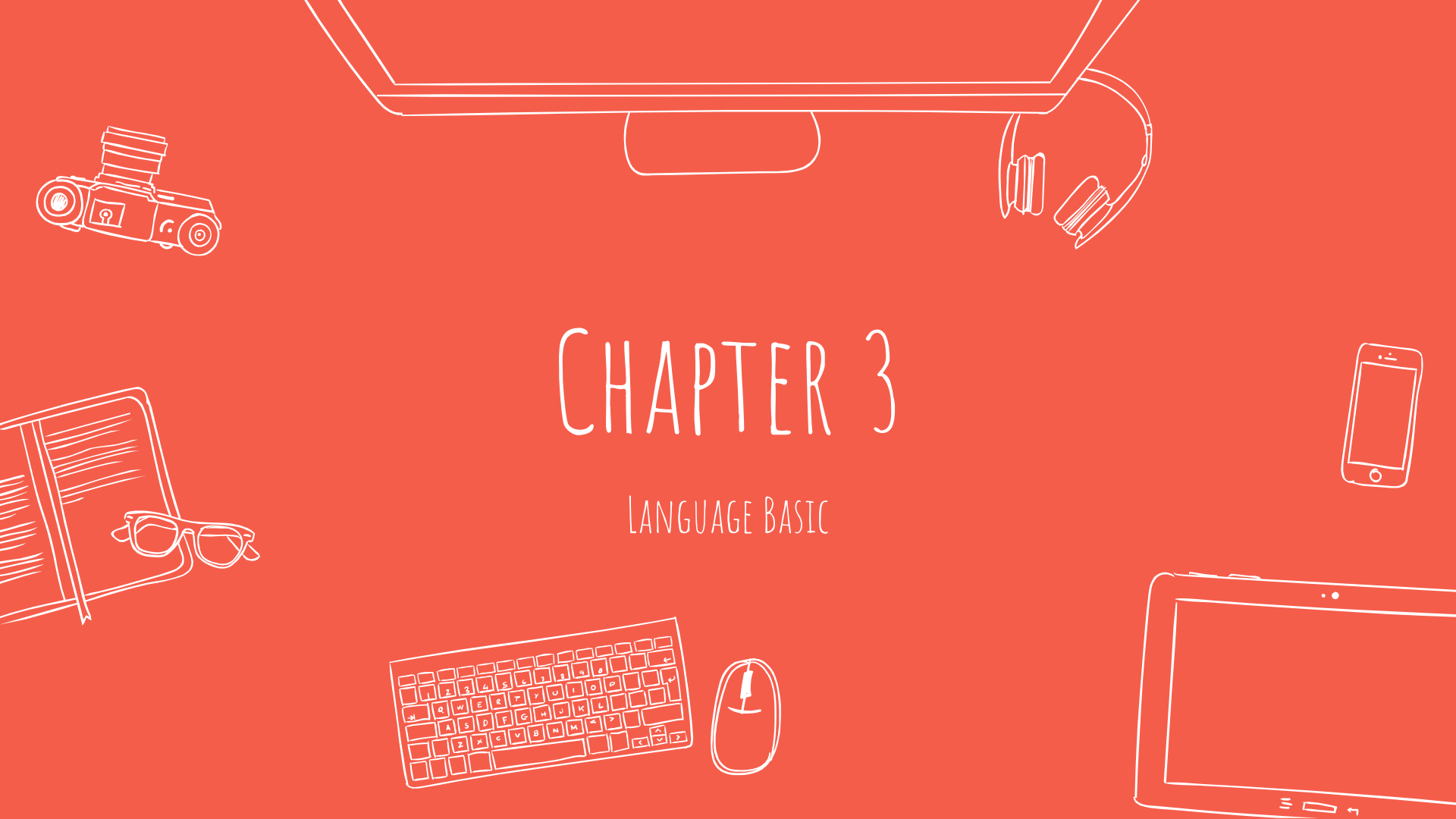


# CHAPTER 3

## LANGUAGE BASIC



# LANGUAGE BASIC

Comments

Variable

Operators

# COMMENTS

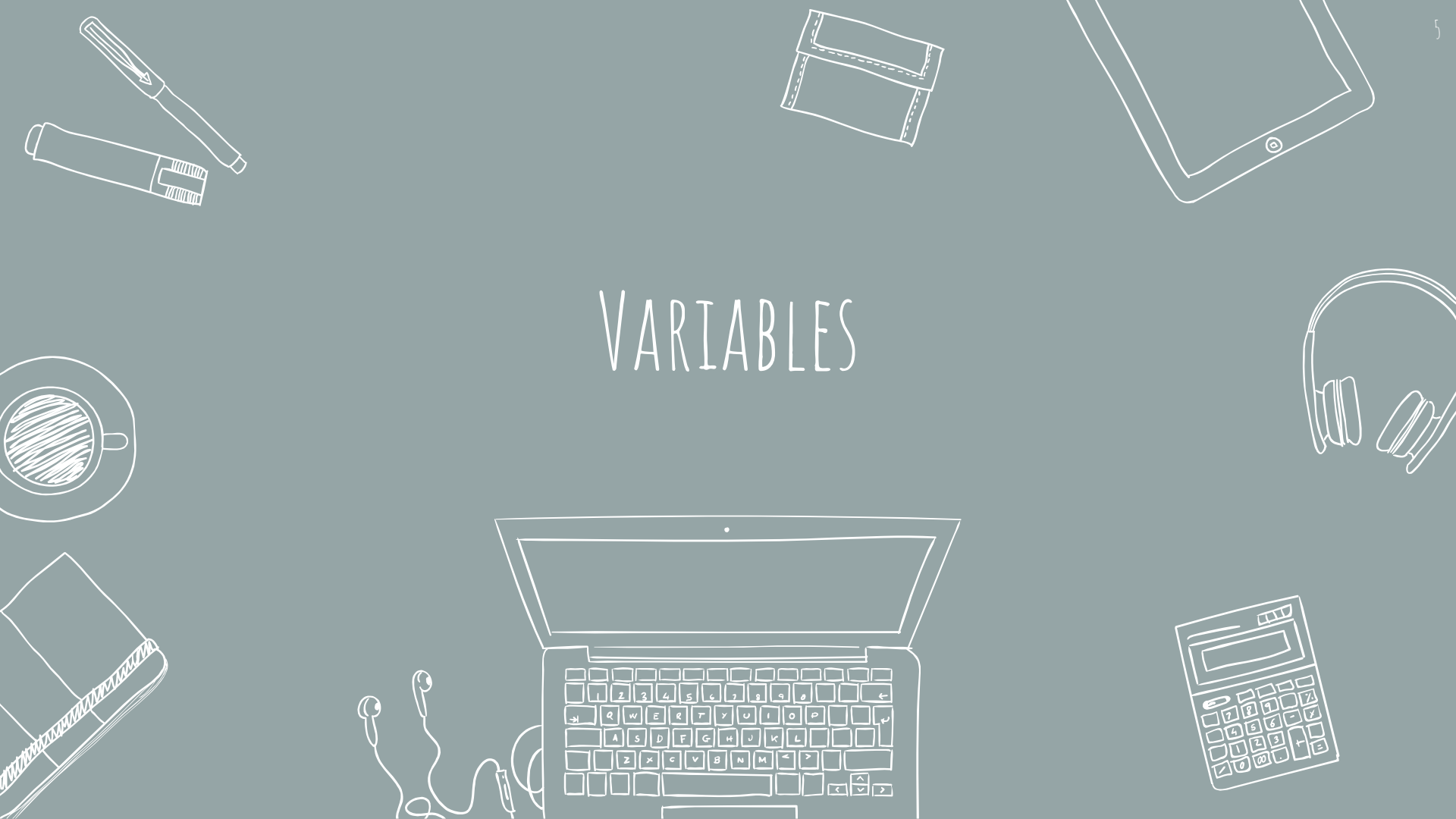


## COMMENT

Java supports following type of comments

- Single line comment
  - E.g. `// This is test`
- Multi-line comment
  - E.g. `/* This is test */`
- Java Doc comment
  - E.g. `/** Class, Method or variable document comment */`

# VARIABLES



# VARIABLE

Why?

E.g.:

- `int speed = 0;`
- `int gear = 1;`
- `String name = "MyName";`



# VARIABLE NAMES

Case-sensitive

- number and NUMBER are not same



# VARIABLE NAMES

## Variables Name Rules:

- Can begin with a letter, \$ or \_.
- Can not start with number
- Subsequent characters may be letters, digits, \$, or \_
- White space is not permitted





## VARIABLE NAMES

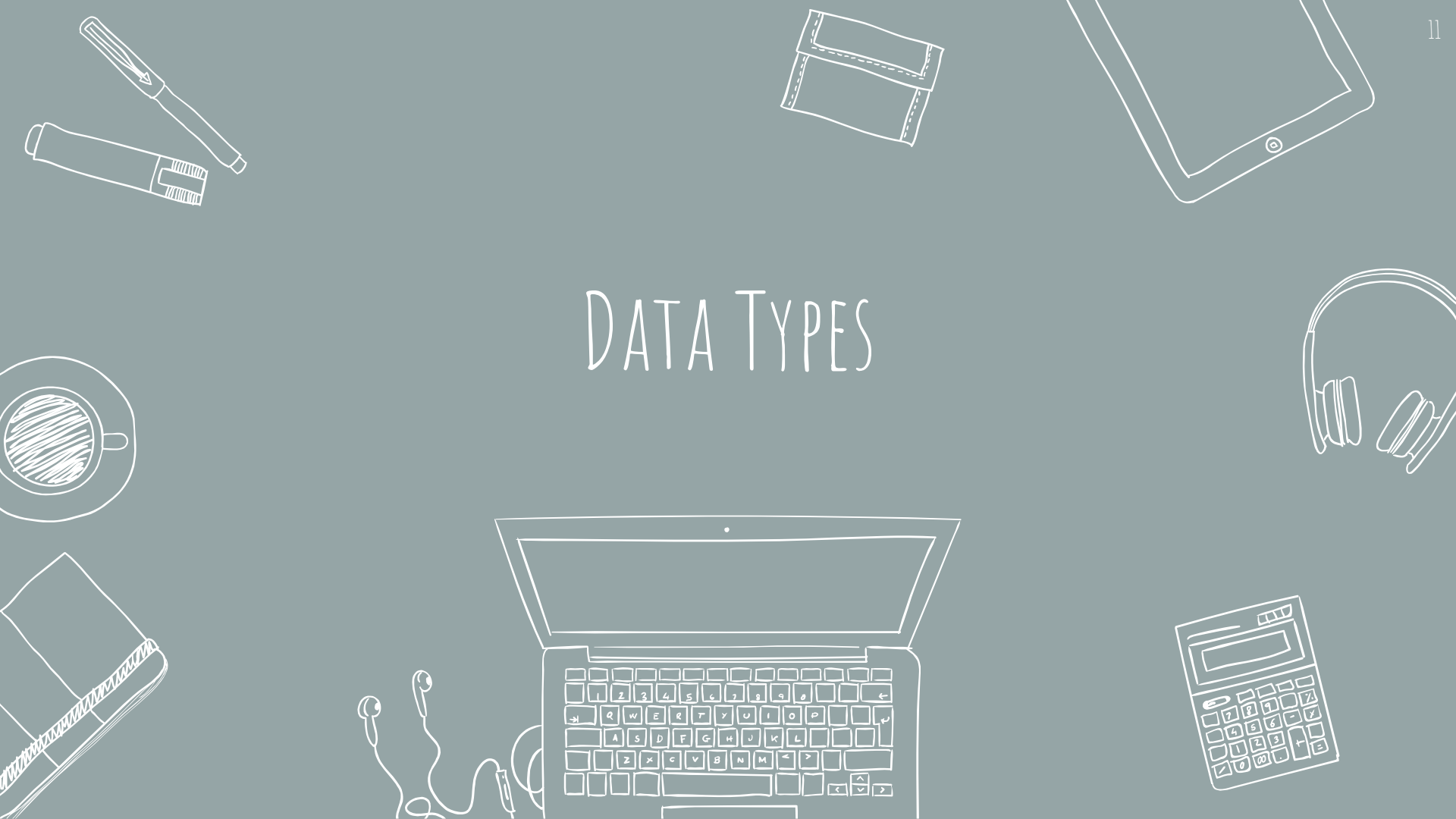
Variable naming convention:

- For constants use all uppercase with underscore character as separator
- Do not use \$ in var name
- Do not start var name with \_
- Use camel case for variable name
- Parameter always start with a prefix
- Use “this” prefix for instance variable.

## VARIABLES EXAMPLE

```
int numberOfStudents;  
float PI_VALUE = 3.14;  
int add (int aFirstNumber, int aSecondNumber) {  
    return aFirstNumber + aSecondNumber -  
    this.index; }  
String NULL_VALUE = null;
```

# DATA TYPES



# DATA TYPE

Primitive Data Types  
Object Data Type

## PRIMITIVE DATA TYPES

**byte:** The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive).

**short:** The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive).

**int:** The int data type is a 32-bit signed two's complement integer. It has a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive).

**long:** The long data type is a 64-bit signed two's complement integer. It has a minimum value of -9,223,372,036,854,775,808 and a maximum value of 9,223,372,036,854,775,807 (inclusive).

## PRIMITIVE DATA TYPES

**float:** The float data type is a single-precision 32-bit IEEE 754 floating point. **This data type should never be used for precise values, such as currency.** For that, you will need to use the `java.math.BigDecimal` class instead.

**double:** The double data type is a double-precision 64-bit IEEE 754 floating point. For decimal values, this data type is generally the default choice. **As mentioned above, this data type should never be used for precise values, such as currency.**

**boolean:** The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditions.

**char:** The char data type is a single 16-bit Unicode character. It has a minimum value of `'\u0000'` (or 0) and a maximum value of `'\uffff'` (or 65,535 inclusive).

## DEFAULT VALUE

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

# LITERALS

```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 100;
int i = 100000;
// The number 26, in hexadecimal
int hexVal = 0x1a;
// The number 26, in binary
int binVal = 0b11010;
double d1 = 123.4;
// same value as d1, but in scientific notation
double d2 = 1.234e2;
float f1 = 123.4f;
long creditCardNumber = 1234_5678_9012_3456L;
float pi = 3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
```



# QUIZ

Which of the following  
is *not* a valid comment:

- a. `/** comment */`
- b. `/* comment */`
- c. `/* comment`
- d. `// comment`



# STRING

String is special class in java

This is not a primitive data type

This is a class

String has many utility methods like

- toLower
- trim
- toUpper
- isEmpty

# OPERATORS



# OPERATORS

Operators are special symbols that perform specific operations on one, two, or three operands, and then return a result.

Operators with higher precedence are evaluated before operators with relatively lower precedence.

All binary operators except for the assignment operators are evaluated from left to right; assignment operators are evaluated right to left.

# OPERATORS

## Simple Assignment Operator

= Simple assignment operator

E.g.:

```
numberOfPizza = 3;
```

# OPERATORS

## Arithmetic Operators

- + Additive operator (also used for String concatenation)
- Subtraction operator
- \* Multiplication operator
- / Division operator
- % Remainder operator

E.g.:

```
total = number1 + number2;
```

```
diff = number1 - number2;
```

# OPERATORS

## Unary Operators

- + Unary plus operator; indicates positive value (numbers are positive without this, however)
- Unary minus operator; negates an expression
- ++ Increment operator; increments a value by 1
- Decrement operator; decrements a value by 1
- ! Logical complement operator; inverts the value of a Boolean

E.g.:

```
number1 = 10;
```

```
number1++;
```

```
largeNumber = true;
```

```
smallNumber = ! largeNumber
```

## QUIZ

What will be the output following:

```
int number1 = 10;
```

```
System.out.println("Number " + number1);
```

```
number1++;
```

```
System.out.println("Number " + number1);
```

```
number1--;
```

```
System.out.println("Number " + number1);
```

```
System.out.println("Number " + ++number1);
```

```
System.out.println("Number " + number1++);
```



# OPERATORS

## Equality and Relational Operators

- `==` Equal to
- `!=` Not equal to
- `>` Greater than
- `>=` Greater than or equal to
- `<` Less than
- `<=` Less than or equal to

E.g:

```
number1 = 10; number2 = 20;  
areNumberEqual = (number1 == number2);  
greater = number1 > number2
```

# OPERATORS

## Conditional Operators

&&    Conditional-AND

||    Conditional-OR

?:    Ternary (shorthand for if-then-else statement)

E.g.:

`allNumberMoreThan100 = num1 > 100 && num2 > 100`

`atleastOneNumberMoreThan100 = num1 > 100 || num2 > 100`

`maxNumber = num1 > num2 ? num1 : num2;`

# OPERATORS

## Type Comparison Operator

`instanceof`     Compares an object to a specified type

## Bitwise and Bit Shift Operators

- `~`     Unary bitwise complement
- `<<`     Signed left shift
- `>>`     Signed right shift
- `>>>`   Unsigned right shift
- `&`     Bitwise AND
- `^`     Bitwise exclusive OR
- `|`     Bitwise inclusive OR

# OPERATOR PRECEDENCE

## Operator Precedence

Operators	Precedence
postfix	<code>expr++ expr--</code>
unary	<code>++expr --expr +expr -expr ~ !</code>
multiplicative	<code>* / %</code>
additive	<code>+ -</code>
shift	<code>&lt;&lt; &gt;&gt; &gt;&gt;&gt;</code>
relational	<code>&lt; &gt; &lt;= &gt;= instanceof</code>
equality	<code>== !=</code>
bitwise AND	<code>&amp;</code>
bitwise exclusive OR	<code>^</code>
bitwise inclusive OR	<code> </code>
logical AND	<code>&amp;&amp;</code>
logical OR	<code>  </code>
ternary	<code>? :</code>
assignment	<code>= += -= *= /= % = &amp;= ^=  = &lt;&lt;= &gt;&gt;= &gt;&gt;&gt;=</code>

# EQUALS

= operator works on primitive data type

= Operator doesn't work on class data type

Use "equals" method to compare 2 objects

E.g.

- `5 = 5` returns true
- `String name1 = "Vijay"`
- `String name2 = "Vijay"`
- `name1 == name2` returns false

# QUIZ

30

Which one is valid method definition:

- a. `Public static void main () {}`
- b. `public static void main ()`
- c. `public static void main () {}`
- d. `public static main (String[] args) {}`



THANKS!

**Any questions?**

You can find me at:

`vijay_garry@hotmail.com`

<https://github.com/vijaygarry>

