**First Program**

1. **Better create a separate folder to save all the programs which we are going to write.**

**First.java**

class First

{

    public static void main(String[] args)

    {

        System.out.println("Welcome");

    }

}

Output:

Javac First.java

Java First

Welcome

**Class First{**

**Public static void main(String[] args){**

**System.out.println(“HEllo World!”);**

**}**

**}**

**In a java program how many classes we can write?**

|  |
| --- |
| **Any no.of classes** |

**At the time of run program which .class file name we should give.**

|  |
| --- |
| We have to give the .class file name which contains main method.  Ex: java <main method .class file name> |

**Second.java**

class One

{

    public static void main(String[] args) {

        System.out.println("Welcome One");

    }

}

class Two

{

    public static void main(String[] args) {

        System.out.println("Welcome Two");

    }

}

public class Second

{

    public static void main(String[] args) {

        System.out.println("Welcome Second");

    }

}

Output:

Java One

Welcome One

Java Two

Welcome Two

Java Second

Welcome Second

**In a java program how many public classes we can write?**

|  |
| --- |
| Only one public class |

**Note:** if you have written a public class in a program, the public class name and program name must be same.

**How many methods are there to print the output?**

1. Println()
2. Print()
3. Printf()
4. Format()

|  |  |
| --- | --- |
| **Println** | **Print** |
| 1. It prints the data and moves the cursor to the new line. | 1. It prints the data and doesn’t moves the cursor to the new line. |
| 1. We can pass either zero or one argument to the println() method | 1. We can pass only one argument to the print() method. Remember we can’t pass zero arguments to the print() method. |
|  |  |

**Example on print() method usage**

public class Third {

    public static void main(String[] args) {

        System.out.print("Madhu");

        System.out.print("From");

        System.out.print("Vijayawada");

        //MadhuFromVijayawada

    }

}

Output:

MadhuFromVijayawada

**Backslash Codes or Escape Sequences or Escape Characters**

|  |
| --- |
| We can use escape sequences only with in a string literal |

|  |  |
| --- | --- |
| **Escape sequence** | **Description** |
| \n | New line |
| \t | Tab space |
| \b | Backspace |
| \r | Carriage return |
| \\ | It puts the single back slash |
| \’ | It is used to put the single quotation in a string literal |
| \” | It is used to put the double quotation with in a string. |

**Using \n(escape sequence) in print() method**

public class Third {

    public static void main(String[] args) {

        System.out.print("Madhu\n");

        System.out.print("From\n");

        System.out.print("Vijayawada\n");

        //Madhu

        //From

        //Vijayawada

        //

    }

}

Output:

Madhu

From

Vijayawada

**Example on usage of println() method**

public class Third {

    public static void main(String[] args) {

        System.out.println("Madhu");

        System.out.println("From");

        System.out.println("Vijayawada");

    }

}

Output:

Madhu

From

Vijayawada

**What is a string concatenation?**

|  |
| --- |
| 1. Appending anything to the string   Ex: "a="+100 result is another string i.e "a=100"  We can perform this operation by using concatenation operator called +. |

**Example on println() method and concatenation operator**

public class Third {

    public static void main(String[] args) {

        int a=10;

        int b=20;

        int c=a+b;

        System.out.println("a="+a);

        System.out.println("b="+b);

        System.out.println("c="+c);

    }

}

Output:

a=10

b=20

c=30

**Calling println() method without passing arguments**

public class Third {

    public static void main(String[] args) {

        System.out.print("Hi Friends");

        System.out.println();

        System.out.print("Hello Friends");

        System.out.println();

        System.out.print("Bye Friends");

        //System.out.print(); Error

    }

}

Output:

Hi Friends

Hello Friends

Bye Friends

**Formatting methods**

**What are formatting () methods:**

In jdk1.5 version two new methods are introduced, to replace the print() and println() methods, and these methods are called as formatting methods.

Those are:

* 1. Printf()
  2. Format()

These methods are existed in the java.io.PrintStream class and these methods, format and printf, are equivalent to one another.

**What is the difference between printf() and format() method?**

Actually **printf()** method is internally calls the **format()** method, that is actual code is existed in the format method. The printf() method is just provided for our convenience.

**Some of the converters which are used in format method**

|  |  |  |
| --- | --- | --- |
| Converter | Flag | Explanation |
| D |  | A decimal integer |
| F |  | A float |
| N |  | A new line character appropriate to the platform running the application. You shold always use %n rather than \n |
| tB |  | Full name of month |
| Tb |  | Short name of month |
| Td |  | day of month, if it is single digit, it puts zero before that number |
| Te |  | Day of month, it will not put zero before single digit. |
| Ty |  | 2 digit year |
| tY |  | 4 digit year |
| Tl |  | Hour in 12 hours clock |
| tM |  | Minutes in 2 digits, with leading zeros as necessary |
| tS |  | Seconds |
| Tp |  | Am/pm |
| Tm |  | Month in 2 digits, with leading zeros as necessary |
| tD |  | Date as %tm%td%ty |
|  | 08 | 8 characters in width, with leading zeros as necessary |
|  | + | Includes sign, whether positive or negative |
|  | , | Includes Locale-specific grouping characters. |
|  | - | Left-justified |
|  | .3 | Three places after decimal part |
|  | 10.3 | Ten characters in width, right justified, with three places, after decimal point |

**Printf():**

What is the first argument we can pass to a printf() or format() method?

|  |
| --- |
| We have to pass a string literal as a first argument |

Example on usage of printf() method

public class FormattingMethods

{

    public static void main(String[] args)

    {

        int a=10,b=2,c=0;

        c=a+b;

        System.out.printf("Demo on formatted method.....%n");

        //%d -> converter for decimal value

        System.out.printf("a=%d%n",a);

        System.out.printf("%d + %d = %d",a,b,c);

    }

}

Output:

Demo on formatted method.....

a=10

10 + 2 = 12

**Aligning output using printf() method**

public class FormattingMethods

{

    public static void main(String[] args)

    {

        int a=10,b=2,c=0;

        c=a+b;

        System.out.printf("Demo on formatted method.....%n");

        //%d -> converter for decimal value

        System.out.printf("a=%d%n",a);

        System.out.printf("%-5d + %-5d = %5d",a,b,c);

    }

}

Output:

Demo on formatted method.....

a=10

10 + 2 = 12

**Java Tokens**

**What is token?**

|  |
| --- |
| Each and every individual unit in a program is called as token  Ex: int a=100; //variable declaration and initialization   * + 1. Keyword     2. Identifier     3. Operator     4. Literal or constant or value     5. Separator     6. Special symbol     7. Comments |

**Keywords:**

**What are keyword?**

|  |
| --- |
| 1. Keywords are pre-defined words, which are comes along with java. And provided by inventors of Java. 2. Every keyword has a special meaning and can be used for a specific purpose. 3. For example, to write a for loop we have to use a keyword called for 4. To write condition in a program we have to use if and else keyword 5. We use int keyword to declare a variable   Etc.. |

**List of keywords**

**List of Java keywords**

1. **\_**

**Added in Java 9, the underscore has become a keyword and cannot be used as a variable name anymore.**

**Yes**

**/tmp/Ngz25IYAAC/HelloWorld.java:7: error: as of release 9, '\_' is a keyword, and may not be used as an identifier**

**int \_=100;**

**^**

**ERROR!**

**Note: in jdk 22 version I am not getting error after usage of \_ as an identifier**

public class EscapeSeq

{

    public static void main(String[] args) {

        int \_=1000;

       // System.err.println(\_);

    }

}

Note: we are not getting errors while compilation of this program

1. **abstract**

**A method with no definition must be declared as abstract and the class containing it must be declared as abstract. Abstract classes cannot be instantiated. Abstract methods must be implemented in the sub classes. The abstract keyword cannot be used with variables or constructors. Note that an abstract class isn't required to have an abstract method at all.**

1. **assert (added in J2SE 1.4)[4]**

**Assert describes a predicate (a true–false statement) placed in a Java program to indicate that the developer thinks that the predicate is always true at that place. If an assertion evaluates to false at run-time, an assertion failure results, which typically causes execution to abort. Assertions are disabled at runtime by default, but can be enabled through a command-line option or programmatically through a method on the class loader.**

1. **boolean**

**Defines a boolean variable for the values "true" or "false" only. By default, the value of boolean primitive type is false. This keyword is also used to declare that a method returns a value of the primitive type boolean.**

1. **break**

**Used to end the execution in the current loop body.**

**Used to break out of a switch block.**

1. **byte**

**The byte keyword is used to declare a field that can hold an 8-bit signed two's complement integer.[5][6] This keyword is also used to declare that a method returns a value of the primitive type byte.[7][8]**

1. **case**

**A statement in the switch block can be labeled with one or more case or default labels. The switch statement evaluates its expression, then executes all statements that follow the matching case label; see switch.[9][10]**

1. **catch**

**Used in conjunction with a try block and an optional finally block. The statements in the catch block specify what to do if a specific type of exception is thrown by the try block.**

1. **char**

**Defines a character variable capable of holding any character of the java source file's character set.**

1. **class**

**A type that defines the implementation of a particular kind of object. A class definition defines instance and class fields, methods, and inner classes as well as specifying the interfaces the class implements and the immediate superclass of the class. If the superclass is not explicitly specified, the superclass is implicitly Object. The class keyword can also be used in the form Class.class to get a Class object without needing an instance of that class. For example, String.class can be used instead of doing new String().getClass().**

1. **continue**

**Used to resume program execution at the end of the current loop body. If followed by a label, continue resumes execution at the end of the enclosing labeled loop body.**

1. **default**

**The default keyword can optionally be used in a switch statement to label a block of statements to be executed if no case matches the specified value; see switch.[9][10] Alternatively, the default keyword can also be used to declare default values in a Java annotation. From Java 8 onwards, the default keyword can be used to allow an interface to provide an implementation of a method.**

1. **do**

**The do keyword is used in conjunction with while to create a do-while loop, which executes a block of statements associated with the loop and then tests a boolean expression associated with the while. If the expression evaluates to true, the block is executed again; this continues until the expression evaluates to false.[11][12]**

1. **double**

**The double keyword is used to declare a variable that can hold a 64-bit double precision IEEE 754 floating-point number.[5][6] This keyword is also used to declare that a method returns a value of the primitive type double.[7][8]**

1. **else**

**The else keyword is used in conjunction with if to create an if-else statement, which tests a boolean expression; if the expression evaluates to true, the block of statements associated with the if are evaluated; if it evaluates to false, the block of statements associated with the else are evaluated.[13][14]**

1. **enum (added in J2SE 5.0)[4]**

**A Java keyword used to declare an enumerated type. Enumerations extend the base class Enum.**

1. **extends**

**Used in a class declaration to specify the superclass; used in an interface declaration to specify one or more superinterfaces. Class X extends class Y to add functionality, either by adding fields or methods to class Y, or by overriding methods of class Y. An interface Z extends one or more interfaces by adding methods. Class X is said to be a subclass of class Y; Interface Z is said to be a subinterface of the interfaces it extends.**

**Also used to specify an upper bound on a type parameter in Generics.**

1. **final**

**Define an entity once that cannot be changed nor derived from later. More specifically: a final class cannot be subclassed, a final method cannot be overridden, and a final variable can occur at most once as a left-hand expression on an executed command. All methods in a final class are implicitly final.**

1. **finally**

**Used to define a block of statements for a block defined previously by the try keyword. The finally block is executed after execution exits the try block and any associated catch clauses regardless of whether an exception was thrown or caught, or execution left method in the middle of the try or catch blocks using the return keyword.**

1. **float**

**The float keyword is used to declare a variable that can hold a 32-bit single precision IEEE 754 floating-point number.[5][6] This keyword is also used to declare that a method returns a value of the primitive type float.[7][8]**

1. **for**

**The for keyword is used to create a for loop, which specifies a variable initialization, a boolean expression, and an incrementation. The variable initialization is performed first, and then the boolean expression is evaluated. If the expression evaluates to true, the block of statements associated with the loop are executed, and then the incrementation is performed. The boolean expression is then evaluated again; this continues until the expression evaluates to false.[15]**

**As of J2SE 5.0, the for keyword can also be used to create a so-called "enhanced for loop",[16] which specifies an array or Iterable object; each iteration of the loop executes the associated block of statements using a different element in the array or Iterable.[15]**

1. **if**

**The if keyword is used to create an if statement, which tests a boolean expression; if the expression evaluates to true, the block of statements associated with the if statement is executed. This keyword can also be used to create an if-else statement; see else.[13][14]**

1. **implements**

**Included in a class declaration to specify one or more interfaces that are implemented by the current class. A class inherits the types and abstract methods declared by the interfaces.**

1. **import**

**Used at the beginning of a source file to specify classes or entire Java packages to be referred to later without including their package names in the reference. Since J2SE 5.0, import statements can import static members of a class.**

1. **instanceof**

**A binary operator that takes an object reference as its first operand and a class or interface as its second operand and produces a boolean result. The instanceof operator evaluates to true if and only if the runtime type of the object is assignment compatible with the class or interface.**

1. **int**

**The int keyword is used to declare a variable that can hold a 32-bit signed two's complement integer.[5][6] This keyword is also used to declare that a method returns a value of the primitive type int.[7][8]**

1. **interface**

**Used to declare a special type of class that only contains abstract or default methods, constant (static final) fields and static interfaces. It can later be implemented by classes that declare the interface with the implements keyword. As multiple inheritance is not allowed in Java, interfaces are used to circumvent it. An interface can be defined within another interface.**

1. **long**

**The long keyword is used to declare a variable that can hold a 64-bit signed two's complement integer.[5][6] This keyword is also used to declare that a method returns a value of the primitive type long.[7][8]**

1. **native**

**Used in method declarations to specify that the method is not implemented in the same Java source file, but rather in another language.[8]**

1. **new**

**Used to create an instance of a class or array object. Using keyword for this end is not completely necessary (as exemplified by Scala), though it serves two purposes: it enables the existence of different namespace for methods and class names, it defines statically and locally that a fresh object is indeed created, and of what runtime type it is (arguably introducing dependency into the code).**

1. **package**

**Java package is a group of similar classes and interfaces. Packages are declared with the package keyword.**

1. **private**

**The private keyword is used in the declaration of a method, field, or inner class; private members can only be accessed by other members of their own class.[17]**

1. **protected**

**The protected keyword is used in the declaration of a method, field, or inner class; protected members can only be accessed by members of their own class, that class's subclasses or classes from the same package.[17]**

1. **public**

**The public keyword is used in the declaration of a class, method, or field; public classes, methods, and fields can be accessed by the members of any class.[17]**

1. **return**

**Used to finish the execution of a method. It can be followed by a value required by the method definition that is returned to the caller**

1. **short**

**The short keyword is used to declare a field that can hold a 16-bit signed two's complement integer.[5][6] This keyword is also used to declare that a method returns a value of the primitive type short.[7][8]**

1. **static**

**Used to declare a field, method, or inner class as a class field. Classes maintain one copy of class fields regardless of how many instances exist of that class. static also is used to define a method as a class method. Class methods are bound to the class instead of to a specific instance, and can only operate on class fields. Classes and interfaces declared as static members of another class or interface are actually top-level classes and are not inner classes.**

1. **super**

**Inheritance basically used to achieve dynamic binding or run-time polymorphism in java. Used to access members of a class inherited by the class in which it appears. Allows a subclass to access overridden methods and hidden members of its superclass. The super keyword is also used to forward a call from a constructor to a constructor in the superclass.**

**Also used to specify a lower bound on a type parameter in Generics.**

1. **switch**

**The switch keyword is used in conjunction with case and default to create a switch statement, which evaluates a variable, matches its value to a specific case, and executes the block of statements associated with that case. If no case matches the value, the optional block labelled by default is executed, if included.[9][10]**

1. **synchronized**

**Used in the declaration of a method or code block to acquire the mutex lock for an object while the current thread executes the code.[8] For static methods, the object locked is the class's Class. Guarantees that at most one thread at a time operating on the same object executes that code. The mutex lock is automatically released when execution exits the synchronized code. Fields, classes and interfaces cannot be declared as synchronized.**

1. **this**

**Used to represent an instance of the class in which it appears. this can be used to access class members and as a reference to the current instance. The this keyword is also used to forward a call from one constructor in a class to another constructor in the same class.**

1. **throw**

**Causes the declared exception instance to be thrown. This causes execution to continue with the first enclosing exception handler declared by the catch keyword to handle an assignment compatible exception type. If no such exception handler is found in the current method, then the method returns and the process is repeated in the calling method. If no exception handler is found in any method call on the stack, then the exception is passed to the thread's uncaught exception handler.**

1. **throws**

**Used in method declarations to specify which exceptions are not handled within the method but rather passed to the next higher level of the program. All uncaught exceptions in a method that are not instances of RuntimeException must be declared using the throws keyword.**

1. **transient**

**Declares that an instance field is not part of the default serialized form of an object. When an object is serialized, only the values of its non-transient instance fields are included in the default serial representation. When an object is deserialized, transient fields are initialized only to their default value. If the default form is not used, e.g. when a serialPersistentFields table is declared in the class hierarchy, all transient keywords are ignored.[18][19]**

1. **try**

**Defines a block of statements that have exception handling. If an exception is thrown inside the try block, an optional catch block can handle declared exception types. Also, an optional finally block can be declared that will be executed when execution exits the try block and catch clauses, regardless of whether an exception is thrown or not. A try block must have at least one catch clause or a finally block.**

1. **void**

**The void keyword is used to declare that a method does not return any value.[7]**

1. **volatile**

**Used in field declarations to guarantee visibility of changes to variables across threads. Every read of a volatile variable will be read from main memory, and not from the CPU cache, and that every write to a volatile variable will be written to main memory, and not just to the CPU cache.[20] Methods, classes and interfaces thus cannot be declared volatile, nor can local variables or parameters.**

1. **while**

**The while keyword is used to create a while loop, which tests a boolean expression and executes the block of statements associated with the loop if the expression evaluates to true; this continues until the expression evaluates to false. This keyword can also be used to create a do-while loop; see do.**

**Not used keywords**

1. **const**
2. **goto**
3. **strictfp**

**Conclusion:**

|  |
| --- |
| Now in jdk22 version we are haing only 50 keywords because of \_ is not using as a keword in jdk22 version. |

**Reserved Identifiers or contextual keywords**

**The following identifiers are contextual keywords, and are only restricted in some contexts:**

1. exports
2. module
3. non-sealed
4. open
5. opens
6. permits
7. provides
8. record
9. requires
10. sealed
11. to
12. transitive
13. uses
14. var
15. when
16. with
17. yield

**Identifiers**

|  |
| --- |
| It is a name given to a variable,method, class etc.. |

**Rules to write an identifier**

|  |
| --- |
| * we can use alphabets, digits, under score and dollar sign to write an identifier * should not start with a digit * we can’t use keyword as an identifier |

**Example on usage of contextual keyword as keyword as well as identifier**

public class ContextualKeyword

{

    public static void main(String[] args)

    {

        //syntax:  <data-type>   <var-name>[=value];

        //               int a;

        //               double d=100.50;

        //syntax to declare a variable by using a contextual keyword called var

        //var <var-name>[=value];

        var a=100;

        var b=200;

        var c=a+b;

        System.out.printf("a=%d%n",a);

        System.out.printf("b=%d%n",b);

        System.out.printf("c=%d%n",c);

        int var=1000;

        System.out.printf("var=%d%n",var);

    }

}

Output:

a=100

b=200

c=300

var=1000

**literals**

|  |
| --- |
| Literal is a value used in our program   * 1. integer literals (100,-100)   2. floating point literals( 100.50,-10.50)   3. Boolean literals (true, false)   4. Character literal : ‘a’   5. String literal : “hello”   6. Other literals(null) |

**Operators**

|  |
| --- |
| Operator is a symbol which is used to perform an operation  **The different types of operators supported by Java**   * Arithmetic Operators (+, -, \*, /, %,) * Relational Operators (<, >, <=,>=,==, !=,instanceof) * Logical Operators (&&, ||, !, ) * Bitwise Operators (&, |, ^, ~, <<, >>, >>>) * Assignment operator (=) * Conditional operator (?:) * Short cut operators (--, ++, +=, -=, \*=, /=, %=) * Other operators (new(memory allocation operator), () type cast operator, .(memory access operator) ) |

**Reading Data From Keyboard**

|  |
| --- |
| **In java we can read the data**   * 1. By using System.in.read() method   2. By using the methods of DataInputStream class   3. By using the methods of BufferedReader class   4. By using the methods of Scanner class |

**What is JCL(Java Class Library)?**

|  |
| --- |
| JCL means hundreds of pre-defined classes. |

**What is Java API(Application Programming Interface)?**

|  |
| --- |
| Collection of pre-defined packages |

**What is a package?**

|  |
| --- |
| * Collection of classes * A package is a container, which contains set of classes * A package is a container which contains collection of classes, interfaces, enums and annotations.   **Some pre-defined packages available in Java**   * Java.lang * Java.io * Java.util * Java.text * Java.awt * Java.awt.event * Java.sql * Java.net   Etc.. |

**Scanner class**

Scanner is a pre-defined class existed in java.util package. Which contains methods which are used to read the data from keyboard or a file.

How to use a pre-defined class in a program?

|  |
| --- |
| By importing it |

**Scanner class methods**

|  |  |
| --- | --- |
| Method | Description |
| nextInt() | It is used to read an integer value from keyboard |
| next() | It takes only single word from keyboard |
| nextLine() | We have to use nextLine() method to read total line at a time. |
| Skip() | Skip() method is used to skip a character |
| nextFloat() | To read a float value from the keyboard |

**First example on usage of nextInt() method of Scanner class**

import java.util.Scanner;

public class ScannerDemo1

{

    public static void main(String[] args)

    {

        //to read the data from keyboard we have to pass System.in to the

        //Scanner class constructor

        Scanner scan=new Scanner(System.in);

        System.out.println("Hey Madhu Enter int value:\t");

        int a=scan.nextInt();

        System.out.println("Inkotiyyaanaaa!");

        int b=scan.nextInt();

        int c=a+b;

        System.out.printf("%-5d + %-5d = %-5d",a,b,c);

    }

}

Output:

Hey Madhu Enter int value:

100

Inkotiyyaanaaa!

20

100 + 20 = 120

**Example to read a single word from the keyboard**

import java.util.Scanner;;

public class ScannerDemo2

{

    public static void main(String[] args)

    {

        Scanner scanner=new Scanner(System.in);

        System.out.println("Orey first name ivvara:\t");

        String firstName=scanner.next();

        System.out.println("Last name kooda ivvara babu:\t");

        String lastName=scanner.next();

        String fullName=firstName+"."+lastName;

        System.out.println(fullName);

    }

}

Output:

Orey first name ivvara:

Madhu

Last name kooda ivvara babu:

K

Madhu.K

**Example on usage of nextLine(), nextFloat() , skip() methods**

import java.util.Scanner;;

public class ScannerDemo2

{

    public static void main(String[] args)

    {

        int eno;

        String ename;

        float sal;

        Scanner scanner=new Scanner(System.in);

        System.out.print("Eno:\t");

        eno=scanner.nextInt(); //101\n

        System.out.print("FullName:\t");

        scanner.skip("\r\n");

        ename=scanner.nextLine();

        System.out.print("Sal:\t");

        sal=scanner.nextFloat();

        System.out.printf("Eno:\t%d%n",eno);

        System.out.printf("Ename:\t%s%n",ename);

        System.out.printf("Salary:\t%s%n",sal);

    }

}

Output:

Eno: 101

FullName: madhu k

Sal: 200000

Eno: 101

Ename: madhu k

Salary: 200000.0

**Naming Conventions**

|  |  |  |
| --- | --- | --- |
| **Java element** | **Description** | **Example** |
| class, interfaces, enums and annotations | Every word first letter capital | Scanner,  String  System  DataInputStream |
| Method names and variables | Camel case means first word is in small and after that every word first letter must be capital | next()  nextLine()  getInputStream()  readLine()  Variable name:  eno;  empName;  firstName; |
| Keywords, packages | In small letters | int, class, if, static, try, final, void, while, for, public etc..  package name:  java.lang, java.io, java.util |
| Constants | CAPITAL LETTERS | PI  MAX\_PRIORITY |

**Conditional or control statements**

* If
* If..else
* else..if ladder
* nested if
* switch

**if statement**

* 1. if you want to execute one or more statements if a given condition Is true then we write if statement
  2. if you want to execute more than one statement if a given condition is true, then you have to put those statements in a block, it is a rule.

import java.util.Scanner;

public class IfDemo

{

    public static void main(String[] args)

    {

        System.out.println("Start of the program");

        Scanner scanner=new Scanner(System.in);

        System.out.println("Nee yezentharaa...!");

        int age=scanner.nextInt();

        if(age>=18)

        System.out.println("I cast my vote");

        System.out.println("End of the program");

    }

}

Output:

Start of the program

Nee yezentharaa...!

30

I cast my vote

End of the program

Executing set of statements if the condition is true

import java.util.Scanner;

public class IfDemo

{

    public static void main(String[] args)

    {

        System.out.println("Start of the program");

        Scanner scanner=new Scanner(System.in);

        System.out.println("Nee yezentharaa...!");

        int age=scanner.nextInt();

        if(age>=18)

        {

            System.out.println("You are eligible to vote");

            System.out.println("You can cast your vote");

            System.out.println("Thanks andeeey");

        }

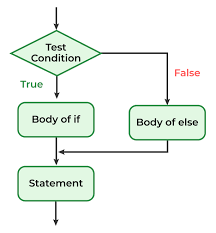
        System.out.println("End of the program");

    }

}

Example on if…else statement

|  |
| --- |
| I want to execute set of statements if the condition is true and also I want to execute set of statement if the condition is false then we have to use if… else statement |



**Example on if…else statement**

import java.util.Scanner;

public class Compare

{

    public static void main(String[] args)

    {

        int a,b;

        Scanner scan=new Scanner(System.in);

        System.out.println("Bangarm enter an int value:\t");

        a=scan.nextInt();

        System.out.println("Bangarm enter another int value:\t");

        b=scan.nextInt();

        if(a>b)

        {

            System.out.println("a is greater than b");

        }else

        {

            System.out.println("b is greater than a");

        }

    }

}

Output:

Bangarm enter an int value:

100

Bangarm enter another int value:

20

a is greater than b

**if..else..if or else…if ladder**

**Write a program to compare two integer**

import java.util.Scanner;

public class Compare

{

    public static void main(String[] args)

    {

        int a,b;

        Scanner scan=new Scanner(System.in);

        System.out.println("Bangarm enter an int value:\t");

        a=scan.nextInt();

        System.out.println("Bangarm enter another int value:\t");

        b=scan.nextInt();

        //a=100

        //b=100

        if(a>b)

        {

            System.out.println("a is greater than b");

        }else if(b>a)

        {

            System.out.println("b is greater than a");

        }else if(a==b){

            System.out.println("a and b are same");

        }

    }

}

Output:

Bangarm enter an int value:

10

Bangarm enter another int value:

10

a and b are same

**Nested If statement**

import java.util.\*;

public class ComparingThree

{

    public static void main(String[] args)

    {

        int a,b,c;

        Scanner scan=new Scanner(System.in);

        System.out.println("Enter 3 integers");

        a=scan.nextInt();

        b=scan.nextInt();

        c=scan.nextInt();

        //a=100

        //b=250

        //c=279

        if(a>b)

        {

            if(a>c)

            {   System.out.println("a is greater than b,c");

            }else

            {   System.out.println("c is greater than a,b");

            }

        }else if(b>c){

            System.out.println("b is greater than a,c");

        }else{

            System.out.println("c is greater than a,b");

        }

    }

}

Output:

Enter 3 integers

10

22

43

c is greater than a,b

**Switch Statement**

* If you want to choose one among many options we can write a switch statement.

public class SwitchDemo

{

    public static void main(String[] args) throws Exception

    {

        System.out.println("Enter a character:\t");

        char ch=(char)System.in.read();

        //System.out.println("ch:\t"+ch);

        switch (ch) {

            case 'a','e','i','o','u','A','E','I','O','U'->System.out.println("Vowel");

            case '1','2','3','4','5','6','7','8','9','0'->System.out.println("Digit");

            default->System.out.println("Consonent");

        }

    }

}

Output:

Enter a character:

A

Vowel

Switch With return value

import java.util.Scanner;

public class SwitchDemo2

{

    public static void main(String[] args)

    {

        int a,b,c,opt;

        Scanner scan=new Scanner(System.in);

        System.out.println("1. add");

        System.out.println("2. sub");

        System.out.println("3. multi");

        System.out.println("4. div");

        System.out.println("5. mod");

        System.out.print("Option:\t");

        opt=scan.nextInt();

        if(opt>=1 && opt<=5)

        {

            System.out.println("Enter 2 int values:\t");

            a=scan.nextInt();

            b=scan.nextInt();

            int result=switch(opt){

                case 1->{yield a+b;}

                case 2->{yield a-b;}

                case 3->{yield a\*b;}

                case 4->{yield a/b;}

                case 5->{yield a%b;}

                default->{throw new RuntimeException("Invalid Option");}

            };

            System.out.println("Result:\t"+result);

        }else{

            System.out.println("Invalid Option");

        }

    }

}

Output:

1. add

2. sub

3. multi

4. div

5. mod

Option: 5

Enter 2 int values:

10

2

Result: 0

**Data types**

**There are 2 types of datatypes**

* 1. Primitive data types
  2. Non-primitive data types (Reference or object data types)

**Primitive data types**

|  |
| --- |
| **There are 8 primitive data types available in Java** |

|  |  |  |  |
| --- | --- | --- | --- |
| **Data type** | **Size** | **Default value** | **WrappeClasses** |
| byte | **1 byte** | 0 | **Byte** |
| short | **2 bytes** | 0 | **Short** |
| int | **4 bytes** | 0 | **Integer** |
| long | **8 bytes** | 0 | **Long** |
| float | **4 bytes** | 0.0 | **Float** |
| double | **8 bytes** | 0.0 | **Double** |
| char | **2 bytes** | ‘\u0000’ | **Character** |
| boolean | **1 bit** | false | **Boolean** |
|  |  |  |  |

**8 primitive data types are divided into 4 categories**

**Integral data types**

* + 1. **byte, short,int and long**

**Floating Point data types**

* + 1. **float and double**

**Character data types**

* + 1. **char**

**Boolean data types**

* + 1. **boolean**

**Example on byte type variable**

public class DataTypes

{

    public static void main(String[] args)

    {

        byte b1=127;

        System.out.format("b1:\t%d%n",b1);

        byte b2=(byte)128;

        //you will loose data

        //byte b2=int

        System.out.format("b2:\t%d%n",b2);

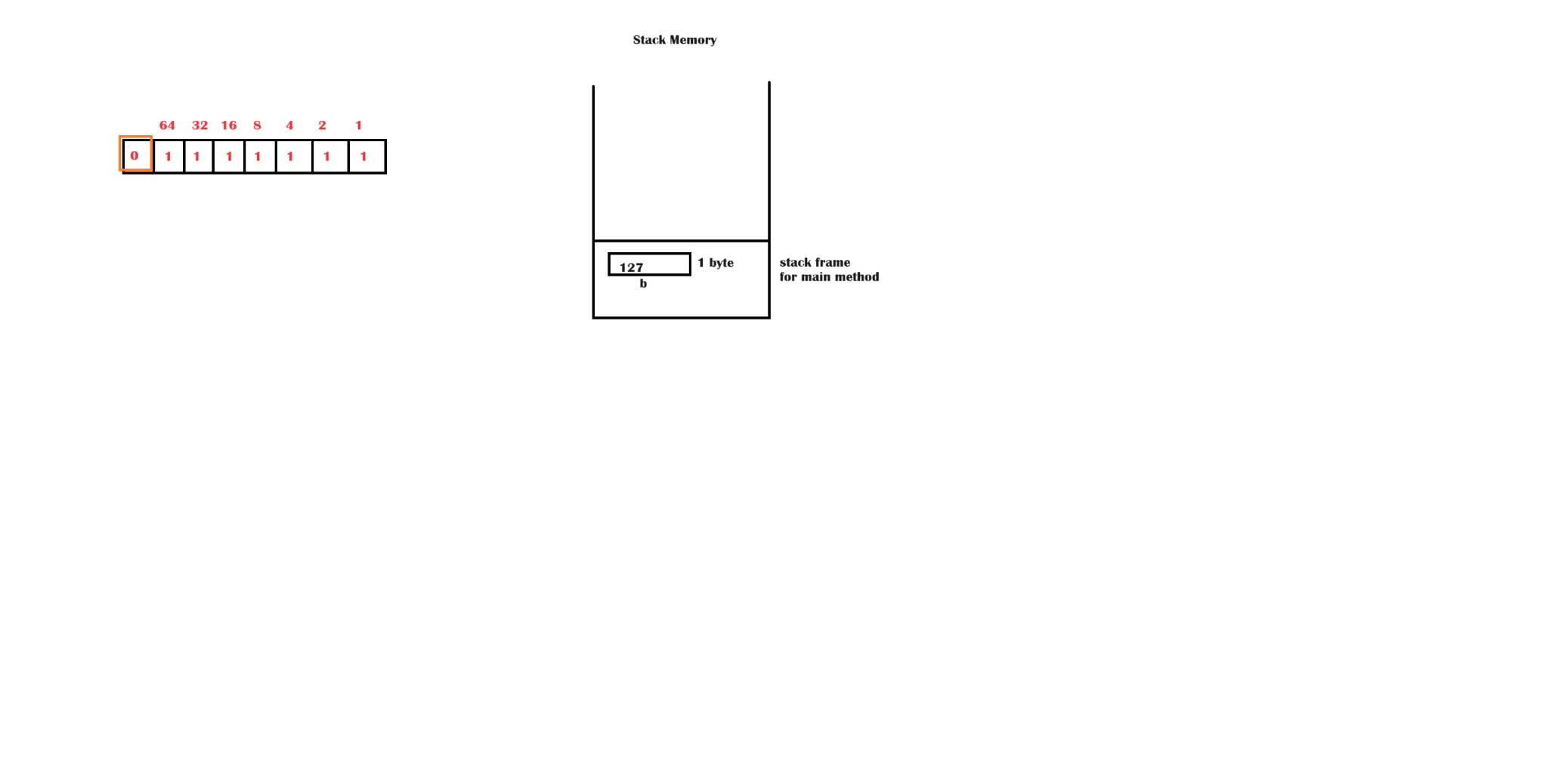
    }

}

Output:

b1: 127

b2: -128



**What is a local variable?**

|  |
| --- |
| * It is a variable which is declared within a block which is existed in a class. * Local variables doesn’t contain with default values |

**Can we use a variable without initializing it?**

|  |
| --- |
| We can’t |

**Getting errors if you try to use local variables without initializing them**

public class DataTypes

{

    public static void main(String[] args)

    {

        byte b1;

        System.out.format("b1:\t%d%n",b1);

        int b2=b1;

    }

}

Output:

PS D:\Trainings\GECFULLSTACK2024\Programs> javac .\DataTypes.java

.\DataTypes.java:6: error: variable b1 might not have been initialized

System.out.format("b1:\t%d%n",b1);

^

1 error

**Non-Primitive Data Types**

By using non-primitive data types (reference data types) we can declare reference variables.

**Based on the values we store in a variable we can divide them into 2 categories.**

* 1. Value type variables
  2. Reference variables

What is a value type variable?

|  |
| --- |
| * In value type variable we can store only a value. We can’t store address in a value type variable. * We can declare a value type variable by using primitive data types.   Ex: int n=100; //value type variable |

**What is a reference variable?**

|  |
| --- |
| * It is a variable where we can store either address or null. * We can declare a reference variable by using reference data type. * As reference data type we can use any one of the following   1. Class name   2. Interface name   3. Enum name   4. Annotation name   5. Any data type with combination of one or more subscripts can be used as reference data type |

**Arrays**

**What is an array?**

|  |
| --- |
| Collection of similar data elements, stored at contiguous memory locations. |

**Types of arrays**

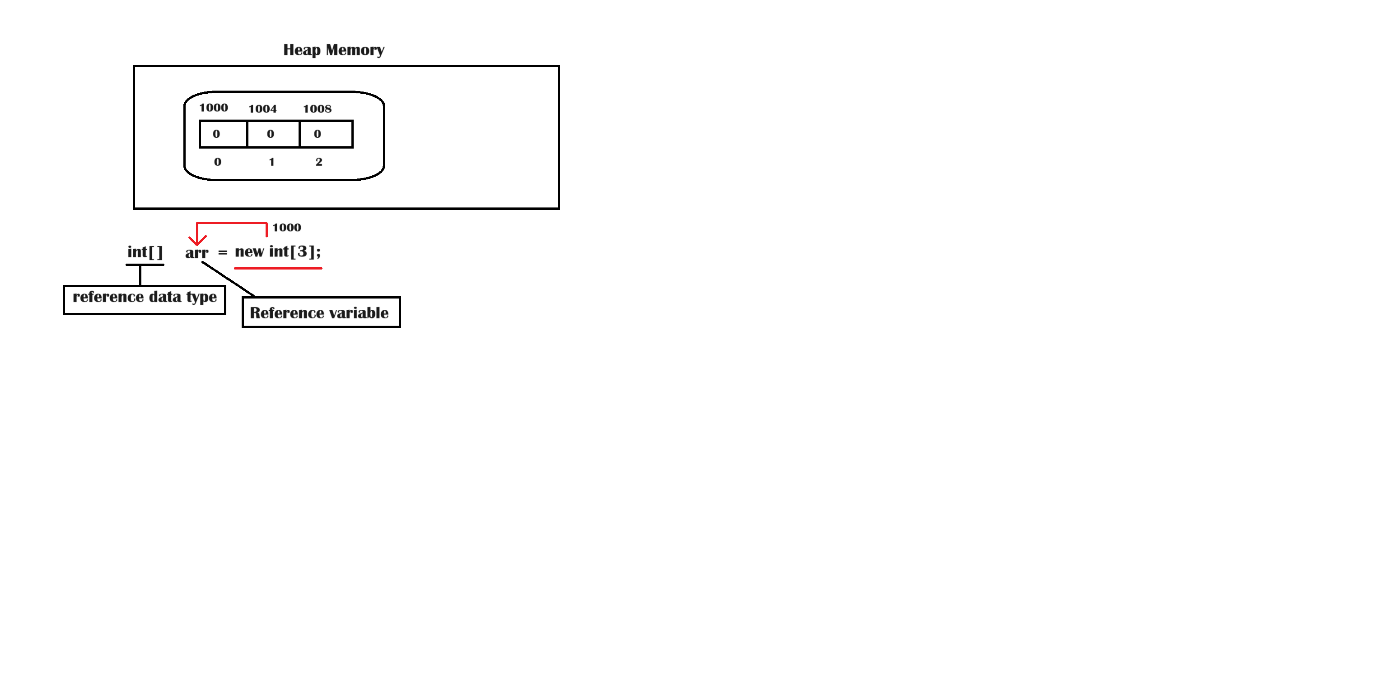
* Single dimensional arrays
* Multi-dimensional arrays

**Single Dimensional Arrays**

|  |
| --- |
| * 1. It is also called as single subscripted array   Ex: new int[3]; |

**What is new?**

|  |
| --- |
| New is a keyword and it is also called as memory allocation operator which allocated memory in heap memory. |



**Iterative Statements**

* **While**
* **Do..while**
* **For**
* **For each**

Example on usage of for loop to get elements of 1D array

public class ArrayDemo1

{

    public static void main(String[] args)

    {

        int[] arr=new int[3];

        //arr is a reference variable

        arr[0]=100;

        arr[1]=200;

        arr[2]=300;

        //0,1,2

        //i=0,1,2,3

        for(int i=0;i<3;i++)

        {

            System.out.println(arr[i]);

        }

        //100

        //200

        //300

        //-

    }

}

Another example on array

import java.util.Scanner;

public class ArrayDemo2

{

    public static void main(String[] args)

    {

        int[] arr=null;

        Scanner scan=new Scanner(System.in);

        System.out.println("Enter the array size:\t");

        int size=scan.nextInt();

        arr=new int[size];

        System.out.printf("Enter %d Elements:\t",size);

        for(int i=0;i<size;i++)

        {   arr[i]=scan.nextInt();

        }

        System.out.println("Elements in an array....");

        for(int i=0;i<size;i++)

        {   System.out.println(arr[i]);

        }

    }

}

Output:

Enter the array size:

5

Enter 5 Elements: 100

200

300

400

500

Elements in an array....

100

200

300

400

500

Foreach

|  |
| --- |
| * 1. For each is an extension of for loop   2. It is used to get the elements from an array or a collection easily.   3. No need to use the index in the foreach to the elements from an array. |

Example to get the elements by using for each

import java.util.Scanner;

public class ArrayDemo2

{

    public static void main(String[] args)

    {

        int[] arr=null;

        Scanner scan=new Scanner(System.in);

        System.out.println("Enter the array size:\t");

        int size=scan.nextInt();

        arr=new int[size];

        System.out.printf("Enter %d Elements:\t",size);

        for(int i=0;i<size;i++)

        {   arr[i]=scan.nextInt();

        }

        System.out.println("Elements in an array....");

        for(int ele:arr)

        {   System.out.println(ele);

        }

    }

}

Output:

Enter the array size:

3

Enter 3 Elements: 100

200

300

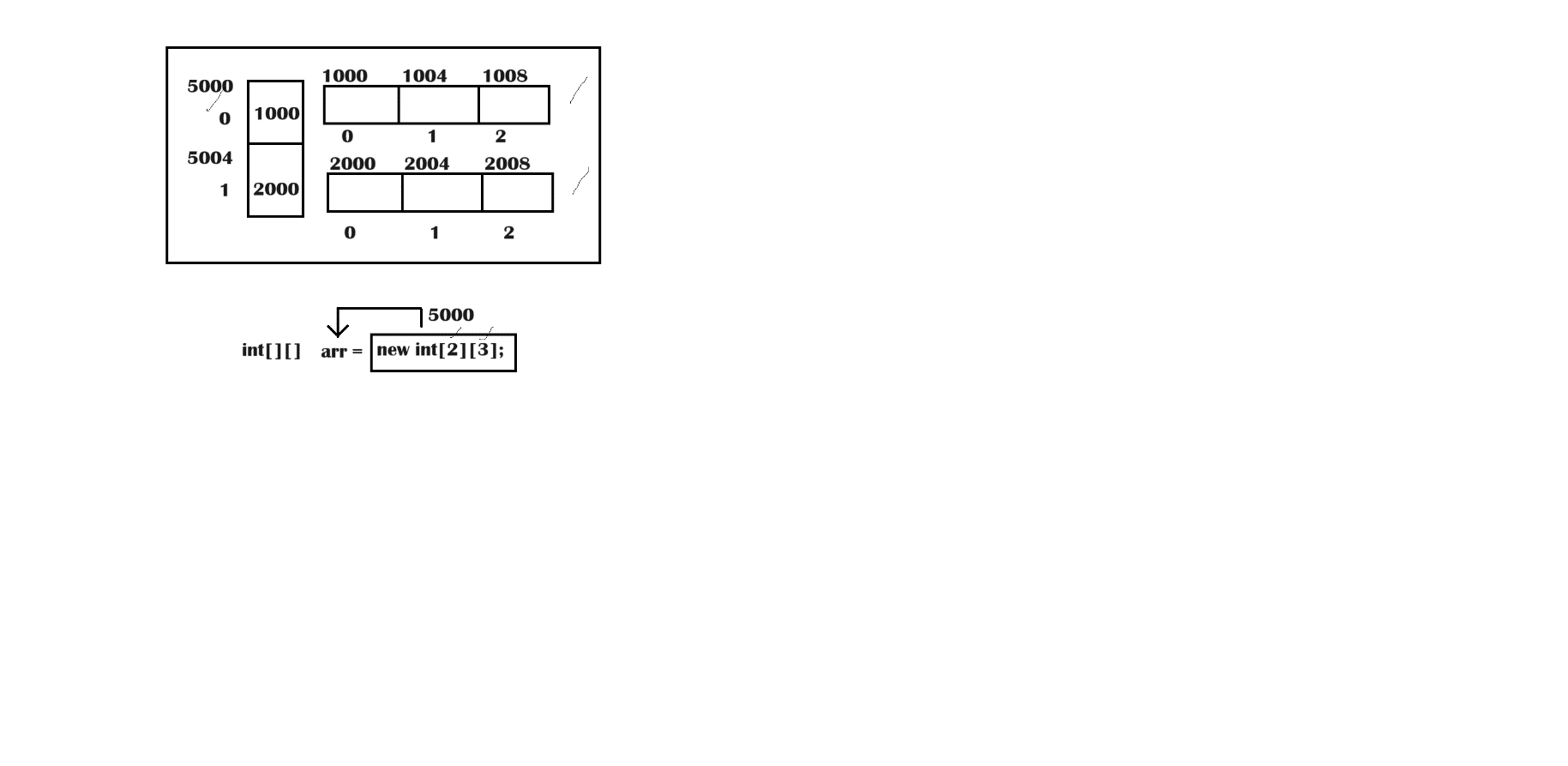
Elements in an array....

100

200

300

**Two dimensional arrays**



|  |
| --- |
| * It is an array with 2 subscripts, where we can store elements in matrix format or in tabular format. * A two-dimensional array is a collection of single dimensional arrays. |

Example on 2d array

import java.util.Scanner;

public class TwoDimArray1

{

    public static void main(String[] args)

    {

            Scanner scan=new Scanner(System.in);

            int[][]arr= new int[2][3];

            System.err.println("arr.length:\t"+arr.length);

            //var arr2= new int[2][3];

            System.out.println("arr[0].length:\t"+arr[0].length);

            System.out.println("arr[1].length:\t"+arr[1].length);

            //arr.length=2

            //i=0,1,2

           for(int i=0;i<arr.length;i++)

           {    //i=0,1

                //arr[i].length=3

                //j=0,1,2,3

                for(int j=0;j<arr[i].length;j++)

                {   System.out.printf("arr[%d,%d]:\t",i,j);

                    arr[i][j]=scan.nextInt();

                }

                System.out.println();

           }

           System.out.println("Elements in 2d Array...");

           for(int i=0;i<arr.length;i++)

           {

                for(int j=0;j<arr[i].length;j++)

                {   System.out.printf("%-8d",arr[i][j]);

                }

                System.out.println();

           }

    }

}

Output:

arr.length: 2

arr[0].length: 3

arr[1].length: 3

arr[0,0]: 10

arr[0,1]: 20

arr[0,2]: 30

arr[1,0]: 40

arr[1,1]: 50

arr[1,2]: 60

Elements in 2d Array...

10 20 30

40 50 60

Jagged Array

|  |
| --- |
| It is an array of arrays where each array size differs. |

Example on Jagged Array

import java.util.Scanner;

public class TwoDimArray1

{

    public static void main(String[] args)

    {

        int[] arr1=new int[]{10,20,30,40,50};

        int[] arr2={100,200,300,400,500};

        System.out.println("Elements in arr1");

        for(int ele:arr1)

        {System.out.print(ele+"\t");

        }

        System.out.println("\nElements in arr2");

        for(int ele:arr2)

        {System.out.print(ele+"\t");

        }

        int[][] arr3={

            arr1,

            arr2

        };

        System.out.println("Elements in 2d array");

        for(int i=0;i<arr3.length;i++)

        {

            for(int j=0;j<arr3[i].length;j++)

            {

                System.out.printf("%-6d",arr3[i][j]);

            }

            System.out.println();

        }

        int[][] arr4={

            {100},

            {100,200},

            {100,200,300}

        };

        System.out.println("Elements in 2d array");

        for(int i=0;i<arr4.length;i++)

        {

            for(int j=0;j<arr4[i].length;j++)

            {

                System.out.printf("%-6d",arr4[i][j]);

            }

            System.out.println();

        }

    }

}

Output:

Elements in arr1

10 20 30 40 50

Elements in arr2

100 200 300 400 500 Elements in 2d array

10 20 30 40 50

100 200 300 400 500

Elements in 2d array

100

100 200

100 200 300

Example to get the elements from 2d array using foreach

import java.util.Scanner;

public class TwoDimArray1

{

    public static void main(String[] args)

    {

        int[][] arr4={

            {100},

            {100,200},

            {100,200,300}

        };

        System.out.println("Elements in 2d array");

        for(int[] arr:arr4)

        {

            for(int ele:arr)

            {

                System.out.printf("%-6d",ele);

            }

            System.out.println();

        }

    }

}

Output:

Elements in 2d array

100

100 200

100 200 300

Example on three dimensional array

public class ThreeDimArray

{

    public static void main(String[] args) {

        int[][] arr1={

            {1,2,3},

            {10,20,30},

            {100,200,300}

        };

        int[][] arr2={

            {11,21,31},

            {110,220,330},

            {1100,2200,3300}

        };

        int[][][] arr3={arr1,arr2};

        for(int[][] arr:arr3)

        {

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

            for(int[] sarr:arr)

            {

                for(int ele:sarr)

                {

                    System.out.printf("%-6d",ele);

                }

                System.out.println();

            }

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

        }

    }

}

Output:

................

1 2 3

10 20 30

100 200 300

................

................

11 21 31

110 220 330

1100 2200 3300

................

**Methods**

**What is a function?**

|  |
| --- |
| * 1. It is a block which contains re-usable set of statements.   2. We write a function to perform a task |

**What is the difference between function and a method?**

|  |
| --- |
| * If we write a function in a class it is called as method * If we write a method outside the class it is called as function. |

**MethodDemo1.java**

import java.util.Scanner;

public class MethodDemo1

{

    static void add()

    {   int a=10,b=20,c=0;

        c=a+b;

        System.out.println("c="+c);

    }

    public static void main(String[] args)

    {

        add();

    }

}

Output:

c=30

**What are parameters?**

|  |
| --- |
| Parameters are variables which are declared within parenthesis of a function. |

**Example on methods**

import java.util.Scanner;

public class MethodDemo1

{   static void add()

    {

        int a=10,b=20,c=0;

        c=a+b;

        System.out.printf("%d + %d = %d%n",a,b,c);

    }

    static void sub(int a,int b)

    {       int c=a-b;

            System.out.printf("%d - %d = %d%n",a,b,c);

    }

    static int multi(int a,int b)

    {

        int c=a\*b;

        //c=20

        return c;

    }

    static int div()

    {

        int a=10,b=2;

        return a/b;

    }

    public static void main(String[] args)

    {

        add();

        sub(100,20);

        int r1=multi(10,2);

        int r2=div();

        System.out.println("r1:\t"+r1);

        System.out.println("r2:\t"+r2);

    }

}

Output:

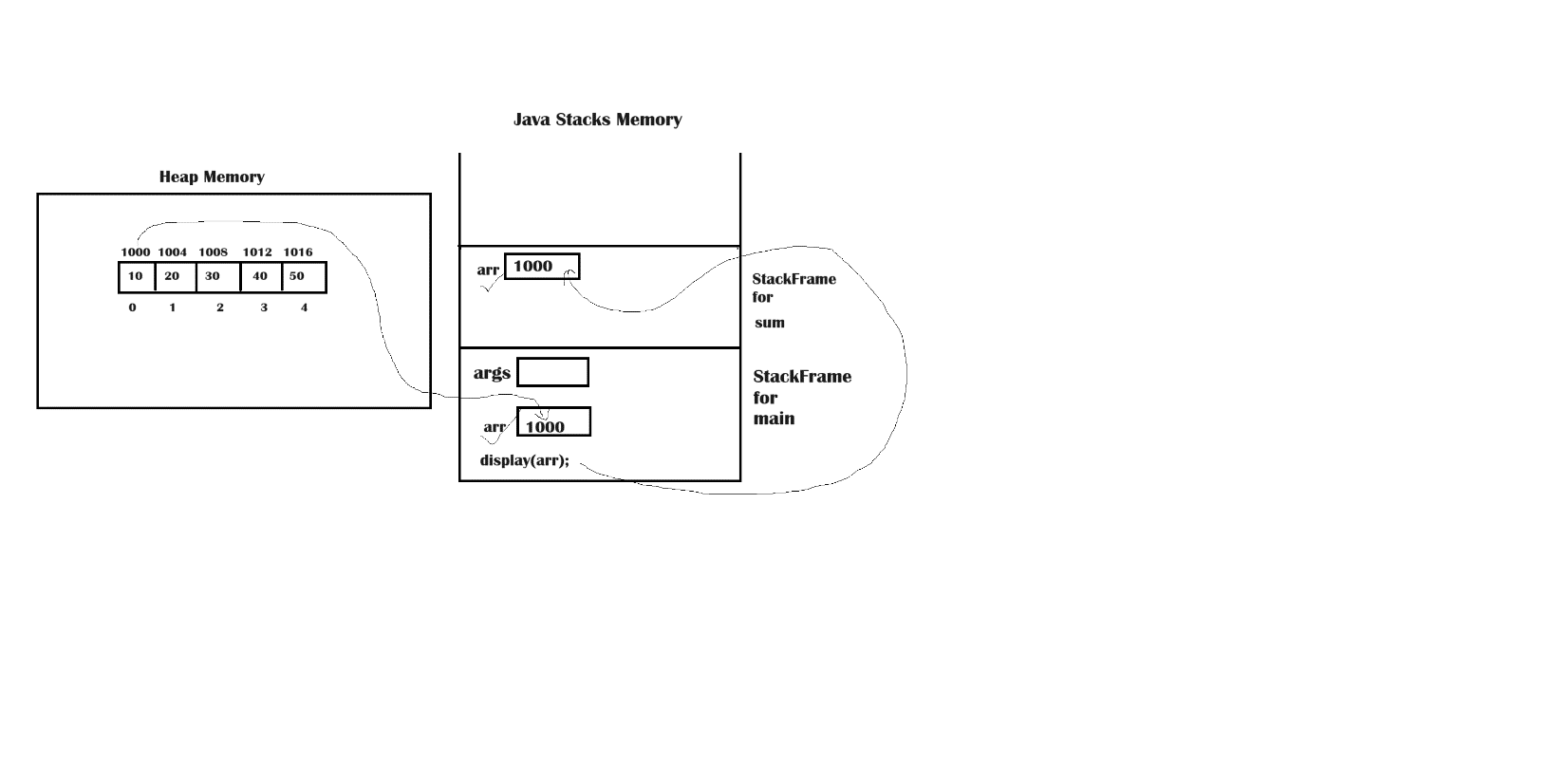
10 + 20 = 30

100 - 20 = 80

r1: 20

r2: 5

**Note:** In Java array will be created in heap memory only.



**Passing array to methods**

public class MethodsDemo2

{

    static void display(int[] arr)

    {

        System.out.println("Elements in an array...");

        for(int ele:arr)

        {

            System.out.printf("%-5d",ele);

        }

        System.out.println();

    }

    static int sum(int[] arr)

    {

        int sum=0;

        for(int ele:arr)

        {

            sum=sum+ele;

        }

        return sum;

    }

    public static void main(String[] args)

    {

        int[]  arr={10,20,30,40,50};

        //arr=1000

        display(arr);  //display(1000)

        //I am passing array address to the display method

        int s=sum(arr);

        System.out.println("s="+s);

    }

}

Output:

Elements in an array...

10 20 30 40 50

s=150

**Concepts we have covered till now**

* + 1. **Introduction**
    2. **Features**
    3. **Jdk installations**
    4. **VSCode installation**
    5. **JCL**
    6. **Java API**
    7. **Escape sequences**
    8. **Naming conventions**
    9. **Compilation**
    10. **Execution**
    11. **Data types**
    12. **If**
    13. **If..else**
    14. **If..else..if**
    15. **Nested if**
    16. **Switch**
    17. **Array**
    18. **1D arrays**
    19. **2d arrays**
    20. **3d arrays**
    21. **Jagged arrays**
    22. **For loop**
    23. **For each**
    24. **While**
    25. **Do..while**
    26. **Functions(methods)**