# Lets Code!

## Fundamentals: Types and Operators

Instructor: Tariq Hook

You can find me on github @code-rhino

# Key Terms

- Reserved Words
- Comments
- Data Types
- Initialize

- Constants
- Operators
- Casting

# A Simple Java Program

```
public class FirstSample {
    public static void main(String[] args) {
        System.out.println("We will not use 'Hello, World!'"); }
}
```

# Java

- Java is case sensitive
- Class names must begin with a letter, and after that, they can have any combination of letters, digits, and underscores.
- By convention, class names start with a capital letter.
- You cannot use a Java reserved word to name a variable or class.
- You need to make the name for the source code the same as the name of the public class
- Whitespace is irrelevant to the Java compiler

# Java

```
public class ClassName {
    public static void main(String[] args) {
        program statements
    }
}
```

# Java

- A Class defines all of the Java code
- In Java (pretty much) everything is an Object
- `main` is merely a special method (a function defined in a class) that is executed when a program runs
- `main` doesn't return anything, thus the `void` return keyword
- `main` doesn't operate on an object, thus the `static` keyword

# Object Stuff

- Since almost everything in Java is an object, for now remember that:
    - Static methods are called on a `Class`, not on objects themselves
        - `SomeObject.someMethod()`
    - Instance methods are called on instances of the objects directly

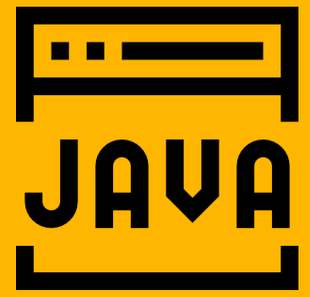
```
SomeObject so = new SomeObject();
so.someInstanceMethod();
```

- Note the construction of `so`

# Java Environment

- Code is compiled with `javac`
    - `javac Example.java`
- Code is compiled to a thing called bytecode and saved in class files
- The bytecode is then run in the JVM (Java Virtual Machine)
    - This means that bytecode can be run on any machine that has the JVM
    - `java Example`

# Comments



# Comments

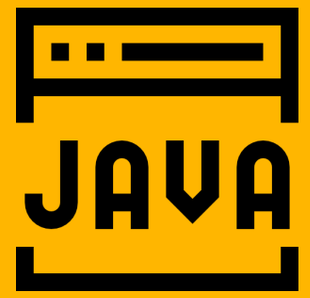
Java has three ways of marking comments:

1. `//` for single line comment
2. `/*` and `*/` for multiline comments
3. using a `/**` to start and a `*/` to end will be used to generate documentation automatically

# Comments

```
/**
 * This is the first sample program in Core Java Chapter 3
 * @version 1.01 1997-03-22
 * @author Gary Cornell
 */
public class FirstSample {
    public static void main(String[] args) {
        System.out.println("We will not use 'Hello, World!'");
    }
}
```

CAUTION: /* */ comments do not nest in Java.




# Data Types

# Data Types

Java is a strongly typed language and every variable must have a declared type.

# There are eight primitive types in Java:

1. int
2. short
3. long
4. byte
5. float
6. double
7. char
8. boolean

# Integer Types

The integer types are for numbers without fractional parts.

| Type | Storage requirements | Range (Inclusive) |
|---|---|---|
| int | 4 bytes | -2,147,483,648 to 2,147,483, 647 (just over 2 billion) |
| short | 2 bytes | -32,768 to 32,767 |
| long | 8 bytes | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| byte | 1 byte | -128 to 127 |

# Floating-Point Types

The floating-point types denote numbers with fractional parts

| Type | Storage requirements | Range (Inclusive) |
|---|---|---|
| float | 4 bytes | Approximately +/-3.40282347E+38F (6–7 significant decimal digits) |
| double | 8 bytes | Approximately +/-1.79769313486231570E+308 (15 significant decimal digits) |

# Floats

Due to repeating decimals and the amount of significant digits in floating point types you can get incorrect values due to roundoff error.

For example, `2.0 - 1.1` does not return `0.9`.

It returns `0.8999999999999999`

# The char Type

Literal values of type char are enclosed in single quotes.

Values of type char can be expressed as hexadecimal values that run from \u0000 to \uFFFF

| Escape sequence | Name | Unicode Value |
|---|---|---|
| \b | Backspace | \u0008 |
| \t | Tab | \u0009 |
| \n | Linefeed | \u000a |
| \r | Carriage return | \u000d |
| \" | Double quote | \u0022 |
| ' | Single quote | \u0027 |
| \\ | Backslash | \u005c |

# The boolean Type

The boolean type has two values, false and true.

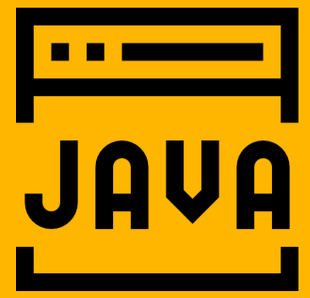
# Big Numbers

If the precision of the basic integer and floating-point types is not sufficient, you can turn to a couple of handy classes in the java.math package: BigInteger and BigDecimal.

```
BigInteger a = BigInteger.valueOf(100);
```

you cannot use the familiar mathematical operators such as + and * to combine big numbers

```
BigInteger c = a.add(b); // c = a + b
BigInteger d = c.multiply(b.add(BigInteger.valueOf(2))); // d = c * (b + 2)
```




# Variables

# Variables

In Java, every variable has a type. You declare a variable by placing the type first, followed by the name of the variable.

```
double salary;
int vacationDays;
long earthPopulation;
boolean done;
```

The semicolon is necessary because a declaration is a complete Java statement.

You can declare multiple variables on a single line:

```
int i, j; // both are integers
```

# Initializing Variables

After you declare a variable, you must explicitly initialize it by means of an assignment statement you can never use the value of an uninitialized variable.

```
int vacationDays;
System.out.println(vacationDays); // ERROR--variable not initialized
```

# Constants

The keyword final indicates that you can assign to the variable once, and then its value is set once and for all. Also known as a constant.

```
public class Constants {
    public static void main(String[] args) {
        final double CM_PER_INCH = 2.54;
        double paperWidth = 8.5;
        double paperHeight = 11;
                System.out.println("Paper size in centimeters: "
          + paperWidth * CM_PER_INCH + " by " + paperHeight * CM_PER_INCH);
    }
}
```

It is probably more common in Java to create a constant so it's available to multiple methods inside a single class. These are usually called class constants.

```
public class Constants2 {
    public static final double CM_PER_INCH = 2.54;

    public static void main(String[] args) {
        double paperWidth = 8.5;
        double paperHeight = 11; System.out.println("Paper size in centimeters: "
            + paperWidth * CM_PER_INCH + " by " + paperHeight * CM_PER_INCH);
    }
}
```

# Operators

# Operators

The usual arithmetic operators

| Operators | Arithmetic |
| --- | --- |
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| % | Modulus |

The / operator denotes integer division if both arguments are integers, and floating point division otherwise.

# Mathematical Functions and Constants

The Math class contains an assortment of mathematical functions that you may occasionally need

```
double x = 4;
double y = Math.sqrt(x);
System.out.println(y); // prints 2.0

double y = Math.pow(x, a);
```

The Math class supplies the usual trigonometric functions:

- Math.sin
- Math.cos
- Math.tan
- Math.atan
- Math.atan2

# Conversions between Numeric Types

19 specific conversions on primitive types are called the widening primitive conversions:

- byte to short, int, long, float, or double
- short to int, long, float, or double
- char to int, long, float, or double
- int to long, float, or double
- long to float or double
- float to double

A widening primitive conversion does not lose information about the overall magnitude of a numeric value.

```
int n = 123456789;
float f = n; // f is 1.23456792E8
```

# Casting

Conversions in which loss of information is possible are done by means of casts.

```
double x = 9.997;
int nx = (int) x;
```

```
double x = 9.997;
int nx = (int) Math.round(x);
```

# Combining Assignment with Operators

`x+=4;` is equivalent to `x=x+4;`

# Increment and Decrement Operators

n++ adds 1 to the current value of the variable n, and n-- subtracts 1 from it

```
int n = 12; n++;
int m = 7;
int n = 7;

int a = 2 * ++m; // now a is 16, m is 8
int b = 2 * n++; // now b is 14, n is 8
```

# Relational and boolean Operators

To test for equality, use a double equal sign, ==.

```
3==7 // is false.
```

Use a != for inequality.

```
3!=7 // is true.
```

Java uses && for the logical "and" operator and || for the logical "or" operator. The exclamation point ! is the logical negation operator.

The && and || operators are evaluated in "short circuit" fashion

The second argument is not evaluated if the first argument already determines the value

```
expression1 && expression2
```

```
x != 0 && 1 / x > x + y //no division by 0
```

The value of expression1 || expression2 is automatically true if the first expression is true, without evaluating the second expression.

Java supports the ternary ?: operator

condition ? expression1 : expression2

x < y ? x : y //gives the smaller of x and y

# Enumerated Types

You can define your own enumerated type

```
enum Size { SMALL, MEDIUM, LARGE, EXTRA_LARGE
};
```

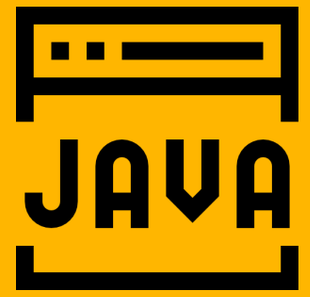
Now you can declare variables of this type:

```
Size s = Size.MEDIUM;
```

# Wrap Up

- Reserved Words
- Comments
- Data Types
- Initialize

- Constants
- Operators
- Casting




# Keep Coding !!!

## Clean Code is Happy Code