Threads in Java

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming Concepts using Java

Week 11

Creating threads in Java

- Have a class extend `Thread`
- Define a function `run()` where execution can begin in parallel
- Invoking `p[i].start()` initiates `p[i].run()` in a separate thread
  - Directly calling `p[i].run()` does not execute in separate thread!
- `sleep(t)` suspends thread for `t` milliseconds
  - Static function — use `Thread.sleep()` if current class does not extend `Thread`
  - Throws `InterruptedException` — later

```
public class Parallel extends Thread{
  private int id;
  public Parallel(int i){ id = i; }
  public void run(){
    for (int j = 0; j < 100; j++){
      System.out.println("My id is "+id);
      try{
        sleep(1000);        // Sleep for 1000 ms
      }
      catch(InterruptedException e){}
    }
  }
}

public class TestParallel {
  public static void main(String[] args){
    Parallel p[] = new Parallel[5];
    for (int i = 0; i < 5; i++){
       p[i] = new Parallel(i);
       p[i].start();  // Start p[i].run()
    }                 // in concurrent thread
  }
}
```

Creating threads in Java

- Have a class extend `Thread`
- Define a function `run()` where execution can begin in parallel
- Invoking `p[i].start()` initiates `p[i].run()` in a separate thread
  - Directly calling `p[i].run()` does not execute in separate thread!
- `sleep(t)` suspends thread for `t` milliseconds
  - Static function — use `Thread.sleep()` if current class does not extend `Thread`
  - Throws `InterruptedException` — later

Typical output

```
My id is 0
My id is 3
My id is 2
My id is 1
My id is 4
My id is 0
My id is 2
My id is 3
My id is 4
My id is 1
My id is 0
My id is 3
My id is 1
My id is 2
My id is 4
My id is 0
...
```

Java threads . . .

- Cannot always extend Thread
  - Single inheritance
- Instead, implement Runnable
- To use Runnable class, explicitly create a Thread and start() it

```
public class Parallel implements Runnable{
  // only the line above has changed
  private int id;
  public Parallel(int i){ ... } // Constructor
  public void run(){ ... }

}

public class TestParallel {
  public static void main(String[] args){
    Parallel p[] = new Parallel[5];
    Thread t[]   = new Thread[5];

    for (int i = 0; i < 5; i++){
       p[i] = new Parallel(i);
       t[i] = new Thread(p[i]);
             // Make a thread t[i] from p[i]
       t[i].start();  // Start off p[i].run()
                      // Note: t[i].start(),
    }                 //   not p[i].start()
  }
}
```

Life cycle of a Java thread

A thread can be in six states

Life cycle of a Java thread

A thread can be in six states

- New: Created but not `start()`ed.

Life cycle of a Java thread

A thread can be in six states

- New: Created but not `start()`ed.
- Runnable: `start()`ed and ready to be scheduled.
  - Need not be actually "running"
  - No guarantee made about how scheduling is done
  - Most Java implementations use time-slicing

Life cycle of a Java thread

A thread can be in six states

- New: Created but not `start()`ed.
- Runnable: `start()`ed and ready to be scheduled.
    - Need not be actually "running"
    - No guarantee made about how scheduling is done
    - Most Java implementations use time-slicing
- Not available to run
    - Blocked — waiting for a lock, unblocked when lock is granted
    - Waiting — suspended by `wait()`, unblocked by `notify()` or `notfifyAll()`
    - Timed wait — within `sleep(..)`, released when sleep timer expires

Life cycle of a Java thread

A thread can be in six states

- New: Created but not `start()`ed.
- Runnable: `start()`ed and ready to be scheduled.
  - Need not be actually "running"
  - No guarantee made about how scheduling is done
  - Most Java implementations use time-slicing
- Not available to run
  - Blocked — waiting for a lock, unblocked when lock is granted
  - Waiting — suspended by `wait()`, unblocked by `notify()` or `notfifyAll()`
  - Timed wait — within `sleep(..)`, released when sleep timer expires
- Dead: thread terminates.

Life cycle of a Java thread

A thread can be in six states — thread status via `t.getState()`

- New: Created but not `start()`ed.
- Runnable: `start()`ed and ready to be scheduled.
  - Need not be actually “running”
  - No guarantee made about how scheduling is done
  - Most Java implementations use time-slicing
- Not available to run
  - Blocked — waiting for a lock, unblocked when lock is granted
  - Waiting — suspended by `wait()`, unblocked by `notify()` or `notfifyAll()`
  - Timed wait — within `sleep(..)`, released when sleep timer expires
- Dead: thread terminates.

Interrupts

- One thread can interrupt another using `interrupt()`
  - `p[i].interrupt();` interrupts thread `p[i]`

Interrupts

- One thread can interrupt another using `interrupt()`
  - `p[i].interrupt();` interrupts thread `p[i]`
- Raises `InterruptedException` within `wait()`, `sleep()`

Interrupts

- One thread can interrupt another using `interrupt()`
  - `p[i].interrupt();` interrupts thread `p[i]`
- Raises `InterruptedException` within `wait()`, `sleep()`
- No exception raised if thread is running!
  - `interrupt()` sets a status flag
  - `interrupted()` checks interrupt status and clears the flag
- Detecting an interrupt while running or waiting

```
public void run(){
  try{
    j = 0;
    while(!interrupted() && j < 100){
      System.out.println("My id is "+id);
      sleep(1000);    // Sleep for 1000 ms
      j++;
    }
  }
  catch(InterruptedException e){}
}
```

More about threads . . .

- Check a thread's interrupt status
  - Use `t.isInterrupted()` to check status of `t`'s interrupt flag
  - Does not clear flag

More about threads . . .

- Check a thread's interrupt status
  - Use `t.isInterrupted()` to check status of `t`'s interrupt flag
  - Does not clear flag
- Can give up running status
  - `yield()` gives up active state to another thread
  - Static method in `Thread`

More about threads . . .

- Check a thread's interrupt status
  - Use `t.isInterrupted()` to check status of `t`'s interrupt flag
  - Does not clear flag
- Can give up running status
  - `yield()` gives up active state to another thread
  - Static method in `Thread`
  - Normally, scheduling of threads is handled by OS — preemptive
  - Some mobile platforms use cooperative scheduling — thread loses control only if it yields

More about threads . . .

- Check a thread's interrupt status
  - Use `t.isInterrupted()` to check status of `t`'s interrupt flag
  - Does not clear flag
- Can give up running status
  - `yield()` gives up active state to another thread
  - Static method in `Thread`
  - Normally, scheduling of threads is handled by OS — preemptive
  - Some mobile platforms use cooperative scheduling — thread loses control only if it yields
- Waiting for other threads
  - `t.join()` waits for `t` to terminate

Summary

- To run in parallel, need to extend `Thread` or implement `Runnable`
  - When implmenting `Runnable`, first create a `Thread` from `Runnable` object
- `t.start()` invokes method `run()` in parallel
- Threads can become inactive for different reasons
  - Block waiting for a lock
  - Wait in internal queue for a condition to be notified
  - Wait for a sleep timer to elapse
- Threads can be interrupted
  - Be careful to check both `interrupted` status and handle `InterruptException`
- Can yield control, or wait for another thread to terminate