Java™ Platform, Micro Edition

Part 4 – Timer, Tasks and Threads

Disclaimer

- **These slides are provided free of charge at http://www.symbianresources.com and are used during Java ME courses at the University of Applied Sciences in Hagenberg, Austria at the Mobile Computing department (http://www.fh-ooe.at/mc)**
- **Respecting the copyright laws, you are allowed to use them:**
  - **for your own, personal, non-commercial use**
  - **in the academic environment**
- **In all other cases (e.g. for commercial training), please contact andreas.jakl@fh-hagenberg.at**
- The correctness of the contents of these materials cannot be guaranteed. Andreas Jakl is not liable for incorrect information or damage that may arise from using the materials.
- This document contains copyright materials which are proprietary to Sun or various mobile device manufacturers, including Nokia, SonyEricsson and Motorola. Sun, Sun Microsystems, the Sun Logo and the Java™ Platform, Micro Edition are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Contents

- Timer
- Threads

Execute tasks at a specified time

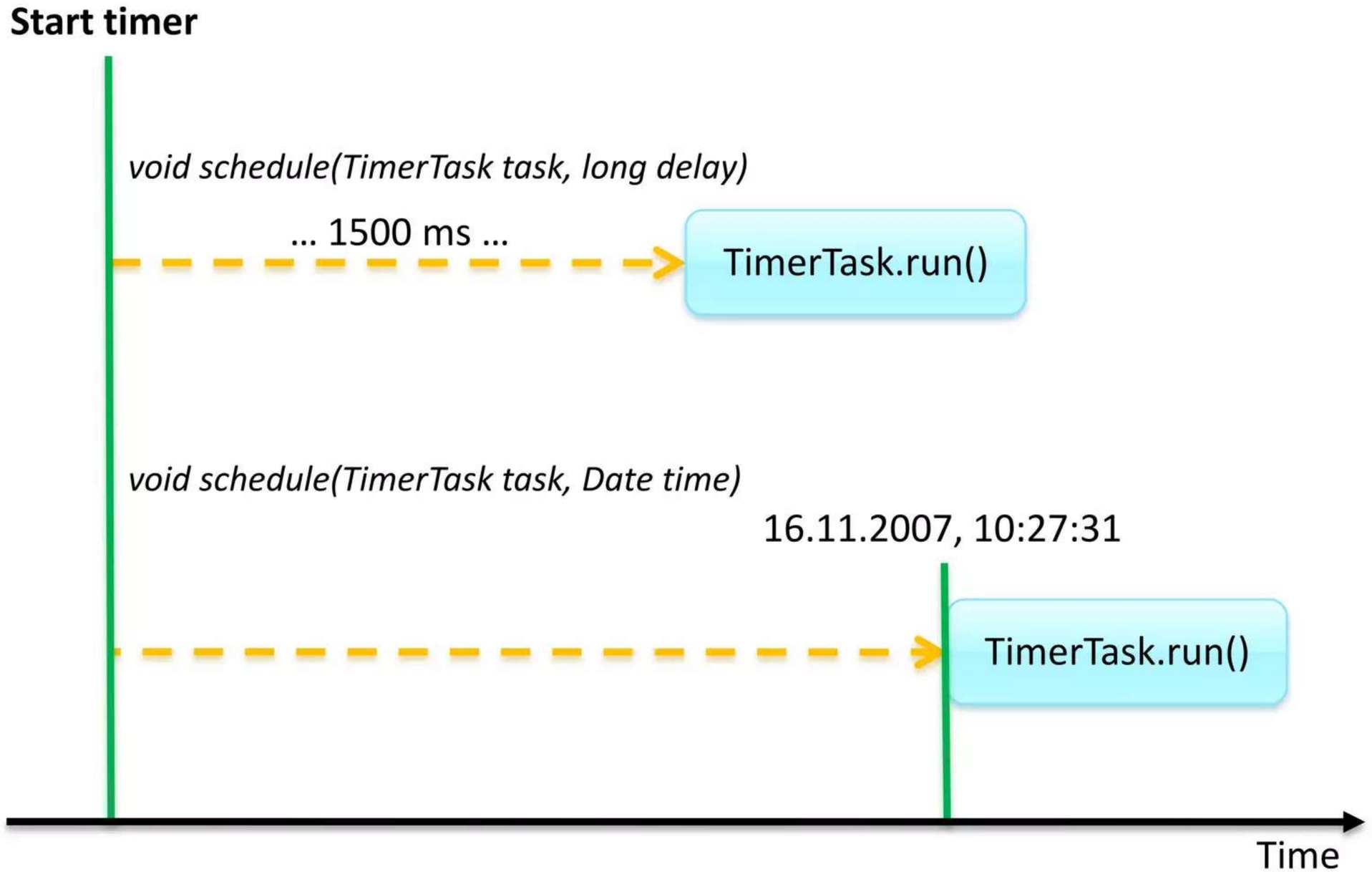
Timer

Timer

- **Asynchronous execution of a task**
  - after a specified **amount of time** (nonrecurring) or …
  - in certain **intervals** (periodic)

- **Two classes** are involved:

Nonrecurring Events

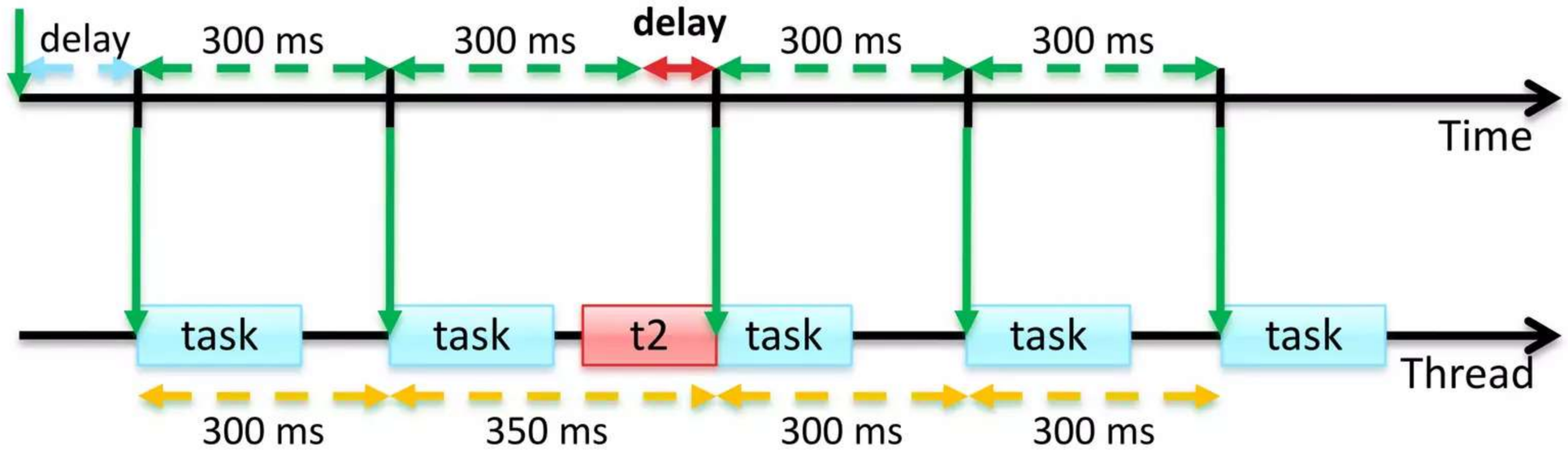


Applications

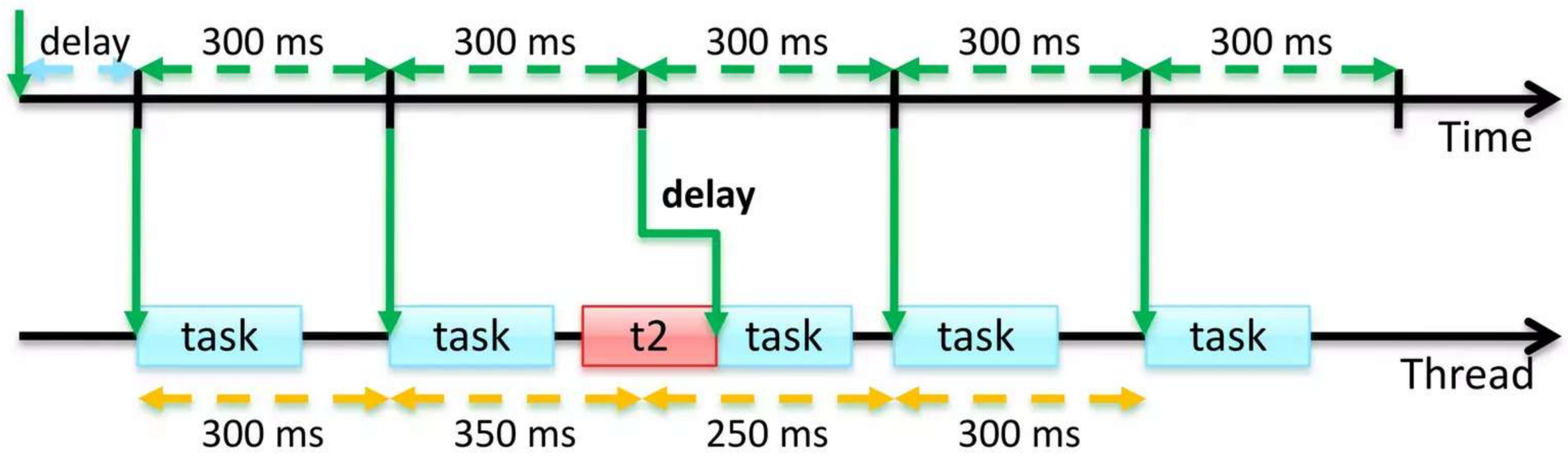
- **After *x* milliseconds**
  - Delayed start of an asynchronous process
    e.g. MIDlet startup: give VM time to draw the splash screen before loading and initializing the game
  - Countdown
- **At a specified point in time**
  - Alarm clock

Periodic Tasks

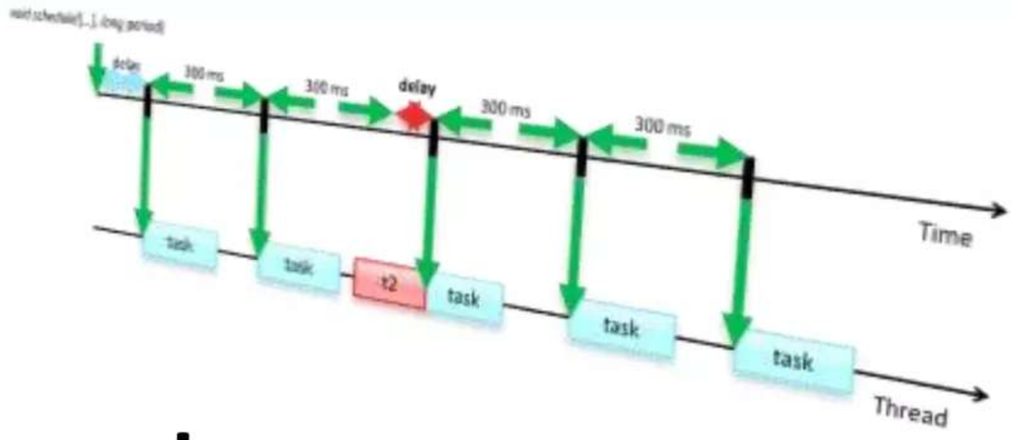
void schedule([…], long period)



void scheduleAtFixedRate([…], long period)

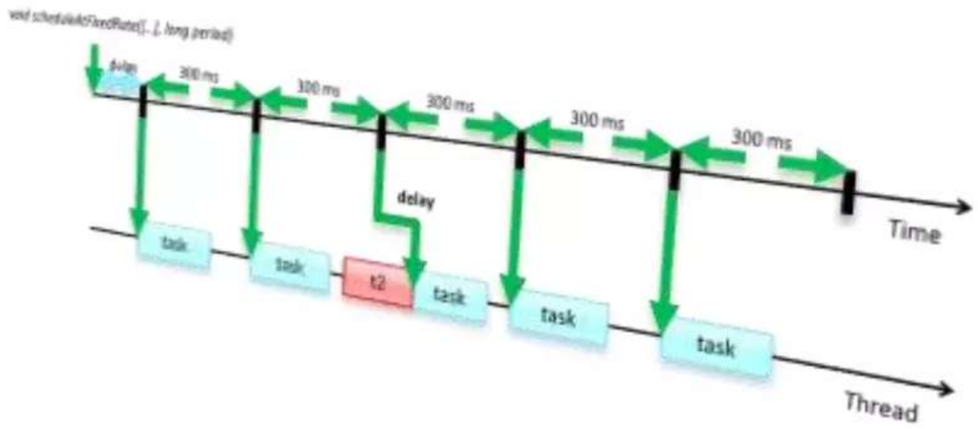


Fixed Delay



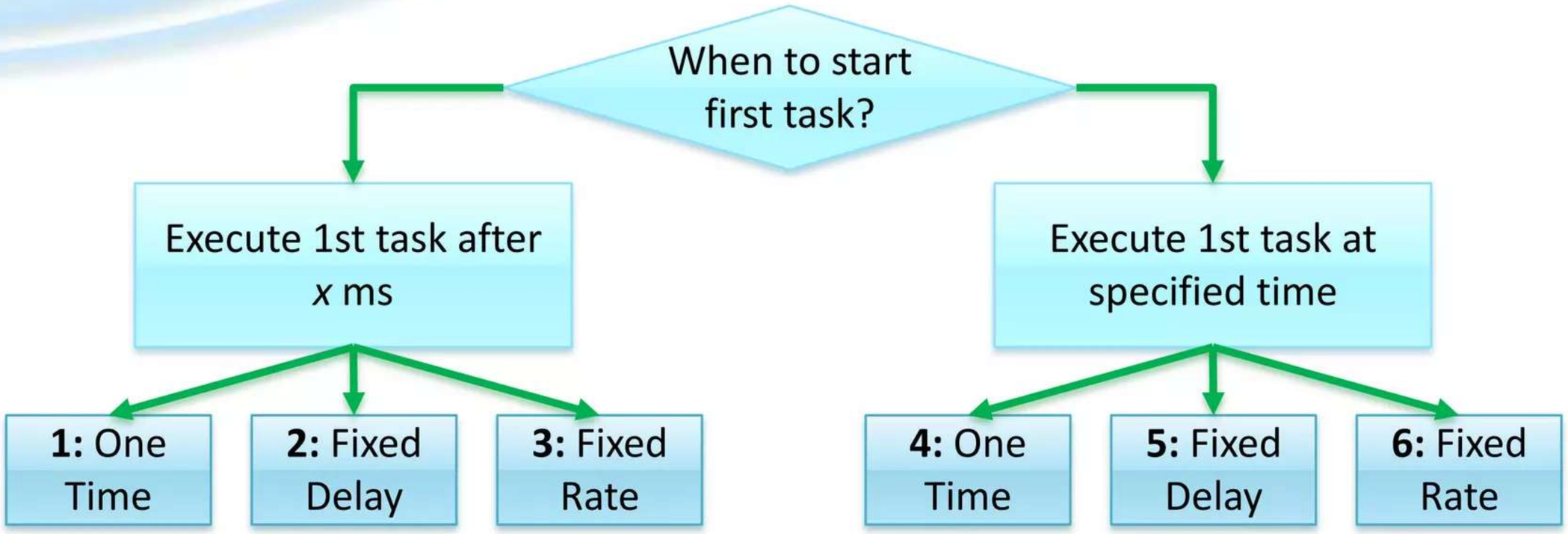
- **If consistency is more important than accuracy**
- Execution is delayed (e.g. by garbage collection):
  - Only one execution is delayed
  - After this, the constant interval is kept up again
- **Suitable for:**
  - Animations: shouldn’t suddenly move faster
- **Unsuitable for:**
  - Clock: Inaccuracies would accumulate

Fixed Rate



- **When accuracy is more important**
- Execution is delayed (e.g. by garbage collection):
  - Two or more subsequent executions are scheduled at shorter intervals to “catch up”
  - None of the events will be dropped
- **(Un)suitable:**
  - Inverse to fixed delay

Combination



1: schedule(TimerTask task, long delay)

2: schedule(TimerTask task, long delay, long period)

3: scheduleAtFixedRate(TimerTask task, long delay, long period)

4: schedule(TimerTask task, Date time)

5: schedule(TimerTask task, Date firstTime, long period)

6: scheduleAtFixedRate(TimerTask task, Date firstTime, long period)

TimerTask

- Create your **own class derived from TimerTask**
- Pass an **instance** of it **to the Timer-object**
- Calls **run()** of your TimerTask-class at the **specified time**
- → implement the abstract run()-method!

Example

```
// Allocate a timer
Timer tm = new Timer();

// Schedule the timer with appropriate scheduling option, eg. in 1000 milliseconds
tm.schedule(new TodoTask(), 1000);

[…]

private class TodoTask extends TimerTask
{
   public final void run()
   {
      // Do something …
   }
}
```

Control an asynchronous task

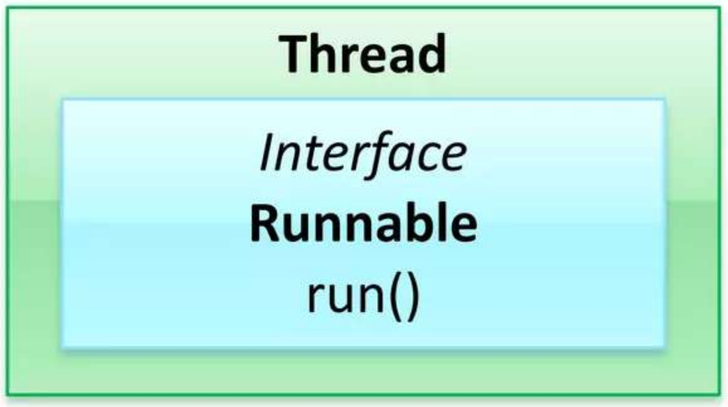
Threads

Why Threads?

- **Examples:**
  - **Game loop**, has to be executed all the time, as often as possible
    - To make animations as smooth as possible.
    - A fixed delay timer would result in a fixed amount of frames per second and not utilize better hardware
  - Animated **progress bar** while loading or processing
  - Connect through **HTTP / Sockets**
    - Connection process can be stopped by JVM to display a security warning
    - Without threads: app. couldn’t process key press anymore, would lock

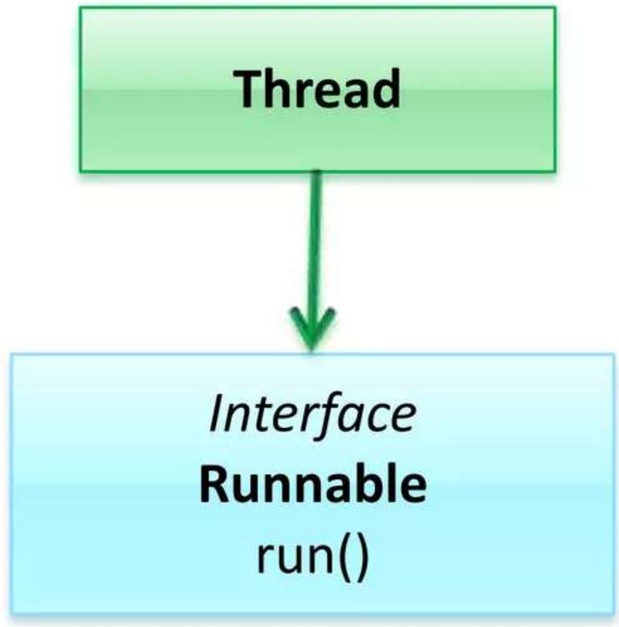
Possibilities

1. Derive your own object from `Thread`, implement `run()`




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

2. Extra (2nd) object implements the `Runnable`-interface, start through `Thread`-object.




```
DoSomething doIt = new DoSomething();
Thread t = new Thread( doIt );
t.start();
```

Example – Thread-Object

```
DoAnotherThing doIt = new DoAnotherThing();
doIt.start();
// …
```

```
public class DoAnotherThing extends Thread {
    public void run() {
        // Here is where you do something
        // Executed within its own thread!
    }
}
```

Example – Runnable-Object

```
DoSomething doIt = new DoSomething();
Thread t = new Thread( doIt );
t.start();
```

```
public class DoSomething implements Runnable {
   private boolean quit = false;

   public void run() {
      while( !quit ) {
         // do something, e.g. process the
         // next frame in a game
      }
   }

   public void quit() {
       quit = true;
   }
}
```

Differences: Runnable / Thread?

- Almost identical.
- **Advantages of the Runnable-variant (interface):**
  - Can be implemented by existing class (no multiple inheritance in Java (`extend Thread`)!)
  - Therefore, saves an additional class

Structure

- run()-method can be used for **single action**
  - Once run() method is left, thread stops and is cleaned up
- Commonly, run() **repeats an action** until some condition is satisfied (e.g., in a game loop)
  - Implement **infinite loop** in run()
  - **To exit the loop and stop the thread:**
    - Repeatedly **query boolean status variable**
    - If set to true through any function → break loop and therefore leave run()-method, thread is stopped

Additional Possibilities

- For more complex threading tasks, use:
  - Synchronisation, Wait, …
- More information:
  - http://developers.sun.com/techtopics/mobility/midp/articles/threading2/

Interactive

… let’s move on to the challenges!