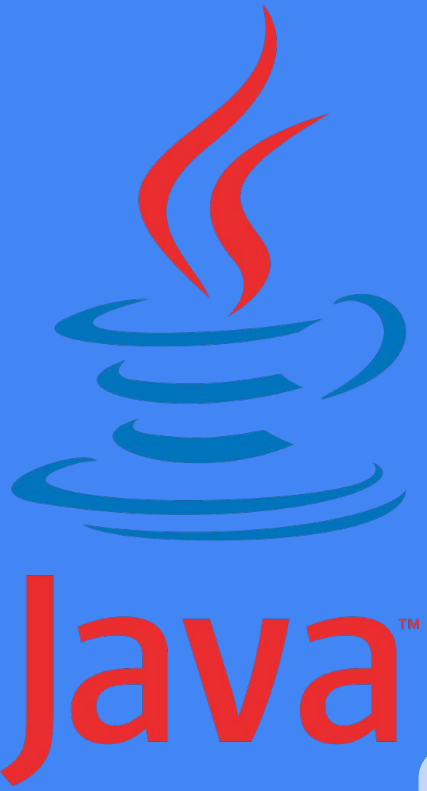
Introduction to Java Programming

Session 2: Processing Input, Variables and OOP Revisited



REVIEW FROM LAST TIME

Basic Java Syntax Rules

- The **class** is the basic unit of Java programming. All code that runs has to be inside of one. We will learn just exactly what a class is later.
- A **method** is a group of code in a class that is grouped together and runs together, such as the *main()* method we just wrote.
- Both **classes** and **methods** include **blocks** of code, which are contained inside of “{...}” curly braces.
- **Statements** do things. **Statements** such as the *System.out.println(“Hello,World!);* require a semicolon to signify the end of the line
- Unlike some languages such as Python, whitespace (spaces, tabs, and line breaks) does not affect the execution of a program

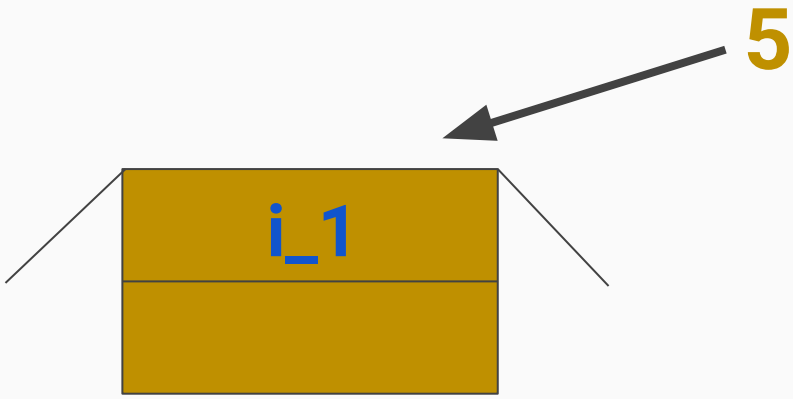
Variable Types

- Integer (int) - a whole number without decimals
- Double (double) - a precise number with decimals
- Float (float) - a less precise decimal number
- Boolean (boolean) - can have the value true or false
- Character (char) - a single character of text
- String (String) - a sequence, or string, of characters that often form a word or phrase, e.g. “Hello, World!”

String is a **Reference Variable**, not a primitive!

Variable Syntax

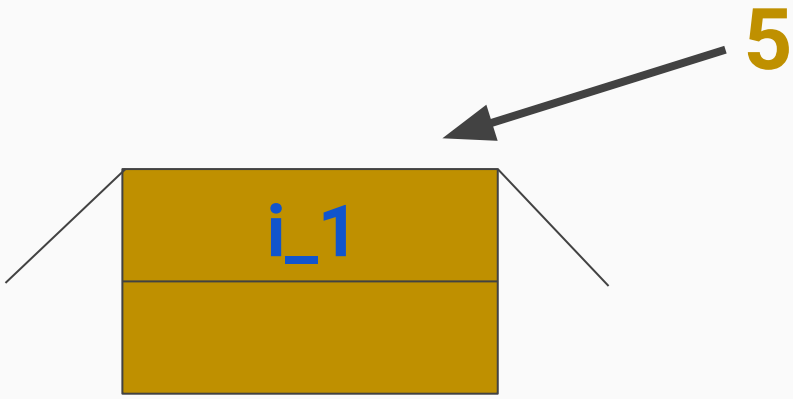
```
int i_1 = 5;
```



Variable Syntax

Declaration Assignment

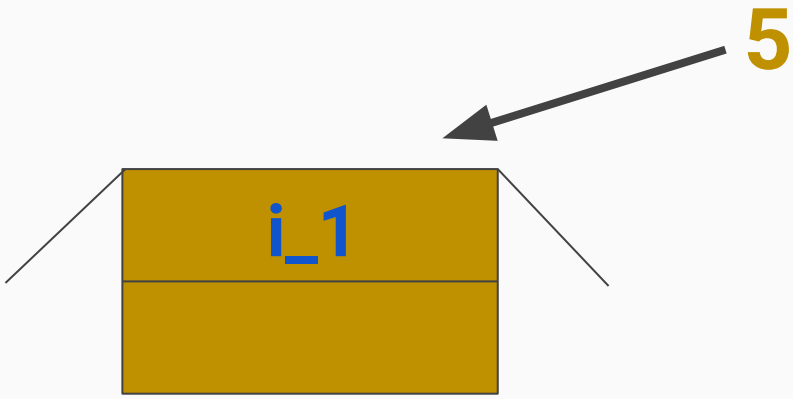
```
int i_1 = 5;
```



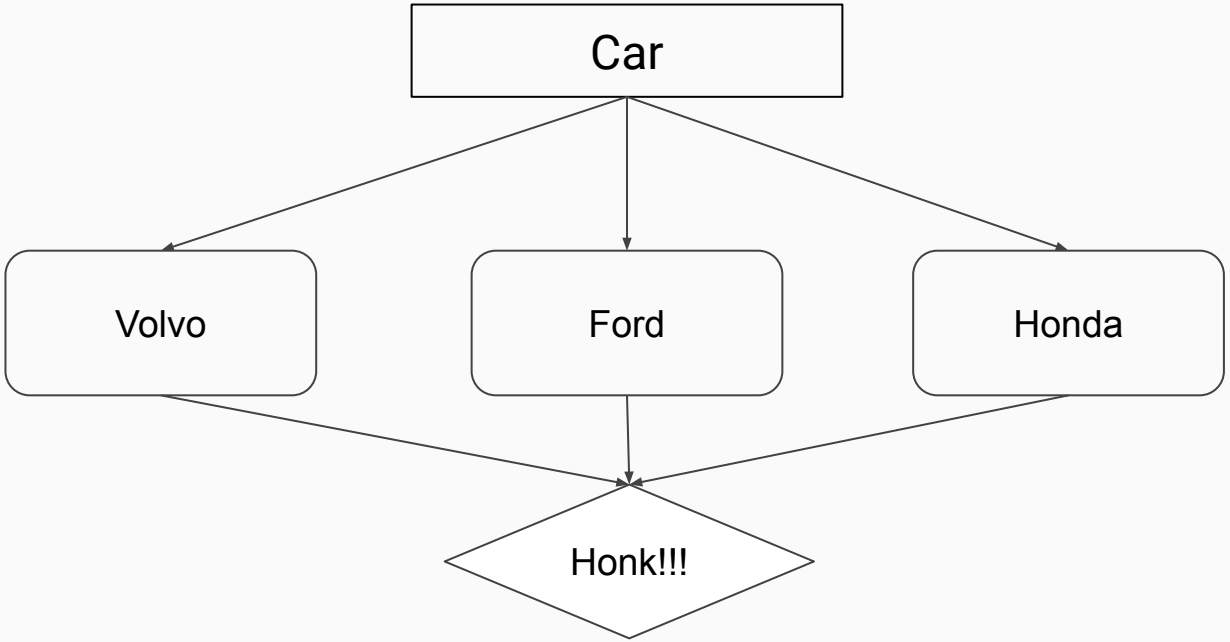
Variable Syntax

Type Identifier Value

```
int i_1 = 5;
```



What is Object Oriented Programming?

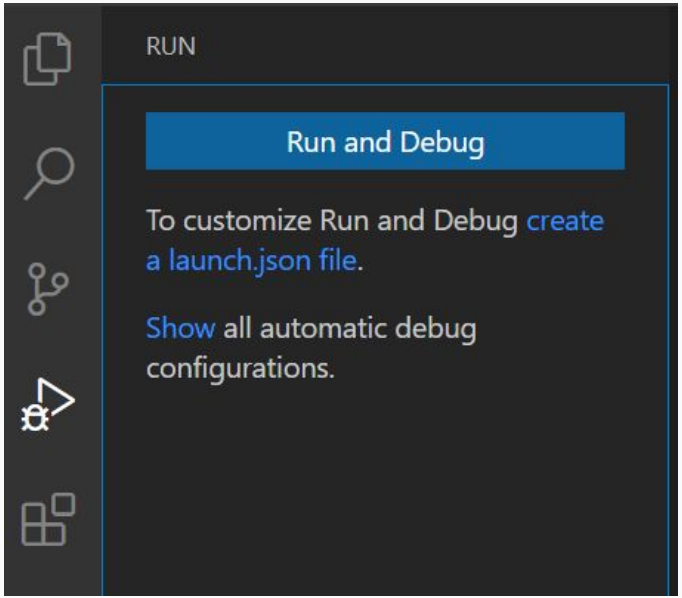


END OF REVIEW

Setting Up VS Code's Console

- This will get rid of the lengthy commands cluttering up the output of the "Hello, World!" program
- We will need to create and modify a "launch.json" file in our working directory
- Luckily, VS Code makes this easy for us
- To begin, we will need to open our run/debug menu and click

create a launch.json file



Setting Up VS Code's Console

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "java",
            "name": "Debug (Launch) - Current File",
            "request": "launch",
            "mainClass": "${file}"
        },
        {
            "type": "java",
            "name": "Debug (Launch)-Tutorial<VsCode Example_ef92b941>",
            "request": "launch",
            "mainClass": "Tutorial",
            "projectName": "VsCode Example_ef92b941"
        }
    ]
}
```

- This will bring up a file like the one to the left. This syntax should look somewhat familiar, but is not really related to Java.
- This is a JSON (JavaScript Object Notation) file, which is used to store information in a human-readable way
- We will only touch the "Current File" configuration
- Each configuration is separated by "{...}" brackets

Setting Up VS Code's Console

- In the configuration whose title includes the phrase "Current File", there should be a line that reads:

```
"mainClass": "${file}"
```

- Add a comma to the end of this line after the last quotation mark, press enter, and on this new line write:

```
"console": "externalTerminal"
```

This is what the configuration should look like when you are finished. Save, and you are done!

Project: Working With User Input

For this project, we will be using Java's built in Scanner class to read input given to us by the program's user.

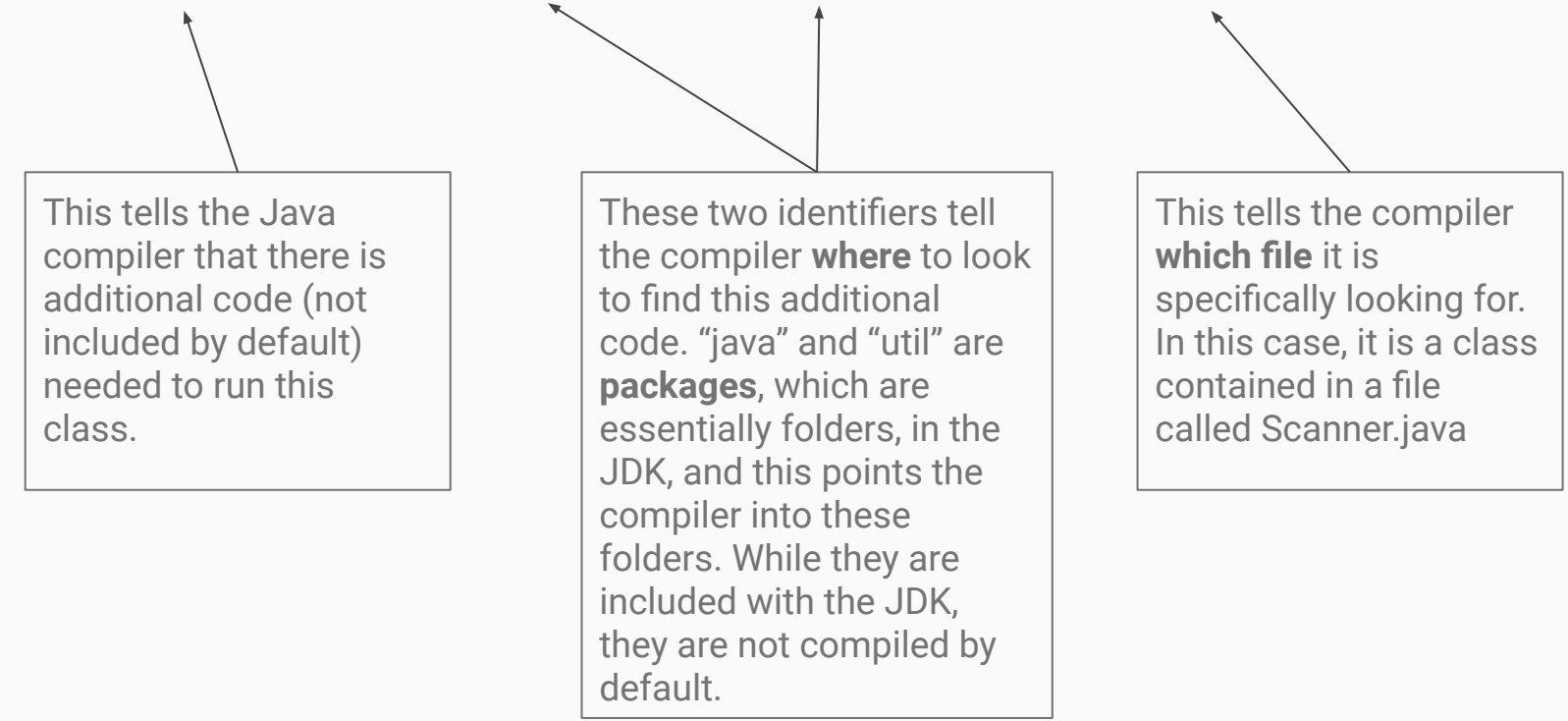
What is the Scanner class? And what does it mean to import code?

I will switch to VS Code and we will work through this project line-by-line, then we will quickly explore what each especially important line is doing.

```
import java.util.Scanner;
public class P1_Input {
    public static void main(String args[]) {
        Scanner inputStream = new Scanner(System.in);
        System.out.print("Write something: ");
        String inString = inputStream.nextLine();
        System.out.println("You wrote: " + inString);
        inputStream.close();
    }
}
```

Imports, Packages, and Java Files

```
import java.util.Scanner;
```



Reference Variables, Objects, and Data Streams

```
Scanner inputStream = new Scanner(System.in);
```

Here, we are declaring a variable called inputStream.

But this isn't one of the types we had talked about being built in to Java!

The equals sign should immediately tell you that this is an assignment operation!

And what's this whole part on the right? What does the "new" keyword mean?

Scanner is a **class**, and inputStream is an **instance** of that class. It is an **object**.

Imports, Packages, and Java Files

```
String inString = inputStream.nextLine();
```

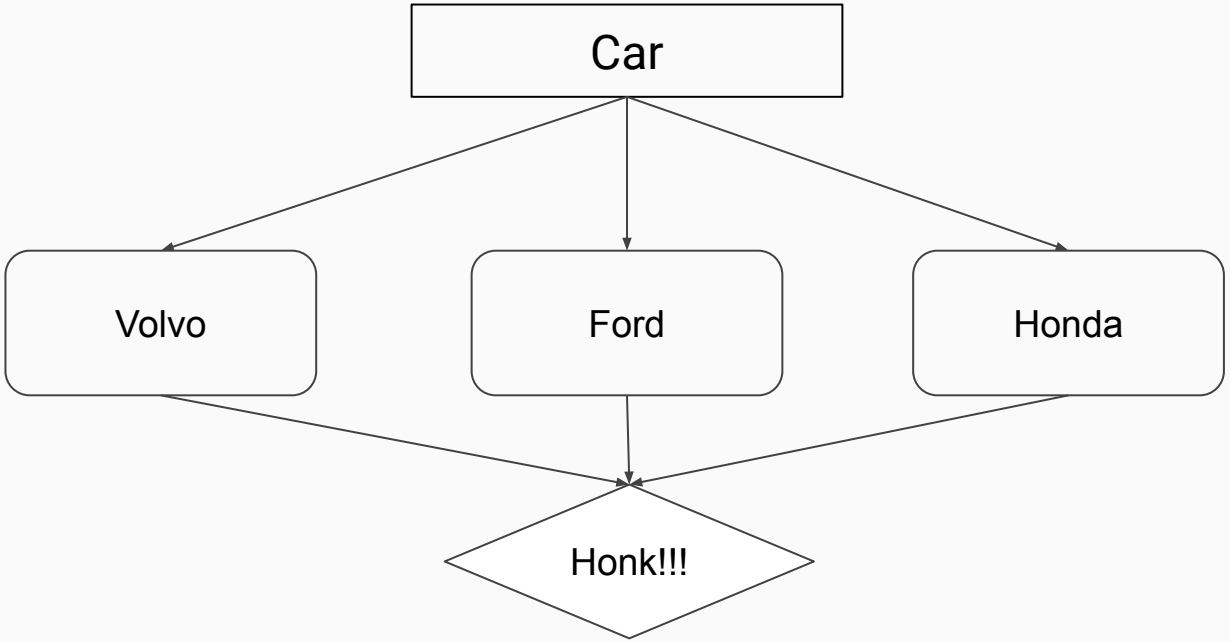
The left side should be familiar now! We are declaring a variable called “inString” that is of type String.

Again, a single equals sign should scream assignment!

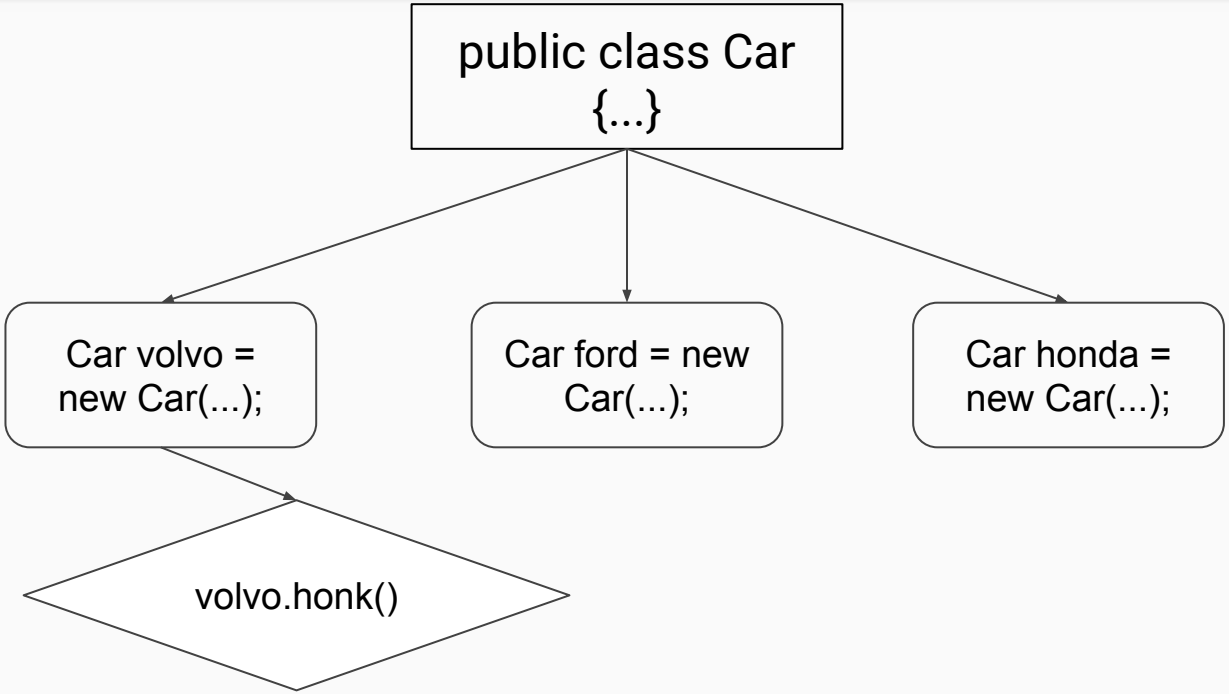
This side is less familiar. Remember **inputStream**? It is an **instance** of the Scanner **class**. The period indicates we are accessing some **field or method**, or value, of the object. In this case, the parentheses indicate we are calling a **method** named “nextLine”

Let’s take another look at that Object Oriented Programming diagram, and hopefully this will make more sense!

What would this look like in Java Syntax?



Object Oriented Programming in Java Syntax



Project: Working With Number Input

Now that we understand a little more about the Scanner class, let's move on to using user input to do more interesting things!

Again, we will move through this program line-by-line together in VS Code.

Then, there are a few key points to be made about equality testing and operations.

```
import java.util.Scanner;
public class P2_Numbers {
    public static void main(String[] args) {
        Scanner inStream = new Scanner(System.in);
        System.out.print("Enter integer a: ");
        int a = inStream.nextInt();
        System.out.print("Enter integer b: ");
        int b = inStream.nextInt();
        System.out.println();
        int c;
        System.out.println(a + " *" + b + " = " + (a * b));
        System.out.println("The same number? " + (a==b));
        c = a + b;
        System.out.println("c is a+b, which equals " + c);
        inStream.close();
    }
}
```

How can we add variables of different types?

```
System.out.println(a + " * " + b + " = " + (a * b));
```

- We are adding strings to integers a and b. How is that possible?
- In this case, the “+” operator is no longer performing addition. It is only addition when both of the operands are numbers.
- Here, because one of the operands is a string, the “+” is instead performing concatenation. Concatenation is an operation that basically pushes two String variables together to form one string.
- a and b can be implicitly converted, or **cast**, to Strings, because every number has a defined representation in text.
- We will get more into types and type-casting later.

The Equality Operator

We haven't seen this before: `(a==b)`

- So far, we have just seen the single equals sign, and it has always represented variable assignment.
- The double equals sign is completely different, and serves to test whether the value on the left is equal to the one on the right.
- Remember the boolean operator? The expression (a==b) evaluates to a boolean value. Either a is equal to b - **true**, or a is not equal to b - **false**.
- An important thing to note is that this will only work as expected with **primitive variables.** With reference variables like Strings and other objects, the equality operator's behavior will be unexpected.

Project: Working With String Input

Working with String variables can be significantly different from working with the primitive variable types.

We will take a brief look at working with Strings and their methods. Remember, Strings are objects, and therefore have useful methods that you can call.

```
import java.util.Scanner;
public class P2_Numbers {
    public static void main(String[] args) {
        Scanner inStream = new Scanner(System.in);
        System.out.print("Enter a word: ");
        String a = inStream.nextLine();
        System.out.println("That word is " + a.length());
        System.out.print("Enter another word, or the same one: ");
        String b = inStream.nextLine();
        System.out.println(a.equalsIgnoreCase(b));
        inStream.close();
    }
}
```

If we have time: If Statements, Case Switches, and Program Flow