Interrupting a loop

1. `break` statement

Let's take a look at an example from the previous lesson:

Code	Explanation

`Scanner console = new Scanner(System.in);`
`boolean isExit = false;`
`while (!isExit)`
`{`
`  String s = console.nextLine();`
`  isExit = s.equals("exit");`
`}`
	The program will read a line from the keyboard, until you enter `"exit"`.

The program reads lines from the console until the word `exit` is entered. If you enter this word, then the `isExit` variable becomes `true`, the loop condition `!isExit` will be `false`, and the loop will end."

Java has a special `break` statement that lets you simplify such logic. If a `break` statement is executed inside a loop, then the loop ends immediately. The program will start executing the statement that follows the loop. The statement is very brief:

```
break;
```

Here's how you can use the `break` statement to rewrite the example we just discussed:

Code	Explanation

`Scanner console = new Scanner(System.in);`
`while (true)`
`{`
`  String s = console.nextLine();`
`  if (s.equals("exit"))`
`  break;`
`}`
	The program will read a line from the keyboard, until you enter `"exit"`.

2. continue statement

But `break` isn't the only Java statement that lets you control a loop's behavior. Java also has the `continue` statement. If you execute a `continue` statement inside a loop, the current iteration of the loop will end ahead of schedule.

Executing the loop body once is called an iteration of the loop.
The `continue` statement interrupts the current iteration of the loop, but unlike the `break` statement, it doesn't end the loop itself. The statement is also brief:

```
continue;
```

The `continue` statement is super convenient in a loop if we want to 'skip' execution of the loop body in certain situations.

Task: We want to write a program that prints numbers from `1` to `20` but skips numbers that are divisible by `7`. This is what this code might look like.

Code	Explanation

`int i = 1;`
`while (i <= 20)`
`{`
`  if ( (i % 7) == 0 )`
`  continue;`
`  System.out.println(i);`
`  i++;`
`}`
	The program displays numbers from `1` to `20`. If the number is divisible by `7` (the remainder of division by `7` is `0`), then we skip displaying the number.

Actually, this code will not work, because `i` will be forever stuck at the number `7`. After all, the `continue` statement skips two other statements: `System.out.println(i)` and `i++`. As a result, once we reach the value `7`, the variable `i` will stop changing and we'll be in an infinite loop.

We wrote the code this way on purpose to illustrate this very common mistake. How do we fix it?

There are two options here:

Option 1: change `i` before executing `continue`, but after `i % 7`

Option 2: always increment `i` at the beginning of the loop. But then `i`'s starting value must be `0`.

Option 1	Option 2

`int i = 1;`
`while (i <= 20)`
`{`
`  if ( (i % 7) == 0 )`
`  {`
`  i++;`
`  continue;`
`  }`
 	
`  System.out.println(i);`
`  i++;`
`}`
	`int i = 0;`
	`while (i < 20)`
	`{`
	`  i++;`
	`  if ( (i % 7) == 0)`
	`  continue;`
	`  System.out.println(i);`
	`}`