## Exercise-1, day-1

This exercise should demonstrate the very basics of Java Thread Programming (Start/stop Threads, Runnable and Sleep)

Write a program that creates and starts 3 different threads.

Task-1 : Should calculate and print the sum of all numbers from 1 to 1 billion (milliard)

Task-2 : Prints the numbers from 1 to 5. Should Pause (sleep) 2 seconds between each output

Task-3 : Prints all numbers from 10 and up. Should pause 3 seconds between each output

The program should stop task-3 after 10 seconds

* Hint-1: To calculate the sum in task-1, use the data type long.
* Hint-2: Let the main thread sleep in 10 seconds, after having started Task-3.
* Hint-3: You can stop Task-3, by controlling the loop in its run-method with a boolean variable which can be changed by the main-thread

## Exercise-2, day-1 (demonstrating the concept Responsiveness)

Exercise that shows how we, **also on single-kernel systems**, can achieve responsiveness using threads.

Clone this project (includes code also for day2 + friday and run BallDemo.  
<https://github.com/cph-dat-sem2/thread-samples.git>

What is the obvious problem with this solution, and how will you describe the problem (a word from the lecture, with NON- in front of it)

Hint: You can stop the program via your IDE’s run-method.

Solve the problem by rewriting the Ball class into a thread so that many balls can be started (as shown in the figure), stopped, and the program can be stopped when we press the red cross.

If you solved the problem, then what have you achieved (a term from today's lecture)

## Exercise-3 - Starting/Stopping/interrupting a thread

class WorkerCancelable implements Runnable{

boolean keepRunning = true;

Thread thread;

public void run() {

thread = Thread.currentThread();

while(keepRunning){

System.out.println("Hello");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

public void cancel() {

keepRunning = false;

}

}

public class ThreadExercise3 {

public static void main(String[] args) throws InterruptedException {

WorkerCancelable worker = new WorkerCancelable();

Thread t = new Thread(worker);

t.start();

System.out.println("DONE");

}

}

**a)** Take a look at the code above, and see whether you anticipate what it does (prints) WITHOUT actually executing the code

Hint: If you did run the code, you will probably have to go to the *Run menu* to stop the program ;-)

**b)** Come up with the necessary changes (WITHOUT running the code) required for the code to print this, when executed:

Hello (wait a second)

Hello (wait a second)

Hello (wait a second)

DONE

**c)** If not already done, copy the code into a project in your IDE, and verify your predictions above

**d)** Interrupting a Thread

In your run method, change the sleep time from 1 second to 60 seconds

Add the necessary changes to print out a single “Hello” followed by a “Done” after 4 seconds.

**Hint:** Interrupting the thread will unblock it, figure out where to do that.