Java Introduction

- Program: set of instruction that performs some specific task
- program can be written using any programming language
- programs are accept input, process it and produce the output

Java Program

- 1995 by James Gosling and his team
- initially they were named as "Green Talk" -> "OAK" --> "Java"
- Latest Version Java18 release on Mar 2022

Features of Java

* Portable - Write Once Run Anywhere (WORA) Platform Independent java ----> bytecode
* Object Oriented - Realtime Entity
* Multithreaded - allows to run more than one thread at a time
* Robust - Automatic Memory Management (Automatic Garbage Collection)
* Strongly Typed
* Standard Libraries
* Security

Java Editions

- JavaSE - Java Standard Edition
- JavaEE - Java Enterprise Edition
- JavaME - Java Mobile Edition

JDK-JRE-JVM

Source code ----> compiler ----> bytecode ----> JVM
(.Java) (.class)

JDK- Java Development Kit

* To develop an application
* It includes JRE, Java Compiler, Java Interpreter, javap and other

JRE: Java Runtime Environment

* It provides support to execute the Java application
*It includes JVM, and other supporting files like class file

JVM: Java Virtual Machine

* Responsible for execution of Java code

Anatomy

- **Methods:** small piece of code that perform some specific task
- **Class:**

    * It is a container
    * Collection of data members and member functions
    * Always start with capital case

- **Package:**
    - stores similar type of classes and interfaces
    - avoid naming collision
    - **Types**
        * Builtin Package or Library Package (io,sql,util, lang)
        * User Defined Package

Variables

- name for memory location
- can change the value of variable during the execution of program
- **Rules:**
    - It is case sensitive sum, SUM, Sum are different variables
    - begins with letter, $ or _ (underscore) sum, $sum, _sum
    - always better to start with letter
    - space cannot be provided between variable name
    - may be the combination of digits
    - It should not be a keyword
    - better to start with small case
    - better to use short and meaningful name
- **Types**
    - Local variable:
        * Declared inside the method
        * Do not hold default value
        * Access specifiers are not allowed
        * Scope of variable is within method
        * Stored in stack memory
    - Instance variable:
        * Declared inside the class but outside the method
        * Hold default value
        * May use access specifiers
        * Scope of variable is within a class
        * Stored in heap memory
    - Static variable
        * Stored in heap memory

Keywords

- reserved words
- 51 keywords in java

Constants

- value do not change during the execution of program
- the value remains same for the entire program

- final keyword used to represent the constant
    final int MAX=18;
- capital case for constant name
    final int AGE=18;
    final int MAX AGE=18;

Data Types

- it is a kind of value that a variable or constant can hold
    Ex: int a; char c; -

Types:

* **Primitive**
    - *Numeric*
        = integer
            * byte (1 byte)
            * short (2 bytes)
            * int (4 bytes)
            * long (8 bytes)
        = floating point
            * float (4 bytes)
            * double (8 bytes)
    - *Character* (2 bytes)
    - *Boolean* (1 bit)
* **Nonprimitive**
    - array
    - string
    - class

Type Conversion:

- converting one type of data into another type of data
- **Types**
    * **Widening Conversion:** byte --> short --> int --> long
        - implicit conversion byte b=20; int n=b;
    * **Narrowing Conversion:** long --> int ---> short --> byte
        - Explicit conversion double d=21.5; int a=(int)d;

- **Conversion of string to primitive type:**

```
String s="20";
int n=Integer.parseInt(s);

int m=123;
String s=String.value0f(m);
```

Automatic Type Promotion

- all byte and short values are automatically promoted to int

```
byte b1=53;
byte b2=120;
int n=bl+b2;
```

- if one operand is long then whole expression is promoted to long
- if one operand is float then whole expression is promoted to float
- if one operand is double then the result is also double

Operators

- **Arithmetic:** used in mathematical calculations (+, -, *, /, %)
- **Relational:**
  - identify relationship between two operands (<, <=, >, >=, ==, !=)
  - outcome of this relationship is boolean
- **Logical:** (&&, ||, !)
  - Can perform more than one operation
  - Can be performed on boolean values
- **Assignment:** (=, +=, -=, %=, /=)
  - Assign RHS value to LHS
- **Unary:** (+, -, ++, --)
  - Only one operand is used with operator
  - Pre increment or post increment
- **Bitwise:** (!, &, |, ~, ^)

Conditional Statements

- Decision making statements
- **If:** handles only true condition
- **If else:** handles both true and false conditions
- **If else if:** handles multiple conditions
- **Nested if:** condition within another condition

Switch case

Multiway branching statement
Break: exit the loop
Continue: stops the current iteration

Looping statements

- Execute same set of statement repeatedly
- **for:** entry conditional checking
- **while:** entry conditional checking
- **do while:** exit conditional checking

String

- sequence of character
- immutable in nature i.e., once string object is created, it cannot be modified
- **Syntax**
    - - using new keyword
      String s1=new String("Hello");
      String s2=new String("Hello"); // even though s1 and s2 have same value, separate memory will be created when we create a string using new keyword

    - - using String literal
      String s3="Hello";
      String s4="Hello"; // s3 and s4 both are point to the same memory location as both are created using string literal

      String s5="hello";

    - S1.equals(s2); // true
      s1.equals(s4); // true
      s3.equals(s4); //true

    - s1==s2; //false
      s1==s4; //false
      s3==s4; //true

    - **equals()** method compares the content or value of the string
    - == compares the memory location

String Functions:

- concat()
- equals()
- equalsIgnoreCase()
- length()
- indexOf(char)
- index0f(string)
- lastIndex0f(char)
- replace()
  split()
- startsWith()
- subString()
- toLowerCase()
- toUpperCase()

StringBuffer:

- sequence of character
- It is Mutable
- append()
- threadsafe

```
StringBuffer sBuffer=new StringBuffer("Hello");
sBuffer.append(" Kumar");
```

StringBuilder:

- sequence of character
- It is Mutable
- append()
- It is not ThreadSafe

```
StringBuilder sBuilder=new StringBuilder("Hello");
sBuilder.append(" kumar");
```

Array

Stores similar types of data in continuous memory location

```
int marks[]= new int[100];
```

Types:

- One dimensional Array

  ```
  int marks[]= new int[100];
  ```

- Multi dimensional Array

  ```
  Int marks[][]= new int[3][3];
  ```

Wrapper classes

- Provides mechanism to convert primitive type data into object and vice versa
- **Autoboxing:** convert primitive type into object types

  ```
  int a= 28;
  Integer i= a;
  ```

- **Unboxing:** convers object types into primitive type

  ```
  Integer a= new Integer(30);
  int i= a;
  ```

- **Uses:**
  - Java collection framework
  - Objects are used to change the original value

Primitive type	Wrapper classes
int	Integer
byte	Byte
short	Short
long	Long
float	Float
double	Double

Date and Time

Helps to handle date and time

Local date and time

java.time package need to be included

- **LocalDate:** represents date (yyyy-mm-dd)
- **LocaTime:** represents time (hh-mm-ss-ns)
- **LocalDateTime:** represents both (yyyy-mm-dd hh-mm-ss-ns)

```
LocalDateobj= LocalDate.now();

obj.getDay0fMonth();
obj.getMonth();
obj.getYear();

obj.isLeapYear();

obj.plusDays(3);
obj.minusDays(2);

LocalTime obj1=LocalTime.now();
```

Constructor

- It has special type of method
- Method name should be same as that of the class name
- Invoked while object is being created
- Do not have return type

- **Types**
  - **Default:**

```
public class Employee{
        private int empId;
        private string empName;
```

```
        // default constructor
        public Employee(){
        ……
        }
        Public void display(){
                ……
        }
        public static void main(Strinh[] args){
                Employee emp= new Employee();
                emp.display();
        }
}
```

- **Parameterized:**

```
        public class Employee{
                private int empId;
                private string empName;

                // parameterized constructor
                public Employee(int empId, String empName){
                        this.empId= empId;
                        this.empName= empName;
                }
                Public void display(){
                        ……
                }
                public static void main(Strinh[] args){
                        Employee emp= new Employee(101, “ABC”);
                        emp.display();
                }
        }
```