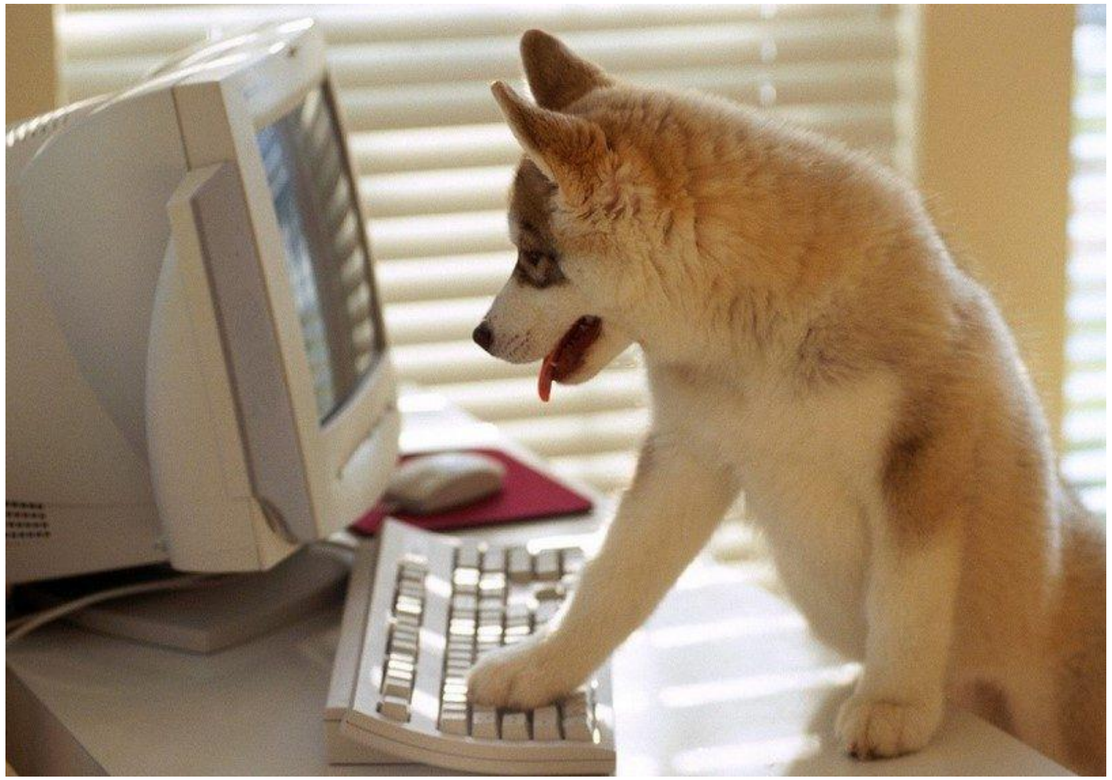
Programming Tutorial [Basics]



Lambda Expressions

```
JButton btn = new JButton("Hello");
btn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
           System.out.println("Hello World!");
       }
});
```

```
JButton btn = new JButton("Hello");
btn.addActionListener((e) ->
          {System.out.println("Hello World!");}
);
```

Scheduler

Scheduler = Decides which task is currently active and how long it stays active
Determined by:

- Priority (*Preemption*)
- Time (*Time-Slicing*)
- Combination of both

ThreadGroup

`ThreadGroup` for grouping several Threads

Operations can be performed on all threads of the `ThreadGroup` at once

```
public class MyRunnable implements Runnable{
    ...
}

…

ThreadGroup tg = new ThreadGroup(“My ThreadGroup”);
Thread t1 = new Thread(tg,new MyRunnable(),"one");
Thread t2 = new Thread(tg,new MyRunnable(),"two");
Thread t3 = new Thread(tg,new MyRunnable(),"three");

tg.list(); //prints all threads from tg

Thread.currentThread().getThreadGroup().interrupt();
```

ThreadGroup

`ThreadGroup` for grouping several Threads

Operations can be performed on all threads of the `ThreadGroup` at once

```
public class MyRunnable implements Runnable{
    ...
}

…

ThreadGroup tg = new ThreadGroup(“My ThreadGroup”);
Thread t1 = new Thread(tg,new MyRunnable(),"one");
Thread t2 = new Thread(tg,new MyRunnable(),"two");
Thread t3 = new Thread(tg,new MyRunnable(),"three");

tg.list(); //prints all threads from tg

Thread.currentThread().getThreadGroup().interrupt();
```

ExecutorService

`ExecutorService:`

- Thread management
- Running of asynchronous Tasks
- Running as many Tasks as we want

```
ExecutorService ex = Executors.NewSingleThreadExecutor();
ex.submit(() -> System.out.println(“Hello World!”));
```

Executors

`newCachedThreadPool()`
- Unspecified number of Threads will all be executed at the same time

`newFixedThreadPool(int nThreads)`
- Fixed number of Threads will be executed at the same time

`newScheduledThreadPool(int corePoolSize)`
- Schedule Tasks to run after a delay or to execute repeatedly

`newSingleThreadExecutor()`
- Just one Thread will be executed (good for sequential task completion)

`newSingleThreadScheduledExecutor()`
- Same like `ScheduledThreadPool` but just with one task at a time

`newWorkStealingPool()`
- Based on WorkStealing Algorithm; uses all available processors

Shutdown

```
try {
    System.out.println("Shutting down Executor...");
    executor.shutdown();
    executor.awaitTermination(5, TimeUnit.SECONDS);
}
catch (InterruptedException e) {
    System.err.println("Tasks interrupted");
}
finally {
    if (!executor.isTerminated()) {
        System.err.println("Cancel running tasks...");
    }
    executor.shutdownNow();
    System.out.println("Executor shut down");
}
```

Shutdown

```
try {
    System.out.println("Shutting down Executor...");
    executor.shutdown();
    executor.awaitTermination(5, TimeUnit.SECONDS);
}
catch (InterruptedException e) {
    System.err.println("Tasks interrupted");
}
finally {
    if (!executor.isTerminated()) {
        System.err.println("Cancel running tasks...");
    }
    executor.shutdownNow();
    System.out.println("Executor shut down");
}
```

Threads

Recall:

Thread
- abstract class
- `void run()`

Runnable
- Interface
- Uses core implementation of Thread
- `void run()`

Callable<T>
- Generic Interface
- `T call()`

Threads

Recall:

Thread
- abstract class
- `void run()`

Runnable
- Interface
- Uses core implementation of Thread
- `void run()`

Callable<T>
- Generic Interface
- `T call()`

How to retrieve the returned value of the call-Method of a Callable?

Future

```
public class MyCallable implements Callable<String>{...}

...

Callable c = new MyCallable();
ExecutorService executor = Executors.newSingleThreadExecutor();
Future<Integer> future = executor.submit(c);

System.out.println("future done? " + future.isDone());

String result = future.get();

System.out.println("future done? " + future.isDone());
System.out.print("result: " + result);
```

Future

```
public class MyCallable implements Callable<String>{...}

...

Callable c = new MyCallable();
ExecutorService executor = Executors.newSingleThreadExecutor();
Future<Integer> future = executor.submit(c);

System.out.println("future done? " + future.isDone());

String result = future.get(1, TimeUnit.Seconds);

System.out.println("future done? " + future.isDone());
System.out.print("result: " + result);
```

Batch

```
ExecutorService executor =
Executors.newWorkStealingPool();

List<Callable<String>> callables = Arrays.asList(
       () -> "task1",
       () -> "task2",
       () -> "task3");

executor.invokeAll(callables)
    .stream()
    .map(future -> {
       try {
          return future.get();
       }
       catch (Exception e) {
          throw new IllegalStateException(e);
       }
    })
    .forEach(System.out::println);
```

```
Callable<String> callable(String result, long sleep) {
    return () -> {
       TimeUnit.SECONDS.sleep(sleep);
       return result;
    };
}

ExecutorService executor = Executors.newWorkStealingPool();

List<Callable<String>> callables = Arrays.asList(
    callable("task1", 2),
    callable("task2", 1),
    callable("task3", 3));

String result = executor.invokeAny(callables);
System.out.println(result);
```

Scheduling

Schedule Tasks

```
schedule(Callable<V> callable, long delay, TimeUnit unit)
```

- Tasks runs after given delay
- Returns object of `ScheduledFuture<?>` that consists of `getDelay()`-method

```
scheduleAtFixedRate(Runnable command,
                    long initialDelay,
                    long period,
                    TimeUnit unit)
```

- Executes Tasks with a fixed time rate

```
scheduleWithFixedDelay(Runnable command,
                       long initialDelay,
                       long delay,
                       TimeUnit unit)
```

- Similar to `scheduleAtFixedRate` but delay applies to time between end of task and start of new one

Executors

In this course we will use Github Classroom

1. Get a Github Account if you don’t have one
2. Go to: https://classroom.github.com/a/mxEfsZxC (or scan the QR Code with your phone)
3. Authorize Github and accept the assignment
4. Click on the repository