Threads in Java

J. Pöial

Multi-threaded programs

Threads - processes **inside** one Java program (lightweight processes):

- parallel
- related to each other (share the same address space)
- cooperate using synchronization tools

Critical section – part of a program that cannot be interrupted by other thread

In Java

- Class **Thread** (methods **start**, **run**, **interrupt**, **join**, **yield**, **sleep**, **isAlive**, **setPriority**, **getPriority**, **setDaemon**, ...)
- Interface **Runnable** (method **run**)
- Methods in class **Object** – **wait**, ..., **notify**, **notifyAll**
- **synchronized** blocks and methods
- priority system and timesharing
- Class **Runtime** (method **exit**)
- Class **ThreadGroup**

Thread states

- NEW - a thread that has not yet started is in this state.
- RUNNABLE - a thread executing in the Java virtual machine is in this state.
- BLOCKED - a thread that is blocked waiting for a monitor lock is in this state.
- WAITING – a thread that is waiting indefinitely for another thread to perform a particular action is in this state.
- TIMED_WAITING - a thread that is waiting for another thread to perform an action for up to a specified waiting time is in this state.
- TERMINATED - a thread that has exited is in this state.

Life cycle of the thread

- Create

  Process executed in thread is given as public void run() method that does not throw exceptions

  – create a subclass of Thread and override the run() method

  – use interface Runnable to create a new thread:

  `Thread t = new Thread (runnableObject);`

  run() method of the **`runnableObject`** is used

- Start

  `t.start();` // run() is executed

Life cycle of the thread

- Thread runs until it:
  - sleeps (sleep or wait is called, suspend and resume are now considered deprecated)
  - waits for a lock to enter some critical section
  - blocks for I/O (e.g. accept or read is called)
  - gives up its time slot (yield)
  - is preempted by higher priority thread
  - is switched to another thread with the same priority in case of time-slicing scheme
  - terminates

Life cycle of the thread

- Stop

  normal end – return from the **`run`** method

  deprecated **`stop`** method is called

  uncaught runtime exception occurs
- Sleep – go idle for given time (ms)
- Interrupt - wake up (**`InterruptedException`** is thrown); method `interrupt()` may not work correctly

Synchronization

- join – sleep until target thread finishes
- critical section in Java – synchronized block/method. Object lock (monitor)
- wait, notify, notifyAll
  - wait – sleep and open the lock until notify is sent to this object (from some other thread)
  - notifyAll – wakes all waiting threads, the winner starts execute, others go back sleeping

Priorities

- Each thread is given an integer priority, programmer can use constants **`Thread.NORM_PRIORITY`**, **`Thread.MAX_PRIORITY`**, **`Thread.MIN_PRIORITY`**
- Priority of a new thread is inherited from the thread that created it
- Threads with lower priority always wait for a thread with higher priority to terminate (whenever higher priority thread becomes runnable it preempts others)

Priorities

- Threads with the same priority:
  - round robin – priority and synchronization rules apply but "parallel" execution caused by time sharing is not guaranteed
  - time slicing – parallel or pseudo-parallel execution (all threads get chance)

  Thread might give up its time slot using the yield method

Priorities

- When a thread terminates:
  - higher priority thread always takes over
  - if there are no higher priority threads switch to the "next" runnable thread in this priority class
  - if there are no runnable threads of current priority switch to the next lower priority class and choose a runnable thread

Priorities

- Usually increasing the priority of an "important" thread does NOT help in performance. Consider decreasing priorities of some other threads instead: **`getPriority`** and **`setPriority`** are the methods to use:

```
Thread bgTask = new Thread (runnable_object);
int prio = Thread.NORM_PRIORITY -
  (Thread.NORM_PRIORITY-Thread.MIN_PRIORITY)/2;
bgTask.setPriority (prio);
bgTask.start();
```

Server threads and thread groups

- Server threads are "weak" – JVM terminates if all remaining threads are daemon threads

  Methods: setDaemon, isDaemon
- ThreadGroup is a set of threads (actually a tree because elements can be groups). Several operations can be applied to the whole group

Example 1

Producer-consumer task. We have a limited buffer, writer threads (producers) and reader threads (consumers).

Rules:

- cannot read from the empty buffer – wait
- cannot write when the buffer is full – wait
- cannot change the buffer during reading - lock
- cannot change the buffer during writing - lock

Example 2

Applet clock