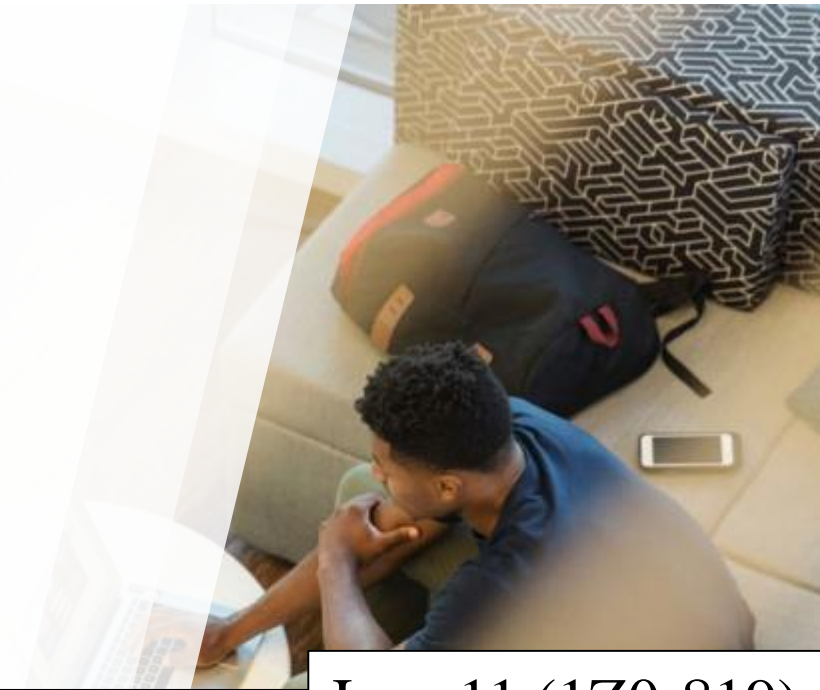


A photograph of four students in a library setting. A young man in a grey t-shirt is smiling and looking towards a young woman with glasses who is looking at a laptop. Another young woman is in the foreground, and a young man is partially visible on the right. They are all gathered around a table with books and a laptop. The background is filled with bookshelves. The image has a semi-transparent blue overlay on the left side and a semi-transparent red overlay at the bottom.

Concurrency

Concurrency



Concurrency

Java 11 (1Z0-819)

✓ Create worker threads using Runnable and Callable, and manage concurrency using an ExecutorService and `java.util.concurrent` API

✓ Develop thread-safe code, using different locking mechanisms and `java.util.concurrent` API

Concurrency

- What is concurrency?
 - executing tasks at the same time
- Multiple CPU's (plus multi-core processors); true parallel processing
 - three chefs working at the same time on a meal
- Single CPU – multitasking
 - one chef on his/her own preparing a meal
- Multi-threading – separate parts of your program can use independent threads

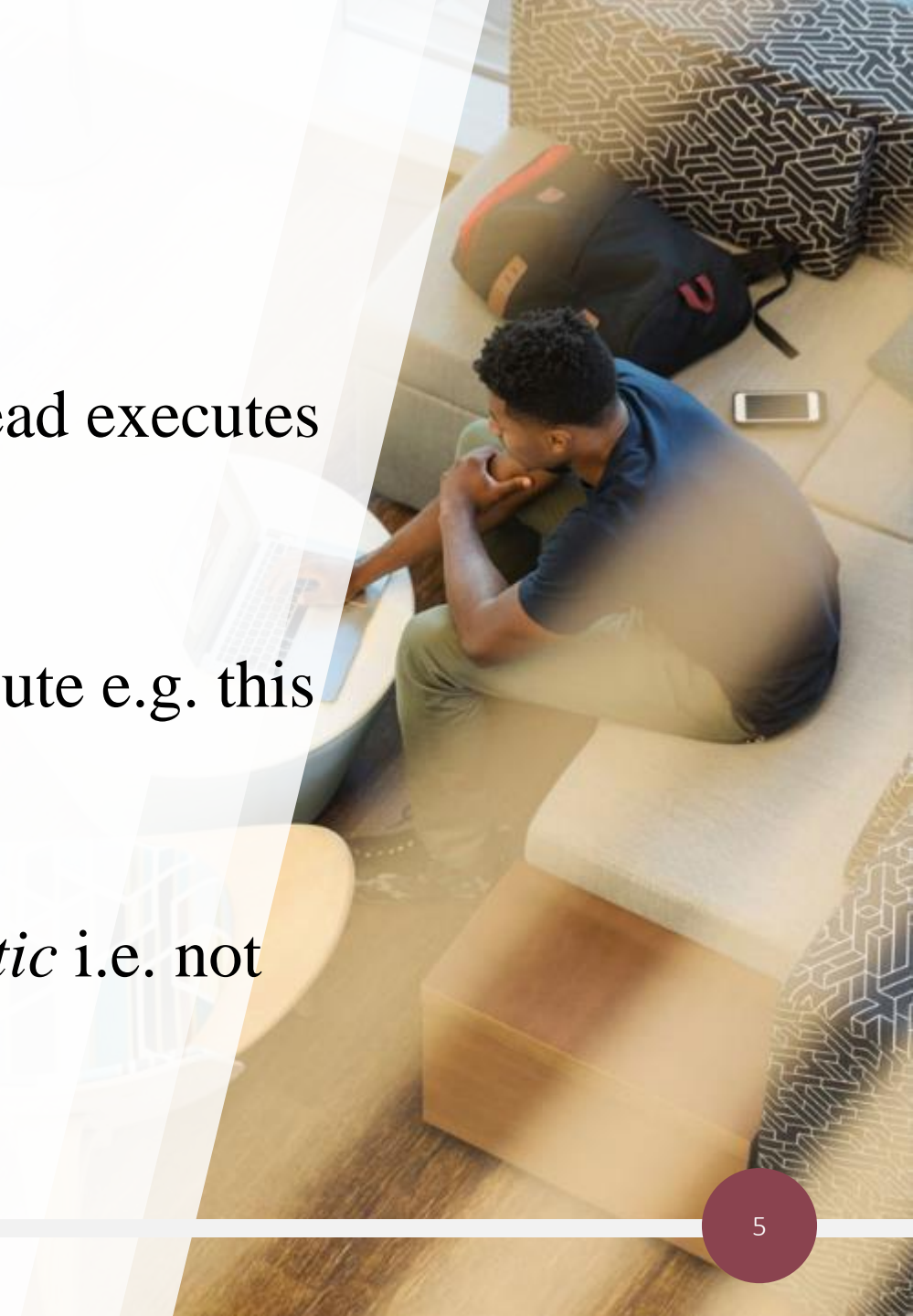


Concurrency

Advantages	Disadvantages
better performance	shared resources must be handled carefully
faster response time	data races, deadlock and livelock

Definition of Terms

- A *process* consists of one or more threads.
- A *thread* is the smallest unit of execution. A thread executes tasks.
- A *task* defines the work that the thread will execute e.g. this is often a lambda.
- The order of thread execution is *non-deterministic* i.e. not guaranteed.



Creating Threads

- extend *Thread*
- implement *Runnable*
- implement *Callable* (requires *ExecutorService*)



extending *Thread*

```
package lets_get_certified.concurrency.creating_threads;

public class MyThread extends Thread{
    @Override
    public void run(){
        System.out.println("run(): "+getName());
    }
    public static void main(String[] args) {
        new MyThread().start();
        System.out.println("main(): "+Thread.currentThread().getName());
    }
}
```

```
main(): main
run(): Thread-0
```

```
run(): Thread-0
main(): main
```

implementing *Runnable*

```
public class MyRunnable implements Runnable{

    @Override
    public void run() {
        System.out.println("run(): "+Thread.currentThread().getName());
    }
    public static void main(String[] args) {
        new Thread(new MyRunnable()).start();
        System.out.println("main(): "+Thread.currentThread().getName());
    }
}
```

```
main(): main
run(): Thread-0
```

```
run(): Thread-0
main(): main
```


implementing *Runnable* (lambda version)

```
package lets_get_certified.concurrency.creating_threads;

public class UsingLambdaAsRunnable {
    public static void main(String[] args) {
        Thread t = new Thread( () -> System.out.println("run(): "+
                                                         Thread.currentThread().getName()) );
        t.start();
        System.out.println("main(): "+Thread.currentThread().getName());
    }
}
```

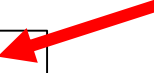
```
main(): main
run(): Thread-0
```

```
run(): Thread-0
main(): main
```

Calling *run()* directly

```
package lets_get_certified.concurrency.creating_threads;

public class UsingLambdaAsRunnable {
    public static void main(String[] args) {
        Thread t = new Thread( () -> System.out.println("run(): "+
                                                         Thread.currentThread().getName()) );
        //      t.start();
        t.run();
        System.out.println("main(): "+Thread.currentThread().getName());
    }
}
```



```
run(): main
main(): main
```

sleep() and *join()*

TimeBomb.java

