Solutions to exercises from lecture 4 on Thread Programming

Ex. 1:

```
public class SlowPrinterEx{
        public static void main( String[ ] args ){
                int length = 20;
                final String[ ] s1 = new String[ length ];
                final String[ ] s2 = new String[ length ];
                final String[ ] s3 = new String[ length ];

                for( int n = 0; n < length; n++ ){
                        s1[n] = "A";
                        s2[n] = "B";
                        s3[n] = "C";
                }
                long startTime = System.currentTimeMillis( );
                SlowPrinter.blockingPrint( s1 );
                SlowPrinter.blockingPrint( s2 );
                SlowPrinter.blockingPrint( s3 );
                long t = ( System.currentTimeMillis( ) - startTime ) / 1000;
                System.out.println( );
                System.out.println( "Time for 3 blockingPrints: " + t + " seconds" );
                System.out.println( );


                startTime = System.currentTimeMillis( );
                SlowPrinter.nonBlockingPrint( s1 );
                SlowPrinter.nonBlockingPrint( s2 );
                SlowPrinter.nonBlockingPrint( s3 );
                while (SlowPrinter.busy != 0){}         //wait for printing to complete or t
will be wrong
                t = (System.currentTimeMillis( ) - startTime ) / 1000;
                System.out.println( );
                System.out.println( "Time for 3 nonBlockingPrints: " + t + " seconds" );

        }
}

class SlowPrinter{

        public static void blockingPrint( String[ ] s ){
                for( int n = 0; n < s.length; n++ ){
                        System.out.print( s[n] );
                        try{
                                Thread.sleep( 100 );
                        }
                        catch( InterruptedException e ){
                                // Never happens
                        }
                }
        }

        static int busy = 0; // Is 0 when no threads are doing non-blocking printing

        public static void nonBlockingPrint( final String[ ] s ){
                busy++;
                final Thread t = new Thread(){
                        public void run(){
                                for( int n = 0; n < s.length; n++ ){
                                        System.out.print( s[n] );
                                        try{
                                                Thread.sleep( 100 );
                                        }
                                        catch( InterruptedException e ){
                                                // Never happens
                                        }
                                }
                                busy--;
                        }
                };
```

```
                t.start();
        }
}
```

Ex. 2:

(a) Two threads are implemented using anonymous inner classes. Each thread will write when it starts and when it stops.
The thread t1 waits for 2 seconds and then dies, t2 waits for t1 to end and dies. When the main thread is executed it will start t1 and t2 and wait for t2 to die.
So what happens is: t1 is started and paused, t2 is started and waits for t1. After 2 sec. t1 dies and allows t2 to die. When t2 dies the main thead is done as well.

(b)

```
package dk.itu.oop.lecture4;

public class JoinSolution{
        public static void main(String args[]){
                System.out.println( "main started" );
                Thread t1 = new Thread (new Join1());
                System.out.println( "t1 made" );
                Thread t2 = new Thread (new Join2(t1));
                System.out.println( "t2 made" );
                t1.start();
                t2.start();
                try{
                        t2.join( );
                }
                catch( InterruptedException e ){
                }
                System.out.println( "main finishes" );
        }
}

class Join1 implements Runnable{
        public void run(){
                System.out.println( "t1 started" );
                try{
                        Thread.sleep( 2000 );
                }catch( InterruptedException e ){ }
                System.out.println( "t1 finishes" );
        }
}

class Join2 implements Runnable{
        Thread t1;
        public Join2(Thread t1){
                super();
                this.t1 = t1;
        }

        public void run(){
                System.out.println( "t2 started" );
                try{
                        t1.join( );
                } catch( InterruptedException e ){ }
                System.out.println( "t2 finishes" );
        }
}
```

Ex. 3:

(a) The main thread and thread A exist. Thread A has the lock on obj1.

(b) The main thread and threads A and B exist. Thread A has the lock on obj1 and thread B has the lock on obj2.

(c) The main thread and threads A and B exist. Thread A still has the lock on obj1 and is waiting for the lock on obj2. Thread B has the lock on obj2.

(d) The main thread and threads A and B exist. Thread A still has the lock on obj1 and is waiting for the lock on obj2. Thread B has the lock on obj2 and is waiting for the lock on obj1.

(e) A deadlock.

Ex. 4:

```
/*
 * Ex. 4.a We only synchronize inside the 'receiveTask' of Secretary. We could synchronize the whole method, but it is
 * always a good idea to restrict the synchronization as much as possible. In this case it makes no difference...
 */

class Task {
        String task;
        public Task(String task) { this.task = task; }
        public String toString() { return task; } }

class Boss extends Thread {
        boolean running = true;
        Secretary secretary;
        Boss(Secretary secretary) { this.secretary = secretary; }

        // ex 4.a
        public void run() {
                while(running) {
                        try { sleep( 500 + (long)(Math.random()*2000) );        }
catch(InterruptedException e){}
                        secretary.receive(new Task("(boss) write a letter"));

                        try { sleep( 500 + (long)(Math.random()*2000) );        }
catch(InterruptedException e){}
                        secretary.receive(new Task("(boss) clean my desk"));

                        // ex 4.c
                        if(secretary.tasks.size() > 7) {
                                System.out.println("giving flowers");
                                secretary.receive(new Flowers());
                        }
                }
        }
}

class Student extends Thread {
        boolean running = true;
        Secretary secretary;
        Student(Secretary secretary) { this.secretary = secretary; }

        // ex 4.a
        public void run() {
                while(running) {
                        try { sleep( 500 + (long)(Math.random()*2000) );        }
catch(InterruptedException e){}
                        secretary.receive(new Task("(student) Do my homework"));

                        try { sleep( 500 + (long)(Math.random()*2000) );        }
catch(InterruptedException e){}
                        secretary.receive(new Task("(student) proof-read my assignment")); }

                }
        }

class Secretary extends Thread {
        long taskTimeLength = 1000;
        boolean running = true;
```

```
        java.util.ArrayList<Task> tasks = new java.util.ArrayList<Task>();

        public void receive(Task task) {
                // ex 4.a
                synchronized(this) {
                        tasks.add(task);
                        notify(); // ex 4.b ... wake up the secretary
                }
        }

        // ex 4.c
        public void receive(Flowers f) {
                new SpeedUpThread( 300, 10000 ).start();

        }


        public void run() {
                while(running) {
                        if(tasks.size() > 0) {                          // If we assume a
precondition that there is only one secretary and that no other threads remove tasks,
                                Task t = tasks.remove(0);               // it is ok not to
syncronize these two lines.
                                System.out.println("working on '" + t + "'..");
                                try { Thread.sleep( taskTimeLength );
}catch(InterruptedException e){}
                                System.out.println(tasks.size() + " tasks left...");
                        }
                        else {  // ex 4.b
                                try {
                                        synchronized(this) { wait(); } // we must gain
control of the monitor in order to be able to wait
                                } catch(InterruptedException e){}
                        }
                }
        }


        // ex 4.c This thread temporarily speeds up the tempo of the secretary
        class SpeedUpThread extends Thread {
                long newSpeed, duration;
                SpeedUpThread(long newSpeed, long duration) {
                        this.newSpeed = newSpeed;
                        this.duration = duration;
                }

                public void run() {
                        long oldSpeed = taskTimeLength;
                        taskTimeLength  = newSpeed;
                        try { sleep(duration); } catch(InterruptedException e) {}
                        taskTimeLength = oldSpeed;
                }
        }


        public static void main(String[] args) {
                Secretary secr = new Secretary();
                Student s1 = new Student(secr);
                Boss b = new Boss(secr);
                secr.start();
                s1.start();
                b.start();

        } }


// ex 4.c - Maybe a bit artificial with an empty class, but it makes it easy to extend in
the future
class Flowers {}
```

Ex. 5:

```
public class MyTimer extends Thread {
        MyTimerListener timerListener;
        int timeBetweenCalls;
        int repetitions;

        public MyTimer (MyTimerListener timerListener, int timeBetweenCalls, int
repetitions){
                this.timerListener = timerListener;
                this.timeBetweenCalls = timeBetweenCalls;
                this.repetitions = repetitions;
        }

        public void run(){
                int i=0;
                try{
                        while(i<repetitions){
                                this.sleep(timeBetweenCalls);
                                timerListener.timeAction();
                                i++;
                        }
                } catch(InterruptedException e){ System.out.println("Interrupted!"); }
        }

        public static void main( String[ ] args ){

                TimeUserX t1 = new TimeUserX();
                TimeUserO t2 = new TimeUserO();

                MyTimer timer1 = new MyTimer(t1,500,25);
                MyTimer timer2 = new MyTimer(t2,3000,3);
                timer1.start();
                timer2.start();
        }
}

public class TimeUserO extends Thread implements MyTimerListener {
        public void timeAction() {
                System.out.println("O");
        }
}

public class TimeUserX extends Thread implements MyTimerListener {
        public void timeAction() {
                System.out.println("X");
        }
}
```