

What is System.out.println?

Package

Naming conventions

Access modifier

Method, calling a method and method structure.

IDE

JDK

Edition of JAVA

API

About JAVA

Why learns JAVA

Syllabus

1. Java packages, classes and methods.

2. Public, private and static in Java

3. The void return type in Java

4. Programming styles and errors

5. Creating classes and methods

6. Basic overview

-Method

-Loops

-Conditions

-Operators

-Data types

7. Introduction to variable

-Assignment

-Declaration

-Memory

8.1 Constants in Java

8.2 Using of final keyword

9. Identifier in JAVA

- Tips

- Case sensitivity in JAVA

- Legal or Illegal

- Naming rules

10. Introductions to Data Type

- Characters

- Real numbers

11. The int data type in JAVA

- Initializing int variables

- Range of an int variable

- integers and the int data type

12. The byte, short & long data type in Java

13. Integer data type in Java source code.

14. Bytes and Values

- Bits and Memory

- Size of different integer data types

- Calculation the range

- Which data type to use

15. The double and float Data Type in Java with example

16. The char Data Type in Java with example

17. Boolean data type in Java with example

18. String data types and calling string method

19. Concatenating string in java

20. Primitive and reference type

21. Instantiating a string object with example

22. String are immutable in java

- Immutable objects

-Using new with strings

-Source code

23. Scanner class in java with example

24. Reading input from the keyboard

25. Printing a number simple code

26. Printing name and age

27. Literals in Java simple code

28. Assignment operator

29. Arithmetic operator

30. Increment and Decrement operator

31. Casting

-Implicit casting

-Explicit casting

32. The division operator in Java details

33. The divisor operator example

34. The relational operator in java

35. Logical operator in java(and, or, not)

36. Conditional operator in Java

37. Difference between switch and if else

39. Making simple calculator using if else

40. Lucky number (first tow digit addition = last two digit addition)

41. While loop

42. do while loop

43. structure of for loop

44. for loop printing star and multiplication table

45. Sum of the values

46. Sum of the strict divisors

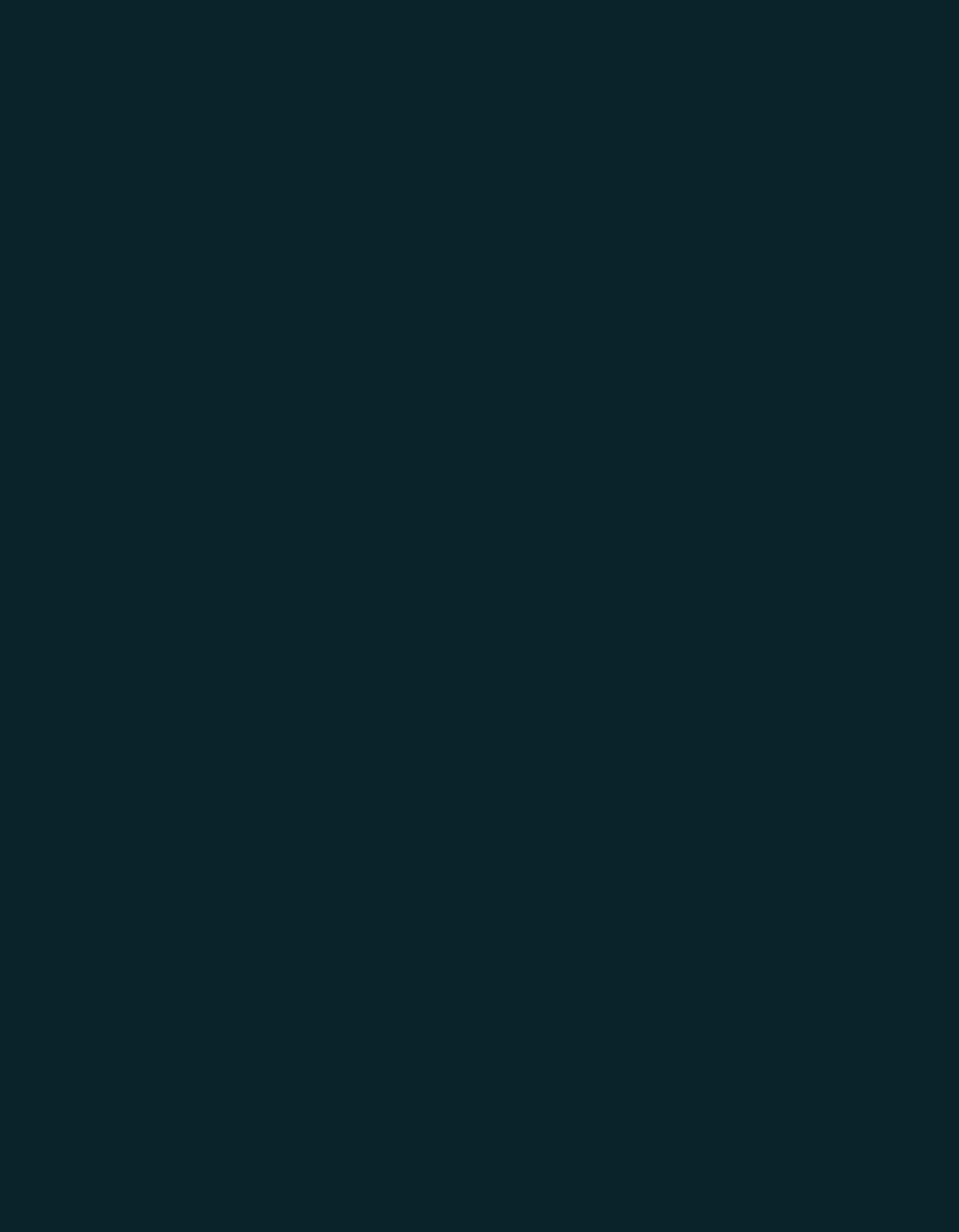
47. Prime number

49. finding max and min

- 50. display the sum of the digit
- 51. Fibonacci number
- 52. Display string with a space
- 53. Reverse string
- 54. String palindrome or not
- 55. Display 1\n22\n333\n4444
- 56. Displaying different type of star
- 57. Displaying different type of star
- 58. Displaying different type of star
- 59. Scope and local variable in java
- 60. Method in java
 - Method in java
 - Value returning in java
- 61. Passing arguments by value.
- 62. Passing arguments by reference
- 63. Method overloading
- 64. Write a method that gets a name and age from the user.
- 65. Prints the prime numbers between two numbers.
- 66. Single dimension array-1(Creating and printing array)
- 67. Single dimension array-2
- 68. Array class(sorting)
- 69. Array class (Searching)
- 70. Array class>equals)
- 71. Array class (filling and toString)
- 72. Variable length arguments (int... numbers)
- 73. Fills an array with n integers
- 74. Fill an array with n integer using point.
- 75. Display the sum, product and average of an integer array.
- 76. display the number of occurrences of an element in the array.

77. Display the maximum and minimum element of an array.
78. Places the odd element of an array before the even element.
79. 2D array (Create and printing)
80. 2D array printing row by row
81. 2D array printing column by column
82. Printing arrays using `toString` method
83. Printing a 2D array using method.
84. Ragged arrays
85. Print the sum of each row and each column of an 2D array.
86. Prints the maximum of each row and each column in a 2D array.
87. Print the maximum of each row using method.
88. ArrayList-Using the `add()`, `get()`, `set()`, `remove()`, `size()` method.
89. Printing an array list using for loop and using of sort method of collections classes
90. Using for each loop for printing an arraylist or array.
91. Create a list of unique elements taken from the user. Sort and print this element.
92. Creating a menu program with this options(add, remove, display, exit).
93. Creating a class in JAVA(Circle class)
94. Static variable and method(With example)
95. Constructor
 - Default constructor
 - Default values
96. Introduction to classes and object
97. Visibility modifiers with example
98. Immutable class and immutable object
99. This keyword in java
 - This
 - Inner classes
100. UML class in java
101. UML rectangle

102. UML account class



WHAT IS SYSTEM.OUT. ?

- out is an object of the 'PrintStream' Class.
- out has the print() and println() methods.
→ use ':' to access print()/println() of out.
- out refers to the standard output device. (Screen)
- System is a Class (pascal case).
- out is inside System (field).
→ use ':' to access out of System.
→ System.out.println().

ESO ACADEMY

CALLING PRINT()

Displays its parameter on the console window

'In' is to break to a new line.

PACKAGE

A container for Classes



REMEMBER

- Every Java program contains at least one Class.
- Pascal case is used with classes.
- Camel case is used with methods.
- A method exists inside a class.
- `main()` is the starting point of execution of our program.

NAMING CONVENTIONS

How to write names in programming

Pascal case convention:

→ ThisIsAName

Camel case convention:

→ thisIsAName

Snake case convention:

→ this_is_a_name

ESO ACADEMY

ACCESS MODIFIERS

Specify how to access Classes & Methods

Discussed later in OOP

- Public
- Private
- Protected ↴
- Default

CALLING A METHOD

Is basically using the method

```
method_name( give parameters );
```

→ The code block of this method will be executed.

Note: the `main()` method is automatically called when we run our java program.

→ it is the first method that is called.

→ it is the starting point of execution of our program

ESO ACADEMY

METHOD STRUCTURE

Each method consists of 4 main parts

```
return_type method_name( parameters ) {  
    code block  
}
```

Note: every method is written inside a Class.

→ A class is a container of methods.



METHOD

Group of instructions to do a specific task

Examples:

- A method to add two numbers.
- A method to say Hi to the user.
- A method to get the user's name.

We have a special method called '**main**'.

'class' is a keyword (syntax).

CLASS

A blueprint to create OBJECTS

OBJECT

An instance of a CLASS

IDE

Integrated development environment

A program that allows us to:

- Write | source code.
- Compile | machine code.
- Debug | tools to find errors.
- Build | files that can be executed by JVM.
- Run | execute our program.

JDK

Java development kit

- Set of programs that enable us to develop our programs.
- Contains JRE (Java Runtime Environment) that is used to run our programs.
- JRE & JDK contain JVM (Java Virtual Machine).
- JVM executes our java programs on different machines.
→ java is independent .


EDITIONS OF JAVA

Java comes in three editions

- Java Standard Edition (SE): develop applications that run on desktop.
- Java Enterprise Edition (EE): develop server-side applications.
- Java Micro Edition (ME): develop applications for mobile devices.

We will use Java SE.
It is the foundation of all other editions.

ISO ACADEMY

API

Application programming interface

- Also known as a 'library'.
- Contains predefined Java code that we can use to develop Java programs.
 - Faster and easier development process | no need to write everything from scratch.

ABOUT JAVA

- Developed at Sun Microsystems which was purchased by Oracle in 2010.
- General-purpose & powerful programming language.
- Used for developing software that run on mobile, desktop, and servers.
- Machine independent.

WHY LEARN JAVA

- High-level , General-purpose , O-O programming language.
→ Easy & used to develop any kind of programs.
- Very Popular.
→ A huge online community for getting help.
- Can be used in Android development.
- C based language.
→ learn C/C++/C# easier.

SYLLABUS

Programming Fundamentals Object-Oriented Programming

- Variables
- Data types
- Operators
- Conditions
- Loops
- Methods
- Concrete Classes
- Objects
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Exceptions
- Abstract Classes
- Interfaces
- File I/O

1. Java packages, classes and methods.

```
package javaapplication;

public class Java {

    public static void main(String[] args) {

        System.out.println("Before sayHi");
        kibria();
    }

    public static void kibria() {
        System.out.println("sayHi");
    }
}

/*
We want to access our system class so we use the dot operator.
And inside the system class we have the out object. And inside
the out object we have a method called println(). If we want to
access the println() method we must use the dot operator.
*/
package javaapplication;

public class JavaCode {

    public static void main(String[] args) {

        Java.kibria();
    }
}

/*
Jodi Java class er kibria() method taky private kory dei tokon
JavaCode class hoty r access kora jaby na.
*/
```

2. Public, private and static in Java

PUBLIC ACCESS MODIFIER:

Specify how to access class, methods and fields.

The access level is everywhere-

Inside class, Outside class, Inside package, Outside package.

PRIVATE ACCESS MODIFIER:

The access level is only inside the class.

STATIC NON-ACCESS MODIFIER:

We can access field or method using the class name.

Ex. `System.out` here `out` is a static field of `System`.

3. The void return type in Java

THE VOID RETURN TYPE

main() has a void return type.

- void means nothing.
- main() does not return a value.

Example:

- printUserName();
 - We just want to print the name of the user.
 - We do not want to get any value from this method.
 - The return type is void.
- Note that every method has a very specific role.
 - getUserName();
 - printUserName();

RETURN TYPES

Examples:

- getUserName();
 - Get the name of the user and return the value.
 - The return type is a text (String).
- getUserAge();
 - Get the age of the user and return the value.
 - The return type is a number (int, double, ...).

RETURN TYPES

The type of data that a method returns/gives

- A method can return/give a value.
- Consider this mathematical function: $f(x) = x + 1$
 - x is a parameter to this function.
 - $x + 1$ is the return value of this function.
 - This function returns a number (return type).
- The same thing applies for methods/functions in programming.
 - A method can return a value.

4. Programming styles and errors

COMMON ERRORS

```
public class Main {  
    public static void Main(string[] args) {  
        System.out.println("Neso Academy Java Course!");  
    }  
}
```

Misspelling Names – Syntax Error

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Neso Academy Java Course!");  
    }  
}
```

COMMON ERRORS

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Neso Academy Java Course!");  
    }  
}
```

Missing Braces – Syntax Error

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("Neso Academy Java Course!");  
    }  
}
```

LOGICAL ERRORS

```
public class Main {  
    public static void main(String[] args) {  
        System.out.print("10 + 2 = ");  
        System.out.println(10 / 2);  
    }  
}
```

A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the following command-line session:

```
C:\Users\Ali Badran\Desktop>javac Main.java ← No problem  
C:\Users\Ali Badran\Desktop>java Main ← No problem  
10 + 2 = 5 ← Mathematically wrong → Logical error  
C:\Users\Ali Badran\Desktop>
```

The output "10 + 2 = 5" is highlighted with a red box and an arrow points from it to the text "Mathematically wrong → Logical error" located below the command prompt.

RUNTIME ERRORS

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println(1 / 0);  
    }  
}
```

A screenshot of a Windows Command Prompt window titled "Command Prompt". The window shows the following command-line session:

```
Microsoft Windows [Version 10.0.17134.885]  
(c) 2018 Microsoft Corporation. All rights reserved.  
C:\Users\Ali Badran>cd Desktop  
C:\Users\Ali Badran\Desktop>javac Main.java ← No problem  
C:\Users\Ali Badran\Desktop>java Main  
Exception in thread "main" java.lang.ArithmException: / by zero  
at Main.main(Main.java:3)  
C:\Users\Ali Badran\Desktop>
```

The output "Exception in thread" and "java.lang.ArithmException: / by zero" are highlighted with a red box and an arrow points from it to the text "The compiler does not detect runtime errors" located to the right of the command prompt.

→ The compiler does not detect runtime errors

SYNTAX ERRORS

```
public class Main {  
    public static main(String[] args) {  
        System.out.println("Neso Academy Java Course!");  
    }  
}
```

```
C:\ Command Prompt  
C:\Users\Ali Badran>cd Desktop  
C:\Users\Ali Badran\Desktop>javac Main.java  
Main.java:2: error: invalid method declaration; return type required  
    public static main(String[] args) {  
           ^  
Main.java:3: error: unclosed string literal  
        System.out.println("Neso Academy Java Course!");  
                           ^  
2 errors
```

→ The compiler reports syntax errors

BLOCK STYLES

```
public class Main {  
    public static void main(String[] args) {  
        System.out.println("hello");  
    }  
}
```

→ End-of-line style
Used in java API
Source code

```
public class Main  
{  
    public static void main(String[] args)  
    {  
        System.out.println("hello");  
    }  
}
```

→ Next-line style

5. Creating classes and methods

```
package javaapplication23;

public class Java {

    public static void sayName(){
        System.out.println("Golam kibria");
    }

    public static void sayAge(){
        System.out.println("21");
    }
}

package javaapplication23;

public class JavaCode {

    public static void main(String[] args) {

        Java.sayName();
        Java.sayAge();

    }
}
```

Output:
Golam kibria
21

6. Java basic in overview

METHODS

Divide our code into smaller pieces

- Less code in main().
- Our program will be easier to maintain and debug.
- Our program will be easier to understand.
- Makes the development process easier by breaking our program to smaller pieces and solving them one by one.

LOOPS

Used to run some code more than once

- While loop.
- Do while loop.
- For loop.
- For each loop.

CONDITIONS

Used to control what to execute in our program

- if a condition is true → do something.
- If it is not true → do another thing.

OPERATORS

Arithmetic, relational, and logical operators

- Arithmetic: to do some calculations. (+, -, ...)
- Relational: to do some comparisons. (<, >=, ...)
- Logical: to combine conditions together. (AND, OR, ...)

DATA TYPES

We have different types of data

Examples:

- Text.
- Numbers.
- Boolean. (true/false)
- User defined types. (Car, Person, ...)

These types are also for variables.

7. Introduction to variable in Java

Always initialize your variables!

- Less errors.
- Less code.
- More readable code.

BE CAREFUL

Do not re-define variables

Consider the following code:

```
public static void main(String[] args) {  
    String myJob; // Ok  
    String myJob; // Not Ok  
    String myJob = "Programmer"; // Not Ok  
}
```

- Each variable has a unique name.
- When assigning a value to a variable do not define it again.

ASSIGNMENT

Used to store/put a value inside a variable

– We can assign a value to a variable by using the **assignment operator** (**=**).

– **variableName = expression**

→ myJob = “Programmer”;

→ “Programmer” will be stored inside myJob.

– An **expression** is anything that produces/gives a value.

Examples: (1 + 3), (4 * 2).

DECLARATION

– To declare multiple variables of the same type:

TYPE NAME1, NAME2;

– String myName, myJob;

→ myName & myJob are two variables that can store a String.

– A variable must be declared before it can be used.

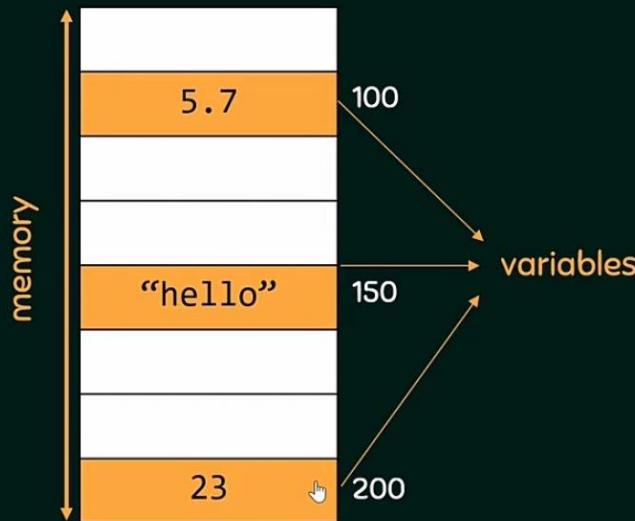
DECLARATION

Allocating space inside our memory

- To allocate space in our memory we declare a variable
→ `TYPE NAME;`
- The type of the variable should be compatible with the data inside it
→ to store a String inside a variable, the variable's type should be String
- `String myName;`
→ Declared a variable called `myName` and can store a String

MEMORY

A variable is like a box in the computer's memory



VARIABLES

Used to store values in our computer's memory

- Our computer has a memory.
- To store values in this memory we need to reserve some space.
 - Use a variable.
- Each variable has a specific type.
- It is called a variable because the value inside it can change.

8. Constants in Java

BENEFITS OF USING CONSTANTS

1. The value will not be changed by accident.
2. You don't have to type the same value if it is used multiple times.
3. A descriptive name for a constant makes the program easy to read and understand.

CONSTANTS

A variable whose value can not be changed

- To define a constant we use the `final` keyword
- Constants can be used like any other variable
- Constants names are written in upper case and using the snake case convention
- You will get a syntax error if you try to change the value of a constant.



```
//USNG OF FINAL KEYWORD
package javaapplication23;

public class Java {

    public static void main(String[] args) {

        final String companyName = "Neso Academy";
        System.out.println(companyName);
    }
}
```

9. Identifier in Java

TIPS

- Use clear and descriptive names
→ `numberOfStudents, userName`
- Avoid using abbreviations
→ `numStuds, uName`
- Do not use the \$ character when naming identifiers

CASE SENSITIVITY IN JAVA

- `area, Area, ARea, and AREA` are four different identifiers
- `X` is different than `x`
- `Main` is different than `main`

EXAMPLES

Legal

1. `$2`
2. `Person`
3. `area`
4. `radius`
5. `Point`
6. `hello`

Illegal

1. `2A`
2. `a+b`
3. `user name`
4. `int`
5. `main`
6. `System`

NAMING RULES

Every identifier must obey some rules

- Can contain letters, digits, underscores (_), and dollar signs (\$)
→ myName1, my_Name2, my\$
- Must start with a letter, or an underscore, or a dollar sign.
It cannot start with a digit and it can not contain spaces.
→ hello, i, _myName, \$1, \$myAge → OK
→ 1hello, @hi, *name, my age → NOT OK
- Cannot be a reserved word
→ main, class, String, ...

IDENTIFIERS

Identifiers are the names that identify the elements in a program

- Names of classes
- Names of methods
- Names of variables

myName, myJob, sayName, Main, ...

CHARACTERS

All characters on the keyboard – AND MORE

'a', '5', '-', '*', '?', '\$', ';' , ',', '

Characters are surrounded with single quotes

REAL NUMBERS

Numbers with a decimal part

1.5, 2.25, -4.7, -9.9, 1.0, 0.0

An integer can be a real number

1.0, 2.0, 100.0, -4.0, -9.0, 0.0

INITIALIZING INT VARIABLES

```
int i1 = 5;
int i2 = 10;
int i3 = -9;
int i4 = 2147483647;
int i5 = -2147483648;

int i6 = 2147483648; // ERROR, > MAX
int i7 = -2147483649; // ERROR, < MIN
```

RANGE OF AN INT VARIABLE

Interval of values that can be stored in an int variable

- Declare an int variable:
→ `int number;`
- All numbers in the interval `[-2147483648, 2147483647]` can be stored in the variable `number`
- All numbers greater than `2147483647` or less than `-2147483648` can not be stored in `number`

→ Error ↴

INTEGERS & THE INT DATA TYPE

An integer is a number that does not have a decimal part

- Examples:

4

1

100

-23

- In java, `int` is a data type used to work with integers

12. The byte, short & long data type in Java

TYPE CONVERSION

byte → short → int → long

A long can store an int, a short, and a byte

An int can store a short and a byte, but it can not store a long

A short can store a byte, but it can not store an int or a long

A byte can only store a byte

BYTE

A type used with integers

All numbers in the interval [-128, 127] can be stored in a byte variable



```
byte b1 = -128; // OK
byte b2 = 127; // OK
byte b3 = 100; // OK

byte b4 = -129; // ERROR
byte b5 = 128; // ERROR
```

SHORT

A type used with integers

All numbers in the interval [-32768, 32767] can be stored in a short variable

```
short s1 = 5000; // OK
short s2 = 32767; // OK

short s3 = 32768; // ERROR
```

LONG

A type used with integers

- All numbers in the interval
[-9223372036854775808L, 9223372036854775807L]
can be stored in a long variable

- A letter 'l' or 'L' should be added to tell the compiler that a number is a long and not an int

```
long l1 = -9223372036854775808L; // Ok
long l2 = -1839; // Ok
long l3 = -9223372036854775808; // ERROR
```

EXAMPLES

```
byte b1 = 5;  
short s1 = 10;  
int i1 = 20;  
long l1 = 100;
```

```
l1 = b1 + s1 + i1; // OK, long = int  
i1 = s1 + b1; // OK, int = short  
s1 = b1; // OK, short = byte  
  
i1 = l1; // NOT OK, int = long  
s1 = i1; // NOT OK, short = int  
b1 = i1; // NOT OK, byte = int
```

13. Integer data type in Java

```
package javaapplication23;

public class Java {

    public static void main(String[] args) {

        byte b = 5;
        short s = 20;
        int i = 100;
        long g = 999L;

        System.out.println(b);
        System.out.println(s);
        System.out.println(i);
        System.out.println(g);

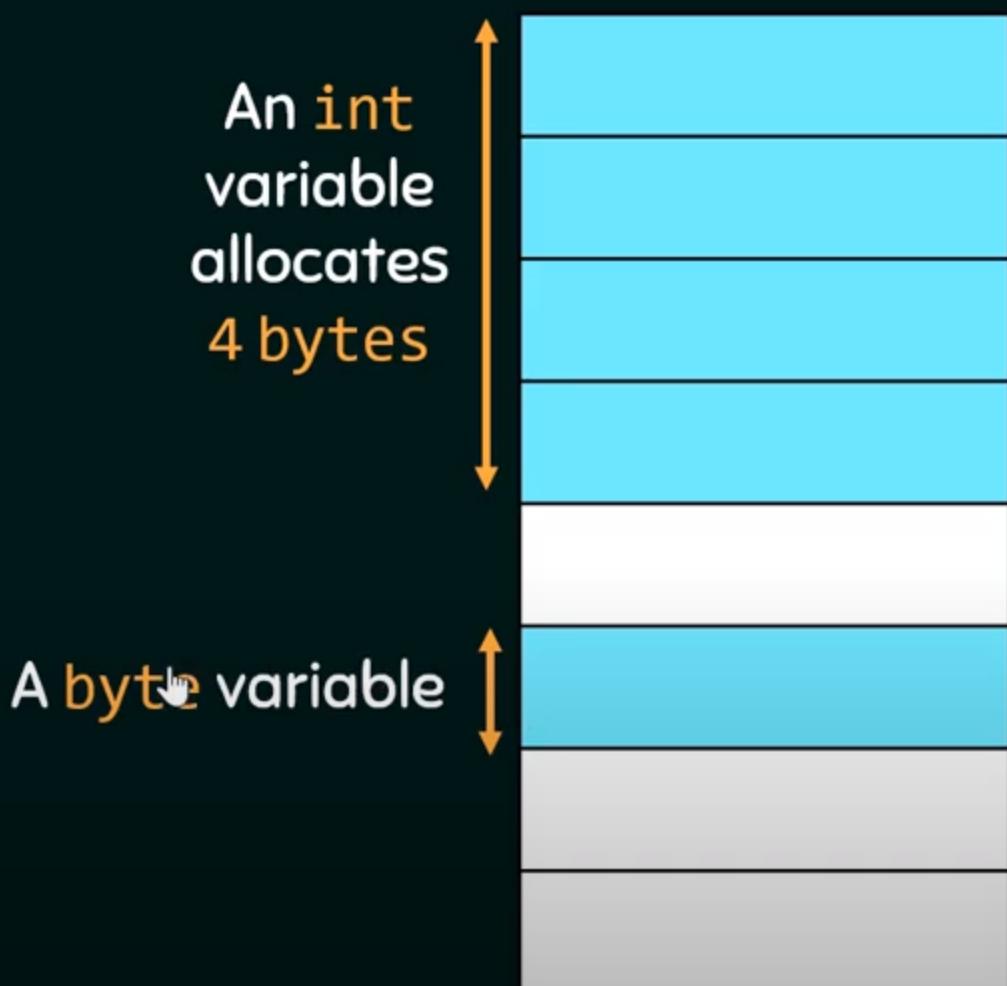
    }
}
```

14. Bytes and Values in Java

BITS & MEMORY

- A **bit** is the smallest unit to measure memory
- A **byte** is equal to **8bits**
- Our **memory** is divided into **bytes**
- Each variable reserves a certain number of bytes in memory
 - More bytes means larger values
 - A **long** reserves more bytes than an **int**

BITS & MEMORY



SIZE OF DIFFERENT INTEGER DATA TYPES

$$1B = 8b$$

- byte: 1B
→ 8b
- Short: 2B
→ 16b
- int: 4B
→ 32b
- long: 8B
→ 64b

CALCULATING THE RANGE

$$[-2^{\text{nbOfBits}-1}, (2^{\text{nbOfBits}-1}) - 1]$$



A short takes 2B in memory

→ 16b

$$\rightarrow \text{Range} = [-2^{16-1}, 2^{16-1} - 1]$$

$$= [-2^{15}, 2^{15} - 1]$$

$$= [-32,768, 32,767]$$

WHICH DATA TYPE TO USE?

Do not waste memory

– Working with small numbers?

→ do not use a **long**, use what you need (**byte**, **short**, **int**)

– Need larger numbers?

→ use a **long**.

– Do not panic, just use what you need ☺.



```
public class HelloWorld{  
  
    public static void main(String []args){  
  
        System.out.println(Integer.MAX_VALUE);  
        System.out.println(Short.MIN_VALUE);  
    }  
}
```

Output:

```
2147483647  
-32768
```

15. The double and float Data Type in Java

THE DOUBLE DATA TYPE

A type used with floating-point numbers

- Declare a double variable:

→ `double number;`

- Range:

$[-1.7976931348623157 \times 10^{308}, 1.7976931348623157 \times 10^{308}]$

- Smallest positive non-zero number:

4.9×10^{-324}

~~- A double takes 8B in memory~~

THE FLOAT DATA TYPE

A type used with floating-point numbers

- Declare a float variable:

→ `float number;`

- Range:

$[-3.4028235E38F, 3.4028235E38F]$

- Smallest positive non-zero number:

1.4×10^{-45}

- A float takes 4B in memory

```
package javaapplication23;

public class Java {

    public static void main(String[] args) {

        float f = 12.3F;
        //we can also store a integer value in float.

        double d = 12.4;
        //we can also write 12.4F without having any problem.

        //The float will converted double and then store the value in d

        System.out.println(f);
        System.out.println(d);
    }
}
```

16. The char Data Type in Java

UNICODE ENCODING SCHEME

Computers see characters as binary numbers

- Each character is encoded then stored in memory
 - Encoding is mapping each character to its binary representation with the help of an encoding scheme
- Java uses the Unicode encoding scheme
 - Character to Unicode to Binary
 - Binary to Unicode to Character

THE CHAR DATA TYPE

A type used with characters

- Declare a char variable:
 - `char c;`
- Range: `[‘\u0000’, ‘\uffff’]` OR `[0, 65,535]`
 - Each number represents a character
- Characters are put inside single quotes
 - ‘A’, ‘*’, ‘;’, ‘4’, ...
- A char takes 2B in memory

EXAMPLES

Unicode range: [‘\u0000’, ‘\uffff’] OR [0, 65535]

```
'A' → '\u0041' , 65
'a' → '\u0061' , 97
';' → '\u003b' , 59
```

```
public static void main(String[] args) {
    char c1 = 'A'; // 'A'
    char c2 = 65; // 'A'
    char c3 = '\u0041'; // 'A'

    System.out.println(c1); // A
    System.out.println(c2); // A
    System.out.println(c3); // A
}
```

```
package javaapplication23;

public class Java {

    public static void main(String[] args) {

        char c1 = 'A';
        char c2 = 65;
        char c3 = '\u0041'; //unicode
        System.out.println(c1);
        System.out.println(c2);
        System.out.println(c3);

        int x = 'A';
        int y = '\u0041';
        System.out.println(x);
        System.out.println(y);
    }
}
```

Output:

A
A
A
65
65

17. Boolean data type in Java

THE BOOLEAN DATA TYPE

A type used with Boolean/logic values

- Declare a boolean variable:
→ `boolean c;`
- Possible values: `true` OR `false`
- The size of a boolean variable is not precisely defined, it depends on JVM

```
package javaapplication23;

public class Java {

    public static void main(String[] args) {

        boolean b1 = true;
        boolean b2 = false;
        boolean b3 = 2 > 1;
        boolean b4 = 2 < 1;

        System.out.println(b1);
        System.out.println(b2);
        System.out.println(b3);
        System.out.println(b4);

        boolean isAlive = true;
        if(isAlive)
            System.out.println("Alive");
        else
            System.out.println("Not Alive");
    }
}

//boolean helps us to create condition in our program.
//so we will be execute certain condition of code depending on the
condition
//so we have more control what our program does
```

Output:

true

false

true

false

Alive

18. String data types and calling string method

text.toUpperCase() → Create a new String “THIS IS SOME TEXT”
→ The method will return the new String
→ The new String can be stored inside a variable
→ The new String is just like any other String

```
package JavaCode;

public class Javaa {

    public static void main(String[] args) {

        String s = "Golam kibria";
        String s1 = s.toUpperCase();
        System.out.println(s1);
        System.out.println(s.toLowerCase());
        System.out.println(s.length());
        System.out.println(s.isEmpty());
        //System.out.println(s.isBlank()); //output: False
        System.out.println(s.charAt(3));
        System.out.println(s.indexOf('m'));
        System.out.println(s.charAt(s.length() - 1));
        System.out.println(s.lastIndexOf('a'));
        String s2 = s.concat(" Ezaz");
        System.out.println(s2);
    }
}
```

Output:

GOLAM KIBRIA

golam kibria

12

false

a

4

a

11

Golam kibria Ezaz

19. Concatenating string in java

CONCATENATION

Adding Strings to each other

```
public static void main(String[] args) {  
    String part1 = "Neso";  
    String part2 = "Academy";  
  
    System.out.println(part1 + part2); // NesoAcademy  
    System.out.println(part1 + " " + part2); // Neso Academy  
}
```

When `+` is used with Strings it acts as a concatenation operator

When `+` operator is used only with numbers it acts as an addition operator

```
package JavaCode;  
public class Javaa {  
  
    public static void main(String[] args) {  
  
        String s1 = "Golam";  
        String s2 = " Kibria";  
        System.out.println(s1 + s2);  
        System.out.println("My favourite number is: " + 5);  
        //automatically convert 5 to "5"  
        System.out.println("My favourite number is: " + 5 + 3);  
        System.out.println("Golam".concat(" Kibria"));  
        System.out.println("Golam".concat(" Kibria").concat(" Ezaz"));  
        System.out.println("Golam".concat(" Kibria").isEmpty());  
    }  
}
```

Output:

Golam Kibria

My favourite number is: 5

My favourite number is: 53

Golam Kibria

Golam Kibria Ezaz

False

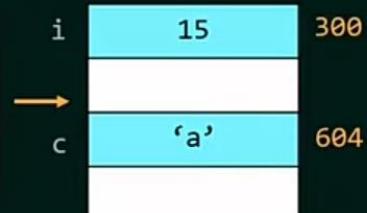
20. Primitive and reference type

PRIMITIVE TYPES

Types that hold simple values

byte, short, int, long, float, double, and char are primitive types

```
public static void main(String[] args) {  
    int i = 15;  
    char c = 'a';  
}
```



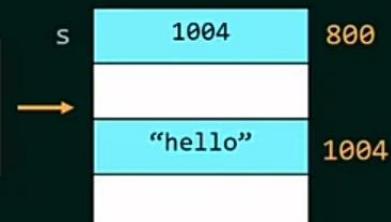
The variable contains the value

REFERENCE TYPES

Types that hold complex values (Objects)

String is a reference types

```
public static void main(String[] args) {  
    String s = "hello";  
}
```



The variable contains the address of the value.

The variable references the value.

s → "hello"

PRIMITIVE TYPES VS REFERENCE TYPES

String s2 = s1;

Create a new variable s2 and assign the value of s1 to it

i1	5	100
s2	1008	500
i2	5	200
s1	1008	300
	“hello”	1008

s1 and s2 are different, but they reference the same value

i1 5
i2 5

s1 → “hello”
s2 → “hello”

21. Instantiating a string object

INSTANTIATING AN OBJECT

Creating an object of a class

```
ClassName objectName = new ClassName(Parameters);
```

A special method called Constructor

```
Point point = new Point(1, 2);
```

```
Cat tom = new Cat("Tom");
```

= VS new

```
public static void main(String[] args) {  
    String name1 = "Neso Academy";  
    String name2 = "Neso Academy";  
    String name3 = new String("Neso Academy");  
}
```

name1 → "Neso Academy"
name2 → "Neso Academy"

name3 → "Neso Academy"

```
package JavaCode;

public class Javaa {

    public static void main(String[] args) {

        String s1 = "Golam kibria";
        String s2 = new String("Golam kibria");
        System.out.println(s1);
        System.out.println(s2);
    }
}
```

Output:

```
Golam kibria
Golam Kibria
```

New is a special keyword in java used to create new object.

And we can use the new keyword in order to create new string cuz string is a class.

22. String are immutable in java

IMMUTABLE OBJECTS

Objects whose contents can not be changed

- A constant is a variable whose value can not change
- An immutable object is an object whose content can not be changed
- Immutable objects are created from immutable classes
- The String Class in Java is Immutable
 - The content of String objects in Java can not be changed

ADEMI

USING NEW WITH STRINGS

Strings are immutable → no need to use new

- We would want to use new because it allows us to create a separate object that references a separate value.
- When working with Strings this is not needed because the original String will not be modified because Strings are immutable



```
package JavaCode;

public class Javaa {

    public static void main(String[] args) {

        String s1 = "Old value";
        s1 = "New value";
        System.out.println(s1);
        //The ole value is now garbage. So it will be remove from the
        memory by the java garbage collector.

        String str1 = "Dhaka";
        String str2 = str1;
        str2 = "Bangladesh";
        System.out.println(str1);
        System.out.println(str2);
    }
}

Output:
New value
Dhaka
Bangladesh
```

23. Scanner class in java

INPUT METHODS

Methods used to read specific types of data from the keyboard

```
input.next(); // Read a String  
input.nextInt(); // Read an integer  
input.nextDouble(); // Read a double  
  
.nextByte(), .nextShort(), .nextLong(), .nextFloat(), .nextBoolean()
```

When one of these methods is called, the program will pause execution and wait for the user to enter a value, the entered value will be returned by these methods.

Note: we don't have .nextChar()

```
package JavaCode;

import java.util.Scanner;

public class Javaa {

    public static void main(String[] args) {

        //      String userName = " Kibria";
        //      System.out.println("Golam"+userName);
        Scanner input = new Scanner(System.in);
        //      System.out.println(input.next());
        //      program will pause until i type a string

        System.out.println("Enter your name: ");
        System.out.println("Your name is: " + input.next());
        System.out.println("Your full name is: " + input.nextLine());
    }
}

//System.in mutotw use kora hoy keyboard hoty data read korar jonno.
//input.next() dara bujay ami dot operator er sahajjy next() method e access
korbo
//next() method read untill the first space
```

Output:

```
Enter your name:
golam kibria
Your name is: golam
Your name is: kibria
```

24. Reading input from the keyboard

```
package JavaCode;

import java.util.Scanner;

public class Javaa {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        System.out.println(input.nextDouble());

        float f = input.nextFloat();
        System.out.println(f);

        System.out.println(input.nextBoolean());
    }
}
```

Output:

```
1
1.0
2
2.0
true
true
```

25. Printing a number

```
package JavaCode;
```



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

```
public class Javaa {
```

```
    public static void main(String[] args) {  
  
        Scanner input = new Scanner(System.in);  
        System.out.print("Enter your favourite integer: ");  
  
        System.out.println(input.nextInt() +" is my favourite integer");  
    }  
}
```

Output:

```
Enter your favourite integer: 5
```

```
5 is my favourite integer
```

26. Printing name and age

```
package JavaCode;

import java.util.Scanner;

public class Javaa {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        System.out.println("Enter your name and age: ");

        System.out.println(input.nextLine() + " You are! " +
        input.nextInt() + " years old");
    }
}
```

Output:

```
Enter your name and age:
```

```
Golam kibria
```

```
20
```

```
Golam kibria You are!20 years old
```

27. Literals in Java

```
package JavaCode;

public class Javaa {

    public static void main(String[] args) {

        System.out.println(12.4);
        System.out.println(12.4F);
    }
}
```

Output:

12.4

12.5

28. Assignment operator

ASSIGNMENT OPERATOR

An operator to assign a value/expression to a variable

- Using '='.

- lvalue = rvalue;
 - the lvalue should be a variable
 - the rvalue is an expression that evaluates to a value
 - rvalue is evaluated and then the result is stored in lvalue
 - lvalue can be used in rvalue

Example:

```
int x = 3 + 1; → 3 + 1 is evaluated and then the result is stored in x
```

- We can assign multiple variables in the same statement:

```
int y;  
int x = y = 4 + 1;
```

This is done from right to left

- Step 1: $y = 4 + 1$
- Step 2: $x = y$
- x and y are equal to 5

```
int i1 = 1; → evaluates to 1  
double d = 5.0; → evaluates to 5.0  
int i2 = i1 = 4 + 1; → evaluates to 5
```

```
double d;  
int i1, i2;  
  
System.out.println(d = 5.0); // 5.0  
System.out.println(i1 = 3); // 3  
System.out.println(i2 = i1 = 5); // 5  
  
int i1 = 2;  
i1 *= 5 + 1;
```

→ Equivalent to:

```
i1 = i1 * (5 + 1) // 12
```



→ Not to:

```
i1 = i1 * 5 + 1 // 11
```

29. Arithmetic operator

OPERATOR PRECEDENCE

All operations are done from **left to right** in the following order:

1. The operations between the parenthesis
2. The multiplication and the division
3. The addition and the subtraction

- $2 * (1 + 5) - 12 / (4 + 2)$
- $2 * 6 - 12 / 6$
- $12 - 2$
- $\downarrow 10$

30. Increment and Decrement operator

THE INCREMENT OPERATOR

Used to increase the value of a variable by 1

- Increment operator: `++`
- Can be placed before the variable (pre-increment).
→ `++i`
- Can be placed after the variable (post-increment).
→ `i++`

```
int i = 4;  
i++;  
System.out.println( i ); // 5
```

```
int i = 4;  
++i;  
System.out.println( i ); // 5
```

`i++` VS `++i`

```
int i = 1;  
int j = ++i;  
System.out.println("i is " + i + ", j is " + j ); // i is 2, j is 2
```

BUT

```
int i = 1;  
int j = i++;  
System.out.println("i is " + i + ", j is " + j ); // i is 2, j is 1
```

`++i` increments the
value of `i` by 1 and uses
the **new** value in the
statement

`i++` increments the
value of `i` by 1 and uses
the **original** value in the
statement

```
int i = 1;  
int j = i++ + 4;  
System.out.println("i is " + i + ", j is " + j ); // i is 2, j is 5
```

```
int i = 1;  
int j = ++i + 4;  
System.out.println("i is " + i + ", j is " + j ); // i is 2, j is 6
```

THE DECREMENT OPERATOR

Used to decrease the value of a variable by 1

- Increment operator: `--`
- The same rules apply.

```
int i = 4;  
System.out.println( i-- ); // 4
```

```
int i = 4;  
System.out.println( --i ); // 3
```

```
package JavaCode;

public class Javaa {

    public static void main(String[] args) {

        //      int i = 4;
        //      i++;
        //      System.out.println(i); //5 cuz i++ means i = i+1

        //      int i = 4;
        //      ++i;
        //      System.out.println(i); //5 cuz i er man saty saty bery jaby pre
        increment a

        //      int i = 4;
        //      int j = i++;
        //      System.out.println("i = "+i);//5
        //      System.out.println("j = "+j);//4
        //      i++ increment the value of i by 1 and use the original value of i

        //      int i = 4;
        //      int j = ++i;
        //      System.out.println("i = "+i);//5
        //      System.out.println("j = "+j);//5
        //      ++i increment the new value of i by 1 and use the new value in the
statement.

        //      int i = 4;
        //      int j = i++ + 1;
        //      System.out.println("i = " + i);//5
        //      System.out.println("j = " + j);//5

        //      int i = 4;
        //      int j = ++i + 1;
        //      System.out.println("i = " + i);//5
        //      System.out.println("j = " + j);//6

        //      int i = 2;
        //      System.out.println(i--);//2
        //      post decrement: original value of i

        //      int i = 2;
        //      System.out.println(--i);//1
    }
}
```

31. Casting

CASTING

Converting a data type to another type

- **Implicit casting:** Happens automatically when converting from a narrower range data type to a wider range data type

→ converting an **int** to a **double/float/long**

→ converting a **float** to a **double**

- **Explicit casting:** Does not happen automatically. Should be done by the programmer when converting from a wider to a narrower data type

→ converting a **double/float/long** to an **int**

→ converting a **double** to a **float**

IMPLICIT CASTING

```
double d1 = 4; // int → double
double d2 = 5.7f; // float → double
long l1 = 100; // int → long
```

Implicit casting happens because:

- The range of a **double** is wider than an **int**
- The range of a **double** is wider than a **float**
- The range of a **long** is wider than an **int**

EXPLICIT CASTING

```
int i1 = 4.5; // ERROR  
int i2 = 8L; // ERROR  
float f1 = 4.5; // ERROR
```

Implicit casting can not happen because:

- The range of an `int` is narrower than a `double`
- The range of an `int` is narrower than a `long`
- The range of a `float` is narrower than a `double`

→ Explicit casting is used

EXPLICIT CASTING
(new data type) expression

```
int i1 = (int) 4.5;  
int i2 = (int) 8L;  
float f1 = (float) 4.5;
```

The programmer tells Java to do the casting:

- `(int) 4.5 → 4` (data loss)
- `(int) 8L → 8`
- `(float) 4.5 → 4.5F`

NOTE

```
int i1 = 4.5;
```

This statement produces a syntax error.

We are trying to store a **double** inside a variable of type **int**.

↓

Wider Range

↓

Narrower Range

```
int i1 = (int) 4.5;
```

NOTE

```
double d1 = 4.5 + 3;
```

Every arithmetic operator should be applied between values of the same type.

→ 3 will be automatically casted to 3.0

→ 4.5 + 3.0 will be calculated and stored in d1

Also:

```
double d1 = 4.5 + (double)3;
```

```
double d = 4.2;  
int i = (int) d;  
// i = 4, d = 4.2
```

The variable being casted does not change.

32. The division operator in Java

DIVIDING INTEGERS

Dividing an **int** by an **int** gives an **int**

```
System.out.println(1 / 2); // 0
```

Note:

```
double d = 1 / 2; // double = int → implicit casting
```

```
int i = 1 / 2; // int = int
```

DIVIDING DOUBLES

Dividing a **double** by a **double** gives a **double**

```
System.out.println(1.0 / 2.0); // 0.5
```

Note:

```
double d = 1.0 / 2.0; // double = double
```

```
int i = 1.0 / 2.0; // int = double → ERROR
```

DIVIDING FLOATS

Dividing a float by a float gives a float

```
System.out.println(1.0f / 2.0f); // 0.5f
```

Note:

```
double d = 1.0f / 2.0f; // double = float →  
implicit casting
```

```
int i = 1.0f / 2.0f; // int = float → ERROR
```

DIVIDING INTEGERS AND DOUBLES

The integer will be casted to a double automatically

```
System.out.println(1 / 2.0); // 0.5
```

```
System.out.println(1.0 / 2); // 0.5
```

Note:

```
double d = 1 / 2.0; // double = double
```

```
int i = 1 / 2.0; // int = double → ERROR
```

DIVIDING FLOATS AND DOUBLES

The **float** will be casted to a **double** automatically

```
System.out.println(1.0f / 2.0); // 0.5
```

```
System.out.println(1.0 / 2.0f); // 0.5
```

Note:

```
double d = 1.0f / 2.0; // double = double
```

```
float f = 1.0f / 2.0; // float = double → ERROR
```

DIVIDING FLOATS AND INTEGERS

The **integer** will be casted to a **float** automatically

```
System.out.println(1.0f / 2); // 0.5f
```

```
System.out.println(1 / 2.0f); // 0.5f
```

Note:

```
double d = 1.0f / 2; // double = float →  
implicit casting
```

```
int i = 1.0f / 2; // int = float → ERROR
```

33. The divisor operator example

```
int i1 = 1 / 2; // OK, int = int
int i2 = 1.0 / 2; // ERROR, int = double
int i3 = (int) (1.0 / 2); // OK, int = int
int i4 = (int) (1.0f / 2); // OK, int = int
int i5 = (int) 1.0f / 2; // OK, int = int
int i6 = 1.0f / (int) 2; // ERROR, int = float
double i7 = 1.0 / 2; // OK, double = double
double i8 = 1 / 2; // OK, double = int
double i9 = (double) 1 / 2; // OK, double = double
float i10 = 1.0f / 2.0f; // OK, float = float
float i11 = 1 / 2; // OK, float = int

int i1 = 3;
int i2 = 2;
double d1 = 2;

System.out.println(i1 / i2); // 1
System.out.println(i1 / d1); // 1.5
System.out.println((double) i1 / i2); // 1.5
System.out.println(i1 / (double) i2); // 1.5
System.out.println((double) i1 / (double) i2); // 1.5
System.out.println((double) (i1 / i2)); // 1.0
```

34. The relational operator in java

RELATIONAL OPERATORS

Operators used to make comparisons

- Equality operator: `==`
- Inequality operator: `!=`
- Greater than operator: `>`
- Greater than or equal operator: `>=`
- Less than operator: `<`
- Less than or equal operator: `<=`

All these operators produce a Boolean value when used

```
boolean b1 = (1 == 1); // true
boolean b2 = (2 == 3); // false
boolean b3 = (2 + 3) == (6 - 1); // true
```

The parenthesis are optional.
Just to make the code more clear and readable.

```
boolean b1 = (1 != 1); // false
boolean b2 = (2 != 3); // true
boolean b3 = (2 + 3) != (6 - 2); // true
```

```
boolean b1 = (1 > 1); // false
boolean b2 = (2 > 3); // false
boolean b3 = (2 + 3) > (6 - 2); // true
```

```
boolean b1 = (1 >= 1); // true
boolean b2 = (2 >= 3); // false
boolean b3 = (2 + 3) >= (6 - 2); // true
```

```
boolean b1 = (1 < 1); // false
boolean b2 = (2 < 3); // true
boolean b3 = (2 + 3) < (6 - 2); // false
```

```
boolean b1 = (1 <= 1); // true
boolean b2 = (2 <= 3); // true
boolean b3 = (2 + 3) <= (6 - 2); // false
```

35. Logical operator in java

LOGICAL OPERATORS

Operators used to construct complex conditions

- Logical AND operator: &&
- Logical OR operator: ||
- Logical NOT operator: !

These operators are used between Boolean values, and the result is also a Boolean value.



- **AND**: condition_1 && condition_2 && condition_n
→ true only if all conditions are true
- **OR**: condition_1 || condition_2 || condition_n
→ true if at least one condition is true
- **NOT**: negates a Boolean value
→ !true = false
→ !false = true



```
boolean b1 = true && true; // true
boolean b2 = b1 && false && true; // false
boolean b3 = b2 || true; // true
boolean b4 = !b2; // true
boolean b5 = !(b4 && b2); // true
```

```
if isRaining || isCold → wear a jacket
if n ↴>= 1 && n <= 10 → n is between 1 and 10
```

36. Conditional operator in Java

CONDITIONAL OPERATOR

An operator that evaluates an expression based on a condition

boolean-expression ? expression1 : expression2

Evaluates to:

expression1 if boolean-expression is true
expression2 if boolean-expression is false

```
int a = 10;  
int b = 15;
```

```
int max = a > b ? a : b;  
System.out.println(max); // 15
```

```
String name = "Neso Academy";
```

```
System.out.println( name.isEmpty() ? "The name is not  
valid" : name ); // Neso Academy
```

```
String name = "";
```

```
System.out.println( name.isEmpty() ? "The name is not  
valid" : name ); // The name is not valid
```

```
String name = "Neso Academy";  
  
System.out.println( !name.isEmpty() ? name : "The name is  
not valid" ); // Neso Academy
```

37. Difference between switch and if else

SWITCH VS IF ELSE

```
if (n == 10)
    System.out.println("= 10");
else if (n == 90)
    System.out.println("= 90");
else if (n == -3)
    System.out.println("= -3");
else
    System.out.println("none of the above");
```

Only one block will be executed

```
switch (n) {
    case 10:
        System.out.println("= 10");
    case 90:
        System.out.println("= 90");
    case -3:
        System.out.println("= -3");
    default:
        System.out.println("none of the above");
}
```

More than one case may be executed

If the break keyword is executed we will break out from the switch statement in other words we will stop executing the switch statement.

```
public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    int n = s.nextInt();

    switch (n % 2) {
        case 0:
            System.out.println("even");
            break;
        default:
            System.out.println("odd");
    }
}
```

```
public static void main(String[] args) {  
    Scanner s = new Scanner(System.in);  
    int n = s.nextInt();  
  
    switch (n % 2) {  
        case 0:  
            System.out.println("even");  
            break;  
        case 1:  
            System.out.println("odd");  
    }  
}
```

39. Making simple calculator using if else

```
public static void main(String[] args) {  
    System.out.print("enter num1 op num2 (5 * 7): ");  
  
    Scanner s = new Scanner(System.in);  
    double d1 = s.nextDouble();  
    char op = s.next().charAt(0);  
    double d2 = s.nextDouble();  
  
    if( op == '+' )  
        System.out.println((int)(d1 + d2));  
    else if( op == '-' )  
        System.out.println((int)(d1 - d2));  
    else if( op == '/' )  
        System.out.println(d1 / d2);  
    else if( op == '*' )  
        System.out.println((int)(d1 * d2));  
  
    else  
        System.out.println("Invalid operator");
```

40. Lucky number

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

System.out.println("enter a four-digit number: ");
int n = s.nextInt();

if(!(n > 999 && n < 10000)) {
    System.out.println(n + " is not a four-digit number");
}

else {
    // ABCD
    int fourthDigit = n % 10; // D
    int thirdDigit = (n / 10) % 10; // ABC % 10 = C
    int secondDigit = (n / 100) % 10; // AB % 10 = B
    int firstDigit = (n / 1000) % 10; // A % 10 = A

    if(firstDigit + secondDigit == thirdDigit + fourthDigit)
        System.out.println("lucky");
    else
        System.out.println("not lucky");
}
```

41. While loop

Each execution is called an iteration

Write a program that reads an integer between 1 and 10 from the user

```
public class Main {  
    public static void main(String[] args) {  
        Scanner s = new Scanner(System.in);  
  
        int n = s.nextInt();  
        while(n < 1 || n > 10) {  
            System.out.print(n + " is not between 1 and 10. Try again: ");  
            n = s.nextInt();  
        }  
  
        System.out.println(n + " is between 1 and 10.");  
    }  
}
```

42. do while loop

```
int i = 1;
do {
    System.out.println("hello " + i);
    i++;
} while(i <= 3);
```

WHILE VS DO WHILE

while: check condition then execute
do while: execute then check condition

```
int i = 100; // i <= 5 is false
while(i <= 5) {
    System.out.println("hello " + i);
    i++;
}
```

→ 0 Iterations

```
int i = 100; // i <= 5 is false
do {
    System.out.println("hello " + i);
    i++;
} while(i <= 5);
```

→ 1 Iteration

SCHULE

3:24 / 6:53

```
package com.NesoAcademy;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        int n;
        do {
            System.out.print("enter a number between 1 and 10: ");
            n = s.nextInt();
        } while (n < 1 || n > 10); I

        System.out.println(n + " is between 1 and 10.");}
    }
}
```

43. structure of for loop

The variable can be initialized outside:

```
int i = 1;  
for( ; i <= 5; i++) {  
    System.out.println("Hello " + i);  
}
```

```
System.out.println("Hello " + i); // Hello 6  
                                ↓
```

i can be accessed outside the loop.

INFINITE FOR LOOP

```
for( ; ; )  
    System.out.println("Hello");
```

```
for( ; true; )  
    System.out.println("Hello");
```

FOR LOOP \leftrightarrow WHILE LOOP

```
int i = 1;
for ( ; i <= 5; ) {
    System.out.println("Hello");
    i++;
}
```

```
int i = 1;
while ( i <= 5 ) {
    System.out.println("Hello");
    i++;
}
```

44. for loop printing star and multiplication table

EXAMPLE

```
for(int i = 1; i <= 5; i++) {  
    for (int j = 1; j <= i; j++)  
        System.out.print("*");  
  
    System.out.println();  
}
```

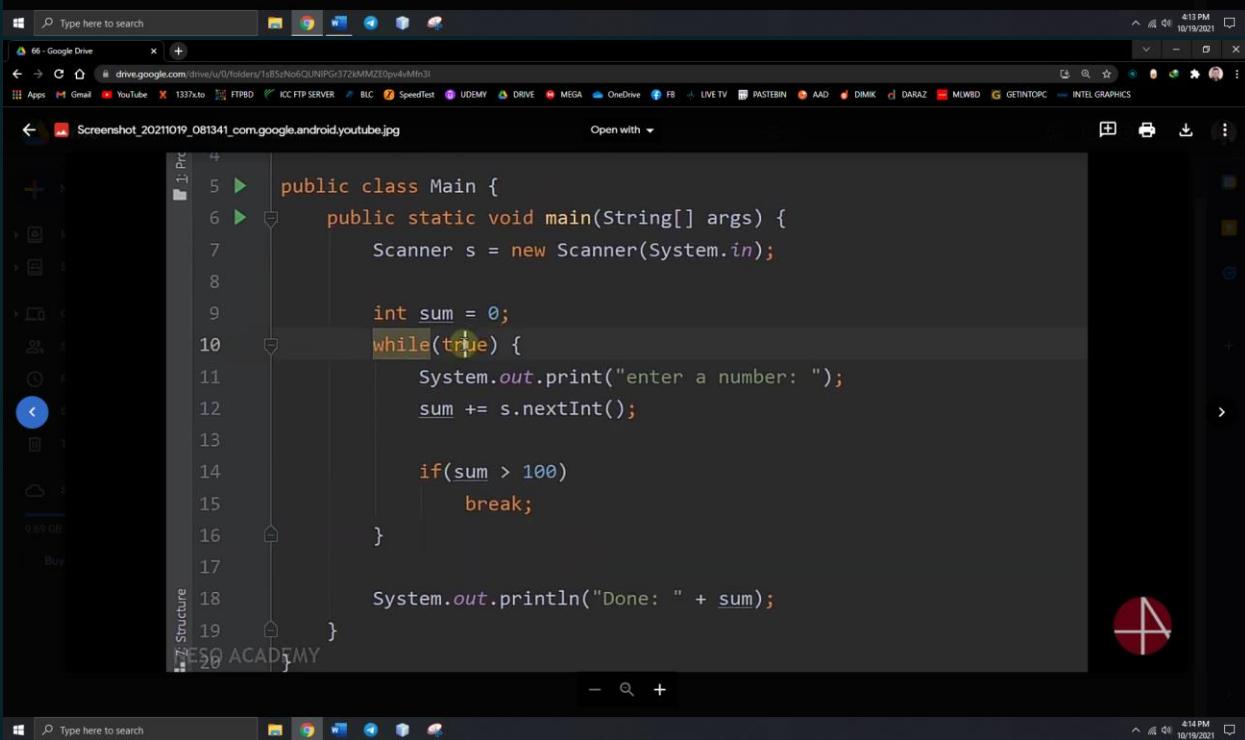
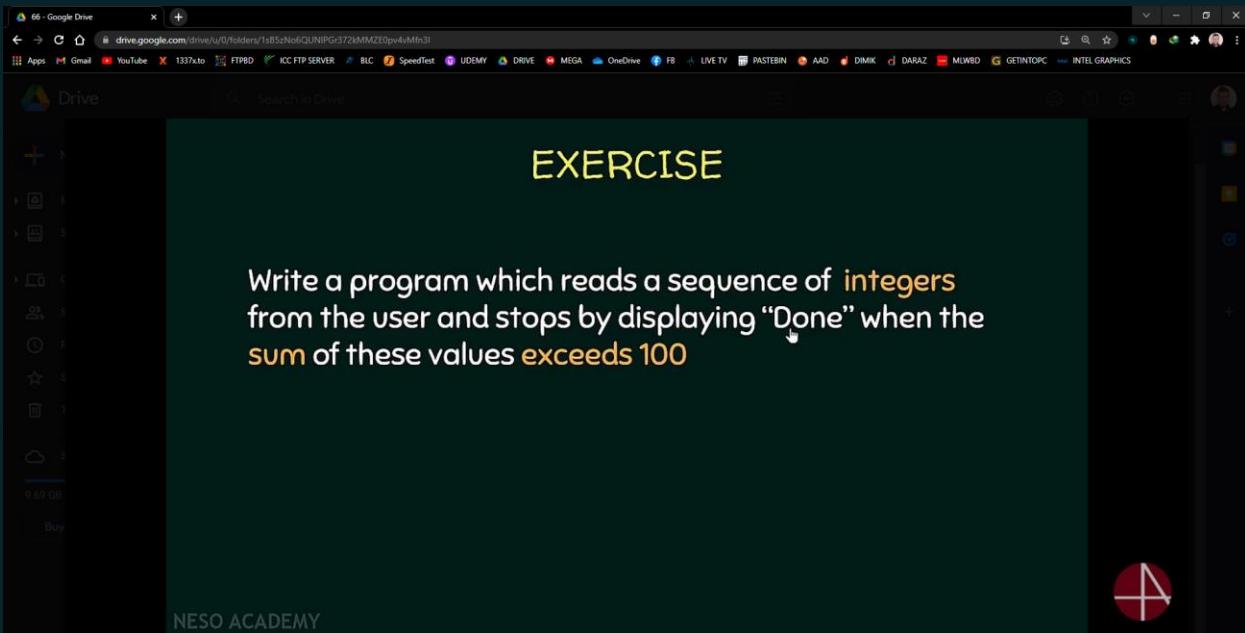


```
*  
**  
***  
****  
*****
```

MULTIPLICATION TABLE

```
for(int i = 1; i <= 10; i++) {  
    for(int j = 1; j <= 10; j++)  
        System.out.print((i * j) + " ");  
  
    System.out.println();  
}
```

45. sum of the values



```
public class Main {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        int sum = 0;
        while(true) {
            System.out.print("enter a number: ");
            sum += s.nextInt();

            if(sum > 100)
                break;
        }

        System.out.println("Done: " + sum);
    }
}
```

This is the same Java code as the one in the Google Drive folder, displayed in a code editor window. The code reads integers from the user until the sum exceeds 100, at which point it prints "Done" and the total sum.

A screenshot of a Java code editor (likely IntelliJ IDEA) running on a Windows 10 desktop. The code in the editor is:

```
int sum = 0;
while(sum <= 100) {
    System.out.print("enter a number: ");
    sum += s.nextInt();
}
System.out.println("Done: " + sum);
```

The run output shows the program's execution:

```
enter a number: 100
enter a number: 10
Done: 110
```

Process finished with exit code 0

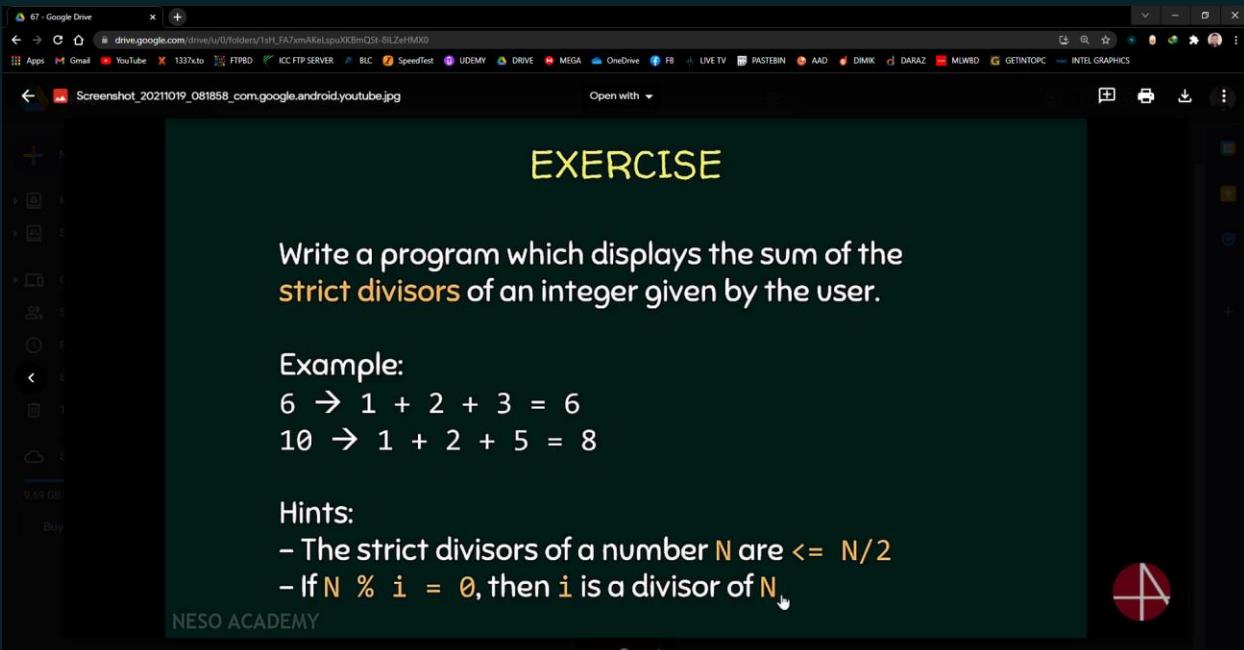
A screenshot of a Java code editor (likely IntelliJ IDEA) running on a Windows 10 desktop. The code in the editor is:

```
int sum = 0;
do {
    System.out.print("enter a number: ");
    sum += s.nextInt();
} while(sum <= 100);
System.out.println("Done: " + sum);
```

The run output shows the program's execution:

```
enter a number: 50
enter a number: 50
enter a number: 50
Done: 150
```

46. Sum of the strict divisors



```
5 > public class Main {
6 >     public static void main(String[] args) {
7 >         Scanner s = new Scanner(System.in);
8 >
9 >         System.out.print("enter a number: ");
10 >        int n = s.nextInt();
11 >
12 >        int sum = 0;
13 >        for(int i = 1; i <= n/2; i++)
14 >            if(n % i == 0)
15 >                sum += i;
16 >
17 >        System.out.println(sum);
18 >    }
}
```

A screenshot of a Java code editor (likely Eclipse or IntelliJ IDEA) displaying a Java program. The code is a prime number checker that prints all divisors of a given number. The code is as follows:

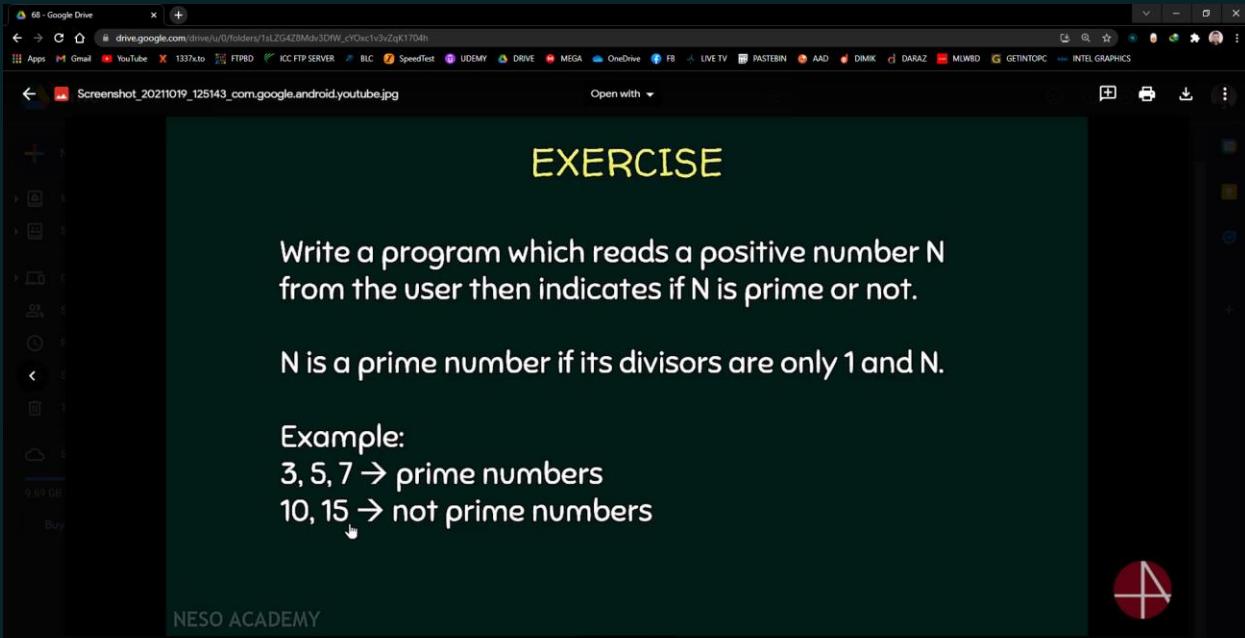
```
int sum = 0;
for(int i = 1; i <= n/2; i++)
    if(n % i == 0)
        System.out.print(i + " ");
System.out.println(sum);
```

The code is run in the terminal, and the output is:

```
enter a number: 20
1 2 4 5 10
Process finished with exit code 0
```

Windows Taskbar at the bottom:

47. Prime number



```
public class Main {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        System.out.print("enter a number: ");
        int n = s.nextInt();

        int sum = 1;
        for(int i = 2; i <= n/2; i++)
            if(n % i == 0)
                sum += i;

        System.out.println(sum == 1 ? "prime" : "not prime");
    }
}
```

A screenshot of a Java code editor displaying a program to check if a number is prime. The code uses a Scanner to read input from System.in, prints a prompt, reads an integer n, and then iterates from 2 to n/2 to check for factors. If a factor is found, it sets isPrime to false and breaks the loop. Finally, it prints "prime" or "not prime" based on the value of isPrime.

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

System.out.print("enter a number: ");
int n = s.nextInt();

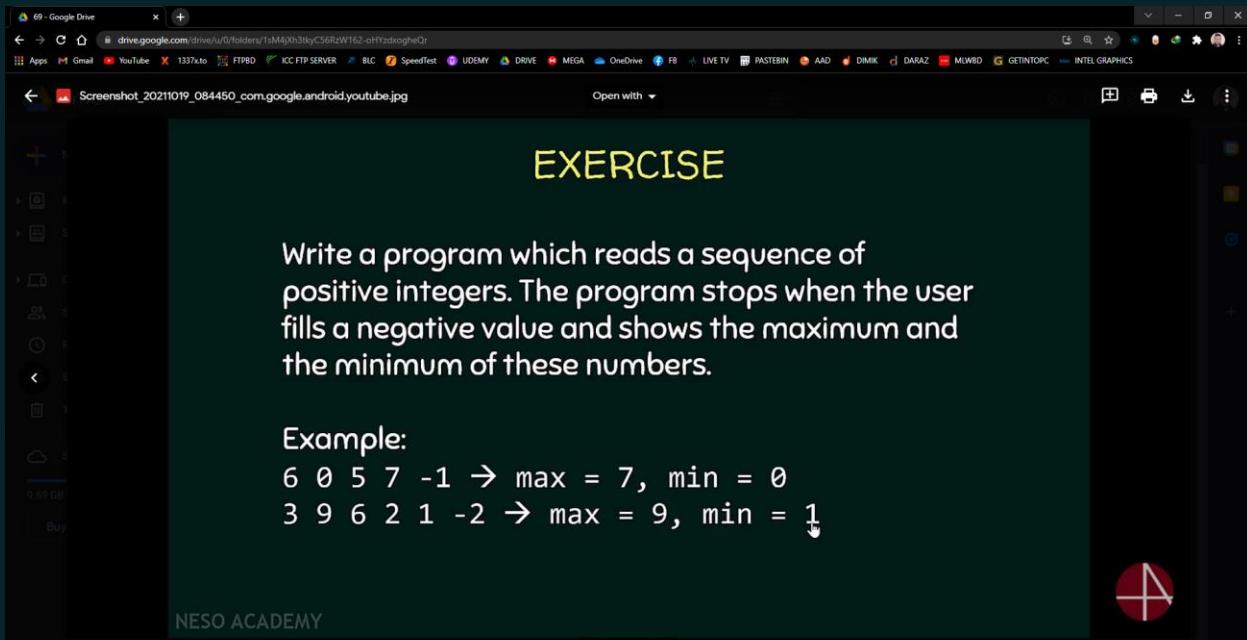
boolean isPrime = true;
for(int i = 2; i <= n/2; i++) {
    if(n % i == 0) {
        isPrime = false;
        break;
    }
}

System.out.println(isPrime ? "prime" : "not prime");
```

Type here to search

4:17 PM 10/19/2021

49. finding max and min



```
int n = s.nextInt();
int max = n;
int min = n;

while(true) {
    n = s.nextInt();

    if(n < 0)
        break;

    if(n > max)
        max = n;

    if(n < min)
        min = n;
}

System.out.println("min= " + min + ", max= " + max);
```

The code is a Java program that reads integers from standard input. It initializes variables `max` and `min` to the first input value. It then enters a loop where it repeatedly reads the next integer. If the integer is less than zero, it breaks out of the loop. Otherwise, it checks if the integer is greater than `max` or less than `min` and updates `max` or `min` accordingly. Finally, it prints the values of `min` and `max`.

A screenshot of a Java code editor displaying the following code:

```
int n = s.nextInt();
int max = n;
int min = n;

if(n >= 0) {
    while(true) { [ ] 
        n = s.nextInt();

        if(n < 0)
            break;

        if(n > max)
            max = n;

        if(n < min)
            min = n;
    }

    System.out.println("min= " + min + ", max= " + max);
} else
    System.out.println("invalid number");
}
```

The code is a Java program that reads integers from standard input. It initializes variables `n`, `max`, and `min` to the first input. A `while(true)` loop continues to read integers until one is less than zero, at which point it breaks. It then updates `max` and `min` accordingly. Finally, it prints the minimum and maximum values. If the first input is less than or equal to zero, it prints "invalid number".

Else invalid number

A screenshot of a Java code editor displaying the following code:

```
int n = s.nextInt();
int max = n;
int min = n;

if(n >= 0) {
    while(true) {
        n = s.nextInt();

        if(n < 0)
            break;

        max = n > max ? n : max;
        min = n < min ? n : min;
    }

    System.out.println("min= " + min + ", max= " + max);
} else
    System.out.println(n + " is invalid");
}
```

This version of the code is identical to the previous one, except for the update logic inside the loop. Instead of using separate `if` statements to update `max` and `min`, it uses ternary operators (`? :`) to assign `n` directly to either `max` or `min` based on whether `n` is greater than or less than the current value of `max` or `min`.

50. display the sum of the digit

Write a program which displays the sum of digits of an integer read from the user.

Example:

$108 \rightarrow 1 + 0 + 8 = 9$

$1123 \rightarrow 1 + 1 + 2 + 3 = 7$

$12345 \rightarrow 1 + 2 + 3 + 4 + 5 = 15$

```
package com.NesoAcademy;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        int n = s.nextInt();
        int sum = 0;
        while( n > 0 ) {
            sum += n % 10; // A
            n /= 10; // B
        }

        System.out.println("sum= " + sum);
    }
}
```

51. Fibonacci number

The screenshot shows a dark-themed web browser window. The title bar says "71 - Google Drive". The address bar shows "drive.google.com/drive/u/0/folders/1t3-Zfe8526iNAC4ck4jO_hSeQEH50EVQ". The page content is titled "EXERCISE". It contains text explaining the Fibonacci sequence: "The Fibonacci sequence is a sequence of numbers where the first two numbers in the sequence are 1 and 1. Then, each additional Fibonacci number is the sum of the two previous numbers in the sequence: 1, 1, 2, 3, 5, 8, 13, 21, ...". Below this, there is a challenge: "Write a program that reads an integer n and displays the n^{th} Fibonacci number". The footer of the page says "NESO ACADEMY".

The screenshot shows an IDE window with a Java file named "Main.java" open. The code is as follows:

```
int n = s.nextInt();

int result = 0;
int v1 = 1;
int v2 = 1;
for (int i = 1; i <= n - 2; i++) {
    result = v1 + v2;
    v1 = v2;
    v2 = result;
}

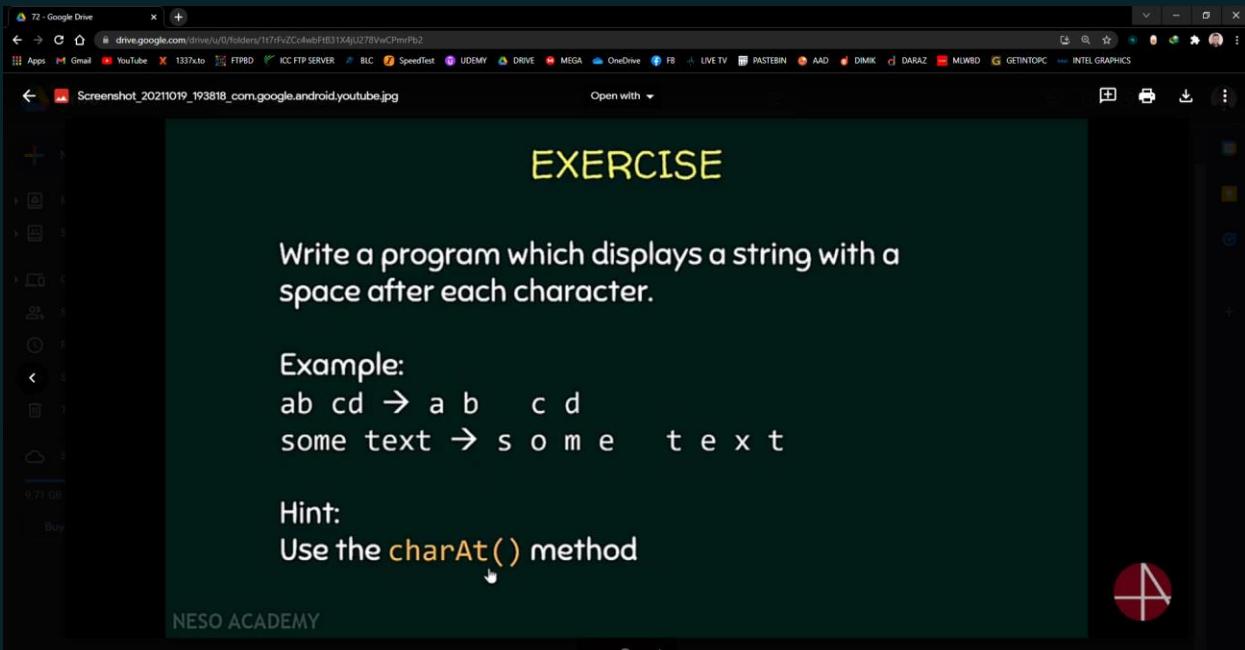
System.out.println("result = " + (result == 0 ? 1 : result));
```

The "Run" tab shows the output of running the program with input "5":

```
"C:\Program Files\Java\jdk-13.0.1\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2020.3.3\lib\idea_rt.jar" -Dfile.encoding=UTF-8 Main
enter a positive integer n: 5
result = 5
```

The footer of the IDE says "NESO ACADEMY".

52. display string with a space



The screenshot shows an IDE (Android Studio) displaying the Java code for the exercise. The code is as follows:

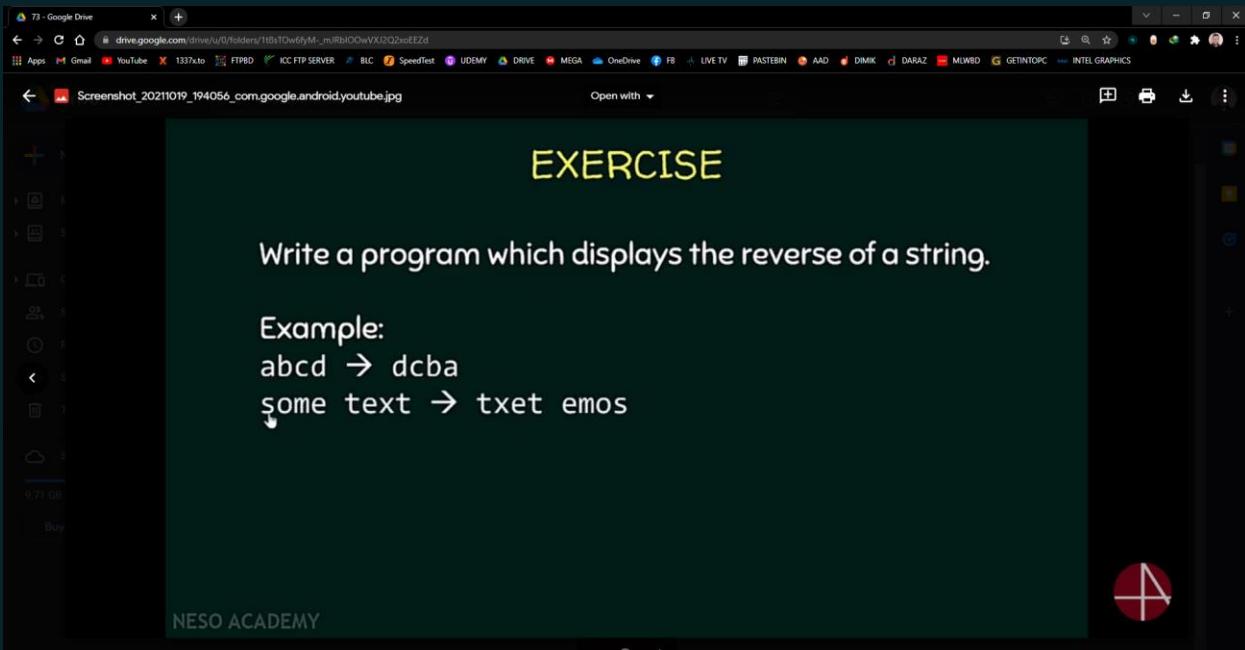
```
package com.NesoAcademy;

public class Main {
    public static void main(String[] args) {
        String str = "some text";
        for(int i = 0; i < str.length() - 1; i++)
            System.out.print(str.charAt(i) + " ");
    }
}
```

The Run tab shows the output of the program:

```
"C:\Program Files\Java\jdk-13.0.1\bin\java.exe" "-javaagent:C:\Program Files\J
some t e x t
NESO ACADEMYProcess finished with exit code 0
```

53. reverse string



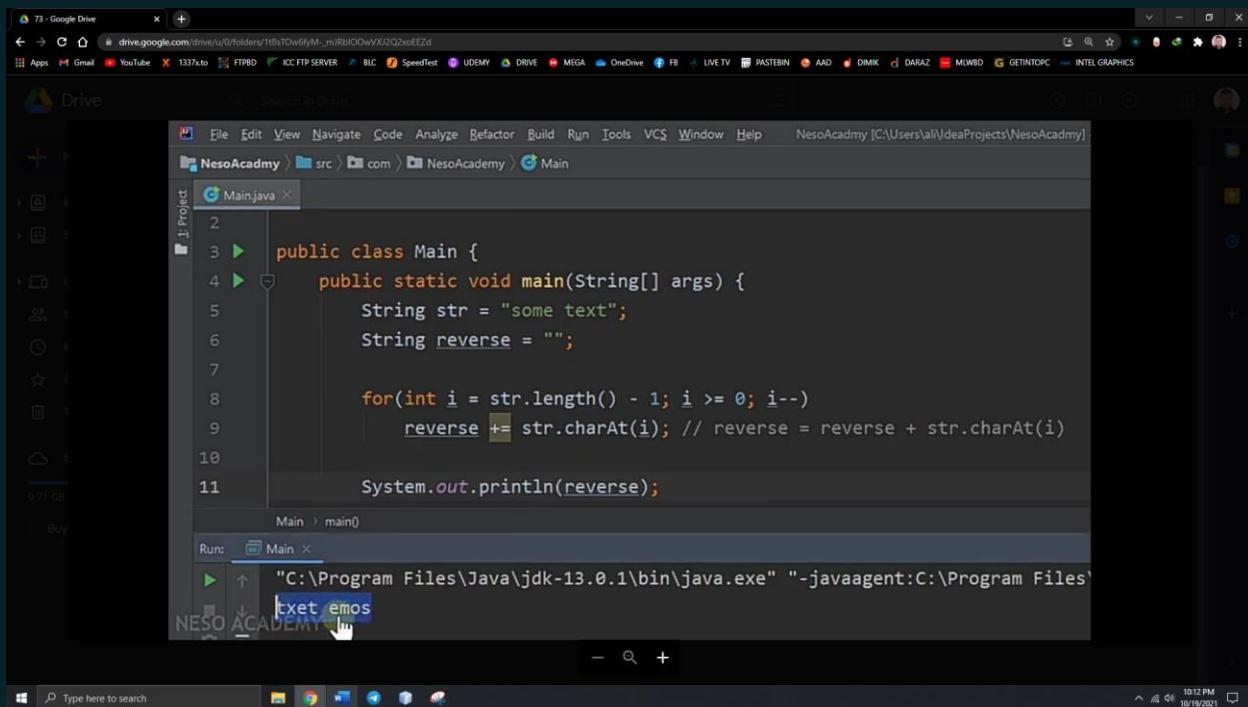
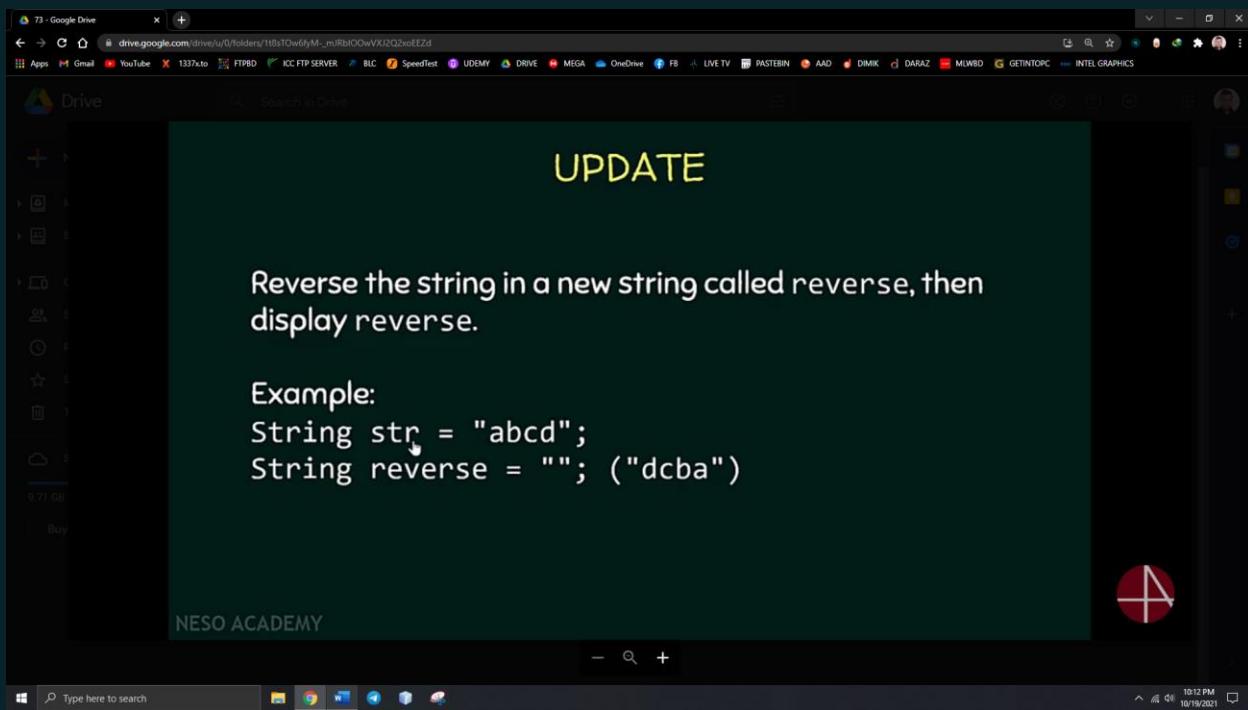
The screenshot shows an IDE window with the following Java code in a file named Main.java:

```
package com.NesoAcademy;

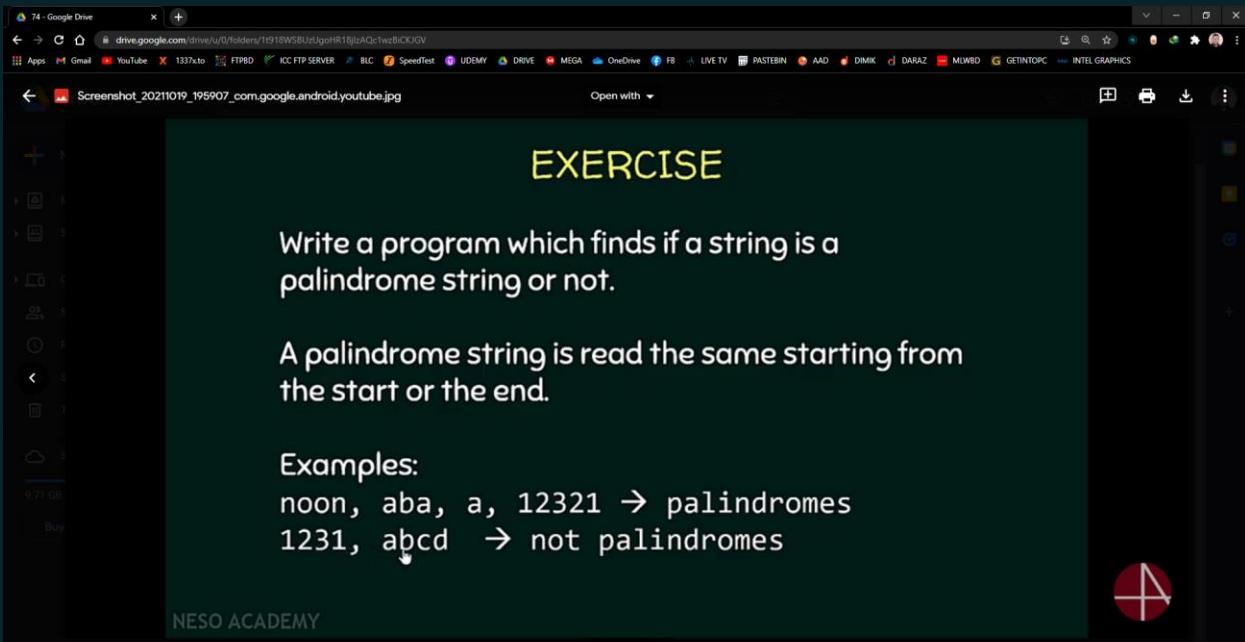
public class Main {
    public static void main(String[] args) {
        String str = "some text";

        for(int i = str.length() - 1; i >= 0; i--)
            System.out.print(str.charAt(i));
    }
}
```

The code uses a for loop to iterate through the characters of the string "some text" from the end to the beginning, printing each character using `System.out.print(str.charAt(i))`. The output in the terminal shows the string "txet emos" being printed.



54. String palindrome or not



```
public class Main {
    public static void main(String[] args) {
        String str = "aba";
        boolean isPalindrome = true;
        for(int i = 0, j = str.length() - 1; i < j; i++, j--) {
            if(str.charAt(i) == str.charAt(j))
                continue;
            isPalindrome = false;
            break;
        }
        System.out.println(isPalindrome ? "palindrome" : "not palindrome");
    }
}
```

55. display the following

Write a program that reads a positive integer N from the user and displays the following:

1
22
333
4444

...

NNNN...N

```
package com.NesoAcademy;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int n = s.nextInt();

        for(int i = 1; i <= n; i++) {
            for(int j = 1; j <= i; j++)
                System.out.print(i);

            System.out.println();
        }
    }
}
```

56.

Write a program that reads a positive integer N
and displays the following:

Example for N = 6:

```
*      ↴  
**  
***  
****  
*****  
*****
```

Example for N = 3:

```
*  
**  
***
```

```

public class Main {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int n = s.nextInt();

        for(int i = 1; i <= n; i++) {
            for(int j = 1; j <= n - i; j++)
                System.out.print(" ");

            for(int k = 1; k <= i; k++)
                System.out.print("*");
            System.out.println();
        }
    }
}

```

Example for N = 6:

```

*
 ***
 *****
 ******
 *****

```

Example for N = 3:

```

*
 ***
 *****

```

$2*i - 1$

```
public class Main {  
    public static void main(String[] args) {  
        Scanner s = new Scanner(System.in);  
        int n = s.nextInt();  
  
        for(int i = 1; i <= n; i++) {  
  
            for(int j = 1; j <= n - i; j++)  
                System.out.print(" ");  
  
            for(int j = 1; j <= 2*i - 1; j++)  
                System.out.print("*");  
  
            System.out.println();  
        }  
    }  
}
```

57.

Write a program that reads a positive integer N
and displays the following:

Example for N = 6:

```
*  
 * *  
 *   *  
 *       *  
 *           *  
*****
```

Example for N = 3:

```
*  
 * *  
*****
```

```
public class Main {  
    public static void main(String[] args) {  
        Scanner s = new Scanner(System.in);  
        int n = s.nextInt();  
  
        for(int i = 1; i <= n; i++) {  
  
            for(int j = 1; j <= n - i; j++)  
                System.out.print(" ");  
  
            for(int j = 1; j <= 2*i - 1; j++)  
                if(i == n)  
                    System.out.print("*");  
                else  
                    if(j == 1 || j == 2*i - 1)  
                        System.out.print("*");  
                    else  
                        System.out.print(" ");  
  
            System.out.println();  
        }  
    }  
}
```

58.

Write a program that reads a positive integer N and displays the following:

Example for N = 5:

```
*****  
*****  
*****  
*****  
*****
```

Example for N = 3:

```
***  
***  
***
```

```
public class Main {  
    public static void main(String[] args) {  
        Scanner s = new Scanner(System.in);  
        int n = s.nextInt();  
  
        for(int i = 1; i <= n; i++) {  
            for(int j = 1; j <= n; j++)  
                System.out.print("*");  
  
            System.out.println();  
        }  
    }  
}
```

Example for N = 5:

```
*****  
*   *  
*   *  
*   *  
*****
```

Example for N = 3:

```
***  
*  *  
***
```

```
public class Main {  
    public static void main(String[] args) {  
        Scanner s = new Scanner(System.in);  
        int n = s.nextInt();  
  
        for(int i = 1; i <= n; i++) {  
            if(i == 1 || i == n)  
                for(int j = 1; j <= n; j++)  
                    System.out.print("*");  
            else  
                for(int j = 1; j <= n; j++)  
                    if(j == 1 || j == n)  
                        System.out.print("*");  
                    else  
                        System.out.print(" ");  
            System.out.println();  
        }  
    }  
}
```

Example for N = 5:

```
*****  
***  
***  
***  
*****
```

Example for N = 3:

```
***  
*  
***
```

```
for(int i = 1; i <= n; i++) {  
    if(i == 1 || i == n)  
        for(int j = 1; j <= n; j++)  
            System.out.print("*");  
    else  
        for(int j = 1; j <= n; j++)  
            if(j == 1 || j == n)  
                System.out.print(" ");  
            else  
                System.out.print("*|");  
  
    System.out.println();  
}  
}
```

59. Scope and local variable in java

- The scope of the variable is the part of the program where the variable can be used or referenced.
- A variable define inside a method is called a local variable.
- The scope of a local variable starts from its declaration and continues to the end of the block that contains the value.
- A local variable must be declared or assigned a value before it can be used.
- Parameters are also local variables; their scope is the whole method.
 - ❖ args is the local variable for the main method.
 - ❖ We can declare the same variable but in different block.

60. Method in java

VOID METHOD: A method that doesn't return a value

```
package neso;

public class Neso {

    public static void main(String[] args) {

        sayHi();

    }

    public static void sayHi() {

        System.out.println("Hi");

    }

}
```

Output: Hi

VALUE RETURNING METHOD:

```
package neso;

public class Neso {

    public static void main(String[] args) {

        int a = 5, b = 5;

        System.out.println("Sum is: "+sum(5, 5));

        // System.out.println(sum(3, 4) + sum(5, 6));

    }

    public static int sum(int x, int y){

        return x + y;

    }

}
```

Output: Sum is: 10

Public: This keyword is used in order to make the method public. Because a public method can access from outside the class.

Static: This is a keyword. We will be able to call the method from inside the `main` method.

61. Passing arguments by value.

```
package neso;

public class Neso {

    public static void main(String[] args) {

        int x = 1; //this x is local for main method

        increment(x);

        System.out.println(x);//1

        //Arguments are the actual value.

    }

    public static void increment(int x){

        x++;

        System.out.println(x);//2

    }

}
```

Output:

2

1

62. Passing arguments by reference

```
package neso;

import java.awt.Point;

public class Neso {

    public static void main(String[] args) {

        Point p1 = new Point();
        p1.x = 1;
        p1.y = 2;
        System.out.println("x = " + p1.x + ", y = " + p1.y);

        change(p1);
        System.out.println("x = " + p1.x + ", y = " + p1.y);
    }

    public static void change(Point p2) {

        p2.x = 2;
        p2.y = 1;
    }
}

/*


- Passing arguments by reference is apply to all objects.
- p1 is the reference type so it contains the address.
- p1 & p2 are refer to the same point that we created the main method.


*/
```

63. Method overloading

```
package neso;

public class Neso {

    public static void main(String[] args) {

        System.out.println(sum(2, 3));

        System.out.println(sum(2, 3, 4));

        sayHi();

        sayHi("Golam kibria");

    }

    public static int sum(int x, int y) {

        return x + y;

    }

    public static int sum(int x, int y, int z) {

        return x + y + z;

    }

    public static void sayHi() {

        System.out.println("Hi");

    }

    public static void sayHi(String args) {

        System.out.println("Hi " + args);

    }

}
```

Output:

5

9

Hi

Hi Golam Kibria

Overloading:

Writing the same method with different parameter.

Overloading method can have different return type.

Overloading method must have different parameters.

64. Write a method that gets a name and age from the user.

```
package neso;

import java.util.Scanner;

public class Neso {

    public static void main(String[] args) {

        System.out.print("Enter your name and age: ");
        System.out.println(getName() + " " + getAge());
    }

    public static String getName() {

        Scanner input = new Scanner(System.in);

        String name = input.nextLine();

        return name;

        //return new Scanner(System.in).nextLine();
    }

    public static int getAge() {

        Scanner input = new Scanner(System.in);

        int age = input.nextInt();

        return age;
    }
}
```

Output:

Enter your name and age: kibria

20

kibria 20

65. Prints the prime numbers between two numbers.

```
package neso;

public class Neso {

    public static void main(String[] args) {

        primePrintBetween(10, 50);

    }

    public static boolean isPrime(int n) {

        for (int i = 2; i <= n / 2; i++) {

            if (n % i == 0) {

                return false;

            }

        }

        return true;

    }

    public static void primePrintBetween(int start, int end) {

        for (int i = start; i <= end; i++) {

            if (isPrime(i)) {

                System.out.println(i + " ");

            }

        }

    }

}
```

Output:

11

13

17

19

23

29

31

37

41

43

47

66. Single dimension array-1

Arrays:

- A collection of variables of the same data type.
- An array in java is an object.
- An array variable references a group of data.
- The size of an array is fixed.

Default values:

When an array is created its element are assigned the following default values-

- 0 for the numeric data types.
- \u0000 for char types.
- False for Boolean types.
- Null for reference type.

Creating and printing an array

```
package neso;

public class Neso {

    public static void main(String[] args) {

        /*
            creating array:
            datatype[] arrayName = new datatype[];
            int[] number = new int[];
        */
        double[] number = {1.2, 4.2, 4.5, 5, 6};

        /*This will print the address*/
        System.out.println(number);

        /*This will print the array*/
        for (int i = 0; i < number.length; i++) {
            System.out.println(number[i] + " ");
        }

        /*Alternative way to print an array*/
        System.out.println("\n");
        double[] num = new double[3];
        num[0] = 1.2;
        num[1] = 2.2;
        num[2] = 3.3;
        for (int i = 0; i < num.length; i++) {
            System.out.println(num[i] + " ");
        }
    }
}
```

OUTPUT:

```
[ D@15db9742
1.2
4.2
4.5
5.0
6.0
```

```
1.2
2.2
3.3
```

67. Single dimension array-2

```
package neso;

import java.awt.Point;

public class Neso {

    public static void main(String[] args) {

        char[] ch = {'a', 'b', 'c', 'd'};
        System.out.println(ch);
        /*The indicate must be between 0 and Length-1
         ArrayIndexOutOfBoundsException*/

        /*Arrays are passed by reference*/
        int[] num = {1, 2};
        System.out.println("Before changing: ");
        for (int i = 0; i < num.length; i++) {
            System.out.println(num[i] + " ");
        }
        change(num);
        printArray(num);

        /*An object is destroyed when it is no longer referenced by a
         variable. But this time the object is referred by variable that's why it will
         not be destroyed*/
        Point p = getPoint();
        System.out.println(p);

        int[] n = getNumber();
        printArray(n);
    }

    /*num and num are refer to the same address. That's why the value of the
     variable must be change after execution the change method*/
    public static void change(int[] num) {
        num[0] = 2;
        num[1] = 1;
    }

    public static void printArray(int[] num) {
        System.out.println("After Changing: ");
        for (int i = 0; i < num.length; i++) {
            System.out.println(num[i] + " ");
        }
    }
}
```

```

public static Point getPoint(){
    return new Point(1, 2);
    /*1 is called absessa and 2 is called ordinate*/
}

public static int[] getNumber(){
    int[] n = {1, 2, 3, 4, 5};
    return n;
    /*return new int[] {1, 2, 3, 4, 5};*/
}
}

/*
-Anonymous arrays: An array without a variable referencing it.
-Array is passed by reference to a method.
-If we want to return an array from a method the address of the
array is returned.
-As long as we have a variable that is referencing to the array,
the array is still existing in the memory and it will not be destroyed.
*/

```

OUTPUT:

abcd

Before changing:

1

2

After Changing:

2

1

java.awt.Point[x=1,y=2]

After Changing:

1

2

3

4

5

68. Array class(sorting)

A class that contains some static methods that are used with arrays.

```
package neso;

import java.util.Arrays;

public class Neso {

    public static void main(String[] args) {

        int[] numbers = {4, 3, 6, 1, 7, 8};
        Arrays.sort(numbers);
        for (int i = 0; i < numbers.length; i++) {
            System.out.print(numbers[i] + " ");
        }

        System.out.println("\n");
        char[] ch = {'e', 'a', 'A', 'f', 'D'};
        Arrays.sort(ch);
        for (int i = 0; i < ch.length; i++) {
            System.out.print(ch[i] + " ");
        }

        System.out.println("\n");
        int[] unicode = {'e', 'a', 'A', 'f', 'D'};
        Arrays.sort(unicode);
        for (int i = 0; i < unicode.length; i++) {
            System.out.print(unicode[i] + " ");
        }

        System.out.println("\n");
        int[] num = {4, 2, 6, 5, 7, 1}; //1, 4+1
        Arrays.sort(num, 1, 5);
        for (int i = 0; i < num.length; i++) {
            System.out.print(num[i] + " ");
        }

        System.out.println("\n");
        String[] str = {"hi", "hello", "kibria"};
        Arrays.sort(str);
        for (int i = 0; i < str.length; i++) {
            System.out.print(str[i] + " ");
        }
        System.out.println("\n");
    }
}
```


69. Array class(Searching)

```
package neso;

import java.util.Arrays;

public class Neso {

    public static void main(String[] args) {

        int[] number = {5, 4, 3, 2, 1, 0, -1};
        Arrays.sort(number); // -1, 0, 1, 2, 3, 4, 5

        System.out.println(Arrays.binarySearch(number, 4));
        System.out.println(Arrays.binarySearch(number, 3));
        System.out.println(Arrays.binarySearch(number, -3));
        System.out.println(Arrays.binarySearch(number, 6));

        System.out.println("\n");
        String[] str = {"a", "b", "c"};
        System.out.println(Arrays.binarySearch(str, "a"));
        System.out.println(Arrays.binarySearch(str, "c"));
        System.out.println(Arrays.binarySearch(str, "A"));
        System.out.println(Arrays.binarySearch(str, "d"));
    }
}
```

OUTPUT:

```
5
4
-1
-8
```

```
0
2
-1
-4
```

70. Array class>equals)

```
package neso;

import java.awt.Point;
import java.util.Arrays;

public class Neso {

    public static void main(String[] args) {

        int[] num1 = {1, 2, 3, 4, 5};
        int[] num2 = {1, 2, 3, 4, 5};
        int[] num3 = {7, 8, 9, 10};
        System.out.println(num1 == num2); //= compare the address.
        System.out.println(Arrays.equals(num1, num2));
        System.out.println(Arrays.equals(num1, num3));

        System.out.println("\n");
        //compare content to the string
        String[] str1 = {"a", "b", "c"};
        String[] str2 = {"a", "b", "c"};
        System.out.println(str1 == str2);
        System.out.println(Arrays.equals(str1, str2));

        System.out.println("\n");
        Point[] p1 = {new Point(1, 2), new Point(3, 4)};
        Point[] p2 = {new Point(1, 2), new Point(3, 4)};
        Point[] p3 = {new Point(6, 7), new Point(8, 9)};
        System.out.println(p1 == p2);
        System.out.println(Arrays.equals(p1, p2));
        System.out.println(Arrays.equals(p1, p3));

        System.out.println("\n");
        String s1 = new String("Hello");
        String s2 = new String("Hello");
        System.out.println(s1 == s2);
        System.out.println(s1.equals(s2));

        System.out.println("\n");
        Point point1 = new Point(1, 2);
        Point point2 = new Point(1, 2);
        System.out.println(point1 == point2);
        System.out.println(point1.equals(point2));

    }
}
```

OUTPUT:

false
true
false

false
true

false
true
false

false
true

false
true

71. Array class(filling and `toString`)

```
package neso;

import java.awt.Point;
import java.util.Arrays;

public class Neso {

    public static void main(String[] args) {

        //fill(array, value)
        int[] num1 = new int[8];
        Arrays.fill(num1, 3);
        for (int i = 0; i < num1.length; i++) {
            System.out.print(num1[i] + " ");
        }

        System.out.println("\n");
        //fill(array, fromindex, toindex, value)
        int[] num2 = new int[8];
        Arrays.fill(num2, 3, 7, 4);
        for (int i = 0; i < num2.length; i++) {
            System.out.print(num2[i] + " ");
        }

        System.out.println("\n");
        String[] str = new String[3];
        Arrays.fill(str, "hi");
        for (int i = 0; i < str.length; i++) {
            System.out.print(str[i]+ " ");
        }

        System.out.println("\n");
        Point[] p = new Point[3];
        Arrays.fill(p, new Point(0, 0), new Point(1, 2));
        for (int i = 0; i < p.length; i++) {
            System.out.println(p[i]+ " ");
        }
        System.out.println("\n");

        /*Alternative way to print array of string*/
        System.out.println(Arrays.toString(str));
        System.out.println(Arrays.toString(p));

        int[] x = {1, 2, 3, 4, 5};
        System.out.println(Arrays.toString(x));
    }
}
```


72. Variable length arguments

```
package neso;

public class Neso {

    public static void main(String[] args) {

        System.out.println(sum(1, 7));
        System.out.println(sum(1, 3, 4));
        System.out.println(sum(6, 7, 8, 9));

        int[] n = {1, 2, 3, 4};
        System.out.println(sum(n));
        /*Anonymous array*/
        System.out.println(sum(new int[]{1, 2, 3, 4}));
    }

    public static int sum(int... numbers) {
        int sum = 0;
        for (int i = 0; i < numbers.length; i++) {
            sum = sum + numbers[i];
        }
        return sum;
    }
}
/*
Only one variable length parameter may be specified in a method.
It must be the last parameter.
Any other parameter must precede it.
We can pass an array or a variable length of arguments to a variable
length parameter.
When invoking a method with a variable length of arguments, java creates
an array and passes the argument to it.
*/
```

Output:

```
8
8
30
10
10
```

All the variable length arguments must be put before the variable LA.

73. Write a program that fills an array with n integers entered by user. (at least 1 number and maximum 20 numbers)

```
package neso;

import java.util.Arrays;
import java.util.Scanner;

public class Neso {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        int n;
        System.out.println("How many numbers? (max 20)");
        n = input.nextInt();

        while (n > 20 || n <= 0) {
            System.out.println("Invalid number....Please try again!!!");
            n = input.nextInt();
        }

        int[] numbers = new int[n];
        fillArrayOfInteger(numbers);
        printArrayOfInteger(numbers);
    }

    private static void fillArrayOfInteger(int[] numbers) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter elements one by one: ");
        for (int i = 0; i < numbers.length; i++) {
            numbers[i] = input.nextInt();
        }
    }

    private static void printArrayOfInteger(int[] numbers) {
        System.out.println("The elements are: ");
        System.out.println(Arrays.toString(numbers));
    }
}
```


74. Fill an array with n integer using point.

```
package neso;

import java.awt.Point;
import java.util.Scanner;

public class Neso {

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        int n;
        System.out.println("How many numbers? (max 20)");
        n = input.nextInt();

        while (n > 20 || n <= 0) {
            System.out.println("Invalid number....Please try again!!!");
            n = input.nextInt();
        }

        Point[] p = new Point[n];
        fillArrayOfInteger(p);
        printArrayOfInteger(p);
    }

    private static void fillArrayOfInteger(Point[] p) {
        Scanner input = new Scanner(System.in);
        for (int i = 0; i < p.length; i++) {
            System.out.println("Enter x and y for point " + (i + 1) + ": ");
            p[i] = new Point(input.nextInt(), input.nextInt());
        }
    }

    private static void printArrayOfInteger(Point[] p) {
        System.out.println("The elements are: ");
        for (int i = 0; i < p.length; i++) {
            System.out.println("(" + p[i].x + ", " + p[i].y + ")");
        }
        //for(Point p1 : p) its alternative way.
    }
}
```


75. Write a program that display the sum, product and average of an integer array.

```
package neso;

import java.util.Scanner;

public class Neso {

    public static void main(String[] args) {

        //int[] num = {1, 2, 3, 4, 5};
        Scanner input = new Scanner(System.in);
        System.out.println("How many number? ");
        int n = input.nextInt();
        int[] num = new int[n];
        for (int i = 0; i < n; i++) {
            num[i] = input.nextInt();
        }
        int sum = 0;
        int product = 1;
        double average;

        for (int i = 0; i < num.length; i++) {
            sum = sum + num[i];
            product = product * num[i];
        }

        average = (double) sum / num.length;
        System.out.println("The sum is: " + sum);
        System.out.println("The product is: " + product);
        System.out.println("The average is: " + average);

    }
}
```

OUTPUT:

```
How many number?
3
1
2
3
The sum is: 6
The product is: 6
The average is: 2.0
```

76. Write a program that display the number of occurrences of an element in the array.

```
package neso;

public class Neso {

    public static void main(String[] args) {

        int[] numbers = {1, 3, 3, 4, 5, 7};
        int searchElement = 3;

        int f = find(numbers, searchElement);
        System.out.println(f + " times");
    }

    public static int find(int[] numbers, int s) {
        int occ = 0;
        for (int i = 0; i < numbers.length; i++) {
            if (numbers[i] == s) {
                occ++;
            }
        }
        return occ;
    }
}
/*
Without using find method:
int occ = 0;
for (int i = 0; i < numbers.length; i++) {
    if(searchElement == numbers[i]){
        occ++;
    }
}
System.out.println(occ +" times");
*/
```

Output:
2 times

77. Display the maximum and minimum element of an array.

```
package neso;

public class Neso {

    public static void main(String[] args) {

        int[] num = {1, 4, 2, 5, 6};

        int min = num[0];
        int max = num[0];

        for (int i = 1; i < num.length; i++) {
            min = (num[i] < min) ? num[i] : min;
            max = (num[i] > max) ? num[i] : max;
        }
        System.out.println("Minimum is: " + min + " & maximum is: " + max);
    }
}

Output:
Minimum is: 1 & maximum is: 6
```

78. Write a program that places the odd element of an array before the even element.

```
package neso;

import java.util.Arrays;

public class Neso {

    public static void main(String[] args) {

        int[] n = {1, 4, 2, 5, 7, 8, 10};
        int[] temp = new int[n.length];

        int j = 0;
        int k = n.length - 1;

        for (int i = 0; i < n.length; i++) {
            if (n[i] % 2 != 0) {
                temp[j++] = n[i];
            } else {
                temp[k--] = n[i];
            }
        }

        for (int i = 0; i < temp.length; i++) {
            n[i] = temp[i];
        }
        System.out.println(Arrays.toString(n));
    }
}

/*
USING ANOTHER METHOD:

copyArray(temp, n);
System.out.println(Arrays.toString(n));
public static void copyArray(int[] temp, int[] n){
    for (int i = 0; i < temp.length; i++) {
        n[i] = temp[i];
    }
}
*/

```

OUTPUT:

```
[1, 5, 7, 10, 8, 2, 4]
```

79. 2D array

```
package student;

public class TwoArray {

    public static void main(String[] args) {

        int[][] num = new int[2][2];

        num[0][0] = 2;
        num[0][1] = 3;
        num[1][0] = 4;
        num[1][1] = 5;

        System.out.print(num[0][0] + " ");
        System.out.println(num[0][1]);
        System.out.print(num[1][0] + " ");
        System.out.println(num[1][1]);

    }
}
```

OUTPUT:

```
2 3
4 5
```

80. 2D array printing row by row

```
package student;

public class TwoArray {

    public static void main(String[] args) {

        int[][] num = {{1, 2}, {3, 4}};

        //printing first row
        System.out.print(num[0][0] + " ");
        System.out.println(num[0][1]);

        //printing second row
        System.out.print(num[1][0] + " ");
        System.out.println(num[1][1]);

        System.out.println();

        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                System.out.print(num[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

OUTPTUT:

```
1 2
3 4
```

```
1 2
3 4
```

81. 2D array printing column by column

```
package student;

public class TwoArray {

    public static void main(String[] args) {

        int[][] num = {
            {1, 2},
            {3, 4}
        };

        //printing first column
        System.out.print(num[0][0] + " ");
        System.out.println(num[1][0] + " ");

        //printing second column
        System.out.print(num[0][1] + " ");
        System.out.println(num[1][1] + " ");

        System.out.println();

        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                System.out.print(num[j][i] + " ");
            }
            System.out.println();
        }
    }
}
```

OUTPUT:

```
1 3
2 4
```

```
1 3
2 4
```

82. Printing arrays using `toString` method

```
package student;

import java.util.Arrays;

public class TwoArray {

    public static void main(String[] args) {

        int[][] num = {{1, 2}, {3, 4}};

        System.out.println(Arrays.toString(num));
        System.out.println(Arrays.deepToString(num));
    }
}
```

Output:

```
[[I@15db9742, [I@6d06d69c]
[[1, 2], [3, 4]]
```

83. Printing a 2D array using method.

```
package student;  
public class TwoArray {  
  
    public static void main(String[] args) {  
  
        int[][] num = getArray();  
        printArray(num);  
  
    }  
  
    public static int[][] getArray() {  
        return new int[][]{{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};  
    }  
  
    public static void printArray(int[][] num) {  
        for (int i = 0; i < 3; i++) {  
            for (int j = 0; j < 3; j++) {  
                System.out.print(num[i][j] + " ");  
            }  
            System.out.println();  
        }  
    }  
}  
  
Output:  
1 2 3  
4 5 6  
7 8 9
```

84. Ragged arrays

```
package student;

public class TwoArray {

    public static void main(String[] args) {

        int[][] num = {
            {1, 2, 3},
            {4, 5},
            {6, 7, 8}
        };

        printArray(num);
    }

    public static void printArray(int[][] num) {
        for (int i = 0; i < num.length; i++) {
            for (int j = 0; j < num[i].length; j++) {
                System.out.print(num[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Output:

```
1 2 3
4 5
6 7 8
```

85. Print the sum of each row and each column of an 2D array.

```
package student;

public class A {

    public static void main(String[] args) {
        int[][] num = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        //printing row by row
        for (int i = 0; i < 3; i++) {
            int sum = 0;
            for (int j = 0; j < 3; j++) {
                sum = sum + num[i][j];
            }
            System.out.println("Sum of row " + (i + 1) + ": " + sum);
        }

        System.out.println();
        //printing column by column
        for (int i = 0; i < 3; i++) {
            int sum = 0;
            for (int j = 0; j < 3; j++) {
                sum = sum + num[j][i];
            }
            System.out.println("Sum of column " + (i + 1) + ": " + sum);
        }
    }
}
```

Output:

```
Sum of row 1: 6
Sum of row 2: 15
Sum of row 3: 24

Sum of column 1: 12
Sum of column 2: 15
Sum of column 3: 18
```

86. Prints the maximum of each row and each column in a 2D array.

```
package student;

public class A {

    public static void main(String[] args) {
        int[][] num = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        //printing row maximum
        for (int i = 0; i < 3; i++) {
            int max = num[i][0];
            for (int j = 1; j < 3; j++) {
                max = (num[i][j] > max) ? num[i][j] : max;
            }
            System.out.println("Maximum of row " + (i + 1) + ": " + max);
        }

        //printing column maximum
        System.out.println();
        for (int i = 0; i < 3; i++) {
            int max = num[i][0];
            for (int j = 1; j < 3; j++) {
                max = (num[j][i] > max) ? num[j][i] : max;
            }
            System.out.println("Maximum of column " + (i + 1) + ": " + max);
        }
    }
}
```

Output:

```
Maximum of row 1: 3
Maximum of row 2: 6
Maximum of row 3: 9

Maximum of column 1: 7
Maximum of column 2: 8
Maximum of column 3: 9
```

87. Print the maximum of each row using method.

```
package student;

public class A {

    public static void main(String[] args) {
        int[][] num = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        for (int i = 0; i < 3; i++) {
            System.out.println("Maximum of row " + (i + 1) + ": " +
getMax(num[i]));
        }
    }

    public static int getMax(int[] num) {
        int max = num[0];
        for (int i = 0; i < num.length; i++) {
            max = (max > num[i]) ? max : num[i];
        }
        return max;
    }
}

//Maximum of column ata try korlei parbo inshah-Allah
```

Output:

```
Maximum of row 1: 3
Maximum of row 2: 6
Maximum of row 3: 9
```

88. Using the add(), get(), set(), remove(), size() method.

```
package student;

import java.util.ArrayList;

public class A {

    public static void main(String[] args) {

        ArrayList<Integer> num1 = new ArrayList<>();
        ArrayList<Double> num2 = new ArrayList<>();
        ArrayList<String> fruit = new ArrayList<>();

        System.out.println("Using the add() method: ");
        fruit.add("Apple");
        fruit.add("Banana");
        fruit.add("Orange");
        System.out.println(fruit);
        fruit.add(2, "Pepy");
        System.out.println(fruit);
        fruit.add(4, "Pineapple");
        System.out.println(fruit);

        System.out.println("\nUsing the get() method: ");
        System.out.println(fruit.get(1));
        System.out.println(fruit.get(3));

        System.out.println("\nUsing the set() method: ");
        fruit.set(1, "Kola");
        System.out.println(fruit);

        System.out.println("\nUsing the remove() & size() method: ");
        System.out.println("Before removing the size is: "+fruit.size());
        fruit.remove(1);
        System.out.println(fruit);
        System.out.println(fruit.size());
        fruit.remove("Orange");
        System.out.println(fruit);
        fruit.clear();
        System.out.println(fruit);
        System.out.println(fruit.size());
    }
}

/*
In an array list we can store object(String, Double, Integer, Boolean,
Character) but not a premitive type(string, double, int, double, char)
*/
```


89. Printing an array list using for loop and using of sort method of collections classes

```
package student;

import java.util.ArrayList;
import java.util.Collections;

public class A {

    public static void main(String[] args) {

        ArrayList<Integer> num1 = new ArrayList<>();
        ArrayList<Double> num2 = new ArrayList<>();
        ArrayList<String> fruit = new ArrayList<>();

        fruit.add("Apple");
        fruit.add("Banana");
        fruit.add("Orange");
        fruit.add("Pineapple");

        for (int i = 0; i < fruit.size(); i++) {
            System.out.println(" " + fruit.get(i));
        }
        System.out.println(fruit);
        Collections.sort(fruit); //alphabetical order.
        System.out.println(fruit);

        num1.add(5);
        num1.add(7);
        num1.add(10);
        num1.add(8);
        System.out.println(num1);
        Collections.sort(num1);
        System.out.println(num1);
    }
}
```

OUTPUT:

```
Apple
Banana
Orange
Pineapple
[Apple, Banana, Orange, Pineapple]
[Apple, Banana, Orange, Pineapple]
[5, 7, 10, 8]
[5, 7, 8, 10]
```

90. Using for each loop for printing an arraylist or array.

```
package student;

import java.util.ArrayList;

public class A {

    public static void main(String[] args) {

        ArrayList<String> name = new ArrayList<>();
        name.add("Rahim");
        name.add("Karim");
        name.add("Kibria");
        name.add("Tausif");

        for (String x : name) {
            System.out.println(x);
        }

        System.out.println();
        String[] student = {"rahim", "kibria", "saim"};
        for (String y : student) {
            System.out.println(y);
        }

        System.out.println();
        int[] num = {1, 2, 3};
        int i = 0;
        for(int z : num){
            System.out.println(z + " is a element of index: "+i);
            i++;
        }
    }
}
/*
    In each iteration, the variable x hold the value of an element inside the
array list or array. Starting from the frist element.
    There is no index
    Safe(Boundaries)
*/
```


91. Create a list of unique elements taken from the user. Sort and print this element. (Many same element thakly oitkahy akta element dory sort korby then print korby)

```
package student;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;

public class A {

    public static void main(String[] args) {

        ArrayList<Integer> integers = new ArrayList<>();

        System.out.println("Enter five integer: ");
        Scanner input = new Scanner(System.in);
        for (int i = 0; i < 5; i++) {
            int temp = input.nextInt();

            if (!integers.contains(temp)) {
                integers.add(temp);
            }
        }
        Collections.sort(integers);
        System.out.println(integers);
    }
}

Output:
Enter five integer:
2 4 1 2 4
[1, 2, 4]
```

92. Creating a menu program with this options(add, remove, display, exit). The program runs infinitely until the user press fourth options.

```
package student;

import java.util.ArrayList;
import java.util.Scanner;

public class A {

    public static void main(String[] args) {

        ArrayList<Integer> num = new ArrayList<>();
        Scanner input = new Scanner(System.in);

        while (true) {
            displayMenu();
            int choice = input.nextInt();

            if (choice == 1) {
                //Add
                System.out.print("Enter an integer: ");
                int n = input.nextInt();
                num.add(n);
                System.out.println("Added");
            } else if (choice == 2) {
                //Remove
                System.out.print("Enter the value you want to remove: ");
                int removeElement = input.nextInt();
                if (num.contains(removeElement)) {
                    num.remove(Integer.valueOf(removeElement));
                    System.out.println("Removed");
                } else {
                    System.out.println("Elements not found");
                }
            } else if (choice == 3) {
                //Display
                System.out.println("Your list: " + num);
            } else if (choice == 4) {
                //Exit
                System.out.println("Good Bye");
                break;
            }
        }
    }
}
```

```
public static void displayMenu() {  
    System.out.println("1. Add");  
    System.out.println("2. Remove");  
    System.out.println("3. Display");  
    System.out.println("4. Exit");  
    System.out.println("Enter your choice: ");  
}  
}
```

OUTPUT:

```
1. Add  
2. Remove  
3. Display  
4. Exit  
Enter your choice:  
1  
Enter an integer: 2  
Added  
1. Add  
2. Remove  
3. Display  
4. Exit  
Enter your choice:  
1  
Enter an integer: 3  
Added  
1. Add  
2. Remove  
3. Display  
4. Exit  
Enter your choice:  
2  
Enter the value you want to remove: 2  
Removed  
1. Add  
2. Remove  
3. Display  
4. Exit  
Enter your choice:  
3  
Your list: [3]  
1. Add  
2. Remove  
3. Display  
4. Exit  
Enter your choice:  
4  
Good Bye  
BUILD SUCCESSFUL (total time: 22 seconds)
```

93. Creating a class in JAVA(Circle class)

```
package student;

import java.awt.Point;

public class Circle {

    Point center;
    double radius;

    Circle() {
    }

    Circle(Point center, double radius) {
        this.center = center;
        this.radius = radius;
    }

    double getParameter() {
        return 2 * Math.PI * this.radius;
    }

    double getArea() {
        return Math.PI * this.radius * this.radius;
    }

    void setCenter(Point center) {
        this.center = center;
    }

    void setRadius(double radius) {
        this.radius = radius;
        //this keyword help me to distinguished the parameter from the attribute.
    }
}

/*
 * The attribute which is called center is equal to the parameter center.
 */
```

```

package student;

import java.awt.*;

public class Test {

    public static void main(String[] args) {
        Circle c1 = new Circle(new Point(1, 2), 3);

        System.out.println(c1.getArea());
        System.out.println(c1.getParameter());
    }
}

```

OUTPUT:
28.27433882308138
18.8495592153876

যদি আমি টেষ্ট ক্লাসটা এইভাবে করি তাহলে নিচের মতো আউটপুট হবেঃ

```

package student;

import java.awt.*;

public class Test {

    public static void main(String[] args) {
        Circle c1 = new Circle(new Point(1, 2), 3);
        Circle c2 = new Circle(new Point(3, 4), 5);

        System.out.println("c1 radius: " + c1.radius);
        System.out.println("c2 radius: " + c2.radius);

        c1.radius = 10;
        c2.radius = 20;
        System.out.println("c1 radius: " + c1.radius);
        System.out.println("c2 radius: " + c2.radius);
    }
}

/*
   Each object is independent from other object. Cause each circle c1
and c2 has its own memory.
*/
Output:
c1 radius: 3.0
c2 radius: 5.0
c1 radius: 10.0
c2 radius: 20.0

```

94. Static variable and method.

STATIC VARIABLES & METHODS

**Static variables and static methods belong to the class,
they are shared between all objects**

- If an object modifies a static variable, all objects of the same class are affected.
- A static variable can be accessed without creating an instance of the class.
- A static method can be called using the same way.
- A static method can not access instance variables or methods.



```
package student;

import java.awt.Point;

public class Circle {

    Point center;
    double radius;
    static int numOfCircle;

    //Each time we use this constructore(Circle) to creater the circle
    object we are incresing the number of objects(numberOfCircle)
    Circle() {
        numOfCircle++;
    }

    Circle(Point center, double radius) {
        numOfCircle++;
        this.center = center;
        this.radius = radius;
    }

    double getParameter() {
        return 2 * Math.PI * this.radius;
    }

    double getArea() {
        return Math.PI * this.radius * this.radius;
    }

    void setCenter(Point center) {
        this.center = center;
    }

    void setRadius(double radius) {
        this.radius = radius;
    }

    static int numOfCircle(){
        return numOfCircle;
    }
}
```

```
package student;

import java.awt.*;

public class Test {

    public static void main(String[] args) {
        //We can access the static variable using the name of the
        class or name of the object.
        System.out.println(Circle.numOfCircle);

        Circle c1 = new Circle(new Point(1, 2), 3);
        System.out.println(Circle.numOfCircle);
        //System.out.println(c1.numOfCircle);

        Circle c2 = new Circle(new Point(3, 4), 5);
        System.out.println(Circle.numOfCircle);
        //System.out.println(c2.numOfCircle);
    }
}

//we use the new keyword in order to called the constructor of a
class.
Output:
0
1
2
```

CONSTRUCTORS

A method used to instantiate and initialize objects

- A constructor must have the same name as the class
- Constructors do not have a return type
- Constructors are invoked using the new operator
- Constructors play the role of initializing objects
- A default constructor is a zero-argument constructor with an empty body

OOP & OBJECTS

Object-oriented programming (OOP)

- Involves programming using objects.
- An object represents an entity/object in the real world.
- An object has a state, a behavior, and an identity.

Examples:

- Student, circle, button, bank account, ...
- To represent a real world Point in programming we create a Point Object.



THE STATE OF AN OBJECT

Represented by the data fields of the object with their current values

- The state of an object is also known as its properties or attributes.
- A point object, for example, has a data field **x** and a data field **y** that characterizes a point.
- A rectangle object, for example, has the data fields **width** and **height**, that characterizes a rectangle.



CLASSES & OBJECTS

A Class defines the data fields and actions of an Object

- To create an Object we need a template/blueprint that defines the data fields and methods that this Object will have
 - Class
 - An object is an instance of a Class
- Objects of the same type are created/instantiated from the same class
- A Java class uses variables to define the data fields and methods to define actions. There are special methods Called Constructors.

CONSTRUCTORS

A method used to create Objects

- A constructor can do anything, it is simply a method, but it is designed to do initializing actions, such as initializing the data fields of an object.

```
Point p1 = new Point(0, 1);

public Point(int x, int y) {
    this.x = x;
    this.y = y;
}
```

DEFAULT CONSTRUCTORS

A constructor that takes no parameters

If we create a class without creating a constructor, Java automatically creates a default constructor that takes no parameters and we will be able to use it to create objects of this class.

DEFAULT VALUES

```
class DefaultValues {  
  
    Point point; // null  
    String str; // null  
    double d; // 0.0  
    int i; // 0  
    char c; // '\u0000'  
    boolean b; // false  
  
}
```

VISIBILITY MODIFIERS

Visibility modifiers can be used to specify the visibility of a class and its members

- **public**: Can be used on classes and class members. Used for unrestricted access, i.e. can be accessed from any other class.
- **private**: Can be used on class members. Used for restricting the access to the defining class, i.e. can be accessed within the class only.
- **protected**
- If no visibility modifier is used, then by default the classes, methods, and data fields are accessible by any class in the same package.

```
package student;

public class Dhaka {

    public int x;
}

//public int x and just int x are the same thing.

package student;

public class TestJava {

    public static void main(String[] args) {

        Dhaka d = new Dhaka();
        d.x = 10;
        System.out.println(d.x);
    }
}

OUTPUT: 10
```

```
package student;

public class Dhaka {

    private int x;

    public void setX(int x) {
        this.x = x;
    }

    public int getX() {
        return x;
    }
}

package student;

public class TestJava {

    public static void main(String[] args) {

        Dhaka d = new Dhaka();
        d.setX(10);
        System.out.println(d.getX());
    }
}

OUTPUT: 10
```

```
package student;

public class TestJava {

    private int x;

    public static void main(String[] args) {

        TestJava d = new TestJava();
        d.setX(10);
        System.out.println(d.getX());
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getX() {
        return x;
    }
}
```

OUTPUT: 10

98. Immutable class and object

IMMUTABLE OBJECTS

An object whose contents cannot be changed

- Sometimes, we want to create an object whose contents cannot be changed once the object has been created. Such an object is called an **immutable object** and its class is an **immutable class**.
- The String class is an immutable class, and String objects are immutable.

IMMUTABLE CLASSES

- All data fields must be private
- There can't be any setter methods
- No getter methods can return a data field that is mutable
 - if there is a getter method that returns an object, this object should be immutable
 - Applies to data fields of reference type (objects)

99. This keyword in java

THIS

The **this** keyword refers to the object itself

Can be used to:

- Reference instance data fields
- Reference instance methods
- Invoke a constructor

100. Inner class in java

INNER CLASSES

A Class inside another Class

We will talk about:

- Member Class
- Static Member Class

An inner class can:

- Access private members of the containing class
- Have a **private** access modifier

100. UML class in java

UML-unified modeling language. We use this language in order to create some diagram in computer science.

+ public -private -----static

101. UML rectangle

RECTANGLE CLASS DIAGRAM

Rectangle

```
-width : double  
-height : double  
+Rectangle()  
+Rectangle(double, double)  
+getArea() : double  
+getPerimeter() : double  
+getters / setters
```

The default value for the width and height is 1.0

```
package student;

public class Rectangle {

    private double width;
    private double height;

    public Rectangle() {
        this(1.0, 1.0);
    }

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    public double getArea() {
        return this.width * this.height;
    }

    public double getParameter() {
        return 2 * (this.width + this.height);
    }

    public double getWidth() {
        return width;
    }

    public void setWidth(double width) {
        this.width = width;
    }

    public double getHeight() {
        return height;
    }

    public void setHeight(double height) {
        this.height = height;
    }
}
```

| ----- |

```
package student;

import java.util.Scanner;

public class TestJava {

    public static void main(String[] args) {

        Rectangle[] r = new Rectangle[3];

        Scanner input = new Scanner(System.in);
        for (int i = 0; i < r.length; i++) {
            System.out.print("Do you want to enter the width and
height? (Y/N)");
            char choice = input.next().charAt(0);
            if (choice == 'y') {
                System.out.print("Please enter the width and height:
");
                r[i] = new Rectangle(input.nextDouble(),
input.nextDouble());
            } else {
                r[i] = new Rectangle();
            }
        }
        System.out.println("The rectangles are: ");
        for (int i = 0; i < r.length; i++) {
            System.out.println(
                "Rectangle " + (i + 1)
                + ": width: " + r[i].getWidth()
                + ", height: " + r[i].getHeight());

            System.out.println(
                "Rectangle " + (i + 1)
                + ": area is: " + r[i].getArea());
            System.out.println();
        }
    }
}
```

```
OUTPUT:  
Do you want to enter the width and height? (Y/N)n  
Do you want to enter the width and height? (Y/N)n  
Do you want to enter the width and height? (Y/N)y  
Please enter the width and height: 2 3  
The rectangles are:  
Rectangle 1: width: 1.0, height: 1.0  
Rectangle 1: area is: 1.0  
  
Rectangle 2: width: 1.0, height: 1.0  
Rectangle 2: area is: 1.0  
  
Rectangle 3: width: 2.0, height: 3.0  
Rectangle 3: area is: 6.0
```

ACCOUNT CLASS DIAGRAM

```
Account
-id : int
-balance : double
-annualInterestRate : double
-dateCreated : Date
+Account(int, double, double)
+withdraw(double) : boolean
+deposit(double) : void
+getters / setters
```

Do not create a setDateCreated() method.

1. Create the following Account object:

```
testAccount : Account
id = 1122
balance = 20,000
annualInterestRate = 4.5
```

2. Withdraw 2500 from the account.
3. Deposit 3000 to the account.
4. Print the account's information.

```
package student;

import java.util.Date;

public class Account {

    private int id;
    private double balance;
    private double annualInterestRate;
    private Date dateCreated;

    public Account(int id, double balance, double annualInterestRate)
    {
        this.id = id;
        this.balance = balance;
        this.annualInterestRate = annualInterestRate;

        dateCreated = new Date();
    }

    public boolean withdraw(double amount) {
        if (balance < amount) {
            return false;
        } else {
            balance = balance - amount;
            return true;
        }
    }

    public void deposit(double amount) {
        balance = balance + amount;
    }
}
```

```
public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public double getBalance() {
    return balance;
}

public void setBalance(double balance) {
    this.balance = balance;
}

public double getAnnualInterestRate() {
    return annualInterestRate;
}

public void setAnnualInterestRate(double annualInterestRate) {
    this.annualInterestRate = annualInterestRate;
}

public Date getDateCreated() {
    return dateCreated;
}
}
```

```
package student;

public class TestAccount {

    public static void main(String[] args) {

        Account a = new Account(100, 20000, 4.5);

        if (a.withdraw(2500)) {
            System.out.println("Withdraw successful");
        } else {
            System.out.println("Withdraw not successful");
        }

        a.deposit(3000);
        System.out.println(a.getId() + " "
                           + a.getBalance() + " "
                           + a.getAnnualInterestRate() + " "
                           + a.getDateCreated().toLocaleString());
    }
}
```

OUTPUT:

```
Withdraw successful
100 20500.0 4.5 Oct 28, 2021 10:16:10 AM
```

Client

```
-id : int  
-name : String  
-phone : String  
-accounts : ArrayList<Account>  
+Client(int, String, String)  
+addAccount(Account) : boolean  
+removeAccount(accountId) : boolean  
+toString() : String  
+getters / setters
```

PROGRAM

1. Create an array of 2 clients
2. Add 2 accounts for the first client and 3 for the second one
3. Print the information of each client using `toString()`

```
package student;

import java.util.Date;

public class Account {

    private int id;
    private double balance;
    private double annualInterestRate;
    private Date dateCreated;

    public Account(int id, double balance, double annualInterestRate) {
        this.id = id;
        this.balance = balance;
        this.annualInterestRate = annualInterestRate;

        dateCreated = new Date();
    }

    public boolean withdraw(double amount) {
        if (balance < amount) {
            return false;
        } else {
            balance = balance - amount;
            return true;
        }
    }

    public void deposit(double amount) {
        balance = balance + amount;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public double getBalance() {
        return balance;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }

    public double getAnnualInterestRate() {
        return annualInterestRate;
    }
}
```

```
}

public void setAnnualInterestRate(double annualInterestRate) {
    this.annualInterestRate = annualInterestRate;
}

public Date getDateCreated() {
    return dateCreated;
}

public String toString() {
    return this.id + " " + this.balance + " " + this.annualInterestRate +
" " + this.dateCreated.toLocaleString();
}
}

/*
-----
-----*/
package student;

import java.util.ArrayList;

public class Client {

    private int id;
    private String name;
    private String phoneNumber;
    private ArrayList<Account> a;

    public Client(int id, String name, String phoneNumber) {
        this.id = id;
        this.name = name;
        this.phoneNumber = phoneNumber;

        a = new ArrayList<>();
    }

    public boolean addAccount(Account acc) {
        a.add(acc);
        return true;
    }
}
```

```
public boolean removeAccount(int accountId) {
    for (Account x : a) {
        if (x.getId() == accountId) {
            a.remove(x);
            return true;
        }
    }
    return false;
}

public String toString() {
    String s = this.id + " " + this.name + " " + this.phoneNumber + "\n";

    for (Account y : a) {
        s = s + y.toString() + "\n";
    }

    return s;
}

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getPhoneNumber() {
    return phoneNumber;
}

public void setPhoneNumber(String phoneNumber) {
    this.phoneNumber = phoneNumber;
}
}

/*
-----
```

```
package student;

public class TestAccount {

    public static void main(String[] args) {

        Client[] c = new Client[2];

        c[0] = new Client(100, "Kibria", "23144345234");
        c[0].addAccount(new Account(1, 100, 1.5));
        c[0].addAccount(new Account(2, 200, 2.0));

        c[1] = new Client(101, "Rahim", "564562435");
        c[1].addAccount(new Account(3, 300, 2.5));
        c[1].addAccount(new Account(4, 400, 3.0));
        c[1].addAccount(new Account(5, 500, 3.5));

        System.out.println(c[0].toString());
        System.out.println(c[1].toString());
    }
}

/*
-----*/
```

OUTPUT:

```
100 Kibria 23144345234
1 100.0 1.5 Oct 28, 2021 11:06:38 AM
2 200.0 2.0 Oct 28, 2021 11:06:38 AM

101 Rahim 564562435
3 300.0 2.5 Oct 28, 2021 11:06:38 AM
4 400.0 3.0 Oct 28, 2021 11:06:38 AM
5 500.0 3.5 Oct 28, 2021 11:06:38 AM
```

UML-4:

Account

```
-id : int  
-balance : double  
-annualInterestRate : double  
-dateCreated : Date  
-client : Client  
-transactions : ArrayList<Transaction>  
+Account(int, double, double, Client)  
+withdraw(double) : boolean  
+deposit(double) : void  
+getters / setters
```

PROGRAM

1. Create an ArrayList of Accounts
2. Display the number of deposits and withdrawals of each account

```
package student;

import java.util.ArrayList;
import java.util.Date;

public class Account {

    private int id;
    private double balance;
    private double annualInterestRate;
    private Date dateCreated;
    private Client c;
    private ArrayList<Transaction> t;

    public Account(int id, double balance, double annualInterestRate, Client
c) {
        this.id = id;
        this.balance = balance;
        this.annualInterestRate = annualInterestRate;
        this.c = c;

        this.t = new ArrayList<>();

        dateCreated = new Date();
    }

    public boolean withdraw(double amount) {
        if (balance < amount) {
            return false;
        } else {
            balance = balance - amount;
            this.t.add(new Transaction('W', amount, this.balance, "Withdraw"
+ amount));
            return true;
        }
    }

    public void deposit(double amount) {
        balance = balance + amount;
        this.t.add(new Transaction('D', amount, this.balance, "Deposit" +
amount));
    }

    public int countTransaction(char type){
        int count = 0;
        for(Transaction z : t){
            if(z.getType()==type){
                count++;
            }
        }
    }
}
```

```
        }
        return count;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public double getBalance() {
        return balance;
    }

    public void setBalance(double balance) {
        this.balance = balance;
    }

    public double getAnnualInterestRate() {
        return annualInterestRate;
    }

    public void setAnnualInterestRate(double annualInterestRate) {
        this.annualInterestRate = annualInterestRate;
    }

    public Date getDateCreated() {
        return dateCreated;
    }

    public Client getC() {
        return c;
    }

    public void setC(Client c) {
        this.c = c;
    }

    public String toString() {
        return this.id + " " + this.balance + " " + this.annualInterestRate +
" " + this.dateCreated.toLocalizedString();
    }
}

/*
-----  

-----  

-----*/
```

```
package student;

import java.util.ArrayList;

public class Client {

    private int id;
    private String name;
    private String phoneNumber;
    private ArrayList<Account> a;

    public Client(int id, String name, String phoneNumber) {
        this.id = id;
        this.name = name;
        this.phoneNumber = phoneNumber;

        a = new ArrayList<>();
    }

    public boolean addAccount(Account acc) {
        a.add(acc);
        return true;
    }

    public boolean removeAccount(int accountId) {
        for (Account x : a) {
            if (x.getId() == accountId) {
                a.remove(x);
                return true;
            }
        }
        return false;
    }

    public String toString() {
        String s = this.id + " " + this.name + " " + this.phoneNumber + "\n";

        for (Account y : a) {
            s = s + y.toString() + "\n";
        }

        return s;
    }

    public int getId() {
        return id;
    }
}
```

```
public void setId(int id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getPhoneNumber() {
    return phoneNumber;
}

public void setPhoneNumber(String phoneNumber) {
    this.phoneNumber = phoneNumber;
}
}

/*
-----
-----
-----
-----*/



package student;

import java.util.Date;

public class Transaction {

    private char type;
    private double amount;
    private double balance;
    private Date date;
    private String description;

    public Transaction(char type, double amount, double balance, String description) {
        this.type = type;
        this.amount = amount;
        this.balance = balance;
        this.description = description;

        date = new Date();
    }

    public char getType() {
```

```
        return type;
    }
}

/*
-----
-----
-----*/
package student;

import java.util.ArrayList;

public class TestAccount {

    public static void main(String[] args) {

        ArrayList<Account> a = new ArrayList<>();
        Client c = new Client(100, "kibria", "4582932");

        a.add(new Account(1, 1000, 1.5, c));
        a.add(new Account(2, 2000, 2.5, c));

        a.get(0).withdraw(2000);
        a.get(0).withdraw(1000);
        a.get(0).deposit(2000);
        a.get(0).withdraw(1000);

        a.get(1).deposit(100);
        a.get(1).deposit(200);

        for(Account account : a){
            System.out.println("Account: "+account.getId()+" : ");
            System.out.println("W:"+account.countTransaction('W'));
            System.out.println("D:"+account.countTransaction('D'));
        }
    }
}

/*
-----
-----
-----*/

```

