NAME: CHARLES CHINEMEREM CHARLES

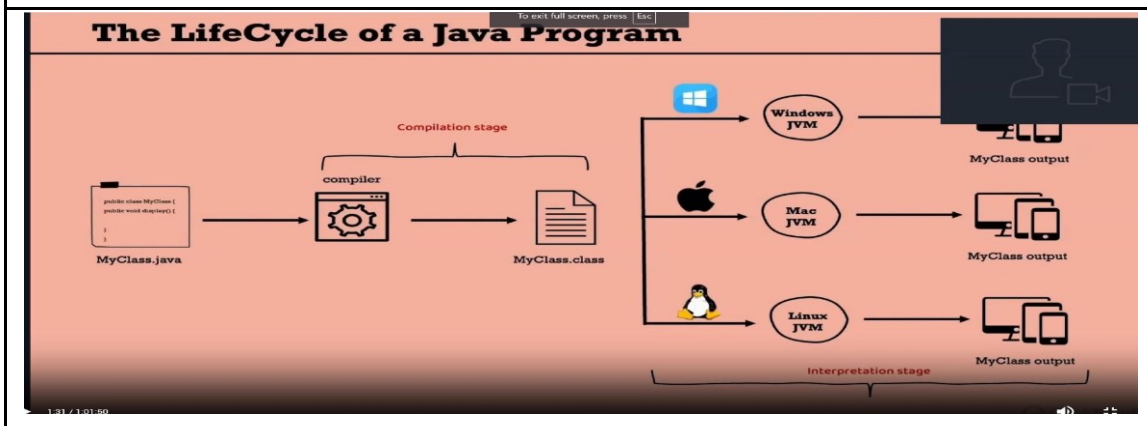
LECTURER: Jarosław Wojciechowski Phd
DATE: 25TH April 2025.
COURSE: OBJECT ORIENTED PROGRAMMING 2.

```
PS C:\Users\ENVY\Java> touch HelloWorld.java
Touch Version 5.0 Copyright (c) 1995-2010 Embarcadero Technologies, Inc.

PS C:\Users\ENVY\Java> javac HelloWorld.java;
PS C:\Users\ENVY\Java> javac HelloWorld.java;
PS C:\Users\ENVY\Java> java HelloWorld
Hello World
PS C:\Users\ENVY\Java>
```

Java Installation and Environment Setup

To begin programming in Java, you'll need to install the Java Development Kit (JDK). I chose to install JDK 17 and configured the system's environment variables (particularly the PATH) to allow running Java commands from any terminal.

I used this code to verify that the installation was successful.

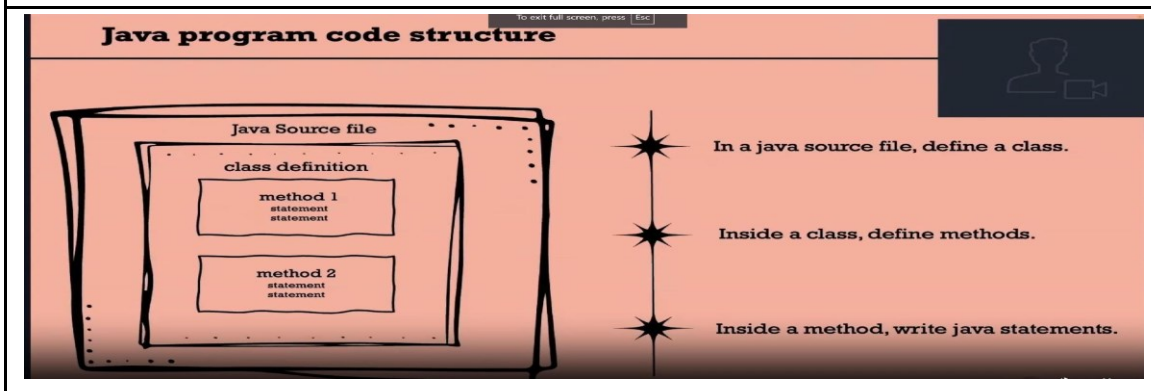


Java Program Lifecycle

The Java program lifecycle involves several key steps:

- **Writing Code**: Developers create the .java source file using a text editor or an Integrated Development Environment (IDE).
- **Compiling**: The javac compiler translates the .java file into bytecode, which is stored in a .class file.
- **Execution**: The Java Virtual Machine (JVM) reads the bytecode and runs the program.
- **JRE (Java Runtime Environment)**: Supplies the libraries and resources required for executing Java programs.
- **JDK (Java Development Kit)**: Encompasses the JRE and adds development tools such as the compiler and debugger.

Each Java file contains a class named after the file. The main method serves as the entry point for the application.



Java Program Structure

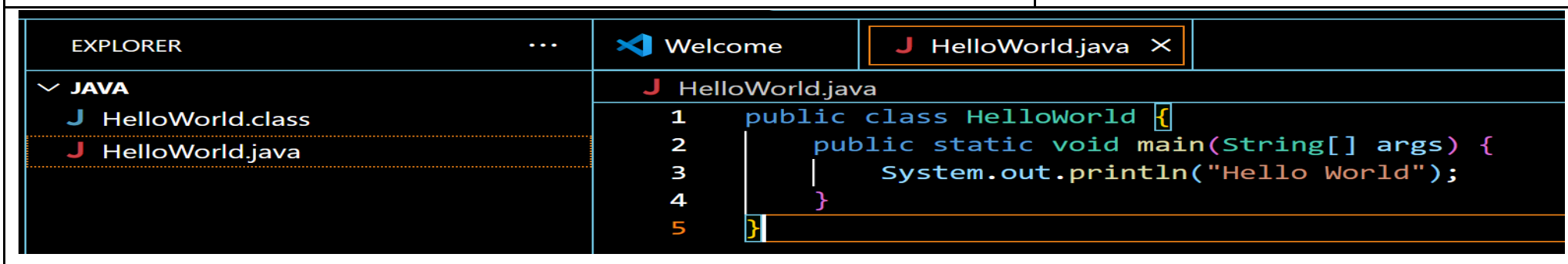
A basic Java program follows a specific structure:

- It starts with a class declaration.
- The main method is where the program begins execution.
- Java syntax requires the use of semicolons, curly braces, and specific naming conventions.

To compile a Java program, open the terminal in the directory containing the .java file and run:

`javac HelloWorld.java`

This command compiles the file. If no errors occur, it creates a .class file, such as HelloWorld.class. This file contains the bytecode that the JVM understands.




```
PS C:\Users\ENVY> jshell
|  Welcome to JShell -- Version 24
|  For an introduction type: /help intro

jshell> /list

jshell> System.out.print("Constance")
Constance
jshell> int myFirstNumber = 5;
myFirstNumber ==> 5

jshell> System.out.print(myFirstNumber);
5

jshell> myFirstNumber = 10;
myFirstNumber ==> 10

jshell> System.out.print(myFirstNumber);
10
jshell> myFirstNumber = 1000;
myFirstNumber ==> 1000

jshell> System.out.print(myFirstNumber);
1000
jshell>  /list
|  Error:
|  illegal start of expression
|   /list
|   ^

jshell> /list

   1 : System.out.print("Constance")
   2 : int myFirstNumber = 5;
   3 : System.out.print(myFirstNumber);
   4 : myFirstNumber = 10;
   5 : System.out.print(myFirstNumber);
   6 : myFirstNumber = 1000;
   7 : System.out.print(myFirstNumber);

jshell> int myFirstNumber = 1000;
myFirstNumber ==> 1000
```

Integer Declaration and Reassignment

The int data type is used to store whole numbers. I reassigned the same variable several times to demonstrate how variables can hold different values throughout the program.

```
jshell> myFirstNumber = 10 + 5;
myFirstNumber ==> 15

jshell> myFirstNumber = (10 + 5) + (2 * 10);
myFirstNumber ==> 35
```

Arithmetic Operations

This section shows how Java handles the order of operations (BODMAS) and how expressions can be evaluated and assigned to variables.

```
jshell> int mySecondNumber = 12;
mySecondNumber ==> 12

jshell> int myThirdNumber = 6;
myThirdNumber ==> 6

jshell> /var
|    int myFirstNumber = 35
|    int mySecondNumber = 12
|    int myThirdNumber = 6


jshell> int myTotal = myFirstNumber + mySecondNumber + myThirdNumber;
myTotal ==> 53
```

Multiple Integer Variables

I declared several integer variables and used them in an arithmetic expression to highlight how values can be combined in Java.

```
jshell> int myMinIntValue = Integer.MIN_VALUE;
myMinIntValue ==> -2147483648

jshell> int myMaxIntValue = Integer.MAX_VALUE;
myMaxIntValue ==> 2147483647

jshell> System.out.print ("Integer Minimum Value = " + myMinIntValue);
Integer Minimum Value = -2147483648
jshell> System.out.print ("Integer Minimum Value = " + Integer.MIN_VALUE);
Integer Minimum Value = -2147483648
jshell> System.out.print("Integer Value Range(" + Integer.MIN_VALUE + "to" + Integer.MAX_VALUE + ")" );
Integer Value Range(-2147483648to2147483647)
jshell> System.out.print ("Busted Max Value = " + (myMaxIntValue + 1));
Busted Max Value = -2147483648
jshell> System.out.print ("Busted Min Value = " + (myMinIntValue - 1));
Busted Min Value = 2147483647
jshell> System.out.print ("Integer Maximum Value = " + Integer.MAX_VALUE);
Integer Maximum Value = 2147483647
```

Integer Limits

`int myMinIntValue = Integer.MIN_VALUE;`

`int myMaxIntValue = Integer.MAX_VALUE;`

Every primitive type in Java has a minimum and maximum limit. These lines demonstrate the lowest and highest values an int can hold. For example, the following code demonstrates an overflow when the value exceeds the maximum limit:

`System.out.print("Busted Max Value = " + (myMaxIntValue + 1));`

```
jshell>  byte myMinByteValue = Byte.MIN_VALUE, myMaxByteValue = Byte.MAX_VALUE;
myMinByteValue ==> -128
myMaxByteValue ==> 127

jshell> short firstShort = 1; int firstInteger = 2;
firstShort ==> 1
firstInteger ==> 2
```

Byte and Short

The byte and short data types are smaller versions of int, useful for optimizing memory usage when storing smaller numbers.

```
jshell> byte byteValue = 10;
byteValue ==> 10

jshell> short shortValue = 20;
shortValue ==> 20

jshell> int intValue = 30;
intValue ==> 30

jshell> long longTotal = 50000L + 10L * (byteValue + shortValue + intValue);
longTotal ==> 50600

jshell>  int sumofThree = byteValue + shortValue + intValue;
sumofThree ==> 60

jshell> longTotal = 50000L + (10 * sumofThree);
longTotal ==> 50600
```

Long Type and Arithmetic

The long data type is used to store very large integers. The L suffix indicates a long literal.

```
jshell> System.out.print("Float Value Range(" + Float.MIN_VALUE + " to " + Float.MAX_VALUE + ")");
Float Value Range(1.4E-45 to 3.4028235E38)
jshell>  int myIntValue = 5; float myFloatValue = 5; double myDoubleValue =5;
myIntValue ==> 5
myFloatValue ==> 5.0
myDoubleValue ==> 5.0

jshell> myFloatValue = 5f
myFloatValue ==> 5.0

jshell> myDoubleValue = 5d
myDoubleValue ==> 5.0

jshell>  float myOtherFloatValue = (float)5.25;
myOtherFloatValue ==> 5.25

jshell> int myIntValue = 5; float myFloatValue = 5f; double myDoubleValue =5d;
myIntValue ==> 5
myFloatValue ==> 5.0
myDoubleValue ==> 5.0
```

Float and Double Types

The float and double data types are used for decimal numbers. The f and d suffixes are used to explicitly declare float and double literals, respectively. For example:

`float myOtherFloatValue = (float)5.25;`

Casting is used here to fix a type conversion error when converting from double to float.

```
jshell> myIntValue = 5 / 2;
myIntValue ==> 2

jshell> myFloatValue = 5f / 2f;
myFloatValue ==> 2.5

jshell> myDoubleValue = 5d / 2d;
myDoubleValue ==> 2.5

jshell> myIntValue = 5 / 3;
myIntValue ==> 1

jshell> myFloatValue = 5f / 3f;
myFloatValue ==> 1.6666666

jshell> myDoubleValue = 5d / 3d;
myDoubleValue ==> 1.6666666666666667
```

Division and Precision

```
myIntValue = 5 / 2; // Result: 2 (integer division)
myFloatValue = 5f / 2f; // Result: 2.5
myDoubleValue = 5d / 2d; // Result: 2.5
```

This section demonstrates how Java handles precision differences among data types:

```
jshell> double numberofPounds = 200d;
numberofPounds ==> 200.0

jshell> double convertedKilograms = numberofPounds * 0.45359237d;
convertedKilograms ==> 90.718474

jshell> System.out.print("Converted Kilograms = " + convertedKilograms);
Converted Kilograms = 90.718474
jshell>
```

Real-world Calculation

```
double numberOfPounds = 200d;
double convertedKilograms = numberOfPounds * 0.45359237d;
```

This example converts pounds to kilograms and demonstrates practical floating-point arithmetic.

```
jshell> double anotherNumber= 3_000_000.4_567_890d;
anotherNumber ==> 3000000.456789
```

Underscore in Numeric Literals

`int oneMillion = 1_000_000;`

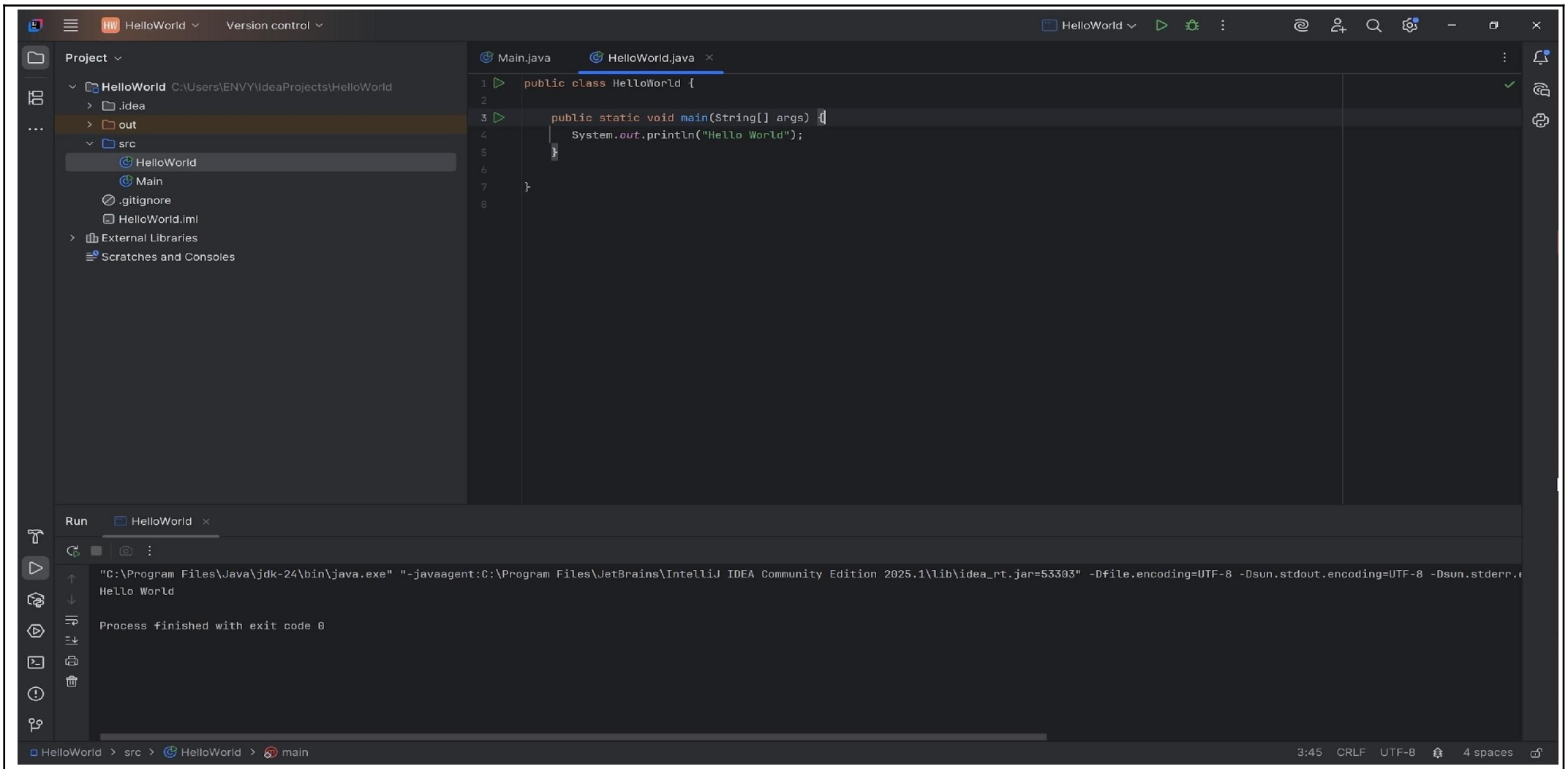
Java allows underscores in numeric literals for better readability, such as:

```
jshell> char mychar = 'D';
mychar ==> 'D'

jshell> char myUnicode = '\u0044'
myUnicode ==> 'D'

jshell>
```

Character and Unicode

The char data type stores a single character:

`char myChar = 'D';`



IntelliJ IDEA Setup

After installing IntelliJ IDEA, I repeated the steps used earlier in Visual Studio Code. I created a new project named HelloWorld.java, typed the code, and manually created the Java class file by right-clicking on the src folder. This was different from using cmd or PowerShell and the javac command to create the .class file.

In Summary:

Action	Tool	Result
Created class via GUI	IntelliJ (right-click src)	Main.java created
Compiled manually	IntelliJ Terminal (javac)	Main.class created
Ran manually	IntelliJ Terminal (java)	Output: "Hey I'm now running…"