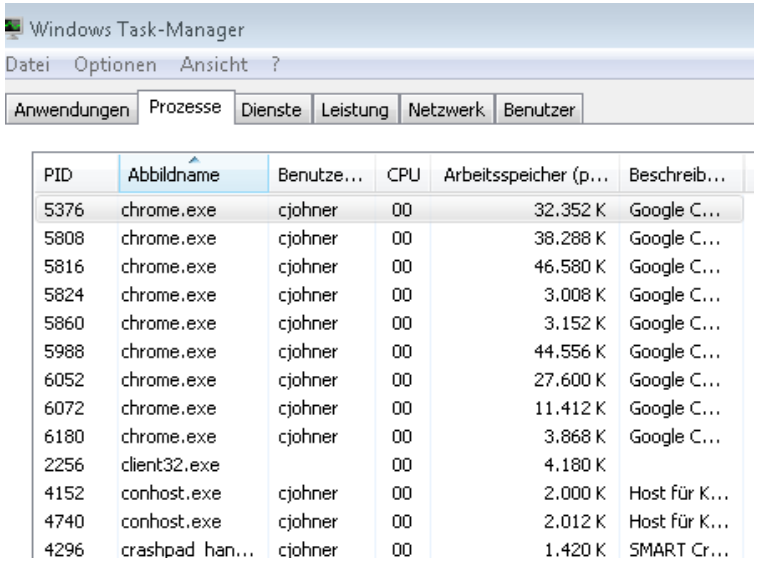
Multi-Threading

Multi-Tasking

Parallel execution of tasks / programs / processes

Any of these processe has

- Process ID (PID)
- Resources e.g. CPU, RAM, Network, ...
- Optionally parent process
- One or more threads



PID	Abbildname	Benutze...	CPU	Arbeitsspeicher (p...	Beschreib...
5376	chrome.exe	cjohner	00	32.352 K	Google C...
5808	chrome.exe	cjohner	00	38.288 K	Google C...
5816	chrome.exe	cjohner	00	46.580 K	Google C...
5824	chrome.exe	cjohner	00	3.008 K	Google C...
5860	chrome.exe	cjohner	00	3.152 K	Google C...
5988	chrome.exe	cjohner	00	44.556 K	Google C...
6052	chrome.exe	cjohner	00	27.600 K	Google C...
6072	chrome.exe	cjohner	00	11.412 K	Google C...
6180	chrome.exe	cjohner	00	3.868 K	Google C...
2256	client32.exe		00	4.180 K	
4152	conhost.exe	cjohner	00	2.000 K	Host für K...
4740	conhost.exe	cjohner	00	2.012 K	Host für K...
4296	crashpad_han...	cjohner	00	1.420 K	SMART Cr...

Multi-Threading

Parallel execution with a process

Example: MS Word

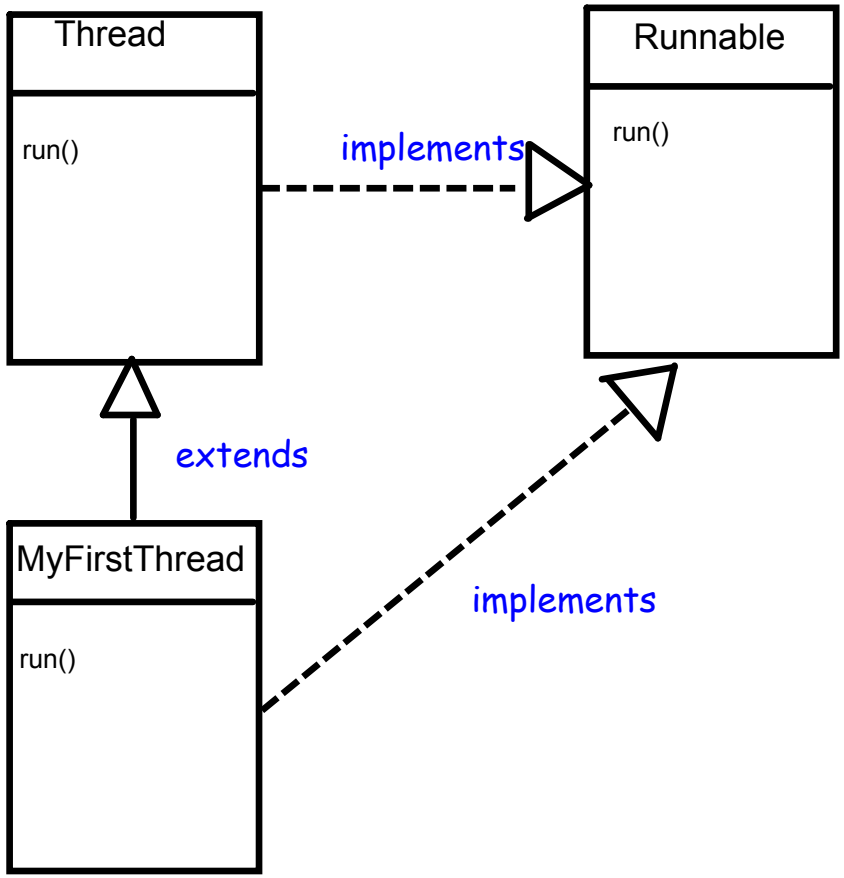
- UI
- Spell checker
- Save (in background)
- Print (in background)
- ....

Another advantage of multi-threading: programs can use multi core CPU.

Implementation of threads in JAVA

1. Inherit from thread
2. Implement Runnable

either or, not both



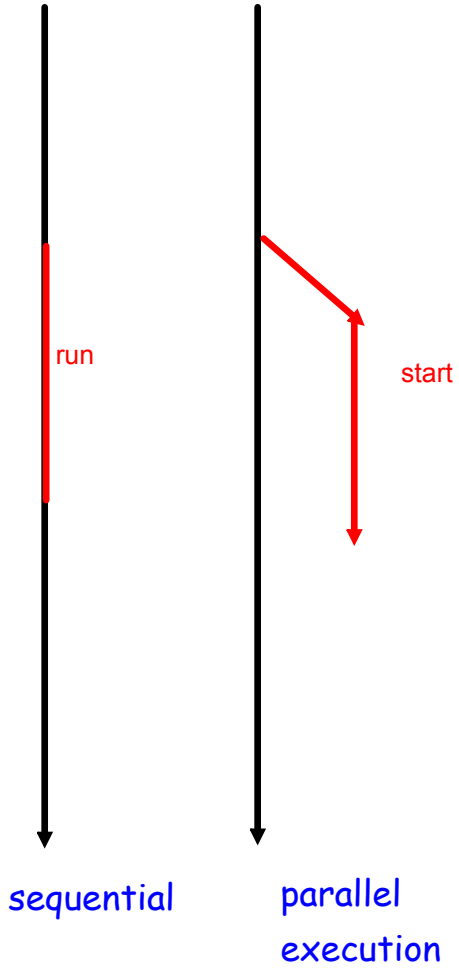
1. alternative: extending "Thread"

MyFirstThread.java | MySecondThread.java | StartThreads.java

```
1  package threads;
2
3  import java.util.GregorianCalendar;
4
5  public class MyFirstThread extends Thread {
6
7      @Override
8      public void run() {
9
10         System.out.println("MyFirstThread start ....");
11
12         for (int i = 0; i < 10000000; i++) {
13             new GregorianCalendar();
14         }
15
16         System.out.println("MyFirstThread ends ....");
17     }
18
19 }
```

MyFirstThread.java | MySecondThread.java | StartThreads.java

```
1  package threads;
2
3  public class StartThreads {
4
5      public static void main(String[] args) {
6          System.out.println("main starts ... ");
7
8          MyFirstThread thread1 = new MyFirstThread();
9
10         //thread1.run();
11         thread1.start();        start
12
13         System.out.println("main ends ... ");
14
15     }
16 }
17
```



2. alternative: implementing "Runnable"

MyFirstThread.java | MySecondThread.java | StartThreads.java

```
1  package threads;
2
3  import java.util.GregorianCalendar;
4
5  public class MySecondThread implements Runnable {
6
7      @Override
8      public void run() {
9          System.out.println("MySecondThread start ....");
10
11         for (int i = 0; i < 10000000; i++) {
12             new GregorianCalendar();
13         }
14
15         System.out.println("MySecondThread ends ....");
16     }
17
18 }
19
```

MyFirstThread.java | MySecondThread.java | StartThreads.java

```
1  package threads;
2
3  public class StartThreads {
4
5      public static void main(String[] args) {
6          System.out.println("main starts ... ");
7
8          MyFirstThread thread1 = new MyFirstThread();
9
10         //thread1.run();
11         thread1.start();
12
13         Thread thread2 = new Thread(new MySecondThread());
14
15         thread2.start();
16
17         System.out.println("main ends ... ");
18
19     }
20 }
21
```

Ending threads

1. method run completes successfully
2. method run terminates with an exception
3. method run is interrupted. However, the flag needs to be evaluated!
4. the main program ends and the thread was started as daemon

```java
package threads;

public class BadAgentThread extends Thread {

    public void run() {
        System.out.println("BadAgentThread start killing.
        while(!interrupted()) {
            System.out.println("booooooom");
        }
    }
}
```

we actually evaluate the flag "interrupted"

MyFirstThread.java MySecondThread.java StartThreads.java BadAgentThread.j

```java
package threads;

public class StartThreads {

    public static void main(String[] args) throws InterruptedException {
        System.out.println("main starts ... ");

//      MyFirstThread thread1 = new MyFirstThread();
//
//      //thread1.run();
//      thread1.start();
//
//      Thread thread2 = new Thread(new MySecondThread());
//
//      thread2.start();

        BadAgentThread killer = new BadAgentThread();
        killer.start();

        Thread.sleep(2000);

        killer.interrupt();

        System.out.println("main ends ... ");

    }
}
```

sleep for 2 seconds

we call "interrupt".

ATTENTION!: If the flag "interrupted" is not evaluated the call of "interrupt()" has no effect!!

StartThreads.java

SOTE1-2018-05-07/src/threads/StartThreads.java

```
2
3  public class StartThreads {
4
5      public static void main(String[] args) throws InterruptedException {
6          System.out.println("main starts ... ");
7
8  //      MyFirstThread thread1 = new MyFirstThread();
9  //
10 //      //thread1.run();
11 //      thread1.start();
12 //
13 //      Thread thread2 = new Thread(new MySecondThread());
14 //
15 //      thread2.start();
16
17         NeverEndingBadAgentThread killer = new NeverEndingBadAgentThread();
18
19         killer.setDaemon(true);
20         killer.start();
21
22         System.out.println("main ends ... ");
23
24     }
25 }
26
```

MyFirstThread.java MySecondThread.java BadAgentThread.java

```
1  package threads;
2
3  public class NeverEndingBadAgentThread extends Thread {
4
5      public void run() {
6          System.out.println("BadAgentThread start killing.... "
7
8          while(true) {
9              System.out.println("booooooom");
10         }
11     }
12
13 }
14
```

the flag setDaemon decides whether a thread is terminated the same moment the main program terminates (if setDaemon == true) or not.

NOTE!: the flag needs to be set before(!) the thread is started.

Waiting for threads

StartThreads.java

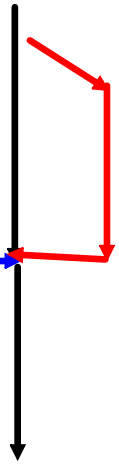
```
1  package threads;
2
3  public class StartThreads {
4
5      public static void main(String[] args) throws InterruptedException {
6          System.out.println("main starts ... ");
7
8          MyFirstThread thread1 = new MyFirstThread();
9
10         thread1.start();
11
12         thread1.join();
13
14         System.out.println("main ends ... ");
15
16     }
17 }
18
```

Console

```
<terminated> StartThreads [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (07.05.2018, 14:51
main starts ...
MyFirstThread start ....
MyFirstThread ends ....
main ends ...
```

the main-thread waits for thread2 to be finshed (run-method ended with or without exception).

here the main thread waits for the other (red) thread



Sequentially invoking the same thread over again:

StartThreads.java

```
1  package threads;
2
3  public class StartThreads {
4
5      public static void main(String[] args) throws InterruptedException {
6          System.out.println("main starts ... ");
7
8          MyFirstThread thread1 = new MyFirstThread();
9
10         thread1.start();
11
12         thread1.join();
13
14         thread1.start();
15
16         thread1.join();
17
18         System.out.println("main ends ... ");
19
20     }
21 }
22
```

Console

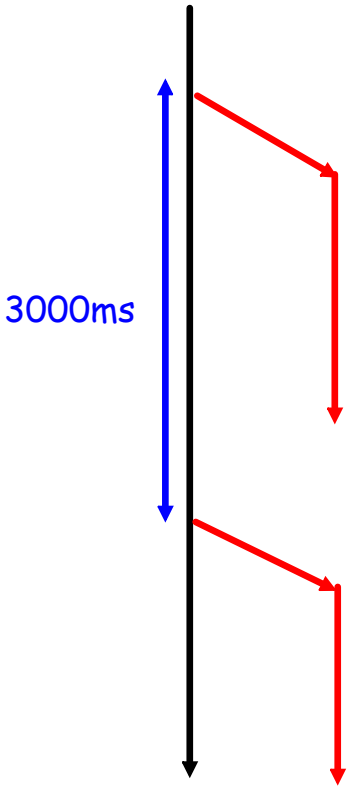
```
<terminated> StartThreads [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (07.05.2018, 14
main starts ...
MyFirstThread start ....
MyFirstThread ends ....
Exception in thread "main" java.lang.IllegalThreadStateException
        at java.lang.Thread.start(Unknown Source)
        at threads.StartThreads.main(StartThreads.java:14)
```

it is not possible to start a thread a second time.

The solution to this problem already exists: ExecutorService

StartThreads.java | StartThreads2.java

```
1  package threads;
2
3  import java.util.concurrent.ExecutorService;
4  import java.util.concurrent.Executors;
5
6  public class StartThreads2 {
7      public static void main(String[] args) throws InterruptedException {
8          System.out.println("main starts ... ");
9
10         MyFirstThread thread1 = new MyFirstThread();
11
12         ExecutorService threadpool = Executors.newCachedThreadPool();
13
14         threadpool.execute(thread1);
15
16         Thread.sleep(3000);
17         System.out.println("Waiting done");
18         threadpool.execute(thread1);
19
20         threadpool.shutdown();
21
22
23         System.out.println("main ends ... ");
24
25     }
26 }
27
```



Synchronization

There are situations where we do NOT want methods (or code) to be invoked in parallel (multi-threaded) but rather sequentially.

Example: a method print() of printer should be used sequentially (> printer queue) and not in parallel).

StartThreads.java StartThreads2.java Printer.java

```
1  package threads;
2
3  public class Printer {
4      public synchronized void print() {
5          System.out.println("Printer start printing");
6
7          //method access the printer hardware
8
9          System.out.println("Printer ends printing");
10     }
11 }
12
```

Synchronizing an entire method

StartThreads.java StartThreads2.java Printer.java

```
1  package threads;
2
3  public class Printer {
4      public void print() {
5          System.out.println("Printer start printing");
6
7          synchronized(new Object()) {
8              //method access the printer hardware
9          }
10
11         System.out.println("Printer ends printing");
12     }
13 }
14
```

Synchronizing just a code block (i.e. a part of a method)