

Sistemi Concorrenti e di Rete LS

II Facoltà di Ingegneria - Cesena

a.a 2008/2009

[module lab 1.3]

CANCELLATION AND SHUTDOWN

STOPPING THREADS AND TASKS

- An activity is *cancellable* if external code can move it to completion before its normal completion
- Main reasons:
 - user-requested cancellation
 - e.g.: via GUI
 - time-limited activities
 - e.g. time-bounded search
 - application events
 - e.g. finding the first solution in a searching process
 - errors
 - shutdown
 - e.g. application shutdown

ASYNCHRONOUS VS. COOPERATIVE APPROACH

- Asynchronous
 - preemption
 - safety problems: state consistency
 - API in Thread class deprecated
- **Cooperative** (or synchronous) approach
 - signaling a request to stop & managing the request
 - *cancellation flag*
 - defining the *cancellation policy* for each task
 - **how** other code can request cancellation, **when** the tasks checks whether cancellation has been requested, and **what** actions the task takes in response to a cancellation request

CANCELLATION EXAMPLE: PRIME GENERATOR

```
public class PrimeGenerator implements Runnable {

    private final List<BigInteger> primes = new ArrayList<BigInteger>();

    private volatile boolean cancelled;

    public void run(){
        BigInteger p = BigInteger.ONE;
        while (!cancelled) {
            p = p.nextProbablePrime();
            synchronized (this){
                primes.add(p);
            }
        }
    }

    public void cancel() { cancelled = true; }

    public synchronized List<BigInteger> get(){
        return new ArrayList<BigInteger>(primes);
    }
}
```

CANCELLATION EXAMPLE: USER SIDE

- Getting one seconds of primes...

```
..  
  
List<BigInteger> aSecondOfPrimes() throws InterruptedException {  
  
    PrimeGenerator gen = new PrimeGenerator();  
  
    new Thread(gen).start();  
  
    try {  
        SECONDS.sleep(1);  
    } finally {  
        generator.cancel();  
    }  
    return generator.get();  
  
}
```

...AND IF THE THREAD IS BLOCKED?

- Suppose that the bounded buffer is full (blocking put...)

```
class StuckedPrimeProducer extends Thread {  
  
    private BoundedBuffer buffer;  
  
    public StuckedPrimeProducer(BoundedBuffer buf){  
        buffer = buf;  
        cancelled = false;  
    }  
  
    public void run(){  
  
        BigInteger p = BigInteger.ONE;  
  
        while (!cancelled){  
            BigInteger value = p.nextProbablePrime();  
            buf.put(value)  
        }  
    }  
  
    public void cancel(){ cancelled = true; }  
}
```

INTERRUPTION

- Acting on thread (boolean) interrupted status to avoid infinite blocking
 - set by interrupt() method
 - cleared by interrupted() method

```
class Thread {  
  
    public void interrupt() {...}  
    public boolean isInterrupted() {...}  
    public static boolean interrupted() {...}  
    ...  
}
```

- Unblock blocked states
 - sleep & wait methods
 - clear the status and throw InterruptedException

INTERRUPTION POLICY

- Interruption policy for each thread
 - interruption management idioms
- Common idiom
 - explicit shutdown state
 - cancel method
 - interruption check in loop
 - management of InterruptedException

INTERRUPTION POLICY: EXAMPLE

```
class PrimeProducer extends Thread {  
    private BlockingQueue<BigInteger> queue;  
  
    public PrimeProducer(BlockingQueue<BigInteger> q){  
        queue = q;  
    }  
  
    public void run(){  
        BigInteger p = BigInteger.ONE;  
        try{  
            while (!Thread.currentThread().isInterrupted()){  
                BigInteger value = p.nextProbablePrime();  
                queue.put(value)  
            } catch (InterruptedException ex){  
                // allow thread to exit  
            }  
        }  
        public void cancel(){  
            interrupt();  
        }  
    }  
}
```

- Note that calling `interrupt` does not necessarily stop the target thread from doing what it is doing
 - it merely delivers the message that interruption has been requested

EXECUTORS SHUTDOWN

- Two ways to shut down for ExecutorService
 - shutdown method for *graceful* shutdown
 - no new tasks are accepted
 - previously submitted tasks are allowed to complete, including those that have not yet begun execution
 - shutdownNow method for *abrupt* shutdown
 - attempts to cancel outstanding tasks
 - does not start any tasks that are queued but not begun
- Task submitted to an ExecutorService after it has been shutdown are handled by a *rejected execution handler*

LifecycleWebServer EXAMPLE

- The web server can be shut down in two ways:
 - programmatically by calling stop
 - through a client request by sending a specially formatted request

```
class LifecycleWebServer {  
    private final ExecutorService exec = ...  
  
    public void start() throws IOException {  
        ServerSocket socket = new ServerSocket(80);  
        while (!exec.isShutdown()) {  
            try {  
                Socket sock = conn.accept();  
                exec.execute(new Runnable(){  
                    public void run(){ handleRequest(conn); }  
                });  
            } catch (RejectedExecutionException ex){  
                if (!exec.isShutdown()){  
                    log("task submission rejected.");  
                }  
            }  
        }  
    }  
  
    public void stop(){ exec.shutdown(); }  
  
    void handleRequest(Socket conn){  
        Request req = readRequest();  
        if (isShutdownRequest(req)) {  
            stop();  
        } else { dispatchRequest(req); }  
    }  
}
```

AWAITING FOR TASK COMPLETION

- ExecutorService provides the awaitTermination method to await for task completion after a shut down
 - LogService example

```
public class LogService {  
  
    private final ExecutorService exec = Executors.newSingleThreadExecutor();  
    private final PrintWriter writer = ...  
  
    public void stop() throws InterruptedException {  
        try {  
            exec.shutdown();  
            exec.awaitTermination(Long.MAX_VALUE,TimeUnits.SECONDS);  
        } finally {  
            writer.close();  
        }  
    }  
  
    public void log(String msg){  
        try {  
            exec.execute(new WriteTask(msg));  
        } catch (RejectedExecutionException ignored){}  
    }  
}
```

“POISON PILLS” TECHNIQUE

- Recognisable objects placed in shared data structures signaling the termination of some process / activity

```
class Consumer extends Thread {  
  
    private BoundedBuffer buffer;  
    private boolean shutdown;  
  
    public Consumer(BoundedBuffer buf){  
        buffer = buf;  
        shutdown = false;  
    }  
  
    public void run(){  
  
        while (!shutdown){  
            try {  
                Item item = buffer.get();  
                if (!item.getDescr().equals("stop")){  
                    // process item  
                } else {  
                    shutdown = true;  
                }  
            } catch (InterruptedException ex){  
            }  
        }  
    }  
  
    public void shutdown(){  
        shutdown = true;  
        interrupt();  
    }  
}
```

INTERRUPTION POLICY: PITFALLS IN JAVA

- Non-interruptible blocking
 - synchronous socket I/O in `java.io`
 - read / write to `InputStream` / `OutputStream`
 - (native) lock acquisition
 - synchronized blocks
- Avoiding non-interruptible blocking
 - for I/O
 - use `java.nio` library
 - Interruptible channels
 - Selectors are interruptible
 - for locking
 - use `java.util.concurrent` library
 - use user-defined mechanisms / media

AVOIDING THREAD LEAKAGE

- Handling abnormal thread termination
 - catching and managing `Throwable` exceptions

```
...
public void run(){
    Throwable thrown = null;
    try {
        while (!isInterrupted()){
            runTask(getTaskFromQueue());
        }
    } catch (Throwable ex){
        throw = ex;
    } finally {
        threadExited(this,thrown);
    }
}
```

UNCAUGHT EXCEPTION HANDLERS

- Uncaught exception handlers
 - `UncaughtExceptionHandler` interface
 - method `setUncaughtExceptionHandler` in `Thread` class

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

- Example with Loggers:

```
public class UEHLogger implements Thread.UncaughtExceptionHandler {  
    public void uncaughtException(Thread t, Throwable e){  
        Logger logger = Logger.getAnonymousLogger();  
        logger.log(Level.SEVERE, "Thread terminated with an exception:"+  
                  +t.getName,e);  
    }  
}
```

JVM SHUTDOWN HOOKS

- JVM hook
 - user defined thread started when JVM shut down orderly tasks: cleanup, finalizations,...
 - not started in case of abnormal termination
 - attached through `addShutdownHook` in `Runtime` class
- Example

```
public void start() {
    Runtime.getRuntime().addShutdownHook(
        new Thread() {
            public void run(){
                try {
                    LogService.this.stop();
                } catch (InterruptedException ignored){}
            }
        });
}
```