COURSEWARE

Iteration

Contents

Overview

Iterations (or **loops**) allow us to repeat a block of code until a specified condition is met.
Each loop will check for a condition and as soon as that condition is broken, the code will stop running that loop.

There are three main loop types in Java:

- `while()`-loop
- `do()-while()`-loop
- `for()`-loop

Whilst all three provide very similar functionality, their implementation is different as they have different syntax and different condition checking.

Tutorial

`while()`-loop

A `while()`-loop is a loop type that primarily works with a boolean value, meaning that whilst the boolean value meets a specified condition the code will loop.
Think of a `while()`-loop as an `if()`- statement that doesn't stop executing until the condition is no longer met.

```
public class Iteration {

    public static void main(String[] args) {

        int catCount = 0;
        boolean notEnoughCats = true;

        while(notEnoughCats) {
            System.out.println("Another cat");
            catCount++;

            if(catCount > 273) {
                notEnoughCats = false;
            }
        }

        System.out.println("Too many cats what do I do");
    }
}
```

In the above example we are declaring and initialising a variable of type `int` with the name `catCount` and a value of `0`.

We are also declaring and initialising a variable of type `boolean` with the name `notEnoughCats` and a value of `true`.

In the main method we are iterating over a block of code while `notEnoughCats` is `true`, once this becomes false the loop will stop executing.

If `notEnoughCats` is `true`, we print `"Another cat"` and add 1 to `catCount`.

Within the loop we have a conditional that checks if `catCount` is greater than `273`, if it is then we set `notEnoughCats` to `false`.

Once we set `notEnoughCats` to `false`, the `while()`-loop's condition returns `false`, and so Java stops executing the loop.

`do()-while()`-loop

A `do()-while()`-loop is every similar to a `while()`-loop, the major difference is that the block inside the loop block will always run once, because the condition is checked after the code is ran.

```
public class Iteration {

    public static int playCount = 0;
    public static boolean playing = true;

    public static void main(String[] args) {
        do {
            System.out.println("Playing");
            playCount++;

            if(playCount > 10) {
                playing = false;
            }
        } while(playing);

        System.out.println("Game Over!");
    }
}
```

In the above example we are declaring and initialised a variable of type `int` with the name `playCount` and a value of `0`.

We are also declaring and initialising a variable of type `boolean`, with the name `playing` and a value of `true`.

In the main method we a running a `do()-while()`-loop that prints `"Playing"`, then adds 1 to the variable `playCount`, then runs the conditional checking if `playCount` is greater than `10`.

If `playCount` is greater than `10`, then we go into the `if()`-statement and set `playing` to `false`.

Java will then run the conditional on the `do()-while()`-loop and check if `playing` is still `true`, if it is then the loop will run again, if it is not then Java will stop executing the loop.

Finally once the loop has stopped we print `"Game Over!"`.

`for()`-loop

The loops we created in the other sections don't need a counter as they work with boolean values and when a condition is no longer met, the loop stops.

Iteration works specifically with a counter rather than a boolean value, for this we use a `for()`-loop.
`for()`-loops have three main features; the initialisation, the condition, and the iterator.

```
public class Iteration {

    public static void main(String[] args) {
        for(int i = 0; i < 10; i++) {
            System.out.println("Hello There!");
        }
    }
}
```

In the above example we have three statements within the `for()`-loop.

The first one - `int i = 0;` - is initialising the counter for the iteration as an integer with the name `i` and the value `0`.

Best practice for the name of your counter variable is to use `i`.

The second statement `i < 10;` is our condition, and tells Java to run the code within the `for()`-loop until `i` is greater than or equal to `10`.

The third statement `i++` is simply telling Java to increment our counter, in this case `i`, by 1 after it executes the last line of code within our for loop.

So this example will end up printing `"Hello There!"` 10 times.

Transfer and Control Statements

We can manipulate the flow of code through a number of keywords:

- **Break**: Break statements break out of the current code block, in loops this means that Java will skip the rest of the loop and move onto executing the rest of the code.
- **Continue**: Continue statements break out of the current iteration of a code block, in loops this means that Java will skip the current iteration of the code block and move onto the next.
- **Return**: Return statements are used in methods to return values according to the methods return type, ending the method.

```
public class Iteration {

    public static void main(String[] args) {
        for(int i = 0; i < 10; i++) {
            if(i == 2) {
                continue;
            }
            if(i == 7) {
                break;
            }
            System.out.println(i);
        }
    }
}
```

In the above example we are running a `for()`-loop starting with `i = 0`, with the condition `i < 10`, and incrementing `i` by 1 after each iteration.

Each iteration will run the first `if()`-statement and check if `i` is equal to `2`.

If it is, then because of the continue keyword Java will skip this iteration and move onto the next one.

If `i` is not equal to `2`, then the iteration will move onto the second `if()`-statement and check if `i` is equal to `7`.

If `i` does equal `7`, then, because of the `break` keyword, Java will skip the rest of the loop and move onto the rest of the code.

If `i` is not equal to 7, then Java will print the value of `i` for us.

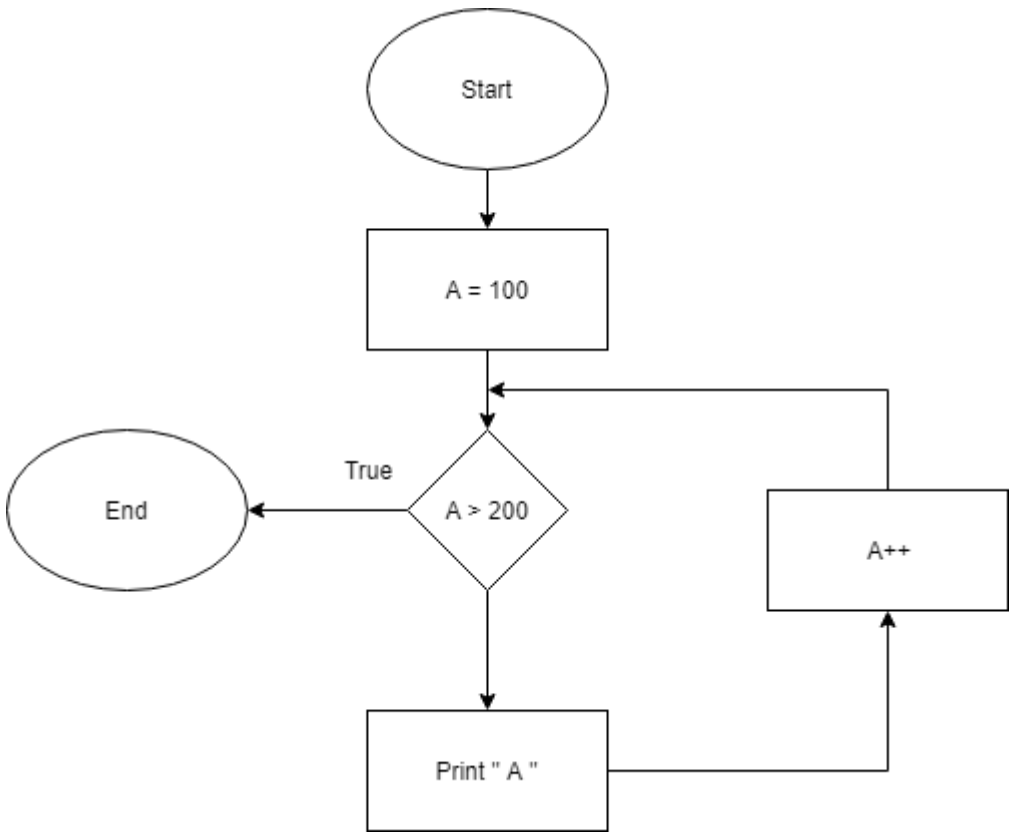
This example will give us the following output:

```
0
1
3
4
5
6
```

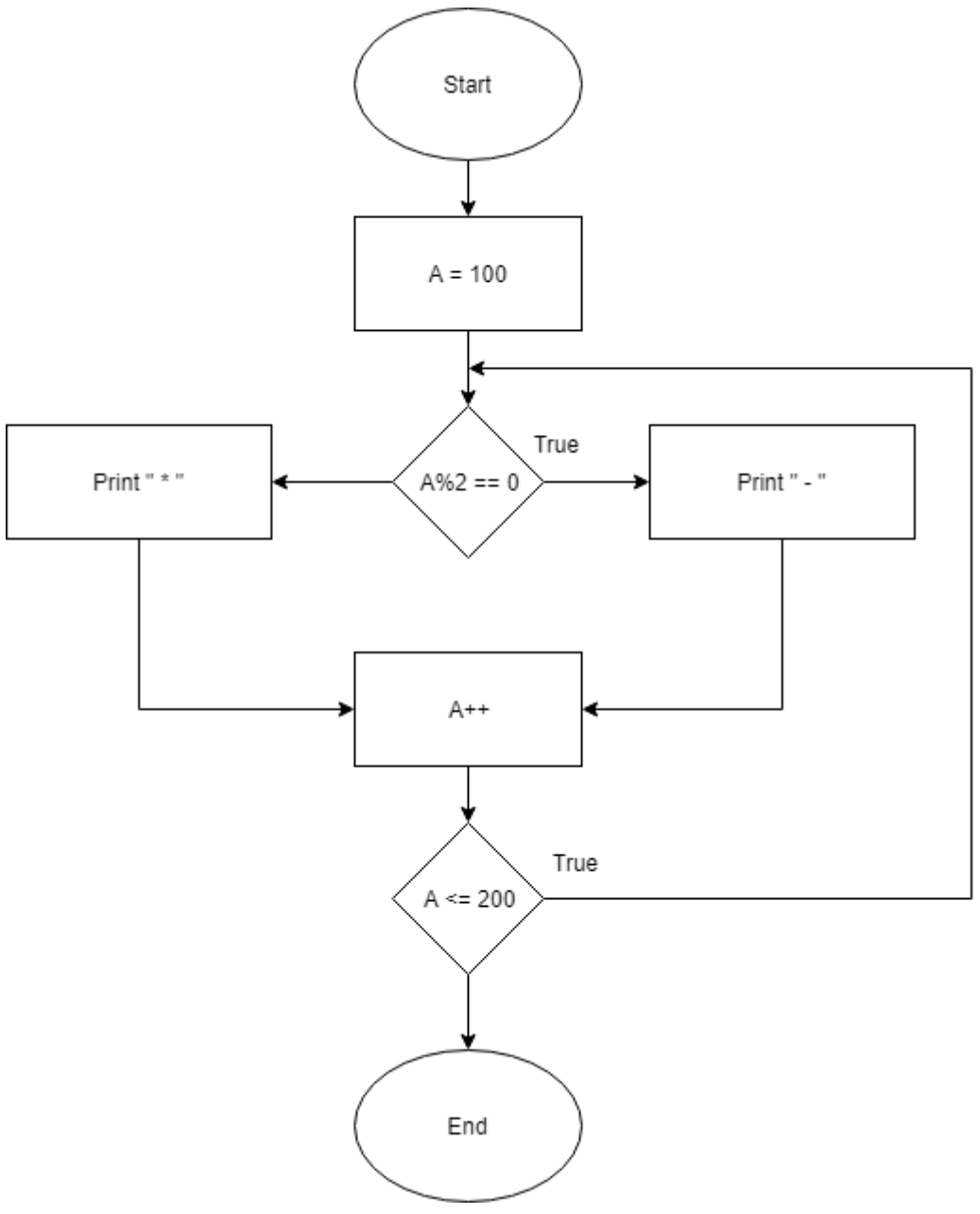
Exercises

Flowcharts

1. Recreate the following flowchart as a project.



2. Recreate the following flowchart as a project.



3. Create a method that can print out the numbers 1-10 10 times each.

4. If you have used a `while()`-loop at any point in these exercises, replace them with `for()`-loops. **Remember you should use a `for()`-loop when you know when the iteration should end.**

5. Create a method to print the numbers 1 to 10 as many times as the value of that number.
   For example; you will print 1 once, and 10 ten times.

Coins

1. Given a cost and an amount, work out the change given in specific coinage.
   For example; the cost is £4.58 and the customer pays with a £20 note so as change they receive:

- 1 £10 note
- 1 £5 note
- 2 20p's
- 1 2p