INFO1113 Object-Oriented Programming

Week 1B: Java Fundamentals
Compilation, Syntax, Types and Ifs

Slides developed by Tyson Thomas

Copyright Warning

Topics

- Java syntax, source code, compilation, executing (s. 4)
- Types (s. 18)
- Command line arguments (s. 25)
- Input and Scanner (s. 29)
- Conditional statements (if) (s. 32)

Hello World

When learning any language we want to be able to output simple text to the screen. So as with all programming languages we are able to write the *classic* “Hello World” program.

```
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }

}
```

Anatomy of a source file

Let's take apart a Java source file (Refer to p. 48-51, Savitch & Mock)

```java
import java.util.Scanner;

public class Anatomy {

    public static void main(String[] args) {

        System.out.println("Hello! This will output to the screen");

        int integerVar = 1;

        double d1 = 1.5;

        Scanner keyboard = new Scanner(System.in);

        String s = "This is a string!";

        s = keyboard.nextLine(); //We have reassigned it

        System.out.println(s);

    }

}
```

Class body, contains attributes and methods inside it.

Method body, scope is defined with curly braces.

Methods must be contained within a class

(For another example, Listing 1.1 on p. 48, Savitch & Mock)

Anatomy of a source file

Let's take apart a Java source file (Refer to p. 48-51, Savitch & Mock)

(For another example, Listing 1.1 on p. 48, Savitch & Mock)

Anatomy of a source file

Let's take apart a Java source file (Refer to p. 48-51, Savitch & Mock)

```
import java.util.Scanner;

public class Anatomy {

    public static void main(String[] args) {

        System.out.println("Hello! This will output to the screen");

        int integerVar = 1;

        double d1 = 1.5;

        Scanner keyboard = new Scanner(System.in);

        String s = "This is a string!";

        s = keyboard.nextLine(); //We have reassigned it

        System.out.println(s);

    }

}
```

Access modifier, allows the following definition to be accessible by others

Class keyword, used when defining a class.

Class name must go next.

Class name, since the public keyword was used, the filename must be

Anatomy.java

(For another example, Listing 1.1 on p. 48, Savitch & Mock)

Anatomy of a source file

Let's take apart a Java source file (Refer to p. 48-51, Savitch & Mock)

```
import java.util.Scanner;

public class Anatomy {

    public static void main(String[] args) {

        System.out.println("Hello! This will output to the screen");

        int integerVar = 1;

        double d1 = 1.5;

        Scanner keyboard = new Scanner(System.in);

        String s = "This is a string!";

        s = keyboard.nextLine(); //We have reassigned it

        System.out.println(s);

    }

}
```

Static modifier allows it to be accessed without the need of an object.

main method, this is the first function invoked in your java program (starting point)

Command line arguments. It is of type String and [] defines it as an Array.

(For another example, Listing 1.1 on p. 48, Savitch & Mock)

Anatomy of a source file

Let's take apart a Java source file (Refer to p. 48-51, Savitch & Mock)

```
import java.util.Scanner;

public class Anatomy {

    public static void main(String[] args) {

        System.out.println("Hello! This will output to the screen");

        int integerVar = 1;

        double d1 = 1.5;

        Scanner keyboard = new Scanner(System.in);

        String s = "This is a string!";

        s = keyboard.nextLine(); //We have reassigned it

        System.out.println(s);

    }

}
```

Outputting to the screen

String literals are defined using double quotes. e.g

"This is a String"

Primitive type int followed by a variable name and initialised to 1.

double type

Reference type String variable set to string literal "This is a string!"

(For another example, Listing 1.1 on p. 48, Savitch & Mock)

Anatomy of a source file

Let's take apart a Java source file (Refer to p. 48-51, Savitch & Mock)

```
import java.util.Scanner;

public class Anatomy {

    public static void main(String[] args) {

        System.out.println("Hello! This will output to the screen");

        int integerVar = 1;

        double d1 = 1.5;

        Scanner keyboard = new Scanner(System.in);

        String s = "This is a string!";

        s = keyboard.nextLine(); //We have reassigned it

        System.out.println(s);

    }

}
```

Imports the **Scanner class** so that it can be used within your **class**

Creates a scanner **object** which allows you to read input from standard input (**stdin**)

(For another example, Listing 1.1 on p. 48, Savitch & Mock)

Anatomy of a source file

Let's take apart a Java source file (Refer to p. 48-51, Savitch & Mock)

```
import java.util.Scanner;

public class Anatomy {

    public static void main(String[] args) {

        System.out.println("Hello! This will output to the screen");

        int integerVar = 1;

        double d1 = 1.5;

        Scanner keyboard = new Scanner(System.in);

        String s = "This is a string!";

        s = keyboard.nextLine(); //We have reassigned it

        System.out.println(s);

    }

}
```

Assigns s to the next line of input that user has given.

Outputting the string using **println**.

(For another example, Listing 1.1 on p. 48, Savitch & Mock)

Compilation

You may have used an interpreted language prior to this course and therefore all that you may be familiar with is:

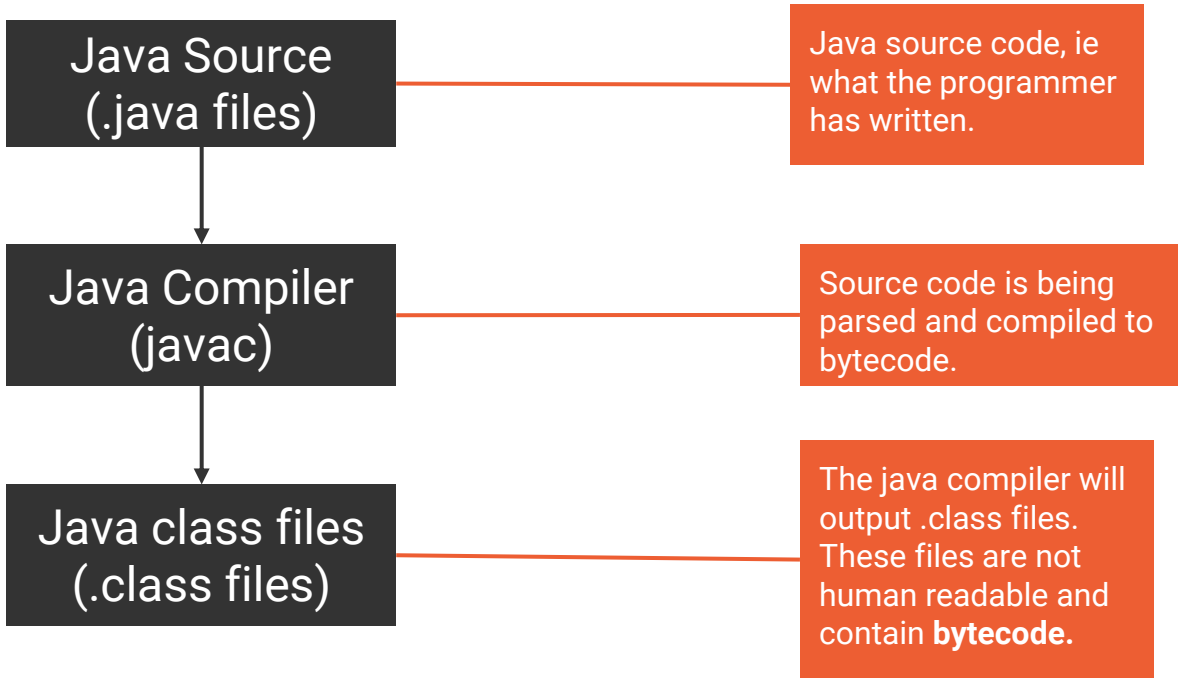
```
> python myprogram.py
Hello Everyone! This is python! Remember me?
```

However! Java is a **compiled** language and therefore there is an extra step we have to take before executing any code. This compilation step will transform the source code to **bytecode.**

```
> javac MyProgram.java
> java MyProgram
Hello Everyone! This is a java program
```

Compilation

```
> javac MyProgram.java
```

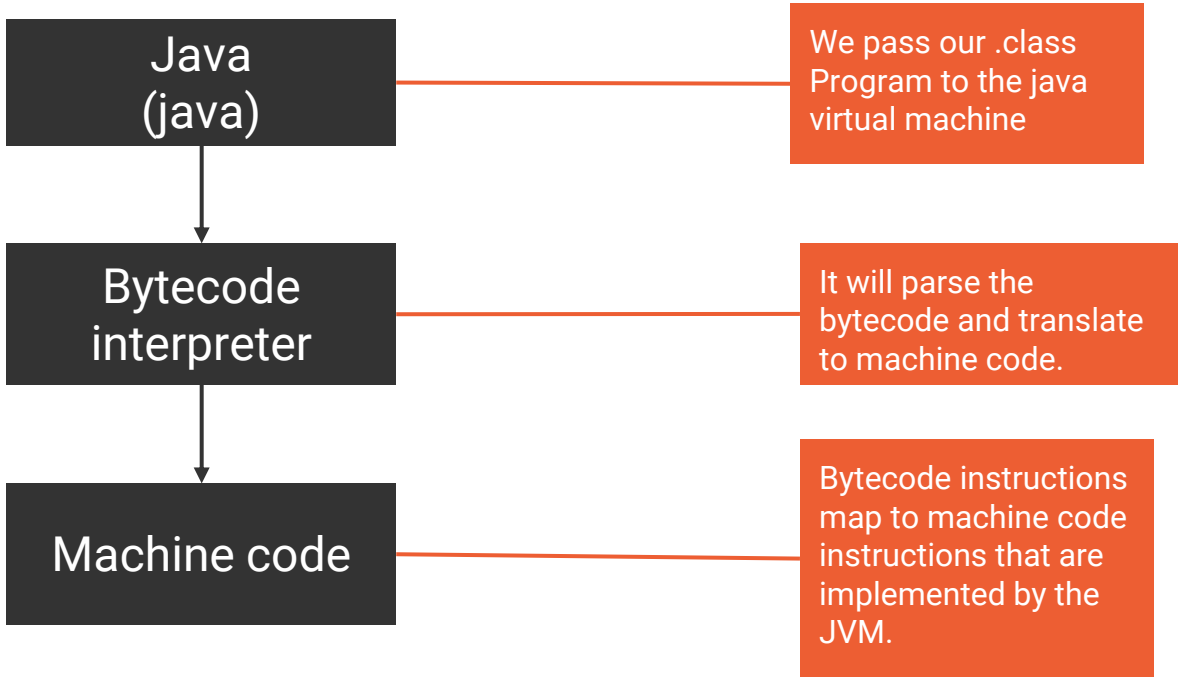


After this process has been successfully completed we are able to now execute our program using the **java virtual machine (jvm)**

(Refer to p. 43-44, Savitch & Mock)

Compilation

```
> java MyProgram
```



During this process our program is executing and is contained within the **java virtual machine**.

(Refer to p. 43-44, Savitch & Mock)

Demonstration:
Compilation and Hello World

Scoping

A block is a collection of statements that is delimited by a pair of curly braces *{ }* .

```
public class Scoping {

    public static void main(String[] args) {

        int x = 0; //x starts here
        {
            int y = 1; //y starts heres
            {
                int z = 2;
            }// z ends here
        } // y ends here

    } //x ends here

}
```

Demonstration:
Scoping and lifetime

Types and variables

Unlike interpreted languages like python, Java has a separation of **primitive** types and **reference** types.

Primitive types

- **int**
- **float**
- **double**
- **char**
- **short**
- **byte**
- **long**
- **boolean**

(Refer to p. 85, Savitch & Mock)

String is a reference as it is an aggregation of the **char** type. (Array of char elements).

Any class you create is a **reference** type.

Why is it called a reference type?

Types and variables

Unlike interpreted languages like python, Java has a separation of **primitive** types and **reference** types.

Primitive types

- **int**
- **float**
- **double**
- **char**
- **short**
- **byte**
- **long**
- **boolean**

(Refer to p. 85, Savitch & Mock)

String is a reference as it is an aggregation of the **char** type. (Array of char elements).

Any class you create is a **reference** type.

Why is it called a reference type?

We are actually working with a binding to a memory address, not the object itself. (Similar to python objects) (Refer to p. 364, Savitch & Mock)

Types and range

Name	Kind	Memory	Range	Type
boolean	boolean	1 byte (1 bit representation)	true or false	**primitive**
byte	integer	1 byte	[-128, 127]	**primitive**
short	integer	2 bytes	[-32768, 32767]	**primitive**
int	integer	4 bytes	[-2147483638, -2147483637]	**primitive**
long	integer	8 bytes	$[\sim -9 \times 10^{18}, \sim 9 \times 10^{18}]$	**primitive**
float	floating-point	4 bytes	$[\pm 3.4 \times 10^{38}, 1.4 \times 10^{-45}]$	**primitive**
double	floating-point	8 bytes	$[\pm 1.8 \times 10^{308}, \pm 4.9 \times 10^{-324}]$	**primitive**
char	character	2 bytes	[0, 65535]	**primitive**
String	string	variable	[0, very long]	**object**

Types and variables

Okay types… why do they exist?

Simply, we are able to confirm the type that is being assigned. The compiler can check the assignment of variables (in the event we are **attempting to assign float to an int or some other nasty assignment**).

Clear allocation of **memory**. With type information the compiler knows how much memory should be allocated for that variable.

Types and variables

Type conversions and casting (**A programmer's headache**)

What would be the output of:

```
public static void main(String[] args) {

    int i = 1;
    double f = 1.0;

    System.out.println(i/2);
    System.out.println(f/2);
    System.out.println(f/i);

}
```

There are specific operations that occur depending on the type and operation.

Type conversions and casting (**A programmer's headache**)

What would be the output of:

```
public static void main(String[] args) {

    int i = 1;
    double f = 1.0;

    System.out.println(i/2);
    System.out.println(f/2);
    System.out.println(f/i);

}
```

Two integers are involved and this is where integer division occurs. Since i is assigned to 1, the divison will be 0 (not 0.5 as .5 cannot be represented as an integer).

Since a double number is involved during the calculation it will promote the integer (2) to a double

Type conversions and casting (**A programmer's headache**)

What would be the output of:

```
public static void main(String[] args) {

    int i = 1;
    double f = 1.0;

    System.out.println(i/2);
    System.out.println(f/2);
    System.out.println(f/i);

}
```

Two integers are involved and this is where integer division occurs. Since i is assigned to 1, the divison will be 0 (not 0.5 as .5 cannot be represented as an integer).

Since a double number is involved during the calculation it will promote the integer (2) to a double

What is the result of this operation?

For further reading: https://docs.oracle.com/javase/specs/jls/se8/html/jls-5.html

Command line arguments

Java inherits C like command line arguments with few differences.

- Program name is **not** included in the arguments.
- Java has a **String** type while C does not.

```
public class CommandLineArgs {

    public static void main(String[] args) {

        String arg1 = args[0];

        System.out.println(arg1+"!");
    }

}
```

(For another example, Listing 1.1 on p. 48, Savitch & Mock)

Command line arguments

Java inherits C like command line arguments with few differences.

- Program name is **not** included in the arguments,
- Java has a **String** type while C does not.

```
public class CommandLineArgs {

    public static void main(String[] args) {

        String arg1 = args[0];

        System.out.println(arg1+"!");
    }

}
```

```
> java CommandLineArgs Hi
Hi!
```

Arrays in java start at index 0.

We have assigned String arg1 to the first element in the args array

(Arguments for the Method main, p. 554, Savitch & Mock)

Demonstration:
Using command line arguments

Scanner and input

Scanner is an abstracted object that provides an interface for **reading** data.

```
import java.util.Scanner;

public class UsingScanner {

    public static void main(String[] args) {

        Scanner keyboard = new Scanner(System.in);

        //We want to read in a line from Standard Input

        String s = keyboard.nextLine();

    }
}
```

Scanner object declared and initialised and we read the next line inputted by the user using the nextLine() method.

(For another example, Listing 1.1 on p. 49, Savitch & Mock)

(Extended Reading, p. 50-52, Savitch & Mock)

Scanner and input

Scanner is an abstracted object that provides an interface for **reading** data.

```
import java.util.Scanner;

public class UsingScanner {

    public static void main(String[] args) {

        Scanner keyboard = new Scanner(System.in);

        //We want to read in a line from Standard Input

        String s = keyboard.nextLine();

    }
}
```

Scanner object declared and initialised and we read the next line inputted by the user using the nextLine() method.

Checkout the following documentation for Scanner:
https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

(For another example, Listing 1.1 on p. 49, Savitch & Mock)

(Extended Reading, p. 50-52, Savitch & Mock)

Demonstration:
Scanner and input

Boolean

Java inherits logical (and other operators) from **C.**

We have the same logical operators for **and, or** and **not.**

```
public static void main(String[] args) {

    boolean t = true;
    boolean f = false;

    boolean exampleAnd = t && t; //This is true
    boolean exampleOr = f || t; //This is true
    boolean exampleNot = !t; //This is false

}
```

logical **and**. It will check if **both** of the expression are true.

logical **or**. It will check if **one** of the expression of either side is true.

This will enact the inverse of the expression. If **t** is **true**, **!t** is **false**

If statements

Similar to other language, Java also has **if** statements however has a strict requirement that statement is of **type boolean**

```
public class IfProgram {

    public static void main(String[] args) {

        boolean t = true;
        boolean f = false;

        if(t) {
            System.out.println("This will be executed");
        }

        if(f) {
            System.out.println("This will not be executed");
        }
    }

}
```

t is a **boolean** type and is assigned the value **true.**

This branch **will execute**

f is a **boolean** type and is assigned the value **false.**

This branch **will NOT execute**

If statements

Similar to other language, Java also has **if** statements however has a strict requirement that statement is of **type boolean**

```
public class IfProgram {

    public static void main(String[] args) {

        int t = 1;
        int f = 0;

        if(t) {
            System.out.println("This will be executed");
        }

        if(f) {
            System.out.println("This will not be executed");
        }
    }
}
```

In other languages true and false are typically defined as 1 and 0 respectively.

However java enforces the **correct type** to be used

We would encounter the following error:
| Error:
| incompatible types: int cannot be converted to boolean

Demonstration:
If statements

See you next time!