

COM2104: Advanced Programming

LECTURE 1: INTRODUCTION

Module description

- **PREREQUISITES:** You should already be able to program in Java (those covered in COM1101)
- Learn object-oriented design techniques
- Practice objected oriented language features such as inheritance, polymorphism etc
- Construct code which is **well-structured** and **maintainable**

Timeline for COM2005 Tentative

WEEK	Content
Week 1	Lecture 1: Get Started
Week 2	Lecture 2: User input & Exception handling
Week 3	Lecture 3: Inheritance& method overriding
Week 4	Lecture 4: Aggregation & Arraylist
Week 5	Lecture 5: Abstract & Interface
Week 6	Lecture 6: Further knowledge about arraylist
Week 7	Lecture 7: Threading (March 8th, mid-term test).

WEEK	Content
Week 8	(Release answer for mid-term test) Lecture 8: Graphical User Interface (GUI)
Week 9	Lecture 9: Generics & Recursion
Week 10	Lecture 10:File IO & Stream
Week 11	Lecture 11: Functional Programming
Week 12	Lecture 12: Enum
Week 13	Recap for final exam.
Week 14	No class.

Contact Information

Dr. Wen Ma (Lecture + Tutorial)

- Department of Computer Science <https://cs.hsu.edu.hk/en/dr-ma-wen/>
- Office: N415-22
- Email: wenma@hsu.edu.hk
- Tel: 3963 5099 (ext.099)
- Consultation hours: anytime by any means, e.g., email, face-to-face / online meeting, phone call or WhatsApp / WeChat)

Structure of class about COM2104

- Lecture notes ~ **1** to **1.5** hours
 - Introduce the related content
- Lab Exercise ~ the **remaining** time.
 - Some simple exercise questions
 - Assignment
- Programming Tool(s)
 - **Eclipse (mainly)**
 - NetBeans (optional)

Class Policy

- No Cell Phone in classroom
 - Do not disturb your classmates
- Attend the classes punctually. Take attendance in each class from week 1. **Attendance rate $\geq 80\%$ for attending final examination.**
- No Plagiarism
 - Plagiarism is a SERIOUS offense in academia
 - VeriGuide: <https://itsc.hsu.edu.hk/veriguide/>

Assessments

- Class participation [5%]
 - Final Attendance **rate** $\geq 80\%$ ---> Get full 5 marks
- Assignments [30%]
 - Individual assignment (10%). **DUE BY 12th April, 23:59.**
 - 10 Lab assignments (20%) for submission. **DUE** at midnight on the day of each class.
- Mid-term Test [15%]
 - 8th March (SAT), 9:30-11:30AM.
- Final Exam [50%]
 - The exact time is pending.



Details about assessment for individual assignment

- Individual assignment (**10%**). **DUE BY 12th April, 23:59.**
- **Submit** .java files.
- **4 short questions** requiring you to program with Java.
- Programming tools:
 - Eclipse **or**
 - NetBeans **or**
 - Other you are familiar with,

Details about assessments for lab assignments

- Each lab assignment will be demonstrated **in class**. So, **attend** the class is **important**.
- Assignment of lecture **1** and **2** will be **released with the answer to you**. No need to submit them.

Assignment	Points
Assignment of lecture 3	2
Assignment of lecture 4	2
Assignment of lecture 5	2
Assignment of lecture 6	2
Assignment of lecture 7	2
Assignment of lecture 8	2
Assignment of lecture 9	2
Assignment of lecture 10	2
Assignment of lecture 11	2
Assignment of lecture 12	2

Details about assessments for mid-term test

- Date: 8th March (SAT), 9:30-11:30AM
- Closed book.
- Bring your calculator
- **15 MC** questions. **1 short question** that require you to **write code** by hands and **2 other short questions** requiring you to **program on computers**.
- Place: **M704&M705**. **Assigned Seat**. (The seats number are pending.)

Textbook Reference

- No official textbook
- Reference book list:
 - 1. Starting Out with Java: From Control Structures through Objects, Tony Gaddis, Pearson Higher Education
 - 2. Java: to program: early objects. how Deitel, Paul J., author.; Deitel, Harvey M., 1945
 - 3. Java, Java, Java Object-Oriented Problem Solving, by R. Morelli and R. Walde.
 - <http://www.cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf>

WARM UP WITH JAVA

JAVA BASICS

Some Online resources

1. W3schools: <https://www.w3schools.com/java/>
2. Oracle's Official Java Tutorials: <https://docs.oracle.com/javase/tutorial/>
3. Codecademy: <https://www.codecademy.com/learn/learn-java>

Variables

- Variables are used to **store any types of data**.
- Variables are named **locations** in **memory**.
- Variables also give context and meaning to the data we're storing. The value 42 could be someone's age, a weight in pounds, or the number of orders placed

Naming a piece of information allows us to use that name later, accessing the information we stored.



Primitive V.S. Non-primitive

- Primitive data type specify the size and type of variable values, and it has no additional methods.
- Non-primitive data types are called reference types because they refer to objects.

Primitive data types refer to int, float, double etc. Non-primitive data types refer to others, like **an object** of one class.



Main difference between primitive and non-primitive data types

- Primitive types in Java are **predefined** and built into the language, while non-primitive types are created by the **programmer** (except for **String**).
- Non-primitive types can be used to **call methods** to perform certain operations, whereas primitive types **cannot**.
- Primitive types start with a lowercase letter (like **int**), while non-primitive types typically starts with an uppercase letter (like **String**).
- Primitive types always hold a value, whereas non-primitive types can be **null**.



Some examples about primitive data types

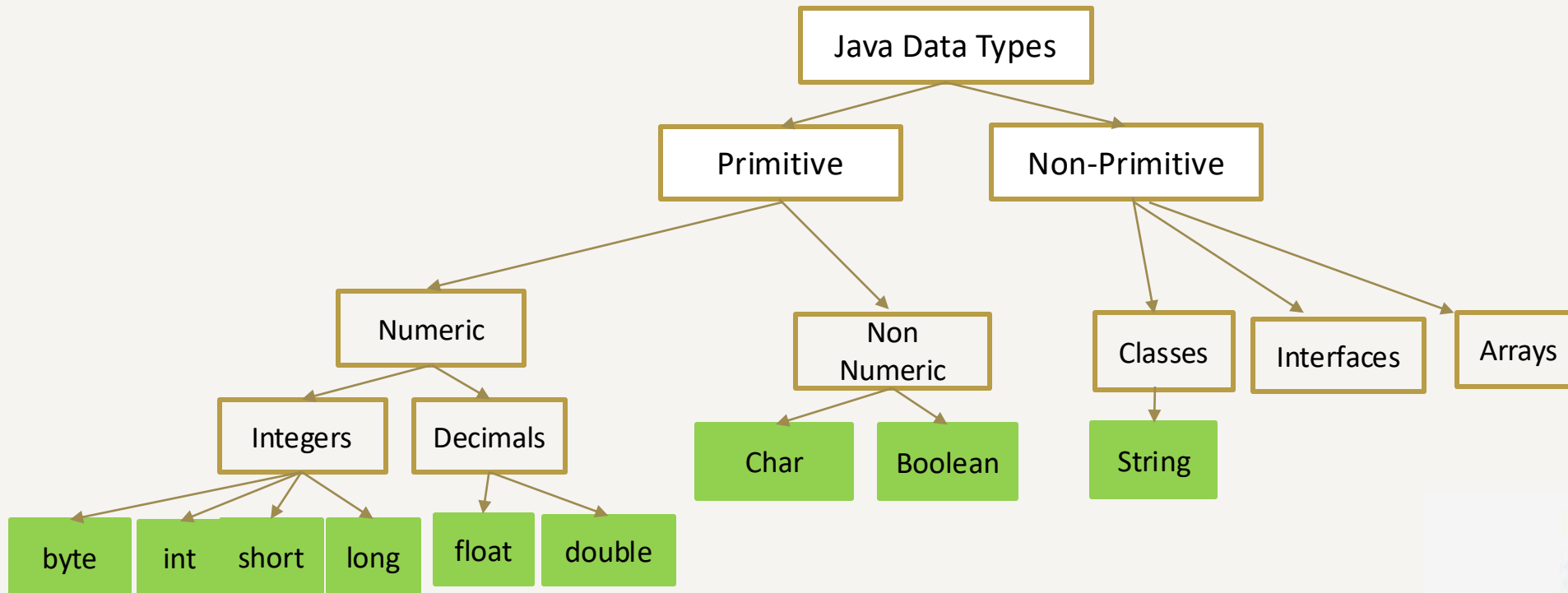
Type	bits	description
byte	1 byte	Small Integers in the range $-128(-2^7)$ to $+127(2^7 - 1)$
short	2 bytes	Integers in the range of $-32,768$ to $+32,767$
int	4 bytes	Integers in the range of $-2,147,483,648$ to $+2,147,483,647$
long	8 bytes	Integers in the range of $-9,223,372,036,854,775,808$ to $+9,223,372,036,854,775,807$
float	4 bytes	Floating-point numbers in the range of approximately -3.4×10^{-38} to $+3.4 \times 10^{38}$
double	8 bytes	Floating-point numbers in the range of -1.7×10^{-308} to $+1.7 \times 10^{308}$
Boolean		true or false (used for decision making)

1 byte = 8 bits

Naming Conventions in Java

Type	Description	Examples
variable	starting with small letters then each word start with a capital	ageCount, sumCount.
constants	using all capital letters	PI, COUNT
method	starting with small letters then each word start with a capital	sumTwoNumbers (), minValue ()
class	starting with a capital	Sample, AddTwoNumbers
package	using all small letters	java.io, java.lang,
keyword	using all small letters	int, short, float, public, void

Data type classification



Remember { meaning
function

✓ Java Reserved Keywords

abstract	double	instanceof	static
assert	else	int	strictfp
boolean	enum	interface	super
break	extends	long	switch
byte	false	native	synchronized
case	for	new	this
catch	final	null	throw
char	finally	package	throws
class	float	private	transient
const	goto	protected	true
continue	if	public	try
default	implements	return	void
do	import	short	volatile
			while

These keywords cannot be used to name a variable



✓ Commenting Code

- Java provides two methods for commenting code.

Comment Style	Description
//	Single line comment. Anything after the // on the line will be ignored by the compiler.
/* ... */	Block comment. Everything beginning with /* and ending with the first */ will be ignored by the compiler. This comment type cannot be nested. E.g., /* * * *.... */

Examples about commenting code

```
//this is a single line comment of the code  
System.out.println("Hello");  
/* This is the comment of the code  
 * Multiple lines comments.  
 * */  
System.out.println("Hello again");
```

✓ Java Escape Sequences (Mainly for printing)

<code>\n</code>	newline	Advances the cursor to the next line for subsequent printing
<code>\t</code>	tab	Causes the cursor to skip over to the next tab stop
<code>\\</code>	backslash	Causes a backslash to be printed
<code>\'</code>	single quote	Causes a single quotation mark to be printed
<code>\"</code>	double quote	Causes a double quotation mark to be printed

Examples for using escape sequences

```
//usage for \n: printing the following content in the next line,  
System.out.println("This is a test message.\nThis is the test again.");  
//usage for \t: add a tab space before printing the following content  
System.out.println("This is a test message.\tThis is the test again.");  
//printing \  
System.out.println("This is a test message.\\This is the test again.");  
//print ' (single quote)  
System.out.println("This is a test message.\'This is the test again.\'");  
//print " (double quotes)  
System.out.println("This is a test message.\"This is the test again.\"");
```

Output:

```
This is a test message.  
This is the test again.  
This is a test message. This is the test again.  
This is a test message.\tThis is the test again.  
This is a test message.'This is the test again.'  
This is a test message."This is the test again."
```


$$\text{Math.min}(5, 3) = 3$$

Math methods

Methods	description	example
Math.min(x,y)	Returns the number with the lowest value	Math.min(5,6)
Math.max(x,y)	Returns the number with the highest value	Math.max(-9,11)
Math.sqrt(x)	Returns the square root of x	Math.sqrt(25)
Math.abs(x)	Returns the absolute value of x	Math.abs(-7.5)
Math.pow(x,y)	Returns the value of x to the power of y	Math.pow(2,3)
Math.ceil(x)	Returns the value of x rounded up to its <u>nearest integer</u>	Math.ceil(2.3)
Math.floor(x)	Returns the value of x rounded down to its nearest integer	Math.floor(-2.3)

Return float
Value

Return 8.0

3.4-5.6
↑
ceil = 2.3

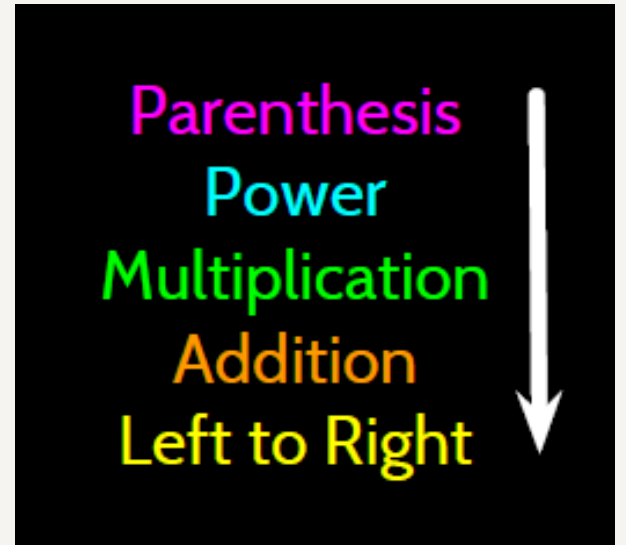
取整

Rounded up to its nearest integer

Operation priority Operator Precedence Rules

From the highest precedence rule to the lowest precedence rule:

- **Parenthesis** are always respected ()
- **Exponentiation** (raise to a power)
- **Multiplication, Division, and Remainder**
- **Addition and Subtraction**
- **Left to right**



Operator Precedence Rules: Cont.

$$X=1+2*3/(4*5) \quad ?$$

$$1+2*3/(4*5) \quad 1. \text{Parenthesis()}$$

$$1+2*3/20 \quad 2. \text{Multiplication}$$

$$1+6/20 \quad 3. \text{Division}$$

$$1+0 \longrightarrow 1 \quad 4. \text{Addition}$$

Parenthesis
Power
Multiplication
Addition
Left to Right



Operator Precedence Rules: Cont.

$X = 1 + \text{Math.pow}(3, 2) * 2 / 4 - 3 ?$

$1 + 9.0 * 2 / 4 - 3$ 1.power

$1 + 18.0 / 4 - 3$ 2.Multiplication

$1 + 4.5 - 3$ 3.Addition

$5.5 - 3 \longrightarrow 2.5$ 4. Subtraction

Parenthesis
Power
Multiplication
Addition
Left to Right



FLOW CONTROL

Logical Operators

Operator	Meaning	Effect
&&	and	Connects two Boolean expressions into one. Both expressions must be true for the overall expression to be true.
	or	Connects two Boolean expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which one.
!	not	The ! operator reverses the truth of a Boolean expression. If it is applied to an expression that is true, the operator returns false. If it is applied to an expression that is false, the operator returns true.

Logical operators return a **Boolean value** for connected Boolean expression(s).

The easiest way to understand logical operators

True **and** False = False

False **and** True = False

True **and** True = True

False **and** False = False

not(True) = False

True **or** False = True

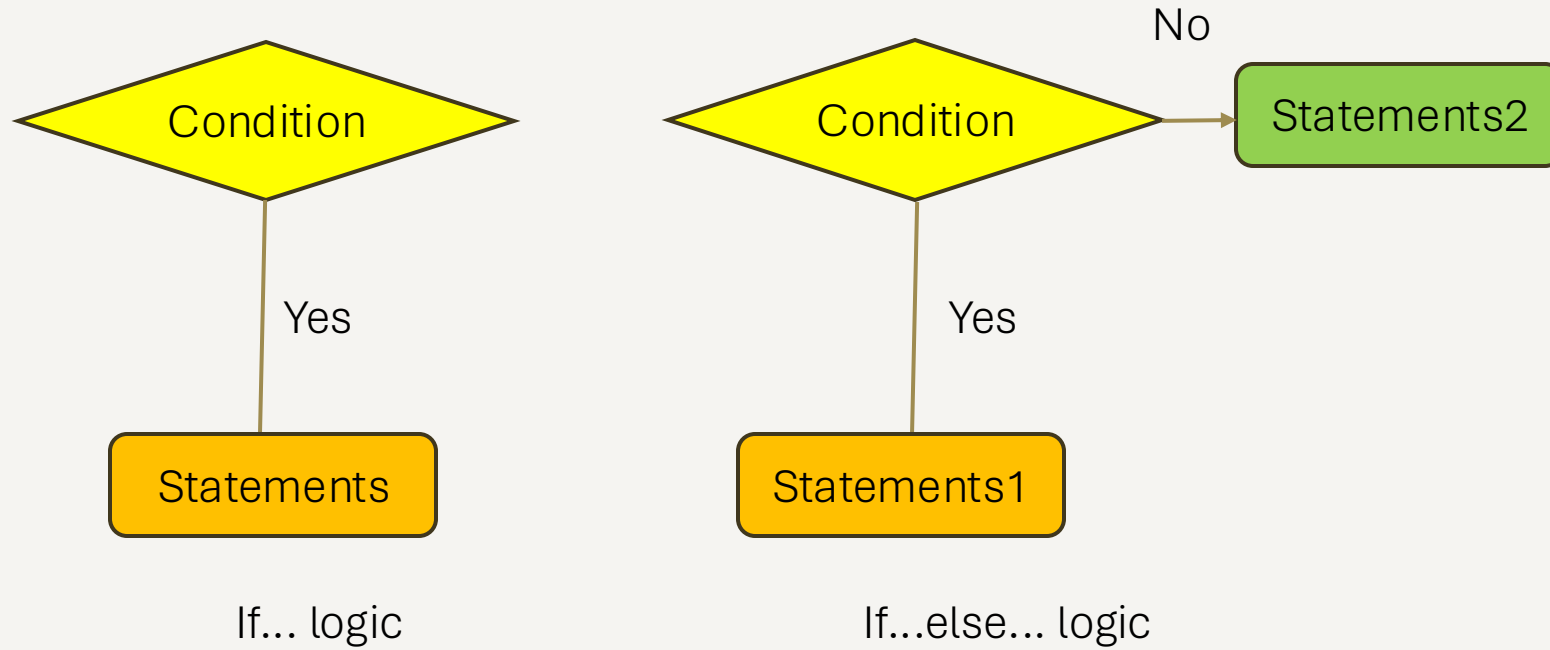
False **or** True = True

True **or** True = True

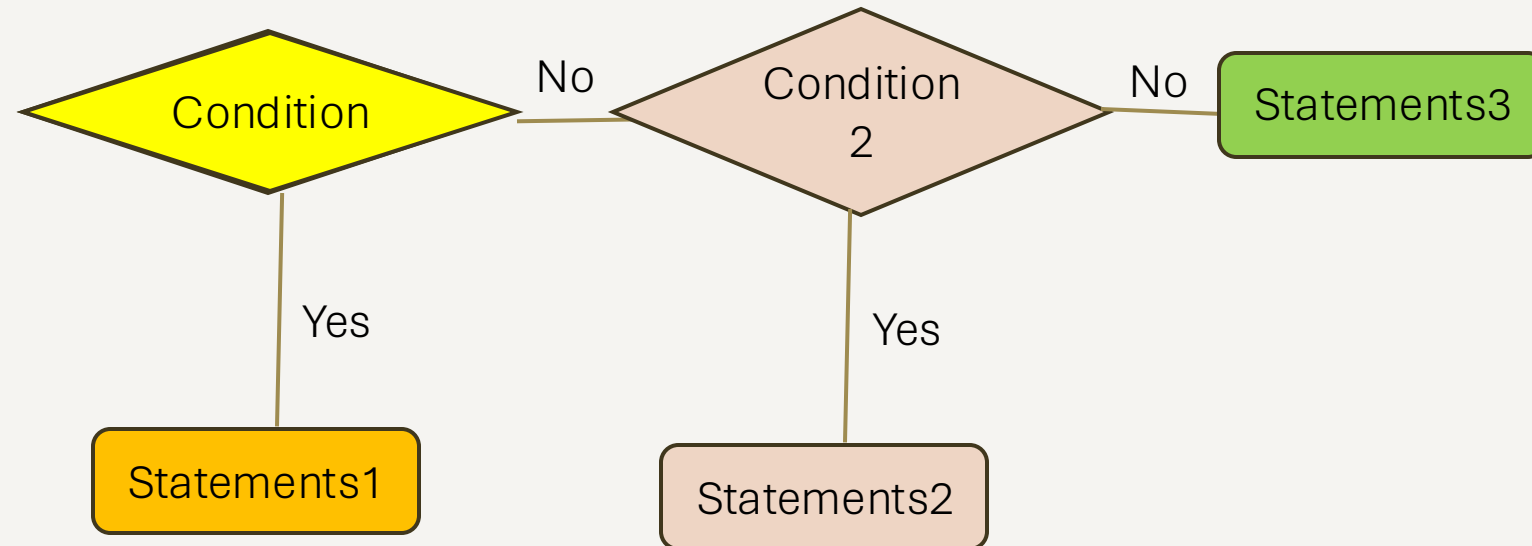
False **or** False = False

not(False) = True

if..., if...else...



if...else if...else... logic



if... , if...else... , if...else if...else... : Syntax

```
if (expression)
{
    Statements;
}
```

```
if (expression)
    statements;
else
    statements;
```

```
if (expression_1)
{
    statement;
}
else if (expression_2) //
{
    statement;
}
```

Insert **as many else if as necessary**

```
else
{
    statement;
    statement; etc.
}
```

Common loop structures in Java

- The `while` loop:
 - Pretest loop
 - Use it where you **do not want the statements to execute if the condition is false** in the beginning.
- The `do-while` loop:
 - Post-test loop
 - Use it where you want the statements to **execute at least one time**.
- The `for` and `for-each` loop:
 - Pretest loop
 - Use it where there is **some type of counting** variable that can be evaluated.

while loop

```
int x = 20;
while(x > 0)
{
    System.out.println("x is greater than 0");
    x--;
}
```

do-while

```
do
{
    System.out.print("Enter a number in the " +
                    "range of 1 through 100: ");
    number = keyboard.nextInt();
} while (number < 1 || number > 100);
```

The for Loop

```
for (number = 1; number <= 10; number++)  
{  
    System.out.println(number + "\t\t" +  
                        number * number);  
}
```

The for-each Loop

```
int arr[]={11,22,44,88,176};  
    //Print the array using for-each loop  
for(int i:arr)  
{  
    System.out.println(i);  
}
```

CLASS & METHODS

Method: No Returning value

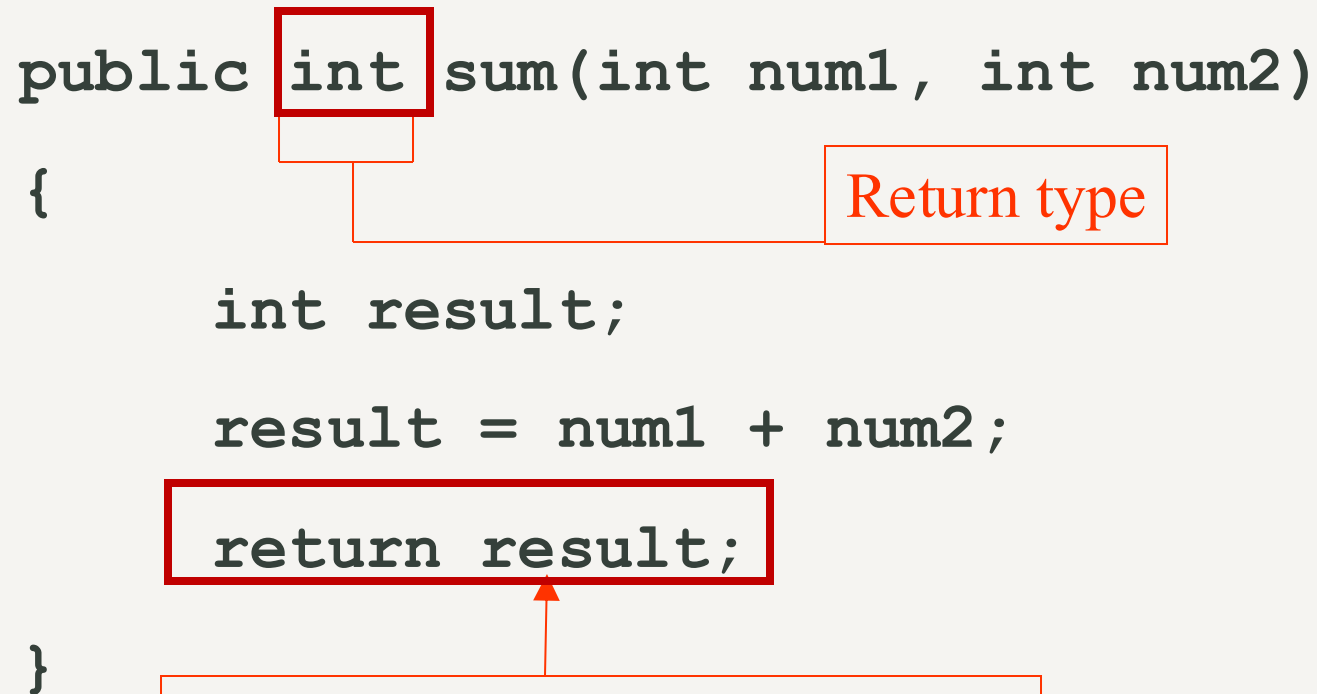
```
public void sum(int num1, int num2)
{
    int result;
    result = num1 + num2;
    Sytem.out.println(result) ;
}
```

using void to represent no value returned

Method: A Value-Returning Method

```
public int sum(int num1, int num2)
{
    int result;

    result = num1 + num2;
    return result;
}
```



The diagram illustrates the components of a value-returning method. A red box highlights the return type 'int' in the method signature. A line connects this box to a label 'Return type'. Another red box highlights the return statement 'return result;'. A line connects this box to a label 'This expression must be of the same data type as the return type'.

The `return` statement causes the method to end execution and it returns a value back to the statement that called the method.

Unified Modeling Language (UML) diagram

- Unified Modeling Language (UML) is a general-purpose modeling language.
- UML is not a programming language, it is rather a visual language.
- The main aim of UML is to define a standard way to **visualize the way a system has been designed**.
- We usually will use UML diagram to **overview the structure** about one class.

<https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/>

UML structure for a class



For attributes and methods, we will use - to represent **private** characteristic, + to represent **public** characteristic, and # to represent **protected** characteristic.

One class example

Constructor



Rectangle

Attributes:

- Private
- + Public

- width : double
- length : double

Methods

- Private
- + Public

+ Rectangle(w : double, len: double)
+ setWidth(w : double) : void → No return value
+ setLength(len : double): void
+ getWidth() : double → Return a double value
+ getLength() : double
+ getArea() : double

```
public class Rectangle{

    private double width;
    private double length;

    public Rectangle(double w, double
len){

        width = w;
        length = len;

    }
    public void setWidth(double w){

        width = w;

    }
    public void setLength(double len){
        length = len;
    }
    public double getWidth(){
        return width;
    }
    public double getLength(){
        return length;
    }
    public double getArea(){
        return length * width;
    }

}
```

Using **static** keyword for a method

```
6 import java.io.*;
7
8 public class StaticExample {
9
10     static int num = 100;
11     static String str = "GeeksForGeeks";
12
13     // This is Static method Exit 'static' can directly call.
14     static void display()
15     {
16         System.out.println("static number is " + num);
17         System.out.println("static string is " + str);
18     }
19
20     // non-static method
21     void nonstatic()
22     {
23         // our static method can accessed
24         // in non static method
25         display();
26     }
27
28     // main method
29     public static void main(String args[])
30     {
31         StaticExample obj = new StaticExample();
32
33         // This is object to call non static method
34         obj.nonstatic();
35
36         // static method can called
37         // directly without an object
38         display();
39     }
40 }
```

- A static method in Java is a method that is part of a class rather than an instance of that class.
- In both static and non-static methods, static methods can be accessed directly.

```
static number is 100
static string is COM2104
static number is 100
static string is COM2104
```

We could access the static method directly without creating an instance.

static Example

obj. ←

instance

Non-Static Without use static word in code line

So obj as instance ~~called~~ ^{need to} the static method.

System.out.printf

- We could indicate the data type we want to print by using System.out.printf
- Three % symbols:
 - **%d**-indicate an int number when printing
 - **%f**-indicate a double or float number when printing
 - **%s**-indicate a String when printing

```
int a = 1;  
System.out.printf("The int number is %d\n",a);  
double b = 1.0;  
System.out.printf("The double number is %f\n",b);  
String s="Hello";  
System.out.printf("The string is %s\n",s);
```

Output

```
The int number is 1  
The double number is 1.000000  
The string is Hello
```





End