C20: Threads

see also: ThreadedBallWorld, DropTest, Tetris source examples

Not covered: advanced stuff like notify/notifyAll

Multi-threading

- Multi-tasking: do multiple things in parallel:
  - Edit your game
  - Surf the web
  - Listen to mp3s
  - …

  but usually only one CPU that switches between tasks
- Multi-threading similar, but inside a single program/process/context

  ➔

  share data directly, same classes/methods/variables

Creating threads: 2 ways

- Subclassing Class Thread

  class ThreadedBallWorld extends Runnable
  - Must implement run() method
  - Use start() on a new instance:
    - (new BallThread()).start();
- Implement interface Runnable:

  class ThreadedBallWorld implements Runnable
  - Must implement run() method
  - Start by wrapping inside a Thread instance:
    - Thread ballThread = new Thread(new BallThread());
    - ballThread.start();
- Both will stop automatically, once run() finishes

Scheduling

- Can be pre-emptive or cooperative,

  ➔Each Thread should regularly call one of

  yield()

  wait()

  sleep()

  to give other threads a chance to run as well
- All event handlers (all listeners) and repaint() execute in the same event-handling thread:

  ➔ methods like "actionPerformed()" should be fast, for more substantial computations they should:
  - Start a new thread
  - Somehow tell another thread what to do

  Otherwise screen repainting will suffer!

Synchronisation

- If two (or more) threads access some value simultanously, and at least one want's to modify the value, things can go wrong:

  a += 10;

  Thread1: (a1) read a, (b1) compute a+10, (c1) write a

  Thread2: (a2) read a, (b2) compute a+10, (c2) write a

  What is the value of a after the following sequence:

  a1,b1,a2,b2,c2,c1

Synchronized methods

```
public synchronized void increment() {
  a += 10;
  }
```

- Only one thread at a time can execute this method on the same instance ➔ no interleaving possible ("atomic action") ➔ no inconsistencies
- but: beware of inefficiencies/deadlocks