Operators and flow control

Arithmetic operators
If statements
Scope
Relational operators
Conditional operators

Arithmetic operators

We've already seen some operators. For example, to concatenate strings, we did the following:

```
"Hello " + name + "!"
```

Arithmetic operators

When dealing with numbers we have the following operators.

Operator	Description	Example
+	Addition	int a = 1 + 2;
-	Subtraction	int a = 1 - 2;
*	Multiplication	int a = 2 * 2;
/	Division	float a = 2 / 3;
%	Modulus	int a = 10 % 3;
++	Increment	a++;
--	Decrement	a--;

Arithmetic operators

Modular arithmetic is a system to calculate the remainder from a division.

X	% 2
0	0
1	1
2	0
3	1
4	0
5	1
6	0
7	1
8	0

X	% 3
0	0
1	1
2	2
3	0
4	1
5	2
6	0
7	1
8	2

X	% 4
0	0
1	1
2	2
3	3
4	0
5	1
6	2
7	3
8	0

Arithmetic operators

```
public static void main(String[] args) {
    // Gather input from user
    Scanner console = new Scanner(System.in);

    System.out.printf("Enter a first number: ");
    float first = console.nextFloat();

    System.out.printf("Enter a second number: ");
    float second = console.nextFloat();

    // Do some calculations
    float sum = first + second;
    float diff = first - second;
    float ratio = first / second;
    float product = first * second;

    // Print to screen
    System.out.println("Sum: " + sum);
    System.out.println("Diff: " + diff);
    System.out.println("Ratio: " + ratio);
    System.out.println("Product: " + product);
}
```

Calculations using basic operators

String concatenation using +

Comparison and assignment

In Java (and many other languages), we use the **=** operator to **assign**, not **compare**

```
public static void main(String[] args) {
    int x = 5;
    x = x + 1;

    System.out.println("The value of x is now " + x);
}
```

```
The value of x is now 6
```

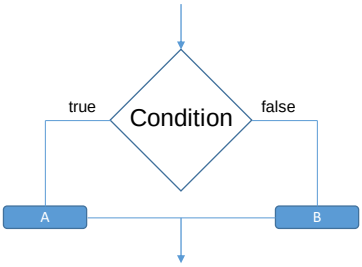
Comparisons are instead performed using the **==** operator

If Statements

if-then-else statements

Using the if statement we can control the execution flow.

”if this is **true**, then I will do this. Otherwise (**else**), I will do this instead”.




```
if (condition) {
    // Do something A
} else {
    // Do something else B
}
```

Anatomy of the if statement

Decomposing the if statement:

The condition is evaluated.
If it's **true…**

```
if (condition) {

    // Do something

} else {

    // Do something else

}
```

…run the code in this scope

… otherwise, run the code in this scope.
Note: Else is optional

Nested if statements

If statements can be nested to any level, but be cautious. Nesting too many if statements is generally a bad idea.

if within **if** "**nested**"

```
if (isThirsty) {
    if (hasBeer) {
        // I'm thirsty and I have a beer!
        // Awesome!
    } else {
        // I'm just thirsty :-(
    }
} else {
    // I'm not thirsty.
}
```

Why is having too many nested if-statements a bad idea?

Comparison and flow control (1)

Using if statements, we can control the flow of our application.

In this example, we're reading a pin code from the user:

```
public static void main(String[] args) {
    // ...
    Scanner in = new Scanner(System.in);

    System.out.printf("Enter your 4 digit pin code: ");
    int secret = in.nextInt();
}
```

```
Enter your 4 digit pin code: 1234
Correct pin code. Welcome!
```

Comparison and flow control (2)

Using the **comparison operator** (**==**), we always get a value that is either **true** or **false**

Either it **is**, or it is **not**.

```
public static void main(String[] args) {
    // ...
    Scanner in = new Scanner(System.in);

    System.out.printf("Enter your 4 digit pin code: ");
    int secret = in.nextInt();

    if (secret == 1234) {
        System.out.printf("Correct pin code. Welcome!\n");
    } else {
        System.out.printf("Wrong pin code!\n");
    }
}
```

Tip: Don't write actual password systems this way. ☺

Comparison and flow control (3)

While the **==** operator means ”is equal to”, you also have the option of using the **inequality operator, !=**, read as ”is not equal to"

These two if-statements are the same:

```
if (secret == 1234) {
    System.out.printf("Correct pin code. Welcome!\n");
} else {
    System.out.printf("Wrong pin code!\n");
}
```

```
if (secret != 1234) {
    System.out.printf("Wrong pin code!\n");
} else {
    System.out.printf("Correct pin code. Welcome!\n");
}
```


Comparison and flow control (4)

When dealing with pure boolean values, you don’t even have to use the **==** operator:

We could have written **canHasCheezBurger == true**, but it would be a waste of character since the statements are equivalent.

```
public static void main(String[] args) {
    boolean canHasCheezBurger = true;

    if (canHasCheezBurger) {
        System.out.println("You can has cheezburger!");
    } else {
        System.out.println("No!");
    }
}
```

Comparison and flow control (5)

Since we don't use the **==** operator, do we need to use the **!=** operator when negating?

No! Simply use the **negating operator**, **!**

Inverted condition

```
public static void main(String[] args) {
    boolean canHasCheezBurger = true;

    if (!canHasCheezBurger) {
        System.out.println("No!");
    } else {
        System.out.println("You can has cheezburger!");
    }
}
```

Same logic, but inverted.

Scope

Notice all the curly braces so far?

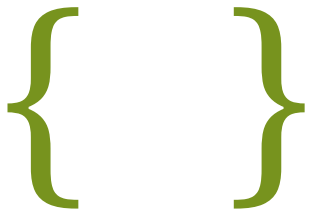
```
public static void main(String[] args)
{
    // This is a scope

    boolean tired = true;

    if (tired) {
        // This is a nested scope
    } else {
        // This is another nested scope
    }
}
```

A pair of braces define a **scope**. The lifetime of variables are tied to these scopes.

Scope



Scope

Java is **lexically scoped**. In this code, **answer** and **error** are only visible inside the function **main**.

```
public static void main(String[] args) {
    // This is a scope

    int answer = 42;
    boolean error = true;

    if (error) {
        // This is a nested scope
        answer = 0;
    }
    else {
        // This is a nested scope
        answer = answer+ 1;
    }
} // "answer" and "error" will cease to exist here
```

answer is available in sub-scopes

Scope

Likewise, any variable declared in a nested scope is visible **within that scope** (and its own nested scopes)

```
public static void main(String[] args) {
    // This is a scope

    int answer = 42;
    boolean error = true;

    if (error) {
        String greeting = "Hello";
    } // "greeting" will cease to exist here
    else {
        String farewell = "Goodbye";
    } // "farewell" will cease to exist here

} // "answer" and "error" will cease to exist here
```

Only exists within the scope

Scope

It is valid to have the same variable name in different scope:

```
if (error) {
    String message = "We got some error!";
    System.out.println(message);
}
else {
    String message = "All OK!";
    System.out.println(message);
}
```

The same variable name but in different scopes.

Relational operators

< > <= >=

Relational operators (1)

Just making comparisons on whether or not two values are equal (==) is often not enough.

We might want to check if a number is in a given range, or if a text string is of a certain length, etc.

```
int x = 10;

// ...

if (x > 5) {
    // x is greater than 5
}

if (x < 5) {
    // x is less than 5
}
```

```
int x = 10;

// ...

if (x >= 5) {
    // x is greater than or equal to 5
}

if (x <= 5) {
    // x is less than or equal to 5
}
```

Flow control and relational ops (1)

Let's write a program that:

Asks the user to input a number between 5 and 10

In other words, we want to:

- Read input from the console
- Evaluate the result and take different actions based on the selection (if)

```
Enter number between 5 and 10: 11
The number 11 does NOT lie between 5 and 10.
```


Flow control and relational ops (2)

1. Read input from the console

```java
public class Main {
    public static void main(String[] args) {
        // Output message on console
        System.out.print("Enter number between 5 and 10: ");

        // Read
        Scanner in = new Scanner(System.in);
        int num = in.nextInt();
    }
}
```

Flow control and relational ops (3)

2. Evaluate the result and take different actions based on the selection

```java
public class Main {
    public static void main(String[] args) {
        // Output message on console
        System.out.print("Enter number between 5 and 10: ");

        // Read
        Scanner in = new Scanner(System.in);
        int numberInput = in.nextInt();

        if (numberInput >= 5) {
            if (numberInput <= 10) {
                System.out.println("The number " + numberInput +
                                   " lies between 5 and 10.");
            } else {
                System.out.println("The number " + numberInput +
                                   " does NOT lie between 5 and 10.");
            }
        } else {
            System.out.println("The number " + numberInput +
                               " does NOT lie between 5 and 10.");
        }
    }
}
```


Flow control and relational ops (4)

Output

```
Enter number between 5 and 10: 11
The number 11 does NOT lie between 5 and 10.
```

Code refactoring (1)

Let's look at the last code block we added:

```
if (numberInput >= 5) {
     if (numberInput <= 10) {
         System.out.println("The number " + numberInput +
                            " lies between 5 and 10.");
     } else {
         System.out.println("The number " + numberInput +
                            " does NOT lie between 5 and 10.");
     }
 } else {
     System.out.println("The number " + numberInput +
                        " does NOT lie between 5 and 10.");
 }
```

Code refactoring (2)

Curly braces aren't required for one-statements code blocks.

```
if (numberInput >= 5)
    if (numberInput <= 10)
        System.out.println("The number " + numberInput +
                           " lies between 5 and 10.");

    else
        System.out.println("The number " + numberInput +
                           " does NOT lie between 5 and 10.");
else
    System.out.println("The number " + numberInput +
                       " does NOT lie between 5 and 10.");
```

Some people prefer to use braces to define the scope explicitly, while others do not.

What are the pros/cons of not using { } ?

Code refactoring (3)

Which if statement does the **else** belong to?

```
if (numberInput >= 5)
    if (numberInput <= 10)
        System.out.println("The number " + numberInput +
                           " lies between 5 and 10.");
else
    System.out.println("The number " + numberInput +
                       " does NOT lie between 5 and 10.");
```

```
if (numberInput >= 5)
    if (numberInput <= 10)
        System.out.println("The number " + numberInput +
                           " lies between 5 and 10.");
    else
        System.out.println("The number " + numberInput +
                           " is more than 4 but is NOT under 10.");
```


Curly Braces: Different Styles

There are two ways to write braces in Java.

```
if (numberInput >= 5) {
    // Code
    // More Code
}

if (numberInput >= 5)
{
    // Code
    // More Code
}
```

The first style by far the most common amongst Java programming and we **recommend** you use this when writing code in Java.

If, despite this, you choose another brace style, the most important thing is to be **consistent** with the way you write your code. Always use the same brace style throughout your software.

Conditional operators

&& ||

Conditional operators (1)

Instead of using nested if statements, a viable option in this case is to use the **conditional operator &&**

```
if (numberInput >= 5) {
    if (numberInput <=10) {
        // Omitted
    } else {
        // Omitted
    }
} else {
    // Omitted
}
```

If numberInput is **greater than or equal to 5 AND** if numberInput is **less than or equal to 10**, then…

```
if (numberInput >= 5 && numberInput <= 10) {
    // Omitted
} else {
    // Omitted
}
```

Conditional operators (2)

The entire application after refactoring:

```
public static void main(String[] args) {
    // Output message on console
    System.out.print("Enter number between 5 and 10: ");

    // Read
    Scanner in = new Scanner(System.in);
    int numberInput = in.nextInt();

    if (numberInput >= 5 && numberInput <= 10) {
        System.out.println("The number " + numberInput +
                           " lies between 5 and 10.");
    } else {
        System.out.println("The number " + numberInput +
                           " does NOT lie between 5 and 10.");
    }
}
```

Conditional operators (3)

Another useful conditional operator is the **or** operator

||

Instead of using the **and** operator:

```
if (numberInput >= 5 && numberInput <= 10) {
    // Within range
} else {
    // Not within range
}
```

We could use the **or** operator to achieve the same thing:

```
if (numberInput < 5 || numberInput > 10) {
    // Not within range
} else {
    // Within range
}
```

Other useful operators

Other useful operators

We've seen assignment expressions such as this before:

```
int num = 5;
num = num + 1; // Increase num by 1

// The following line will output "6" to the console:
System.out.println(num);
```

As an alternative, we could use the **increment** operator, **++**

```
int num = 5;
num++; // Increase num by 1

// The following line will output "6" to the console:
System.out.println(num);
```

Other useful operators

The position of the **++** operator (**before** or **after** the statement) matters in some situations:

```
int num = 5;

// The following line will
// output "6" to the console:

System.out.println(++num);
```

The variable **num** applies the **++** operator and is increased to **6**. After that, it's printed.

```
int num = 5;

// The following line will
// output "5" to the console:

System.out.println(num++);
```

The variable **num** is printed and **then** applies the **++** operator, increasing it to 6.

Other useful operators

Lastly, there is an alternative to statements like the code below, when we want to increase or decrease a number by more than 1.

```
int num = 5;
num = num + 4;

System.out.println(num);
```

As an alternative we can write it using the compound assignments statement:

```
int num = 5;
num += 4; // Instead of using num = num + 4

System.out.println(num);
```

compound assignments.

Other compound assignment operators:

Operator	Instead of
`num *= x;`	`num = num * x;`
`num /= x;`	`num = num / x;`
`num -= x;`	`num = num - x;`
`num %= x;`	`num = num % x;`

Exercise 5

Let's do exercise 5