Zero to Pro Bootcamp – Level 3

Contents

Overview of Java Programming Language

- Java is a high level programming language.
- It is an object oriented programming language.
- Different types of applications such as standalone applications, web Applications and mobile applications can be created using Java Language.

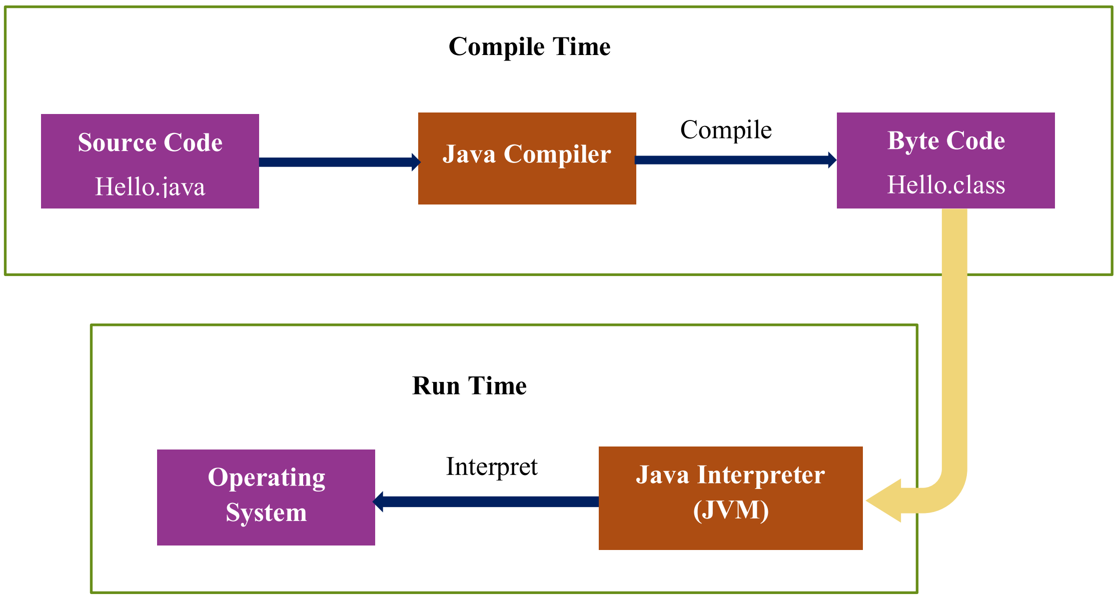
Features of Java

- Platform Independent
- Architectural Neutral
- Portable
- Write One Run Anywhere

Simple Java Program

```
public class Hello {
   public static void main(String[] args)
   {
      // Print statement
      System.out.println("Hello World!");
   }
}
```

How to run Java Program on Machine?



Java's Data Types

There are basically two categories of data types in java.

1. Primitive – for storing simple values such as integer, float etc.
2. Reference – for storing complex objects such as array, string etc.

Primitive Types

Type	Bytes	Range
byte	1	[-128 to 127]
short	2	[-32,768 to 32,767]
int	4	[-2,147,483,648 to 2,147,483,647]
long	8	[-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807]
float	4	Sufficient for storing 6 to 7 decimal digits
double	8	Sufficient for storing 15 decimal digits
char	2	Stores a single character/letter or ASCII values
boolean	1	true/ false

What is Variable?

- A Java variable is a piece of memory that can contain a data value.
- The value stored in a variable can be changed during program execution.

Declaring Variable

- **Syntax:** [access-modifier] data type variable name;
- **Example:**

```
int i,j,k;
float x,y,z;
char ch;
```

Initializing Variable

- You can initialize a variable when you declare it.
- Example: int total = 0;

Types of Java Variable

- Variables can be declared in three places: inside methods, in the definition of method parameters, and outside of all methods.

- There are four types of variables in Java:
    - Local variable
    - Instance variable
    - Static variable
    - Parameters

Local Variable

- A variable defined within a block or method is call local variable.
- It is mandatory to initialize the local variable.
- Access modifiers cannot be used for local variables.
- A local variable cannot be defined with "static" keyword.
- The scope of these variables exists only within the block in which the variable is declared.

Example

```
public class Test {
        void display() {
                // local variable
                int barCode;
                String name = "Coffee";
                System.out.println("BarCode: " + barCode);// compile-time error
                System.out.println("Name: " + name);
        }
        public static void main(String[] args) {
                {
                        String shopName = "ABC de pa to"; // local variable
                }
                System.out.println(shopName);// cannot access it
        }
}
```

- Java 10 introduced "**var**" keyword to declare local variable without data type.
- E.g. var name = "Jeon" instread of String = "Jeon".
- "var" cannot be used for fields, method parameters, and method return types.
- It cannot be used as declaration (e.g. var name;).
- It cannot be initialized null. (E.g. var name = null).

Example

```
public static void main(String[] args) {
        var name = "Jeon";
        var age = 25;
        var mark = "100";
        var address; // cannot use in declaration
        var phone = null; // cannot initialize null
        var sub1 = 30,sub2 =50; // cannot use variable list
}
```

Instance Variable

- Instance variables are defined outside all methods. It is not declared as static.
- Instance variables can be accessed only by creating objects.

Static Variable

- These variables are declared similarly as instance variables, the difference is that it is declared using the "static" keyword.
- Static variables are also called class variables. They can be accessed with the class name.

Example

```
public class Test {
        static String name;// static variable
        int price;// instance variable
        void display() {
                // local variable
                int barCode = 1001;
                name = "Juice";
                price = 1500;
                System.out.println("Code: " + barCode);
                System.out.println("Name : " + name);
                System.out.println("Price : " + price);
        }

}
```

Instance and static variables will be discussed in more details on Java's data members.

Parameters

- A parameter is a variable that is passed to a method when the method is called.
- Parameters are also only accessible inside the method that declares them.

Example

```
public class Test {

        void addNumbers(int a,int b) {// a and b are parameters
                int result = a + b;
                System.out.println(a + " + " + b + " = " + result);
        }
```

```
	public static void main(String[] args) {
		Test t =new Test();
		t.addNumbers(100, 200);
	}

}
```

Java's Constant Variable

- The value of constant variable cannot be changed once it has been assigned.
- ***Syntax*** - static final data-type NAME = VALUE;

Example

```
	static final float RATE = 1.5f;
	static final int MIN_PRICE = 1000;

	public static void main(String[] args) {

		Scanner sc = new Scanner(System.in);
		System.out.print("Enter price : ");
		int price = sc.nextInt();
		if(price < MIN_PRICE) {
			price = MIN_PRICE;
		}

		System.out.println("\nExpense: "+(price * RATE));
		RATE = 0.5f; // compile-time error
	}
```

Data Type Casting

- Convert a value from one data type to another data type is known as type casting.
- There are two major types of casting in java.

1. Widening Type Casting (Implicit Casting)

It is done automatically. It is safe because there is no chance to lose data.

```
byte > short > char > int > long > float > double
```

2. Narrowing Type Casting (Explicit Casting)

It is done manually by the programmer.

```
double > float > long > int > char > short > byte
```

Example

```
/* implicit casting */
   int a = 200;
   long b = a; //implicit casting from int to long
   double c = b; // implicit casting from long to double

   System.out.println(a);
   System.out.println(b);
   System.out.println(c);

   /* explicit casting */
   System.out.println("_____explicit casting_______");
   double d = 57.17;
   int i = (int)d;  // Explicit casting
   System.out.println(d);
   System.out.println(i);
```

OPERATORS

Like other programming languages, Java provides many different types of operators. Some of the types are –

- Arithmetic Operator
  - +, - , *, /, %
- Assignment Operator
  - =, +=, -=, *=, /=, %=
- Comparison Operator
  - ==, !=, >, >= , <, <=
- Logical Operator
  - &&, ||, !
- Ternary Operator
  - (exp1) ? exp2 : exp3
- Increment/Decrement Operator
  - ++, --

Example

```
public static void main(String[] args) {

        int num1 = 100;
        int num2 = 200;

        System.out.println(num1 + " + " + num2 + " = " + (num1 + num2));
        System.out.println(num1 + " - " + num2 + " = " + (num1 - num2));

        System.out.println(num1 + " is equal " + num2 + " is " + (num1 == num2));
        System.out.println(num1 + " is less than " + num2 + " is " + (num1 < num2));

        System.out.println("80 is divisible with 5 and 8 : " + ((80%5 == 0) && (80%8 ==0)));

        String result = (18%2 == 0) ? "even" : "odd";
        System.out.println("18 is " + result);

        num1 += 400;
        num2 -= 100;
        System.out.println("After updating, num1 = " + num1 + ", num2 = " + num2);

}
```

JAVA'S METHOD

- It is a separate code block and that contain a series of statements to perform a particular operations.
- They are useful to improve the code reusability.

General syntax –

```
access-specifier  return-type methodNName(Parameters)
   {
      // Statements to Execute
   }
```

- It has two types: Instance methods and Class or Static methods.
- **Instance method**
    - It is associated with each object.
    - Both instance and static fields can be used inside it.
    - Invoked via object (objName.methodName()).
- **Class or Static method**
    - It is associated with the whole class (identified as "static").
    - Instance fields and "this" and "super" keywords cannot be used inside it.
    - Directly invoked from class level without object creation (className.methodName()).

Example

```
public class Demo {
        int num1 = 10;
        static int num2 = 20;
        static void static_method() {
                // cannot access
                //System.out.println("num1 = " + num1);
                System.out.println("num2 = " + num2);
        }
        void instance_method() {
                System.out.println("num1 = " + num1);
                System.out.println("num2 = " + num2);
        }

        public static void main(String[] args) {

                Demo obj = new Demo();

                obj.instance_method();
                Demo.static_method();
        }
}
```

ACCEPTING VALUES FROM CONSOLE

1. Using BufferedReader Class
2. Using Console Class
3. Using Scanner Class

Using BufferedReader Class

- BufferedReader Class is defined in java.io class so you need to import java.io package.
- It reads text from console.
- **read()** method reads a single character from the console.
- **readLine()** method reads a line of text from the console.

Example

```
// create bufferReader stream
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Enter name : ");
String name = br.readLine(); // read whole line
System.out.print("Enter salary : ");
double salary = Double.parseDouble(br.readLine()); // get double
System.out.print("Enter age : ");
int age = Integer.parseInt(br.readLine()); // get integer

br.close(); // close the stream

System.out.println("\nName : " + name);
System.out.println("Salary : " + salary);
System.out.println("Age : " + age);
```

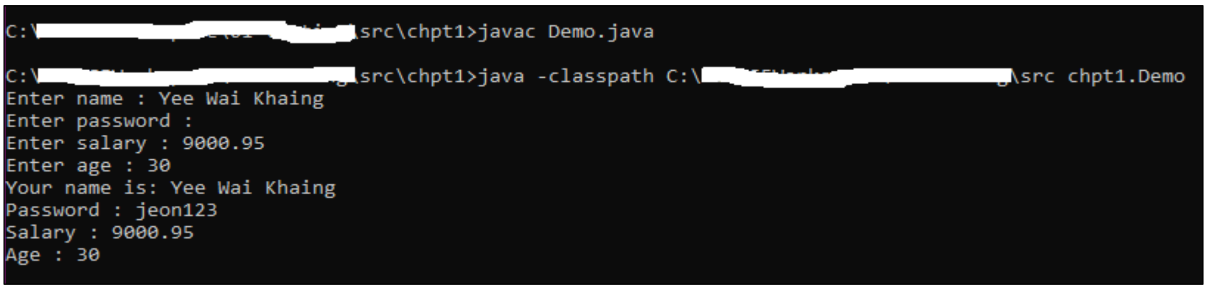
Using Console Class

- Console class is under java.io package.
- It is used to read text from and write to the console.
- **readLine()** method writes a line to the console and then read a line from the console.
- **readPassword()** method is used to read the secure input.
- Does not work in non-interactive environment (such as in an IDE).

Example

```
Console console = System.console(); // get console object
String name = console.readLine("Enter name : "); // read text
char password[] = console.readPassword("Enter password : "); // read password
double salary = Double.parseDouble(console.readLine("Enter salary : ")); // get double
int age = Integer.parseInt(console.readLine("Enter age : ")); // get int

System.out.println ("Your name is: " + name);
System.out.println("Password : " + String.valueOf(password));
System.out.println("Salary : " + salary);
System.out.println("Age : " + age);
```

Output



```
C:\[redacted]\src\chpt1>javac Demo.java

C:\[redacted]\src\chpt1>java -classpath C:\[redacted]\src chpt1.Demo
Enter name : Yee Wai Khaing
Enter password :
Enter salary : 9000.95
Enter age : 30
Your name is: Yee Wai Khaing
Password : jeon123
Salary : 9000.95
Age : 30
```

Using Scanner Class

- Scanner class is under java.util package.
- It provides various methods to read different types of data such as integer, Boolean, double etc.
- **next()** method read the input word as String.
- **nextLine()** method reads entire line of input as String.
- **nextInt()** method reads input value as an Integer.

Example

```
Scanner scanner = new Scanner(System.in); // get scanner stream

System.out.print("Enter name : ");
String name = scanner.nextLine(); // get whole line
System.out.print("Enter salary : ");
double salary = scanner.nextDouble(); // get double
System.out.print("Enter age : ");
int age = scanner.nextInt(); // get integer
```

```
scanner.close();  // close stream
System.out.println("Name : " + name);
System.out.println("Salary : " + salary);
System.out.println("Age : " + age);
```

- Write a Java program that takes input data of your profile
  - Full name(string)
  - email(string)
  - phone(string)
  - education(string)
  - income(double)
  - age(int)
- And display these data.

FLOW CONTROL STATEMENT

Conditional Statement

- if
- if … else
- if .. else if…
- switch - case

Looping Statement

- while
- do - while
- for
- foreach

Jump Statement

- break
- continue
- return

Conditional Statement

- A conditional statement causes the program to change the path of execution based on the value of an expression.
- If the value of the expression is only true, the statement block will be executed.

Syntax 1

```
if (expression)
        [statements]
```

Syntax 2

```
if (expression)
        [statements]
else
        [statements]
```

Syntax 3

```
if (expression)
        [statements]
else if  (condition)
        [statements]
else
        [statements]
```

Example – if statement

```
Scanner sc = new Scanner(System.in);
System.out.print("Enter Language name : ");
String lang = sc.nextLine();

if(lang.equals("java")) {
        System.out.println("Best Programming Language!");
}
sc.close();
```

Example – if else statement

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

System.out.print("Enter email : ");
```

```
String email = sc.nextLine();
System.out.print("Enter password : ");
String pass = sc.nextLine();

if(email.equals("jk@gmail.com") && pass.equals("jeon")) {
        System.out.println("Login success");
}else {
        System.err.println("Email and password don't match!");
}

sc.close();
```

Example – if … else if statement

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

System.out.print("Enter your email : ");
float bmi = sc.nextFloat();
if(bmi < 18.5)
        System.out.println("Underweight");
else if(bmi >= 18.5 && bmi <= 24.9)
        System.out.println("Normal Weight");
else if(bmi >= 25 && bmi <= 29.9)
        System.out.println("Overweight");
else
        System.out.println("Obesity");

sc.close();
```

Switch statement

- Switch statement allows us to replace several nested if-else constructs and thus improve the readability of our code.

Syntax

```
switch ( expression )
{
case value-1: block-1

              break;
case value-2: block-2

              break;
……………..
default :
          default-block
}
```

- In java, the case values allow integer, string, and enum data types.

ulting

Example

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

System.out.print("Enter food name : ");
String name = sc.nextLine();
String category;
switch(name) {
        case "burger" :
                category = "Fast Food";
                break;
        case "pizza" :
                category = "Fast Food";
                break;
        case "sandwich" :
                category = "Fast Food";
                break;
        case "yogurt" :
                category = "Dessert";
                break;
        case "milk tea" :
                category = "Dessert";
                break;
        case "mohinga" :
                category = "Burmese Food";
                break;
        case "sushi" :
                category = "Japanese Food";
                break;
        default :
                category = "unknown";
}
System.out.println(name + " is " + category);
sc.close();
```

- From Java 13, multiple values per case statement are allowed.

Example

```
switch(name) {
        case “burger”, “pizza”, “sandwich” :
                category = “Fast Food”;
                break;
        case “yogurt”, “milk tea” :
                category = “Dessert”;
                break;
        case “mohinga” :
                category = “Burmese Food”;
                break;
        case “sushi” :
                category = “Japanese Food”;
                break;
        default :
                category = “unknown”;
}
```

Switch Expression

- Java 12 introduced switch expression.
- It can be used as an expression just like other Java expression.
- It has no break statement.
- Colon (:) after each case statement has been replaced with a -> operator.

Example

```
var category =
        switch(name) {
                case "burger","pizza","sandwich" -> "Fast Food";
                case "yogurt","milk tea" -> "Dessert";
                case "mohinga"  -> "Burmese Food";
                case "sushi" -> "Japanese Food";
                default -> "unknown";
        };
```

- Java 13 introduced “**yield**” statement for switch expression to specify multiple statements.

Example

```
var category =
        switch(name) {
                case "burger","pizza","sandwich" -> {
                        if(name.equals("pizza"))
                                System.out.println(name + " is an Italian food.");
                        yield "Fast Food";
                }
                case "yogurt","milk tea" -> {yield "Dessert";}
                case "mohinga"  ->  "Burmese Food";
                case "sushi" -> "Japanese Food";
                default -> "unknown";
};
```

Colon (":") can be used instead of "->" symbol.

Example

```
String category = switch(name) {
                case "burger", "pizza", "sandwich" : {
                        if(name.equals("pizza"))
                                System.out.println(name + " is an Italian food.");
                                yield "Fast Food";
                        }
                case "yogurt", "milk tea" : yield "Dessert";
                case "mohinga"  :  yield "Burmese Food";
                case "sushi" : yield "Japanese Food";
                default : yield "unknown";
};
```

Looping Statement

- Looping statements instruct the program to repeat a series of statements as long as a specified condition is satisfied.
- Java provides the following types of loop –
  - while loop
  - do-while loop
  - for loop
  - foreach or Enhanced for loop

while loop

- Pre-test loop
- **Syntax :**

```
while(expression){
    [statements]
}
```

Example:

```
boolean condition =false;
int i=3;
while(!condition) {
        System.out.print(i + "\t");
        i--;
        if (i==0) condition=true;
}
```

do-while loop

- post-test version of while loop
- **Syntax :**

```
do{
    Statements;
} while  (expression);
```

Example:

```
boolean condition =false;
int i=3;
do
{
        System.out.print(i + "\t");
        i--;
        if (i==0) condition=true;
}while(!condition);
```

for loop

- Pre-test loop
- **Syntax :**

```
for (initializer;text-expression;update-expression){
    statement[s]
}
```

Example

```
for (var i = 0;i < 5; i++)
        System.out.print(i + "\t");
```

Enhanced for loop/ for-each loop

- It can be used to iterate over the elements of a collection without knowing the index of each element.
- You also need not know the size of the collection.

Syntax:

```
for(<datatype> <variable_name>:<collection_name>)
{
//Statements;
}
```

Example

```
String languages[] = {"Python", "Java", "Javascript"};
for(String lang : languages)
        System.out.println(lang);
```

Jump Statement

- Jump statements are used to interrupt the normal flow of a program.
- Types of jump statement – break, continue and return statements

break statement

- It breaks out of the 'while' and 'for' loops and the 'switch' statements.

Example

```
var a = 0;
while(true) {
        System.out.print(a + "\t");
        a++;
        if (a == 5)
                break;
}
```

continue statement

- When it executes, it moves the program counter immediately to the next iteration of the

Example

```
for(var x = 1;x < 43;x++)
{
        if(x%7 == 0)
                continue;
        System.out.println(x);
}
```

Assignment 1

Problem solving...

-User may enter the numbers till the user wants.

- at the end, display the count of positive, negative and zero entered.

E.g.

```
How many numbers you want to type : 5

Enter any number: 100
Enter any number: -1
Enter any number: 0
Enter any number: 100
Enter any number: 50
______________________________
Numbers of zero : 1
Numbers of positive number : 3
Numbers of negative number : 1
```

Assignment 2

- Take two user input data:
  - When do you go bed?
  - When do you get up?
- Find the number of sleep-time hours.
- If sleep-time is greater than or equal 5 hrs and less than or equal 8 hrs, display output "You take care your health well!".
- If sleep-time is less than 5 hrs, display output "You are very hardworking!".
- If sleep-time is greater than 8 hrs, display output "You are abnormal!".

The above process will continue taking user data until user enter "exit" word.

ARRAY

- An array is a data structure that holds a fixed number of values of a single type.
- It stores data with index-based.
- The array indexes start at zero.
- Length of the array specifies the number of elements present in the array.

One-dimensional Array

Syntax

```
data-type[] arrayName; // declare
arryName = new data-type[length]; // allocate memory
data-type[] arrayName = new data-type[length];
data-type[] arrayName = new data-type[]{val1,val2,val3};
data-type[] arrayName = {val1,val2,val3,..};
```

Example

```
int[] a = new int[3];     // initialze to default zero
int[] b = new int[] {3, 4, 5}; // Declare and set array element values.
int[] c = {3, 4, 5}; // Alternative syntax.
SomeClass[] d = new SomeClass[10]
int len = a.Length; // number of elements in a
```

- Java provides many useful static methods –
  - Sorting, copying, searching etc.

Example – Java's Useful Methods for array

```
int[] arr1 = {100,20,200,40,90};

// print array
for(int a : arr1)
        System.out.print(a + "  ");

int[] copyArr1 = Arrays.copyOf(arr1, arr1.length); // copy
System.out.println("\nAfter copying : " + Arrays.toString(copyArr1));

int[] copyArr2 = Arrays.copyOfRange(arr1, 1, 3); // copy range (start,end-1)
System.out.println("Aftter range copying : " + Arrays.toString(copyArr2));
```

```
// verify two array equal
System.out.println("arr1 == copyArr1 is " + Arrays.equals(arr1, copyArr1));
System.out.println("copyArr1 == copyArr2 is " + Arrays.equals(copyArr1, copyArr2));

Arrays.sort(arr1); // sort array
System.out.println("After sorting : " + Arrays.toString(arr1));

// searching (if exist, return its index.)
System.out.println("Is 90 in array : " + Arrays.binarySearch(arr1, 90));
// searching (if not exist, return negative value
System.out.println("Is 900 in array : " + Arrays.binarySearch(arr1, 900));

Arrays.fill(arr1, 7); // fill same value
System.out.println("After filling '7' : " + Arrays.toString(arr1));
```

Example – Converting array to Stream

```
int[] arr1 = {100,20,200,40,90};
System.out.println(Arrays.toString(arr1));

int total = Arrays.stream(arr1).sum(); // get total
System.out.println("Total : " + total);

System.out.println("Maximum value is " + Arrays.stream(arr1).max()); // get max value
System.out.println("Minimum value is " + Arrays.stream(arr1).min().getAsInt());
```

- In Java, when array argument is passed to a method, the reference of the array is passed to the method, not value.

Example

```
static void changeValue(int input[]) {
                    input[0] = 100;
}

int[] data = new int[5];
Arrays.fill(data, 7);

System.out.println("Before passing : " + Arrays.toString(data));
changeValue(data);
System.out.println("After passing : " + Arrays.toString(data));
```

Output

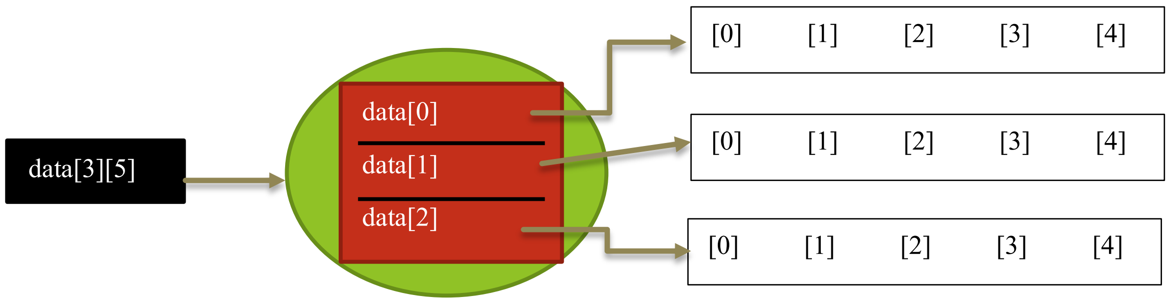
```
Before passing : [7, 7, 7, 7, 7]
After passing : [100, 7, 7, 7, 7]
```

Two-dimensional Array

- Two-dimensional array is a group of elements arranged into many rows and columns.
- In 2D array, member array must have the same number of columns.

Example

Student	Myanmar	English	Math	Chemistry	Physics
Jeon	65	73	86	84	59
Yuri	71	80	90	69	95
Cherry	75	75	99	96	99



How to declare 2D array

Syntax

```
data-type[][] arrayName; // declare
arryName = new data-type[row][col]; // allocate memory
data-type[][] arrayName = new data-type[row][col];
data-type[][] arrayName = new data-type[][]{{},{}};
data-type[][] arrayName = {{},{},…};
```

Example

```
int[][]  marks= {
                {65, 73, 86, 84,59},
                {71, 80, 90, 69,95},
                {75, 75, 99, 96, 99}
        };
// print with for loop
for(var r = 0;r < 3;r++) { // outer loop for row
        for(var c = 0;c < 5;c++) // inner loop for col
                System.out.print(marks[r][c] + "\t");
        System.out.println();
}
System.out.println("-------------------------");
// print with foreach loop
for(int[] row : marks) { // outer loop for row
        for(int col : row) // inner loop for col
                System.out.print(col + "\t");
        System.out.println();
}
System.out.println("-------------------------");
// print another way
System.out.println(Arrays.deepToString(marks));
```

Example

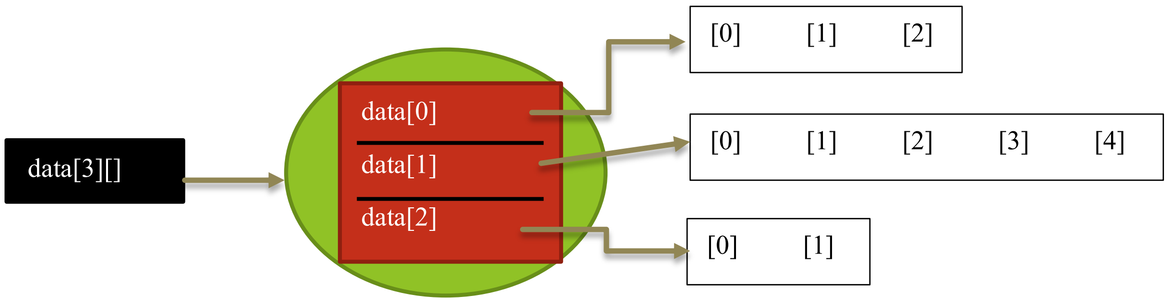
```
String[] names = {"Jeon", "Yuri", "Cherry"};
int[][]  marks= {
                {65, 73, 86, 84,59},
                {71, 80, 90, 69,95},
                {75, 75, 99, 96, 99}
        };
int row = marks.length; // no. of row
int col = marks[0].length; // no. of col
for(var r = 0;r < row;r++) {
        var total = 0;
        for(var c = 0;c < col;c++) {
                total = total + marks[r][c];
        }
        System.out.println("Total marks obtained by student " + names[r] + " : " + total);
        System.out.println("Average mark : " + (float)total/5);
}
```

Output

```
Total marks obtained by student Jeon : 367

Average mark : 73.4

Total marks obtained by student Yuri : 405

Average mark : 81.0

Total marks obtained by student Cherry : 444

Average mark : 88.8
```

Jagged Array

- Unlike two dimensional array, member array of the jagged array allows different numbers of columns.



Create & Initialize Jagged Array

Example

```
int[][] data = new int[2][];
data[0] = new int[]{1,2};
data[1] =new int[4];
int[][] data = {
        {1,2,3},
        {1},
        {1,2}
};
```

Example – Jagged Array with Looping

```
int[][] data = {
        {1,2,3},
        {1,2,3,4},
        {1,2}
};
System.out.println("______ with for looping_____");
for(var r = 0;r < data.length;r++) {
        for(var c = 0; c < data[r].length;c++)
                System.out.print(data[r][c]+"\t");
        System.out.println();
}
System.out.println("______with foreach looping____");
for(var row : data) {
        for(var c : row)
                System.out.print(c + "\t");
        System.out.println();
}
System.out.println(Arrays.deepToString(data));
```

Output

```
______ with for looping_____
1      2      3
1      2      3      4
1      2
______with foreach looping____
1      2      3
1      2      3      4
1      2
[[1, 2, 3], [1, 2, 3, 4], [1, 2]]
```

	lenovo	hp	samsung	acer	dell	asus
core i3	230.21	400.21	294.2	693.33	340.44	691.99
core i5	529.483	920.483	676.66	1594.659	783.012	1591.577
core i7	552.504	960.504	706.08	1663.992	817.056	1660.776
core i9	690.63	1200.63	882.6	2079.99	1021.32	2075.97

- The table shows the price details ($) data of computers.
- User must choose computer specification.
- According to the user's data, calculate the total amount and the result must be displayed with MMK.
- Currency exchange rate is entered by the user.

MATH, DATE-TIME API

Math Class

- The Math class provides helpful static methods for
  - Finding the maximum or minimum of two values
  - Arithmetic operations of two numbers (add, multiply etc.)
  - Rounding values
  - Logarithmic functions
  - And trigonometric functions (sin, cos, tan etc.)

Example

```
// some basic math operation
System.out.println("Absolute value of -7: " + Math.abs(-7));
System.out.println("Result of 3 power 4: " + Math.pow(3, 4));
System.out.println("Square root of 49: " + Math.sqrt(49));
System.out.println("Random value: " + Math.random());
System.out.println("Minimum value: " + Math.min(13, 31));
System.out.println("Maximum value: " + Math.max(13, 31));
System.out.println("100 + 300 = " + Math.addExact(100, 300));

// rounding and other functions
System.out.println("Round(3.6): " + Math.round(3.6));
System.out.println("Floor(3.6): " + Math.floor(3.6));
System.out.println("Ceil(3.2): " + Math.ceil(3.2));

// trigonometric & log functions
System.out.println("Sin(30): " + Math.sin(30));
System.out.println("Log10(10): " + Math.log10(10));
```

Date-Time API

- Java released the new Date-Time Api in Java 8.
- Different classes of the new Date-Time Api are under **java.time** pacakge.
- The most commonly used classes are LocalDate, LocalTime and LocalDateTime.
- All these classes are immutable classes and have private constructors.
- So, directly objects cannot be created and should use the now() or of() methods to create its instances.

LocalDate Class

- The LocalDate class uses the year-month-day structure to display the system date.

Some methods of LocalDate Class

now()	Return the current date.
getYear()	Return the current year
getMonth()	Return the current month
getDayOfWeek()	Return the current day of the week
getDayOfMonth()	Return the current day of the month
getDayOfYear()	Return the current day of the year
isLeapYear()	It is used to check if the year is a leap year
parse(CharSequence text)	Return the local date from a text string such as 2019-10-16.
plusXXX(value)	Return the local date with the specified numbers added.

Example

```
// create local date object
LocalDate now = LocalDate.now();
LocalDate locdate1 = LocalDate.of(2016, 10, 16);
LocalDate locdate2 = LocalDate.parse("2016-10-16");

System.out.println("Current date: " + now);
System.out.println("Yesterday date: " + now.minusDays(1));
System.out.println("Tomorrow date: " + now.plusDays(1));

System.out.println("Current Year : " + now.getYear());
System.out.println("Current Month : " + now.getMonth());
System.out.println("Current day of week: " + now.getDayOfWeek());
System.out.println("Current day of month: " + now.getDayOfMonth());

System.out.println(now + " is leap year : " + now.isLeapYear());
System.out.println(locdate1 + " is leap year : " + locdate1.isLeapYear());
System.out.println(locdate1 + " is same to " + locdate2 + ": " + locdate1.equals(locdate2));
```

LocalTime Class

- This class handles only with time in ISO format without date.

Example

```
// create local time object
LocalTime now = LocalTime.now();
LocalTime time1 = LocalTime.of(11, 03, 45);
LocalTime time2 = LocalTime.parse("04:30");

System.out.println("Current time :" + now);
System.out.println("Previous hour: " + now.minusHours(1).getHour());
System.out.println("Current hour: " + now.getHour());
System.out.println("Current minute: " + now.getMinute());
```

LocalDateTime Class

- LocalDateTime is used to represent a combination of date and time.

Example

```
// create LocalDateTime instance
LocalDateTime now = LocalDateTime.now();
LocalDateTime datetime = LocalDateTime.of(2015,Month.OCTOBER,20,04,30);

System.out.println("Current date time: " + now);
System.out.println("Current Year: " + now.getYear());
System.out.println("Current Month: " + now.getMonth());
System.out.println("Current day : " + now.getDayOfMonth());
System.out.println("Current hour: " + now.getHour());
System.out.println("Currrent minute: " + now.getMinute());
System.out.println("Current second: " + now.getSecond());
```

Using Period and Duration Classes

- Period class represents a quantity of time in terms of years, months and days.
- Duration class represents a quantity of time in terms of seconds and nanoseconds.
- They are widely used to obtain the difference between two dates or times.

Example

```
LocalDate startDate = LocalDate.parse("2019-09-25");
LocalDate endDate = LocalDate.parse("2021-10-31");

int month = Period.between(startDate, endDate).getMonths();
int days = Period.between(startDate, endDate).getDays();
System.out.println("No of months :" + month);
System.out.println("No of days :" + days);

LocalTime startTime = LocalTime.parse("11:30");
LocalTime endTime = LocalTime.parse("12:00");

long seconds = Duration.between(startTime, endTime).getSeconds();
System.out.println("No of seconds:" + seconds);
```

DateTimeFormatter Class

- The DateTimeFormatter class can be used to format date/time objects.
- This class provides various standard formatting options.
    1. Using predefined constants, such as ISO_LOCAL_DATE
    2. Using custom patterns, such as yyyy-MMM-dd
    3. Using localized styles, such as long or medium

1. Using Predefined Constants

```
// predefined patterns
DateTimeFormatter dateFormat = DateTimeFormatter.ISO_LOCAL_DATE;
DateTimeFormatter timeFormat = DateTimeFormatter.ISO_LOCAL_TIME;
DateTimeFormatter datetimeFormat = DateTimeFormatter.ISO_LOCAL_DATE_TIME;

LocalDate date = LocalDate.now();
LocalTime time = LocalTime.now();
LocalDateTime datetime = LocalDateTime.now();
```

```
System.out.println("ISO_LOCAL_DATE Format: " + date.format(dateFormat));
System.out.println("ISO_LOCAL_TIME Format: " + timeFormat.format(time));
System.out.println("ISO_LOCAL_DATE_TIME: " + datetime.format(datetimeFormat));
```

2. Using Custom Pattern

- We can define our own custom date time patterns using ofPattern() method.

Example

```
// custom patterns
DateTimeFormatter f1 = DateTimeFormatter.ofPattern("MMM dd yyyy");
DateTimeFormatter f2 = DateTimeFormatter.ofPattern("hh-m-s");
DateTimeFormatter f3 = DateTimeFormatter.ofPattern("MMMM dd yyyy 'at' h:m:ss");

// LocalDate
LocalDate now = LocalDate.now();
System.out.println("Default Format: " + now);
System.out.println("Custom Format: " + now.format(f1));

// LocalTime
LocalTime time = LocalTime.of(1, 30, 9);
System.out.println("Default Format: " + time);
System.out.println("Custom Format: " + time.format(f2));

// LocalDateTime
LocalDateTime datetime = LocalDateTime.now();
System.out.println("Default Format: " + datetime);
System.out.println("Custom Format: " + datetime.format(f3));
```

Output

```
Default Format: 2021-12-03
Custom Format: Dec 03 2021
Default Format: 01:30:09
Custom Format: 01-30-9
Default Format: 2021-12-03T10:22:19.078798900
Custom Format: December 03 2021 at 10:22:19
```

Some Date-Time Formatting Patterns

Pattern	Example
dd/MM/yy	16/10/21
dd MMM yyyy	16 Oct 2021
MMMM dd yyyy	October 16 2021
yyyy-MM-dd (ISO)	2021-10-16
E, MMM dd yyyy	Fri, Dec 03 2021
h:mm a	1:45 PM
hh:mm:ss	01:45:00
EEEE, MMM dd, yyyy hh:mm:ss a	Friday, Dec 03, 2021 10:36:41 AM
yyyy/MM/dd 'at' HH:mm:ss	2021/12/03 at 10:39:37

3. Using Localized styles

We can also format date time in human readable way using built-in FormatStyle eum.

Example

```
DateTimeFormatter format = DateTimeFormatter.ofLocalizedDate(FormatStyle.LONG);

LocalDate date = LocalDate.now();
System.out.println("_____ LocalDate _____");
System.out.format("LONG: %s\n",date.format(format));
System.out.format("FULL:
%s\n",DateTimeFormatter.ofLocalizedDate(FormatStyle.FULL).format(date));
System.out.format("MEDIUM:
%s\n",date.format(DateTimeFormatter.ofLocalizedDate(FormatStyle.MEDIUM)));
System.out.format("SHORT:
%s\n",date.format(DateTimeFormatter.ofLocalizedDate(FormatStyle.SHORT)));

LocalTime time = LocalTime.now();
System.out.println("________ LocalTime ______");
System.out.format("MEDIUM: %s\n",
time.format(DateTimeFormatter.ofLocalizedTime(FormatStyle.MEDIUM)));
System.out.format("SHORT: %s\n",
time.format(DateTimeFormatter.ofLocalizedTime(FormatStyle.SHORT)));

LocalDateTime datetime = LocalDateTime.now();
System.out.println("_____ LocalDateTme ________");
System.out.format("MEDIUM: %s\n",
datetime.format(DateTimeFormatter.ofLocalizedDateTime(FormatStyle.MEDIUM)));
System.out.format("MEDIUM: %s\n",
datetime.format(DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT)));
```

Assignment 1

- Get current date from your computer.
- If day of week is weekday, display output – "I have no time. I am studying Programming Language!"
- If day of week is weekend, display output – "Today is my holiday!"

E.g.

```
Output1

Fri, Dec 03 2021
************************
I have no time. I am studying Programming
Language.
```

```
Output2

Sun, Dec 05 2021
************************
Today is my holiday!
```

Hint: Use String [] weekdays = {"MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY"}

Assignment 2

- Assume that you have to develop a small food ordering system.
- Initially your system will display available menus and deliverable townships together with their durations.
- You need to choose menu and township to deliver. In your system, you can order in two types (1. order now? Or 2. Preorder?).
- If you make the first option, the app will display your order information detail with arrival time.
- If you make the second option, it will ask for your delivered date. If you enter the date, the app will display your information detail with arrival date.

- In this program, use these data –
  - String[] townships = {"AA","BB","CC","DD"};
  - int[] times = {15,30,10,45};

- String[] menus = {"Pizza", "Burger", "Milk Tea", "Spicy Noodle"};

Sample Output

Initially, the app will display like this. In this step, you need to choose your food and township.

```
****** Available Menu *******
1. Pizza
2. Burger
3. Milk Tea
4. Spicy Noodle
Please choose item: 2
```

```
***** Deliverable Township ******
1. AA (15 mins)
2. BB (30 mins)
3. CC (10 mins)
4. DD (45 mins)
Please choose township: 3
```

And order type will be chosen.

```
****** Order Type *****
1. Order Now?
2. Preorder?
Please choose 1 or 2:1
```

If you made order now option, it will display like this –

```
****** Your Order Information ******
Item Name: Burger
Your Address: CC
Duration: 10 mins
Arrival Time: 09:36:40 PM
******** Thank you for your ordering *******
```

(Current time – 09:26 PM)

If you made preorder option, it will display like this –

```
Enter deliver date (dd): 7
****** Your Order Information ******
Item Name: Burger
Your Address: CC
Arrival Date: Tuesday, Dec 07 2021
******** Thank you for your ordering *******
```

(Current date – 2021-12-03)

STRING & STRINGBUFFER

String

- Series of characters treated as single unit. May include letters, digits, etc.
- String class is immutable object. It cannot be edited but it can be assigned. It also has many constructors.
- A string's contents have been changed, what has really happened is that a new string instance has been created.
- Many operations can be manipulated on String such as copying, concatenating, searching, finding, splitting etc.
- String class provides many useful built-in methods.

Example – 1

```
String s1 = new String(new char[] {'Y','W','K'});
String s2 = new String("Java Programming");

System.out.println("s1: " + s1);
System.out.println("s2: " + s2);
System.out.println("Length of s2: " + s2.length());

// change case
System.out.println("lower case: " + s2.toLowerCase());
System.out.println("upper case: " + s2.toUpperCase());
```

Output

```
s1: YWK
s2: Java Programming
Length of s2: 16
lower case: java programming
upper case: JAVA PROGRAMMING
```

Example – 2

```
// substring
String substr = s2.substring(0, 4);
System.out.println("substring1: " + substr);
System.out.println("substring2: " + s2.substring(5));

// replacing
String result = s2.replace("Java", "PHP");
System.out.println("replace: " + result);

// concat
result = s2.concat(" Language");
System.out.println("concat: " + result);

// finding
System.out.println("Contain 'Java': " + s2.contains("Java"));
```

Output

```
substring1: Java
substring2: Programming
replace: PHP Programming
concat: Java Programming Language
Contain 'Java': true
Contain 'abc': false
End with 'ing': true
Start with 'abc': false
```

```
System.out.println("Contain 'abc': " + s2.contains("abc"));
System.out.println("End with 'ing': " + s2.endsWith("ing"));
System.out.println("Start with 'abc': " + s2.startsWith("abc"));
```

Example – 3

```
// locating
int index = s2.indexOf("a");
System.out.println("index of 'a': " + index);
System.out.println("last index of 'a': " + s2.lastIndexOf("a"));
System.out.println("index of 'abc': " + s2.indexOf("abc"));

String[] splits = s2.split(" ");
System.out.println("Split: " + Arrays.toString(splits));
char[] array = s2.toCharArray();
System.out.println("Char Array: " + Arrays.toString(array));

// converting string from any data type
String rs = String.valueOf(array);
System.out.println("From char array: " + rs);
rs = String.valueOf(1000);
System.out.println("From int: " + rs);
rs = String.valueOf(true);
System.out.println("From boolean: " + rs);
```

Output

```
index of 'a': 1
last index of 'a': 10
index of 'abc': -1
Split: [Java, Programming]
Char Array: [J, a, v, a,  , P, r,
o, g, r, a, m, m, i, n, g]
From char array: Java Programming
From int: 1000
From boolean: true
s1 == s2: false
s2 == s3: false
s2 == s3: true
```

StringBuffer

- It is mutable or dynamically changeable string class without creating new object.
- Many StringBuilder class methods are the same as those of the String class.

Example

```
StringBuffer buffer = new StringBuffer("Java Programming");
buffer.append(" language");
System.out.println("append: " + buffer);
buffer.insert(5, "is a ");
System.out.println("insert: " + buffer);

buffer.replace(0, 4, "PHP");
System.out.println("repalce: " + buffer);
System.out.println("indexof: " + buffer.indexOf("language"));
System.out.println("substring: " + buffer.substring(4));
System.out.println("No of character: " + buffer.length());

buffer.delete(5,10 );
System.out.println("delete: " + buffer);
```

Output

```
append: Java Programming language
insert: Java is a Programming language
repalce: PHP is a Programming language
indexof: 21
substring: is a Programming language
No of character: 29
delete: PHP irogramming language
```

Assignment 1

String paragraph= "NLP techniques are becoming widely popular scientific research areas as well as Information Technology industry. Language technology together with Information Technology can enhance the lives of people with different capabilities. This system implements voice command mobile phone dialer application. The strength of the system is that it can make phone call to the contact name written in either English scripts or Myanmar scripts."

For this paragraph, find the number of sentences and total words.

Assignment 2

- User gives one sentence.
- You verify whether it is question sentence or not.
- If it is question sentence, find the first word of the sent. And verify again the question sentence is whether simple present tense or not.

Assignment 3

- Input data is user's NRCNO.

  (e.g nrcno=" 12/mayana(N)123456" or nrcno="5/yamatha(naing)789412
- Display output like this:

  Township: mayana

  Number: 123456

EXCEPTION HANDLING

What is Exception & Exception Handling?

- Exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions.

Example

```
Scanner sc = new Scanner(System.in);
System.out.print("Enter number 1 : ");
int num1 = sc.nextInt();
System.out.print("Enter number 2 :");
int num2 = sc.nextInt();
int result = num1 / num2;
System.out.println("Result is " + result);
sc.close();
```

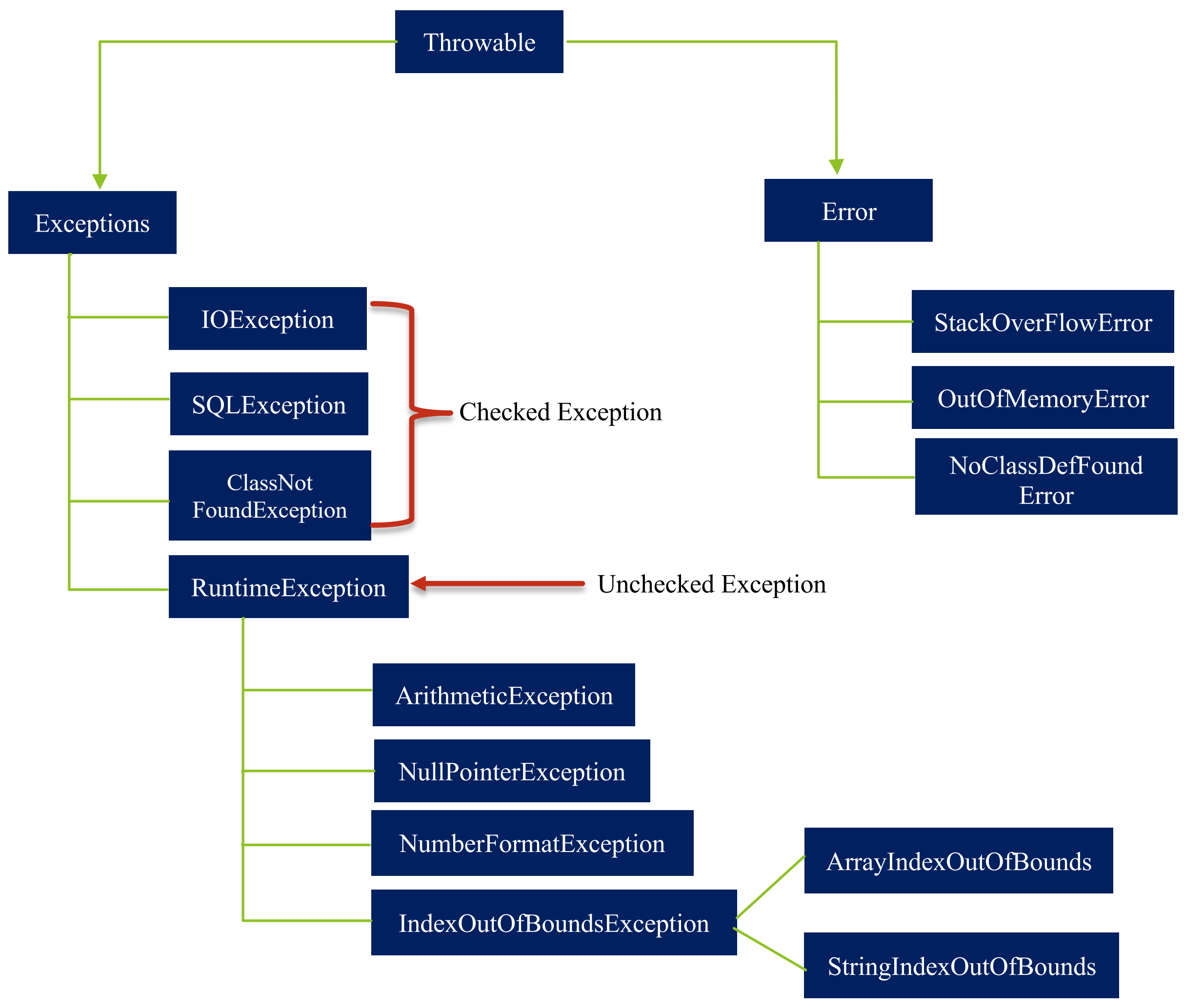
Output

```
Enter number 1 : 10
Enter number 2 :0
Exception in thread "main"
java.lang.ArithmeticException: / by zero
      at chpt1.Demo.main(Demo.java:32)
```

- When the above condition occurred while the program executing, Java create an exception object and throws it (i.e. inform that an error has occurred).
- If programmer has not handled this exception, the program execution terminates and the system generated error message is shown to the user.
- The message is not user friendly so a user will not be able to understand what went wrong.
- Therefore we need to handle exceptions. Java provides many predefined exception classes.

Hierarchy of Java Exception Classes

The java.lang.Throwable class is the root class of the Java Exception classes. It has two subclasses – Exception and Error.



- There are three main categories of exceptional conditions:
  - Checked exceptions
  - Unchecked exceptions / Runtime exceptions
  - Errors

Checked Exception

- Checked exceptions are exceptions that the Java compiler requires us to handle.
- Programmers have to either declaratively throw the exception up the **call stack**, or handle it themselves.

Example

```
BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

System.out.print("Enter your salary: ");
int salary = Integer.parseInt(reader.readLine());
System.out.println("Your salary is: " + salary);
```

This is a checked exception. If you handle/declare it, you will get compilation error.

Unchecked Exception/Runtime Exception

- Unchecked exceptions are exceptions that the Java compiler does not require us to handle.
- Runtime exception and it's all subclasses are unchecked exceptions.
- Compiler will never force you to catch such exception or declare it.

Example

```
Scanner scanner = null;
System.out.println("Enter name: ");
String name = scanner.nextLine();
System.out.println("Your name is " + name);
scanner.close();
```

Output

```
Enter name:
Exception in thread "main" java.lang.NullPointerException: Cannot
invoke "java.util.Scanner.nextLine()" because "scanner" is null
```

Error

- Errors represent serious and usually irrecoverable conditions.
- Some of errors are OutOfMemoryError, StackOverFlowError etc.
- Even though they don't extend RuntimeException, they are also unchecked.

Example

```
static void display(int num) {
        System.out.println("number is " + num);
        display(num);
}
public static void main(String[] args)  {
        display(100);
}
```

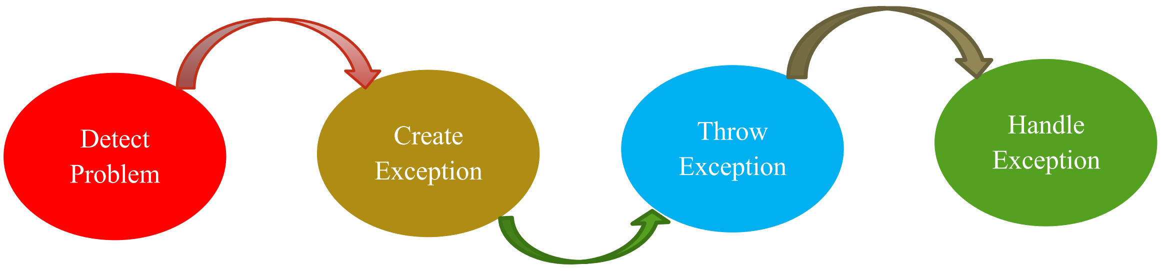
- Your program will get these error message.

Output

```
number is 100
number is 100
number is 100
Exception in thread "main" java.lang.StackOverflowError
        at java.base/java.nio.CharBuffer.<init>(CharBuffer.java:286)……….
```

How to Handle Exception?

- Java has already provided the exception handling mechanism.



- Java exception handling is managed via five keywords: **try, catch, throw, throws, and finally**.

Keyword	Description
try	It is used to monitor for exception occurring. We can't use try block alone. The try block must be followed by either **catch** or **finally**.
catch	It is used to handle the exception. The catch block is executed only when the exception raised inside the try block.
finally	It is executed whether an exception is handled or not.
throw	It is used to manually throw an exception from a method or any block of code. It is mainly used to throw custom exceptions.
throws	It is used to declare exceptions. It is always used with method signature.

Try Block

- Try block identifies a block of statements within which an exception might be thrown.
- The three possible forms of try block are as follows
  - **try-catch** – A try block is always followed by one or more catch blocks.
  - **try-finally** – A try block followed by a finally block.
  - **try-catch-finally** – A try block followed by one or more catch blocks followed by a finally block.

1. try-catch Block

- The simplest and most basic way to handle exceptions is to use the try – catch block.
- If exception is occurred in try block, then the rest of the try block doesn't execute and control passes to the corresponding catch block.
- If the corresponding catch block is not found, default exception handling will be executed.

General Syntax –

```
try {
//code that may raise exception
}catch (ExceptionType1 exOb) {
// code block executed if an exception is thrown
}catch (ExceptionType2 exOb) {
// code block executed if an exception is thrown
}
// ...
```

Example

```
try {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter number1: ");
        int num1 = Integer.parseInt(sc.nextLine());
        /*if exception is raised by about statement,
        the remaining statements will never execute.*/
        System.out.print("Enter number2: ");
        int num2 = Integer.parseInt(sc.nextLine());
        int result = num1/num2;
        System.out.println("Result is " + result);
        sc.close();
} catch (ArithmeticException e) {
        System.err.println(e.getMessage());
}
```

Output 1

```
Enter number1: abc
User input is not number
Outside try-catch clause
```

Output 2

```
Enter number1: 100
Enter number2: 0
/ by zero
Outside try-catch clause
```

Output 3

```
Enter number1: 100
Enter number2: 5
Result is 20
Outside try-catch clause
```

```
catch (NumberFormatException e) {
          System.err.println("User input is not number");
}
// rest program will be executed
System.out.println("Outside try-catch clause");
```

- When exception is handled with multiple catch blocks, the order of the catch blocks is important.

Example

```
try {
     // statements
} catch (Exception e) {
    // this block will catch all exceptions
}
catch (NumberFormatException e) {
  // this block will not be executed
}
```

- To handle multiple Java exceptions with the same logic, you can list them all inside a single catch block.

Example

```
try {
   // statements
 } catch (ArithmeticException  | NumberFormatException e) {
   // do something
 }
```

- If the throwing exception is not handled with the corresponding catch block, the default exception handling mechanism is used.

Example

```
try {
        int[] numbers = {100,200,300};
        System.out.println(numbers[3]);
}
// not  appropriate handler
catch (NumberFormatException e) {
        System.out.println(e.getMessage());
}
// rest program will not execute
System.out.println("Outside try-catch block");
```

Output

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException:
Index 3 out of bounds for length 3
```

2. try-finally Block

- A finally block is usually used for cleaning up resources such as closing connections, stream, etc.
- It is always executed whether an exception occurs within a try block or not.
- It will be executed even if the return statement is used in the try block.

Syntax

```
try {
//code that may raise exception
}
finally{
// this block is always executed
}
// ...
```

Example

```
Scanner sc = new Scanner(System.in);
try {
        System.out.print("Enter salary: ");
        int salary = Integer.parseInt(sc.nextLine());
        if(salary == 0)
                return;
        System.out.println("Your salary is " + salary);
}
finally {
        sc.close();
        // It is always executed
        System.out.println("finally block executed");
```

Output 1

```
Enter salary: 1000000
Your salary is 1000000
finally block executed
Outside try-finally block
```

Output 2

```
Enter salary: abc
finally block executed
Exception in thread "main"
java.lang.NumberFormatException
: For input string: "abc"
```

Output 3

```
Enter salary: 0
finally block executed
```

```
}
// if exception is occurred, it will not executed.
System.out.println("Outside try-finally block");
```

3. try-catch-finally Block

- If an exception is thrown with matching catch block, the catch block is executed, and then finally block is executed.

General Syntax

```
try {
//code that may raise exception
}catch (ExceptionType1 exOb) {
// code block executed if an exception is thrown
}finally {
// It is always executed!
}
// ...
```

Example

```
try {
        String name = null;
        System.out.println("Length: " + name.length());
} catch (NullPointerException e) {
        System.err.println("It does not allocate");
}
finally {
        // always execute
        System.out.println("finally block executed");
}
// rest program will be executed
System.out.println("Outside try-catch-finally block");
```

Output

```
It does not allocate
finally block executed
Outside try-catch-finally block
```

try-with-resource Statement

- The try-with-resources approach is useful for handling resources, because the program closes the used resources automatically.

Example

```
try(Scanner sc = new Scanner(System.in)){
        System.out.println("Enter your name: ");
        String name = sc.nextLine();
        // do not do this
        sc.close();
}catch (Exception e) {
        // TODO: handle exception
}
```

Throwing Exception

- In Java exception handling mechanism, JVM firstly identify the exception, create its object and automatically throws that object.
- If you want to manually throw an exception, "**throw**" keyword is provided in Java.
- The "**throw**" keyword is followed by an instance of Exception class and throws is followed by exception class names.

General syntax

```
throw new ExceptionName();
(or)
ExceptionClass obj = new ExceptionClass();
throw obj;
```

- If checked exceptions are manually thrown, you have to handle it by using try-catch block or using "**throws**" keyword in the signature of the method so that callers will know to handle it.
- If unchecked exceptions are thrown, exception handling is optional.

Example – using try-catch

```
try {
        int[] numbers = { 100,200,300};
        int index = 5;
        if(index >= numbers.length)
                throw new ArrayIndexOutOfBoundsException("Out of Range");
} catch (ArrayIndexOutOfBoundsException e) {
        System.err.println(e);
        System.err.println(e.getMessage());
}
```

Example – using "throws" keyword in the signature of the method

```
static void checkMark(int mark) {
        if(mark < 0 || mark > 100) {
                ArithmeticException ex = new ArithmeticException("Invalid mark");
                throw ex;
        }
}
public static void main(String[] args)  {

        try {
                checkMark(120);
        } catch (ArithmeticException e) {
                System.err.println(e.getMessage());
        }
}
```

Exception Propagation

- An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method.

Example

```
public class ExceptionPropagate {

        static void changeValue(int input[]) {
                input[0] = 100;
        }
        static void test1() {
                System.out.println(100/0);
        }
        static void test2() {
                test1();
        }
        static void test3() {
                test2();
        }
        public static void main(String[] args)  {
                test3();
        }
}
```

Output

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at chpt1.ExceptionPropagate.test1(ExceptionPropagate.java:26)
	at chpt1.ExceptionPropagate.test2(ExceptionPropagate.java:29)
	at chpt1.ExceptionPropagate.test3(ExceptionPropagate.java:32)
	at chpt1.ExceptionPropagate.main(ExceptionPropagate.java:36)
```

User-defined Exception

- You can also define your own exception.
- User-defined exception classes must inherit from either Exception class or one of its standard derived classes.

Example

```
class InvalidAgeException extends Exception{
	public InvalidAgeException() {

	}
	public InvalidAgeException(String msg) {
		super(msg);
	}
}
public class Testing {
	static void validateAge(int age) throws InvalidAgeException {
		if(age < 18)
			throw new InvalidAgeException("age is not valid to smoke");
		else
			System.out.println("You can smoke!");
	}
	public static void main(String[] args)  {
		try (Scanner sc = new Scanner(System.in)){
			System.out.print("Enter your age: ");
			int age = sc.nextInt();
			validateAge(age);
		} catch (InvalidAgeException e) {
			System.err.println("Error Message: " + e.getMessage());
		}
	}
}
```

Assignment 1

- Write a program which will accept an array of numbers (type String) from constructor and then parse to integer.
- Then find the average, max and min by using the appropriate Exception class.

(Exceptions are ArithmeticException, ArrayIndexOutOfBoundException, IllegalArgumentException)

Assignment 2

- Write a program that get a String NRC and displays the division/state, city and number.

  E.g.

  NRC = 12/sakhana(naing)123456

  Outputs:

  Division/state = LL

  City = sakhana

  Number = 123456

- Use String[] division = {“AA”, “BB”, “CC”, “DD”, “EE”, “FF”, “GG”, “HH”, “II”, “JJ”, “KK”, “LL”, “MM”, “NN”};
- Use Exception ArrayIndexOutOfBound and show Message when the input NRC is invalid.

Assignment 3

- Write a new user-defined Exception which will be used in student mark entry system.
- The Exception will prompt when a user enters the mark not between 0 and 100 (inclusive).
- Then write the user program which will test this new exception.

THANK YOU