From JShell to an IDE

Welcome to Section 4.

In this section, we're going to make the switch from JShell to an Integrated Development Environment.

Why do we need an Integrated Development Environment (ID

An IDE is the easiest, least error-prone way to develop, manage and deploy Java classes. It provides many benefits to developers, including

- increased productivity,
- code completion,
- refactoring of code,
- debugging tools,
- version control,
- and team development, just to name a few. And don't worry if you don't understand some of these terms, they will make sense as we progress through the course.

Why do we need an Integrated Development Environment (ID

I like to think that the jump from JShell to an IDE is like the jump from a typewriter to a word processing program on a modern computer.

It's unlikely that you will want to go back once you have experienced the benefits.

What is IntelliJ?

IntelliJ IDEA, is one of several, IDE's, available for Java.

It's also written in Java, developed by JetBrains, and simply known as IntelliJ.

JetBrains offers a free and open-source community edition.

Installing IntelliJ

You now need to install IntelliJ to continue with the course. Thankfully this is an easy thing to do.

I've taken the liberty of recording a separate IntelliJ installation video for Windows, Mac, and Linux. All you need to do is watch the relevant video for the operating system you are running on your computer. All three videos are in this section of the course.

Once you have installed IntelliJ, it needs some configuration. I've recorded a configuration of IntelliJ video which covers all three operating systems. Watch that video after you have installed IntelliJ, and then you will be ready to get started programming in the IntelliJ IDE.

Installing Intellij IDEA for Windows

For this course, we're going to use an IDE called **IntelliJ IDEA**. An **IDE**, or **Integrated Development Environment**, is the tool that we use to write programs.

The main feature is a text editor, that we type our code into. But an IDE also compiles and runs our programs for us. And as you'll see, it also warns us about errors in our code, before we run it.

The IDE we're going to use is called IntelliJ IDEA, created by a company called **JetBrains**. There are other IDEs available, and you can use any that you want for this course.

Installing Intellij IDEA for Windows

If you've already got an IDE that you like, and you're comfortable using, then it's fine to continue using it. Most modern IDEs provide the same, or very similar, features.

If you haven't used an IDE before, I suggest you use IntelliJ IDEA. That way your environment will work the same way as you'll see in the videos.

A reminder, that this video is for Windows - Check out the other videos in this section, if you are on a Mac, or running Linux.

It's time to download and install IntelliJ IDEA, which we need to write Java programs.

Configuring IntelliJ IDEA - WINDOWS, MAC and L

In this video, we want to configure IntelliJ Idea, so that we can actually start some java coding. This video is applicable whether you're running on Windows, Mac, or Linux.

So firstly, we need to link the JDK we installed, Oracle's JDK 17, with IntelliJ IDEA so that they work together. Basically, that means telling IntelliJ, where the JDK is installed, on the computer.

The **JDK** is effectively a **Software Development Kit**, or **SDK**. Whatever you call it, it contains the tools you need, to write programs. The **Java Development Kit** ,includes the tools that enables the computer to understand your java code, and to execute it. It also has a debugger, and we'll be seeing what that is and how to use it, when we've written a program to debug.

Hello World in IntelliJ

So, up until now, we've used the interactive shell included with Java, JShell, to write all of our Java code.

Now we'll start using IntelliJ, the Integrated Development Environment you just installed in the previous videos, to take another look at our first program, "Hello World".

If you haven't installed IntelliJ IDEA yet, please go back to the video for your operating system, and do that now, then come back and continue on with this lecture.

Naming Items in Java

Camel case is the practice of capitalizing the first letter of words in a name for readability, removing spaces or characters such as underscores between the words. Lower camel case only capitalizes the first letter of the second and subsequent words. Upper camel case, also known as Pascal case, capitalizes the first letter of the first word as well.

Identifier	Usage	Recommended	Example
Project Name	IntelliJ Field	Upper Camel Case	FirstJavaProject
Class Name	Java element	Upper Camel Case	NewClass
Method Name	Java element	Lower Camel Case	getData
Variable Name	Java element	Lower Camel Case	firstVariable

Examples are shown in this table. Note that project name is not a Java element, it's part of IntelliJ's configuration. More on this in a future video.

Access Modifiers

The **`public`** Java keyword is what's called an **access modifier**.

An access modifier allows us to define which parts of our code, or even someone else's code, can access a particular element.

```
public class FirstClass {
    public static void main(String[] args) {
        System.out.print("Hello World");
    }
}
```

Class Keyword

The **`class`** keyword is used to define a class. The class name will be the text following the keyword, so **`FirstClass`** in this case.

Notice the **left and right curly braces**, they are used to define the class code block, or class body.

```
public class FirstClass {
    public static void main(String[] args) {
        System.out.print("Hello World");
    }
}
```

What is a Method

A **method** is a collection of statements, one or more, that perform an operation.

We'll be using a special method called the main method, that Java looks for when running a program.

It's the entry point for any Java code, and Java looks for this main method to start and run the program.

You can also create your own methods, as you'll see later.

Your first IntelliJ Challenge

Instead of it printing "Hello World", print "Hello, Tim", or hello whatever your first name is.

if-then Statement

In the last video, we created our first class in Java using the IntelliJ IDE. So now it's time to get back to studying Java.

In this video, we're going to take a look at some more operators, but before that, I need to talk about the **if-then** statement first.

if-then statements in Java

The **if-then** statement is the most basic of all the control flow statements. It tells your program to execute a certain section of code, only if a particular test evaluates to **true**.

This is known as **conditional logic**.

Conditional Logic

Conditional logic uses specific statements in Java to allow us to check a condition, and execute certain code based on whether that condition (the expression) is **true** or **false**.

Let's see how this works in practice.

Assignment Operator (=)

The assignment operator assigns the value of an expression, to the variable to the left of the operator.

```
boolean isAlien = false;
```

So, **`isAlien`** is the variable in this case, and it's been set to false, which is the value of our expression.

Equality Operator (==)

The equality operator tests to see if two operands are considered equal, and returns a boolean value.

```
if (isAlien == false) {
```

So, here **`isAlien`** is being tested against the value false.

Best Practice Rule - Always use a Code Block for If-Then statements

```
boolean isAlien = true;
if (isAlien == false)
    System.out.println("It is not an alien!");
    System.out.println("And I am scared of aliens");
```

Instead of using the **if** statement as we can see here, we should instead use a code block.

The Code Block

A **code block** allows more than one statement to be executed, in other words, a block of code.

The format is:

```
if (expression) {
    // put one or more statements here
}
```

The Logical AND operator and the Logical OR operator

The **and** operator comes in two flavours in Java, as does the **or** operator.

&& is the Logical **and** which operates on **boolean** operands – Checking if a given condition is **true** or **false**.

The **&** is a bitwise operator working at the bit level. This is an advanced concept that we won't get into here.

The Logical AND operator and the Logical OR operator

Likewise || is the Logical **or**, and again it operates on **boolean** operands – Checking if a given condition is **true** or **false**.

The | is a bitwise operator, which is also working at the bit level.

And just like the bitwise **and** operator, we won't be using it as much as their logical counterparts.

We'll almost always be using the logical operators.

Difference Between the Assignment and Equal to Operators

```
int newValue = 50;
if (newValue = 50) {
    System.out.println("This is an error");
}
```

As you can see, we've used the assignment operator (one equal sign) in the if statement.

What we need to do, is to use the "equals to" operator (two equal signs).

Difference Between the Assignment and Equal to Operators

```
int newValue = 50;
if (newValue = 50) {
    System.out.println("This is an error");
}
```

This is what the code should look like:

```
int newValue = 50;
if (newValue == 50) {
    System.out.println("This is an error");
}
```

We're not assigning a value here, instead we want to test if the values are equal to each other.

The NOT Operator

The exclamation mark (!), or **NOT** operator, is also known as the Logical Complement Operator.

It can be used with a boolean variable, to test for the opposite value.

```
boolean isCar = false;
if (isCar) {
```

In the code above, we are testing if the value in is car is true. As you can see on the previous line, we assigned it to be false.

```
boolean isCar = false;
if (!isCar) {
```

If we use the "not" operator, we are testing for the opposite value of the is car variable. We assigned is car on the previous line to false, so not is car, would return true.

The NOT Operator

I'd generally recommend using the abbreviated form, if your variables are booleans, for two reasons.

One, It's much harder to identify the error, if you accidentally use an assignment operator.

As we saw, IntelliJ won't flag this as an error when you're testing a boolean variable, so the only way you'll know you made this common mistake is by discovering your program or output isn't what you expected.

Secondly, the code is more concise, and more concise code can often be more readable code.

Ternary Operator

In previous videos in this section, we've learned how to use the 'if then' statement, as well as experimenting with the Logical **and**, and the Logical **or** operators.

In this video, we're going to take a look at something called the Ternary Operator.

The Ternary Operator (Condition ?: Operator)

The ternary operator has three operands. The only operator currently in Java that does have three. Officially, Java calls it the conditional operator.

The structure of this operator is:

```
operand1 ? operand2 : operand3
```

Ternary Operator ? :

The **ternary** operator is a shortcut to assigning one of two values to a variable, depending on a given condition.

So think of it as a shortcut of the **if-then-else** statement. So far in the course, we've only discussed if-then and not else. I'll be discussing else in the next section when we go deeper into control blocks.

Ternary Operator ? :

Consider this example:

```
int ageOfClient = 20;
String ageText = ageOfClient >= 18 ? "Over Eighteen" : "Still a kid";
System.out.println("Our client is " + ageText);
```

Operand one – **`ageOfClient >= 18`** in this case is the condition we're checking. It needs to return `true`, or `false`.

Operand two – **"Over Eighteen"** is the value to assign to the variable **ageText** , if the condition above is `true`.

Operand three – **"Still a kid"** is the value to assign to the variable **ageText** , if the condition above is `false`.

Ternary Operator ? :

```
int ageOfClient = 20;
String ageText = ageOfClient >= 18 ? "Over Eighteen" : "Still a kid";
System.out.println("Our client is " + ageText);
```

In this particular case, ageText is assigned the value "`Over Eighteen`", because `ageOfClient` has the value 20, which is greater than or equal to 18.

Now it can be a good idea to use parentheses, like this example below, to make the code more readable, particularly in the ternary operator.

```
String ageText = (ageOfClient >= 18) ? "Over Eighteen" : "Still a kid";
```

Ternary Operator ? :

In the first example we looked at in our code, we returned a boolean value from the ternary operation.

```
boolean isDomestic = makeOfCar == "Volkswagen" ? false : true;
```

This was a good way to demonstrate the ternary operator, but wouldn't be something you'd do when writing proper code.

A much simpler way to write this code is shown here:

```
boolean isDomestic = (makeOfCar != "Volkswagen");
```

You can see that this code has the same effect and is quite a bit easier to read.

Challenge

Step 1: create a double variable with a value of 20.00.

Step 2: create a second variable of type double with a value 80.00.

Step 3: add both numbers together, then multiply by 100.00.

Step 4: use the remainder operator, to figure out what the remainder from the result of the operation in step three, and 40.00, will be.

Step 5: create a boolean variable that assigns the value true, if the remainder in step four is 0.00, or false if it's not zero.

Step 6: output the boolean variable just to see what the result is.

Step 7: write an if-then statement that displays a message, 'got some remainder', if the boolean in step five is not true.