Chapter 7
Cancellation and Shutdown

Alfred Theorin, Ph.D. student

Department of Automatic Control

Lund University

Outline

- Cancellation
  - Why and when it is needed
  - How it is accomplished
- Abnormal thread termination
- JVM shutdown

Cancellation: Why or When?

- User-requested
- Time-limited task
- Application event
- Error
- Shutdown

Cancellation: How?

- No safe way to just stop a thread
- Cooperative mechanism
- Cancellation policy
  - How to request cancellation
  - What the task does in response
- Interruption: *interrupted status* in `Thread`

Interruption is usually the most sensible way to implement cancellation.

Interruption

```
public class Thread {
  // Request interruption, set interrupted status
  public void interrupt() { ... }

  // Get interrupted status
  public boolean isInterrupted() { ... }

  // Clear interrupted status
  public static boolean interrupted() { ... }

  ...
}
```

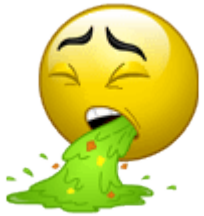
Interruption Example

```
public class MyThread extends Thread {
  public void run() {
    while (!isInterrupted()) {
      doWork();
    }
  }
  ...
}
```

```
MyThread t = new MyThread();
...
t.interrupt();
```

Blocking Operations

```
public class MyThread extends Thread {
  public void run() {
    while (!isInterrupted()) {
      doWork();
      sleepOneHour();
    }
  }
  ...
}
```



Not responsive to interruption.

Don't do this.

Interruptable Blocking Operations

Many blocking operations are interruptable. They terminate prematurely on interruption by clearing the interrupted status and throwing an `InterruptedException`

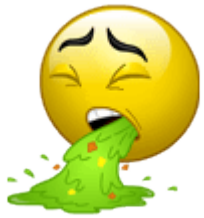
Interruptable Blocking Operations

```
public class MyThread extends Thread {
  public void run() {
    while (!isInterrupted()) {
      doWork();
      try {
        sleep(3600000);
      } catch (InterruptedException e) {
        interrupt();
      }
    }
  }
  ...
}
```

General Purpose Code

General purpose code should never swallow interruption requests.

```
public void sleepOneHour() {
   try {
     sleep(3600000);
   } catch (InterruptedException e) {
   }
}
```



Interruption request is lost. *Don't do this.*

General Purpose Code

General purpose code should never swallow interruption requests.

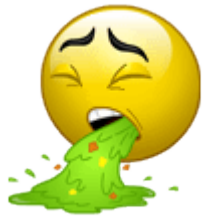
```
public void sleepOneHour()
   throws InterruptedException {
   sleep(3600000);
}
```

Non-interruptable Blocking

- `synchronized`: Cannot be interrupted
- Non-interruptable blocking operations

Non-interruptable Blocking Operations

```
public class SocketThread extends Thread {
  private Socket socket;
  public SocketThread(Socket socket) {
    this.socket = socket;
  }
  public void run() {
    ...
    try {
      int c = socket.getInputStream().read();
    } catch (IOException e) { }
    ...
  }
}
```



Not responsive to interruption requests. *Don't do this.*

Encapsulate Interruption

```
public class SocketThread extends Thread {
  ...
  public void interrupt() {
    try {
      socket.close();
    } catch (IOException ignored) {
    } finally {
      super.interrupt();
    }
  }
}
```

Abnormal Thread Termination

- Uncaught Exceptions in threads
- Default behavior: Print to `System.err`

```
public interface UncaughtExceptionHandler {
  void uncaughtException(Thread t, Throwable e);
}
```

```
Thread.setDefaultUncaughtExceptionHandler(...);
```

JVM Shutdown

Orderly shutdown

- Last nondaemon thread terminates
- Call to `System.exit()`
- Ctrl-C
- …

Abrupt shutdown

- Call to `Runtime.halt()`
- Killed by OS

Shutdown Hooks

- Executed for orderly shutdowns
- Last chance for cancellation and cleanup
- Registered with `Runtime.addShutdownHook(...)`
- Executed concurrently
  - Only use one if order matters
- Should be fast

Summary

- Cancellation – Interruption
  - Interrupted status
  - Blocking operations: Interruptable
  - Blocking operations: Non-interruptable
  - Encapsulate interruption
- Abnormal thread termination
- JVM shutdown