CODEGYM

Java developer in 12 months

Mentor-supported training program from Codegym

MODULE 2. JAVA CORE

Lesson #11
Introduction to threads

CODEGYM

Lesson plan

- Multithreading or multistream?
- Creating and starting new threads
- sleep() method
- Stopping a thread
- Interrupt() method

CODEGYM

Multithreading or multistream?

Threads (or, as they are sometimes incorrectly called, streams) are used when independent parts of a program need to perform some amount of work in parallel with other parts.

How it all works deep inside processors

If a computer has one processor (single-core / single-threaded), then it can only execute one instruction at a time.

Therefore, the processor is constantly switching between threads. It switches to a new thread, executes some of its commands, then switches to the next thread, executes some of its commands, and so on. But since switching between threads occurs hundreds of times per second, from the outside it seems that all the threads work simultaneously.

For multi-core/multi-threaded processors, threads can run concurrently.

CODEGYM

Creating and starting new threads

To generate a new thread:

1. Create an object of the Thread class (a thread).
2. Pass a method to it, which needs to be executed.
3. Call the start method of the created Thread object.

Code	Description

`class Printer implements Runnable{`
`  public void run(){`
`  System.out.println("I'm a printer");`
`  }`
`}`
	A class, which implements the **Runnable** interface.

`public static void main(String[] args){`
`  Printer printer = new Printer();`
`  Thread childThread = new Thread(printer);`
`  childThread.start();`
`}`
	1. Created an object of the Printer class, which has the run method.
	2. Created an object of the Thread class, and passed the printer object to its constructor, whose run() method needs to be started.
	3. Started the thread by calling the **start()** method**.**

Small Java programs usually consist of a single thread called the main thread. But larger programs often run additional threads, also called "child threads".

The main thread executes the main method and exits. An analogue of such a main method for child threads is the run method of the Runnable interface.

The Runnable interface contains a single abstract method - void run(). The Thread class has a Thread(Runnable runnable) constructor to which any object that implements the Runnable interface can be passed.

You must implement the Runnable interface in your class, then implement the run method. The work of a new thread will begin with a call to this method.

CODEGYM

You can combine it all in one class. The Thread class implements the Runnable interface, and it is enough to simply override its run method:

Another approach to creating a new thread	Description

`class Printer extends Thread{`
`private String name;`
`public Printer(String name){`
`  this.name = name;`
`  }`
`public void run(){`
`  System.out.println("I'm " + this.name);`
`  }`
`}`
	Extend from the Thread class, which implements the **Runnable** interface, and override the run() method.

`public static void main(String[] args){`
`  Printer printer = new Printer("John");`
`  printer.start();`
 	
`  Printer printer2 = new Printer("Hannah");`
`  printer2.start();`
`}`
	Create two threads, each of which is based on its own object of the **Printer** type.

CODEGYM

sleep() method

The sleep method is declared as a static method of the Thread class, i.e. it is not attached to any object. The purpose of this method is to make the program "fall asleep" for a while. Here's how it works:

Code	Description

`public static void main(String[] args){`
 	
`  Thread.sleep(2000);`
 	
`}`
	Program starts.
	
	Then pauses for 2 seconds (2 000 miliseconds), and ends.

This method is often used in child threads, when an action needs to be performed repeatedly, but not too frequently.

CODEGYM

Stopping a thread

It is not possible to stop a thread, it can only stop itself.

But you can give the thread a signal, tell it somehow that there is no more work to do, and it needs to terminate.

Also, just as the main thread terminates when the main method exits, in order for a child thread to terminate, it must finish executing the run method.

CODEGYM

interrupt() method

The Thread class includes the isInterrupted variable and the interrupt() termination method.

Code	Description

class Clock implements Runnable {
  public void run() {
  Thread current = Thread.currentThread();
  while (!current.isInterrupted()) {
  try {
  Thread.sleep(1000);
  } catch (InterruptedException e) {
  e.printStackTrace();
  current.interrupt();
  }
  System.out.println("Tik");
  }
  }
}
	Many threads can call the run() method of a single object, so the Clock object in its run() method receives the object of a thread that called it (the «current thread»).
	
	The Clock class will output the word "Tik" once per second, while the isInterrupted variable of the current thread equals false.
	
	Once the **isInterrupted** variable becomes equal to **true**, the **run** method will terminate.

public static void main(String[] args) throws Exception {
  Clock clock = new Clock();
  Thread clockThread = new Thread(clock);
  clockThread.start();
  Thread.sleep(10000);
  clockThread.interrupt();
}
	The main thread starts the child thread – the clock, which must work forever.
	
	Waits 10 seconds and cancels the job, by calling the interrupt() method.
	
	The main and the clock threads terminate.

There is no guarantee that a thread can be stopped. It can only stop itself.

The sleep method has an automatic check for the isInterrupted variable.

If a thread calls the sleep method, then this method will check whether the isInterrputed variable is set to true for the current thread (which called it).

And if it is set to true, then the method will not sleep, and will throw an InterruptedException.

CODEGYM

Homework

Module 2

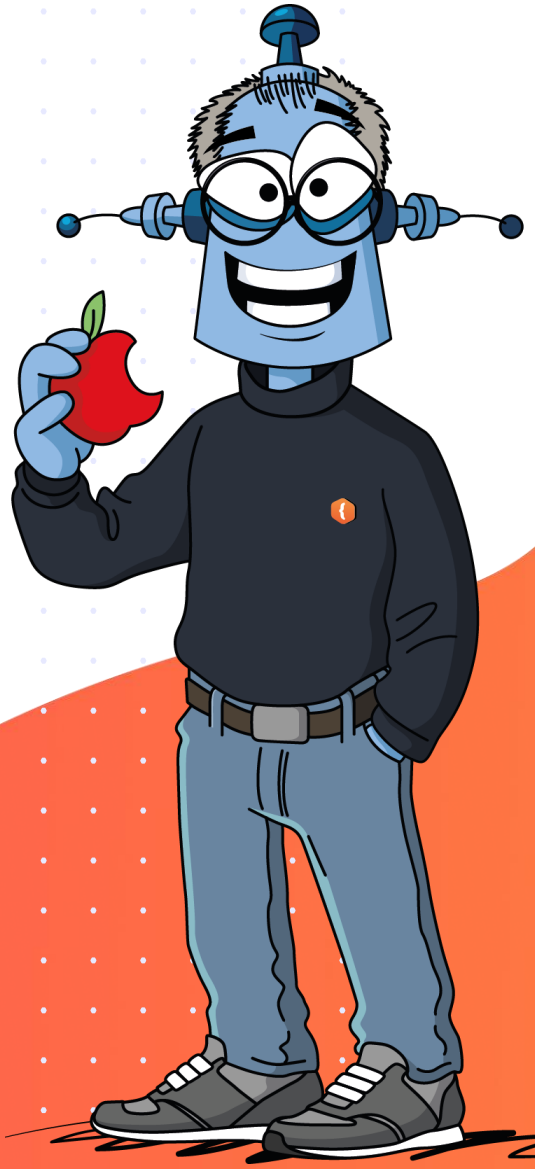
Level 11

Introduction to threads

MODULE 2. JAVA CORE

CODEGYM

Questions and Answers



MODULE 2. JAVA CORE