the CPU is idle--for example, the CPU is doing nothing while the user enters data--it is practical for multiple threads to share the CPU time in single-processor systems.
2. In Java, you can create thread classes by implementing the `Runnable` interface or by extending the `Thread` class. Because it is easier and simpler to use the `Thread` class, it is recommended that you use the `Thread` class unless your class has multiple inheritance. The `Thread` class itself implements the `Runnable` interface. When using the `Runnable` interface, you have to create a thread using `new Thread(this)`. When extending the `Thread` class, you simply create an instance of the user thread class.
3. You use `Thread t = new Thread(this)` to create a thread and use `t.start()` to start the thread. If you replace the `start()` method by the `run()` method in Lines 11-13 in Example 19.1, the `run()` method are executed in sequence. The threads are not executed concurrently.
4. An illegal `java.lang.IllegalThreadStateException` may be thrown because you just started thread and thread might have not yet finished before you start it again.
5. All of the methods shown except `yield()`, `sleep(long)`, and `currentThread()` are instance methods. `sleep()` and `join()` may throw an `InterruptedException`. The methods `stop()`, `suspend()` and `resume()` are deprecated in JDK 1.2.
6. Yes. These methods are defined in the `Object` class and they are used for thread communication.
7. See the section "Controlling Threads and Thread States."

You use the `setPriority()` method to set the priority for a thread. The default priority of the thread is `Thread.NORM_PRIORITY` (5).
8. A thread group is a set of threads. Some programs contain quite a few threads with similar functionality. It is convenient to group them together and perform operations on all the threads in the group.

See the section "Thread Groups" on how to create a thread group and add threads into the group. You can control the threads in the group collectively or individually.

9. You need to start each thread individually and use the `activeCount()` method to count the number of active threads in a group.
10. See the section "Synchronization" for examples and solutions.
11. No. `synchronized (this)` acquires a lock on a thread (an instance of `AddAPennyThread`). Each thread still can access the object `bank` concurrently. To fix the problem, acquire the lock on `bank` using `synchronized (bank)`.
12. `object1` and `object2` are not the same. A lock must be acquired on the receiving object of the `wait()`, `notify()`, and `notifyAll()` methods before invoking these methods.
13. When a thread notify a waiting thread, you cannot assume the `balance >= amount` for the waiting thread. The condition (`balance < amount`) may be still true when the thread is awakened.
14. To override the `init()` method defined in the `Applet` class, you have to use the exact signature. The `init()` method does not claim throwing exceptions.
15. *Deadlock* occurs in the case that two or more threads acquire locks on multiple objects and each has the lock on one object and is waiting for the lock on the other object. The *resource ordering technique* can be used to avoid deadlock.
16. No, because both `timer.sleep()` and `Thread.sleep()` invoke the same `sleep()` method.
17. Create a thread in the `init()` method, start the thread in the `init()` method or in the `start()` method. Suspend the thread in the `stop()` method. Resume the thread in the `start()` method. Destroy the thread by assigning null to the thread variable in the `destroy()` method.
18. The `stringPainted` property. `setOrientation` method