NPTEL

noc24_cs43

Programming in Java

Live Interaction Session 1: 30th Jan 2024

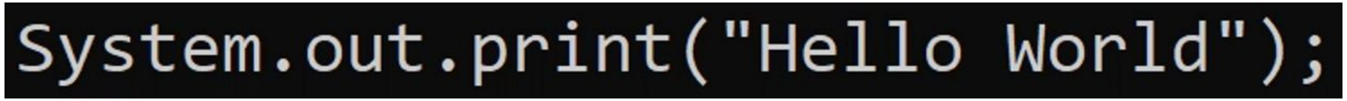
Getting Started

- JShell Tool : Java Shell Tool
- **Nothing beats learning from your own mistakes**, and figuring out how to get a line of code to work, when it wasn't working before.
- This emulates real-world problem solving, and you'll find when you type in your own code, you'll be making all kinds of typos, or omitting or overlooking some characters that are required.
- This is the best way to learn, and pick up all the nuances, of any programming language you're learning.
- If you do encounter a mistake on your own, pause the video, try solving it on your own.

Statement

- It's a complete command to be executed. It can include one or more expressions.

Statement



```
System.out.print("Hello World");
```

- What we've typed above, is a command to print some information to the screen, using syntax provided by the Java language.
- We specified what we wanted Java to print in the parentheses and double quotes – in this case, the text "Hello World" – effectively we're telling Java to print out the words, as we've specified them in the quotes – "Hello World".

JShell : https://docs.oracle.com/en/java/javase/21/jshell/introduction-jshell.html

- These were a few examples, of errors that might occur, when your coding in JShell. Let me encourage you to play with this line of code in JShell, in as many ways as you can think of, to see what kind of errors you might get, or what kind of output is produced.
- This is the whole point of JShell – to provide you with a safe place to test code segments.
- Remember that the key combination 'control c', on windows, or 'control d' on a mac or a linux machine, should cancel what you are in the middle of, and get you back to the JShell prompt.
- Typing forward slash and the word 'exit', or forward slash with the shortcut text ex, will end your JShell session, if you get stuck.
- An example would be **/exit** or **/ex**.

Variables

- Keywords
    - **A keyword is any one of a number of reserved words**, that have a predefined meaning in the Java language.
    - In Java syntax, all code is case-sensitive, and this includes keywords. As we'll soon see, an **int**, all in lowercase, is not the same as **Int**, with a capital I. Here, an int, (all in lowercase) is a keyword in Java.
- Variables
    - **Variables are a way to store information in our computer**.
    - Variables that we define in a program, can be accessed by a name we give them, and the computer does the hard work, of figuring out where they get stored, in the computer's **random access memory**, or RAM.

Declaration Statement

A declaration statement is used to define a variable by indicating the data type, and the name, then optionally to set the variable to a specific value.

Expressions

An expression is a **coding construct, that evaluates to a single value**.

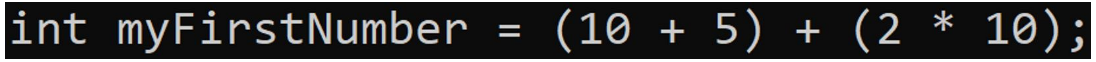
Variable Declarations in JShell

- By declaring a variable again, we are effectively re-declaring a variable, and in normal Java programming, that would not be allowed, and would throw an error.
- Due to its interactive nature, JShell “holds our hand”, and allows the redeclaration to occur without throwing an error.
- For now, just follow along, knowing that re-declaring a variable for a second, or subsequent time, is not allowed, and later in the course, when we switch to a full editor, you'll see what happens when we try and do that.
- Note that we can assign a value to a variable multiple times in Java, but it’s the declaration (which includes the data type) that cannot normally be done a second time for the same variable.

Operators

- Java operators, or just operators, perform an operation (hence the term) on a variable or value.
- Addition, Subtraction, Division, and Multiplication are four common ones that I feel sure you're familiar with, but there are lots more operators you will work with as we go through the course.

Starting out with Expressions



```
int myFirstNumber = (10 + 5) + (2 * 10);
```

Challenge

Your challenge is to create two additional variables in JShell.

Here's what we need:

- One variable called **mySecondNumber**, which is an int, with a value of 12.
- And another variable called **myThirdNumber**, also of type int, with a value of 6.
- Add it in **MyTotal** variable.

Challenge

- First, create a new variable and call it **myLastOne**:
- Its data type should be **int**.
- It should be set to the value of **1000**, minus (or less than) the value in the **myTotal** variable, which we've just talked about in our previous code segment.
- Next, print out the value of the **myLastOne** variable on the line after you declare it.

Java code is case sensitive

- Java code is case sensitive.
- This includes not only keywords and language syntax, but variable names and data types as well.
- **myLastOne** is not the same variable as **MyLastOne** with a capital **M**.
- **int** in lowercase, is not the same as Int with the first letter capitalized, or **INT**, all in uppercase, etc.

Java’s Primitive Type

- In Java, primitive types are the most basic data types.
- The eight primitive data types in Java are shown in the table below, listed by the type of data stored for each:

Whole number	**Real number (floating point or decimal)**
byte short **int** long	float **double**
Single character	**Boolean value**
char	boolean

Wrapper Classes

The wrapper classes for char and int, Character and Integer respectively, are the only two that differ in name (other than that first capitalized letter) from their primitive types.

Primitive	**Wrapper Class**
byte	Byte
short	Short
char	Character
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Overflow and Underflow in Java

- If you try and put a value larger than the maximum value into an int, you'll create something called an Overflow situation.
- And similarly, if you try to put a value smaller than the minimum value into an int, you cause an Underflow to occur.
- These situations are also known as integer wraparounds.

Overflow and Underflow in Java

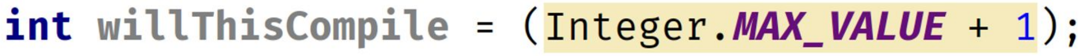
- The maximum value, when it overflows, wraps around to the minimum value, and just continues processing without an error.
- The minimum value, when it underflows, wraps around to the maximum value, and continues processing.
- This is not usually behavior you really want, and as a developer, you need to be aware that this can happen, and choose the appropriate data type.

When will you get an overflow? When will you get an error?

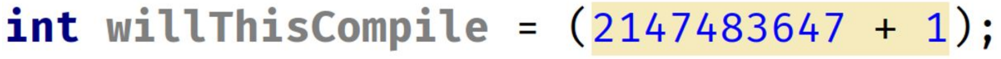
- An integer wraparound event, either an overflow or underflow, can occur in Java when you are using expressions that are not a simple literal value.
- The Java compiler doesn't attempt to evaluate the expression to determine its value, so it DOES NOT give you an error.

When will you get an overflow? When will you get an error?

- Here are two more examples that will compile, and result in an overflow. The second example may be surprising. Even though we are using numeric literals in the expression, the compiler still won't try to evaluate this expression, and the code will compile, resulting in an overflow condition.

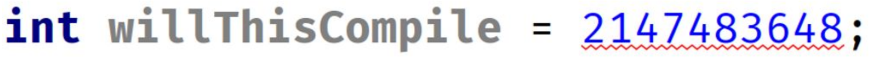


```java
int willThisCompile = (Integer.MAX_VALUE + 1);
```



```java
int willThisCompile = (2147483647 + 1);
```

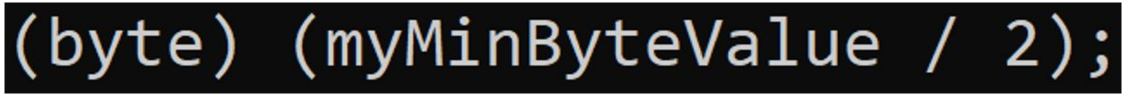
- If you assign a numeric literal value to a data type that is outside of the range, the compiler DOES give you an error. We looked at a similar example previously.



```java
int willThisCompile = 2147483648;
```

Casting in Java

- Casting means to treat or convert a number, from one type to another. We put the type we want the number to be, in parentheses like this:



```
(byte) (myMinByteValue / 2);
```