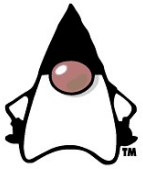
Software Foundations in JAVA (JSF)

Week 8

Ending a thread

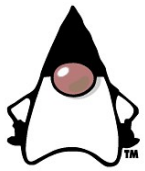
Peter Boots
p.boots@fontys.nl

Erik van der Schriek
e.vanderschriek@fontys.nl



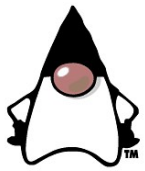
Ending a thread

- 3 ways to end a thread
  - We might use `Thread.stop()`
    - Stops thread right away, no matter what it is doing
    - Dangerous, therefore deprecated
    - Just don't use this
  - We might use a shared boolean
  - We might `interrupt` the thread



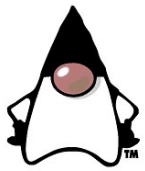
Using a shared boolean

- Assume shared boolean `stop`, initially `false`
  - In the thread, this variable is checked:
    -
```
while(!stop) {
    … // do some action
}
```
  - We can stop the thread by setting `stop` to `true`
    -
```
void StopThread() {
    stop = true;
}
```
- Thread will complete the action, and then stop
- OK, but not when the thread is waiting or sleeping somewhere in the `while` statement



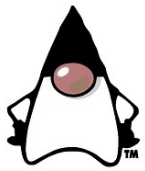
Interrupting a thread

- Send interrupt signal with `t.interrupt()`
- Every thread has an interrupt status
  - initially false (cleared)
- 2 methods to check the interrupt status
  - Static method `interrupted`
    - returns interrupt status of current thread
    - clears interrupt status
  - Non-static method `isInterrupted`
    - returns interrupt status (of any thread)
    - Does not change interrupt status



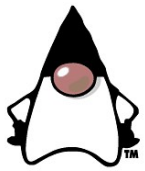
Interrupting a thread

- Effect of interrupt depends on what thread is doing:
  - If thread is blocked in call to `wait`, `join` or `sleep`:
    - interrupt status will be cleared and
    - `InterruptedException` will be thrown
  - Otherwise:
    - interrupt status will be set
- So `wait`, `join` and `sleep` can throw `InterruptedException` (must be catched)



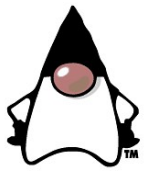
Interrupting a thread

- Suppose, thread executes this:
  - ```
    while(!interrupted()) {
      ......      // do some ‘action’
      c.await();  // wait for some condition
      ......      // do some ‘action’
      sleep(500); // sleep for 500 msec
    }
    ```
- 2 possibilities, when we interrupt this thread:
  - it might be executing the ‘action’ or
  - it might be in the `sleep` or `await`
    ```



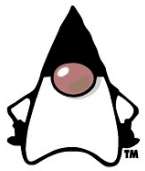
Interrupting a thread

- If the thread was executing the ‘action’ when the interrupt occurred, then:
  - interrupt status will be set,
  - thread will continue normally:
    - finish the ‘action’,
    - do the await and/or sleep, and then
    - end the `while`-statement
  - This is what we expect



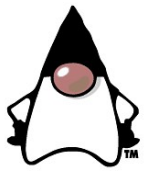
Interrupting a thread

- If the thread was sleeping or waiting when the interrupt occurred, then :
  - `sleep`/`await` ends immediately
    - InterruptedException is thrown
  - interrupt status will not be set
  - `while` statement will not end
  - This is unexpected!
- To solve last situation:
  - Handle `InterruptedException` correctly



Interrupting a thread

- Solution 1: let thread interrupt itself:
  -
```
while(!interrupted()) {
  try {
    ......      // do some action
    c.await();  // wait for some condition
    ......      // do some action
    sleep(500); // wait for 500 msec
  } catch (InterruptedException) {
    Thread.currentThread().interrupt();
  }
}
```
- Stop thread with
  - `t.interrupt();`



Interrupting a thread

- Solution 2: use shared boolean
  - ```
    while(!stop) {
      try {
        ......      // do some action
        c.await();  // wait for some condition
        ......      // do some action
        sleep(500); // wait for 500 msec
      } catch (InterruptedException) { }
    }
    ```
- Stop thread with
  - ```
    t.stop = true;
    t.interrupt();
    ```
- When interrupted on the dots (`.....`), the next `wait` or `sleep` will get `InterruptedException` immediately