What is the difference between JDK and JRE?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| The **JDK (Java Development Kit)** and **JRE (Java Runtime Environment)** are both essential components of Java, but they serve different purposes:  **1. JDK (Java Development Kit)**   * **Purpose:** Used for developing, compiling, and running Java applications. * **Includes:**   + JRE (Java Runtime Environment)   + Java Compiler (javac)   + Development tools (debugger, Javadoc, etc.) * **Who Needs It?** Developers who write and compile Java programs.   **2. JRE (Java Runtime Environment)**   * **Purpose:** Used only for running Java applications. * **Includes:**   + JVM (Java Virtual Machine)   + Java class libraries * **Who Needs It?** End-users who just want to run Java applications but don’t need to develop them.   **Key Difference:**   |  |  |  | | --- | --- | --- | | **Feature** | **JDK (Java Development Kit)** | **JRE (Java Runtime Environment)** | | **Includes JVM?** | ✅ Yes | ✅ Yes | | **Includes JRE?** | ✅ Yes | ❌ No (JRE is a standalone package) | | **Includes Compiler?** | ✅ Yes (javac) | ❌ No | | **Usage** | Development & Execution | Execution only |   **In short:**   * If you want to **develop Java applications**, install the **JDK**. * If you only need to **run Java applications**, install the **JRE**.   Difference Between JDK, JRE, and JVM | Board Infinity  What jdk(Java development kit) contains?  It contains set of tools and JRE  What is JRE(Java Runtime Environment)?  It contains JVM + JCL  What is JVM?  Java Virtual Machine which is used to run the java programs  What is JCL (Java Class Library)  Set of pre-defined classes. |

What is a package?

|  |
| --- |
| 1. A package is a container of classes. 2. A package contains collection classes. 3. A package is a folder which contains classes, interfaces, Enums and annotations.   There are 2 types of packages   1. Pre-defined (provided by inventors of Java) 2. User-defined (Written by programmers)   Some of the pre-defined available in Java are:   1. Java.lang 2. Java.util 3. Java.io 4. Java.awt 5. Java.text 6. Java.awt.event 7. Java.sql 8. Java.net 9. Java.util.stream 10. Etc.. |

What is java API (Application Programming Interface)?

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

We can write comments in 3 ways?

|  |
| --- |
| 1. Single line (//………………) 2. Multi line comments (/\*……….\*/) 3. Documentation comments (/\*\*…………………………………\*/) |

What is the use of “import java.lang.\*” statement in our program?

|  |
| --- |
| It is a statement which is used to import a package. |

What is Java API?

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

What is JCL(Java Class Library)?

|  |
| --- |
| Collection of pre-defined classes. |

What is a keyword?

|  |
| --- |
| It is a pre-defined word, provided by inventors of Java |

What is a class?

|  |
| --- |
| A class is a block which contains functions(methods) and variables(fields/attributes/properties) |

**Sixth.java**

|  |
| --- |
| //this is my sixth program  /\* multi line comments  Author: Balaji B  Date: 08-Feb-2025  \*/  /\*\* documentation comments  Org: Madhu Tech Skills  City: vijayawada  \*/  package p1; //creating a package by using a keyword called package  import java.lang.\*; //importing a package called java.lang  interface MyInterface{}  enum Colors{  }  @interface MyAnnotation{}  class Ayyo{  public static void main(String[] args)  {  System.out.println("Ayyaaaa...");  }  } |

**What is package p1; statement?**

|  |
| --- |
| * It is a statement which is used to create a new package. Here package is a keyword and p1 is the package name. |

**What is the use of “import java.lang.\*;” ?**

|  |
| --- |
| It is used to import all the classes, interfaces, enums and annotations existed in a package called java.lang. |

**Why we import classes of a package?**

|  |
| --- |
| To use those classes, you have to import them |

What is the command to compile the program if it contains package statement?

|  |
| --- |
| Javac -d . Sixth.java  If I compile the program like above then 4 .class files will be created in p1 folder(package)   * + 1. MyInterface.class file     2. My Annotation.class     3. Colors.class     4. Ayyo.class |

**How to run the program if main method class is existed in p1 package?**

|  |
| --- |
| java p1.Ayyo |

**Can we write print statement outside the function in Java?**

|  |
| --- |
| No we can’t, but we can write it in any block which is written in a class. |

**If you write print statement outside the function what error you will get?**

|  |
| --- |
|  |

**Can I declare a variable with in a class and outside the function?**

|  |
| --- |
| **Yes we can declare 2 types of variables**   * Instance variables (it is not declared as static) * Static variables (it is declared as static) |

**Can I write statements other than declarations in a class and outside the functions?**

|  |
| --- |
| No we can write only variable declaration(definition) statements. |

**If I write the “c=a\*s;” statement outside the function within a class what kind of error you will get?**

|  |
| --- |
|  |

**What is String, System which are used in our program?**

|  |
| --- |
| * These are a pre-defined classes existed in java.lang package |

**Can we use String class without importing java.lang package?**

|  |
| --- |
| Yes, we can because java.lang package is the default package, which means the compiler imports it automatically. |

**When the java.lang package will be imported automatically?**

|  |
| --- |
| During compilation |

What is a string?

|  |
| --- |
| A string is nothing but collection of characters represented by pair of double quotations  Ex-1: “ravi teja” -> string literal(value)  Ex-2: “Vamsi” -> string literal  Ex-3: “12345” -> string literal  Ex-4: “+-\*&^@” -> string literal |

What is a concatenation symbol?

|  |
| --- |
| If you find a ‘+’ symbol after or before a string literal it is called as a concatenation symbol |

What is the use of concatenation symbol?

|  |
| --- |
| It appends any value to the string  Ex: “a=”+10 result is “a=10” |

What are the differences between print and println()?

|  |  |
| --- | --- |
| Print | Println |
| 1. It prints the data without moving cursor to the new line | 1. It moves the cursor to the new line after printing the data. |
| 1. We can pass only one argument to the print function | 1. We can pass zero or one argument to the println function |

Third.java

|  |
| --- |
| //this is my sixth program  /\* multi line comments  Author: Balaji B  Date: 08-Feb-2025  \*/  /\*\* documentation comments  Org: Madhu Tech Skills  City: vijayawada  \*/  package p1; //creating a package by using a keyword called package  import java.lang.\*; //importing a package called java.lang  interface MyInterface{}  enum Colors{  }  @interface MyAnnotation{}  class Ayyo  {  static int s=1;  int a=2;  int c=a+s;  //c=a\*s; Error non declaration statements are not allowed here  public static void main(String[] args)  {  int a=2,b=3,c=a+b;  System.out.println("Ayyaaaa...");  System.out.println(); //it displays new line  System.out.println(a,b,c); //error  System.out.print(); //error  }  }  //this is my sixth program  /\* multi line comments  Author: Balaji B  Date: 08-Feb-2025  \*/  /\*\* documentation comments  Org: Madhu Tech Skills  City: vijayawada  \*/  package p1; //creating a package by using a keyword called package  import java.lang.\*; //importing a package called java.lang  interface MyInterface{}  enum Colors{  }  @interface MyAnnotation{}  class Ayyo  {  static int s=1;  int a=2;  int c=a+s;  //c=a\*s; Error non declaration statements are not allowed here  public static void main(String[] args)  {  int a=2,b=3,c=a+b;  System.out.println("Ayyaaaa...");  System.out.println(); //it displays new line  //System.out.println(a,b,c); //error  //System.out.print(); //error  }  }  How to compile the program to create a package?     * + 1. -d: it tells the compiler to create a folder with package name     2. . : it tells where to create the folder (Here . means present folder)   If you compile the above program like below then the package will be create din hari3 folder  Compile: javac -d C:\Users\kalla\OneDrive\Dokumen\hari3 Sixth.java |

What is a variable?

|  |
| --- |
| 1. It a container to store a value 2. Variable is changeable |

How to declare a variable?

|  |
| --- |
| In Java we can declare a variable by using a data type  Syntax: <data-type> <var-name>[=value]; Ex: int a; Ex: float f=10.00f;  Syntax: <data-type> <var-1[=value]>[,var-2[=value],var-3[=value]…var-n[=value]];  Syntax: (from java 10 onwards: var <var-name>[=value]; Ex: var a; Ex: var f=10.00f;  Syntax: (from java10 onwards: var <var-1[=value]>[,var-2[=value], …var-n[=value]]; |

What is a data type?

|  |
| --- |
| Def-1: The type of data we store in a variable is called as data type.  Def-2: it is a keyword or a class name or a combination of data type and one or more subscripts which is used to declare a variable? |

Why we declare a variable?

|  |
| --- |
| To allocate memory in RAM. |

Types of variables?

|  |
| --- |
| Based on the data we store variables are divided into 2 categories   1. Value type variables 2. Reference or object data types |

What are value type variables?

|  |
| --- |
| 1. in these variables we can store the value not address 2. to declare value type variables we have to use primitive data types |

What are reference variables?

|  |
| --- |
| 1. In these variables we can store either address or null. 2. We can declare reference variables, by using class name, interface name, annotation name , enum name or combination of data type and one or more subscripts. |

Example to declare multiple variables

Four.java (no need to execute this program)

|  |
| --- |
| import java.lang.\*; //import statement  class Four  {  public static void main(String args[])  {  int a,b,c; //multiple variables declaration of same type  int x=10,y=20,z=0;// defining varaibles.  byte b;  short s;  int i;  long l;  //we can't declare multiple types of variables in a single statement;  }  } |

What is a local variable?

|  |
| --- |
| 1. It is a variable which is declared within a block existed in a class is called as local variable 2. Local variables are created either in java stack memory or native method stacks memory. 3. Local variables will not be initialized with default values, so you have to initialize them before usage. |

What error you will get if we use local variable without initializing it?

|  |
| --- |
| **Seven.java**  public class Seven {      public static void main(String[] args) {          int a,b,c; //3 local variables          System.out.println(a);      }  }  Compilation error:  Seven.java:4: error: variable a might not have been initialized  System.out.println(a);  ^  1 error |

Why java is called as extensible programming language?

|  |
| --- |
| Because in Java we can use native methods |

What are native methods?

|  |
| --- |
| 1. Methods which are written in other languages like C are called as native methods 2. It is possible to use native methods in Java   Ex: public static native long currentTimeMillis(); method of System class is a native method |

**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.

**Note:** these methods are called as formatting methods because these methods formats the given string by using converters and then prints the formatted string.

What is the first argument we should pass to the formatting method?

|  |
| --- |
| String literal |

How many arguments we can pass to the formatting method?

|  |
| --- |
| 1 or more arguments |

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 should 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():**

It is existed in the PrintStream class; it takes 1 or more arguments.

**FormattingMethods1.java**

|  |
| --- |
| import java.util.Date;  public class Seven {      public static void main(String[] args) {          int a,b,c; //3 local variables          a=10;          b=3;          c=a-b;          System.out.format("Hello%n");          System.out.format("%d-%d=%d%n",a,b,c);          System.out.println(a+"-"+b+"="+c);            //Native method usage          long l=System.currentTimeMillis();          System.out.println("l:\t"+new Date(l));      }  }  **Output:**  Hello  10-3=7  10-3=7  l: Sat Feb 08 16:26:14 IST 2025 |

**FormattingMethodsDemo**

|  |
| --- |
| import java.util.Date;  public class Seven  {      public static void main(String[] args)      {          String name="madhu";          int i=100;          float f=200;          double d=200;          char ch='A';          boolean bl=true;          System.out.printf("name:\t%s%n",name);          System.out.printf("i:\t%d%n",i);          System.out.printf("f:\t%.2f%n",f);          System.out.printf("d:\t%.3f%n",d);          System.out.printf("ch:\t%c%n",ch);          System.out.printf("bl:\t%s%n",bl);      }  }  Output:  name: madhu  i: 100  f: 200.00  d: 200.000  ch: A  bl: true |

**%s can be used for any type of object or value(see the below example)**

|  |
| --- |
| import java.util.Date;  public class Seven  {      public static void main(String[] args)      {          String name="madhu";          int i=100;          float f=200;          double d=200;          char ch='A';          boolean bl=true;          System.out.printf("name:\t%s%n",name);          System.out.printf("i:\t%s%n",i);          System.out.printf("f:\t%s%n",f);          System.out.printf("d:\t%s%n",d);          System.out.printf("ch:\t%s%n",ch);          System.out.printf("bl:\t%s%n",bl);      }  }  **Output:**  name: madhu  i: 100  f: 200.0  d: 200.0  ch: A  bl: true |

**FormatDemo3.java**

|  |
| --- |
| import static java.lang.System.\*;  class FormatDemo3  { public static void main(String args[])  { long n = 261011;  out.format("%d%n",n); // --> "261011"  out.format("%09d%n",n); // --> "000261011"  out.format("%9d%n",n); // --> " 261011"  out.format("%,9d%n",n); // --> " 261,011"  out.format("%+,9d%n%n",n); // --> " +261,011"  }  }  Output:  261011  000261011  261011  261,011  +261,011 |

**FormatDemo4.java(Example on alignment)**

|  |
| --- |
| public class Eight {      public static void main(String[] args) {          double pi =3.141593;          System.out.format("%f%n", pi);       // -->  "3.141593"          System.out.format("%.3f%n", pi);     // -->  "3.142"          System.out.format("%10.3f%n", pi);   // -->  "     3.142"          System.out.format("%-10.3f%n", pi);  // -->  "3.142     "// left justified          System.out.printf("%3f%n",pi);  //it displays total value          System.out.printf("Madhu Tech Skills...");      }  }  **Output:**  3.141593  3.142  3.142  3.142  3.141593  Madhu Tech Skills... |

**FormatDemo5.java**

|  |
| --- |
| import static java.lang.System.\*;  import java.util.\*;  class FormatDemo5  { public static void main(String args[])  { Date date=new Date();  out.println("Actual Date:\t"+date);  out.printf("Month=%tB %n",date);  out.printf("Day=%td%n",date);  out.printf("Year=%tY%n",date);  Calendar c = Calendar.getInstance();  out.format("%tB %td, %tY%n", c, c, c); // --> "July 09, 2013"  out.format("%tb %te, %tY%n", c, c, c); // --> "Jul 9, 2013"  out.format("%tl:%tM %tp%n", c, c, c); // --> "2:34 am"  out.format("%tD%n", c); // --> "07/09/13"//month/day/year  }  } |

**FormatDemo6.java**

|  |
| --- |
| import static java.lang.System.\*;  import java.util.\*;  class FormatDemo6  { public static void main(String args[])  { Date sd=new Date();  out.println("SystemDate:\t"+sd);  out.format("%td-%tB-%tY%n",sd,sd,sd);  out.format("%te-%tb-%ty%n",sd,sd,sd);  out.printf("%tl:%tM:%tS %tp",sd,sd,sd,sd);  }  } |

**Functions**

**What is a function?**

|  |
| --- |
| * A function is a block which contains re-usable set of statements * We write a function to perform a task * Once the function is written, to execute the statements of it, we have to call it |

A function is a block which has a name ends with pair of parentheses and the name is preceded by a data type (it is called as return type). It contains set of statements which are used to perform a task.

What is a static method?

|  |
| --- |
| * 1. If a method is declared by using static keyword then it is called as static method   2. We can call a static method in another static method of same class directly.   Note: we can’t call non-static method (instance method) in a static method directly. Even though it is existed in the same class. |

Who calls the main function?

|  |
| --- |
| JVM calls the main function |

What is the advantage of functions?

|  |
| --- |
| Functions provide several advantages in programming:   1. **Code Reusability** – Functions allow you to reuse code multiple times without rewriting it, reducing redundancy. 2. **Modularity** – They break a large program into smaller, manageable parts, making the code easier to understand and maintain. 3. **Improved Readability** – Well-named functions make the code more readable and self-explanatory. 4. **Easier Debugging** – Errors are easier to find and fix when the code is modular, as you can test individual functions separately. 5. **Scalability** – Functions help in building scalable applications by organizing code efficiently. 6. **Encapsulation** – They allow you to hide implementation details and expose only necessary functionality, improving security and organization. 7. **Avoiding Code Duplication** – Instead of writing the same logic multiple times, you can call a function whenever needed. 8. **Efficient Memory Use** – Functions help optimize memory usage by executing only when called and releasing resources when done. |

**A simple example on functions**

|  |
| --- |
| public class Nine  {      static void add(){          int a=10,b=3,c;          c=a+b;          System.out.printf("%d + %d = %d %n",a,b,c);      }      public static void main(String[] args) {          add();      }  }  **Output:**  10 + 3 = 13 |

**Another example on functions with parameters**

|  |
| --- |
| public class Nine  {      static void add()      {          int a=10,b=3,c;          c=a+b;          System.out.printf("%d + %d = %d %n",a,b,c);      }      static void sub(int a,int b){          int c;          c=a-b;          System.out.printf("%d - %d = %d %n",a,b,c);      }      public static void main(String[] args) {          add();          sub(100,200);      }  }  **Output:**  10 + 3 = 13  100 - 200 = -100 |

**What is a parameter?**

|  |
| --- |
| * + 1. Variables which are declared within parenthesis of a function are called as parameters.     2. Parameters are separated by comma (,)     3. After last parameter no need to give semicolon or comma |

**Example on Types of functions**

|  |
| --- |
| public class Nine  {      //function without parameters and without return type      static void add()      {          int a=10,b=3,c;          c=a+b;          System.out.printf("%d + %d = %d %n",a,b,c);      }      //function with parameters and without return type      static void sub(int a,int b){          int c;          c=a-b;          System.out.printf("%d - %d = %d %n",a,b,c);      }      //function with parameters and with return type      static int multi(int a,int b)      {          int c=a\*b;          return c;      }      //function without parameters and with return type      static int div()      {          int a=10,b=3;          return a/b;      }      public static void main(String[] args) {          add();          sub(100,200);   //to the sub function i am passing two arguments          int x=multi(10, 5); //to the multi() function i am passing two arguments and it is returning the result(int)          System.out.println("x:\t"+x);          int r1=x/2;          System.out.println("r1:\t"+r1);          System.out.println( div() );      }  }  **Output:**  10 + 3 = 13  100 - 200 = -100  x: 50  r1: 25  3 |

**Variable length arguments**

**Example on variable length argument**

|  |
| --- |
| public class Ten  {      static void display(int... enos)      {          System.out.println("..................");          for(int n:enos)          System.out.println(n);      }      public static void main(String[] args)      {          display();          display(1);          display(1,2);          display(1,2,3);      }  }  Output:  ..................  ..................  1  ..................  1  2  ..................  1  2  3 |

**Note:** A method can have variable length parameters with other parameters too, but one should ensure that there exists only one varargs parameter that should be written last in the parameter list of the method declaration. For example:

Another example on variable length arguments

|  |
| --- |
| public class Ten  {      static void display(String ename, int eno,int... marks)      {          System.out.printf(".....%s..with....eno....%d...marks....%n",ename,eno);          for(int mark:marks)          System.out.println(mark);      }      public static void main(String[] args)      {          display("Madhu.K",101,78,77,66,77,88);          display("Manish",102,78,77,66);          display("Lakshman",103,78,77,66);      }  }  **Output:**  .....Madhu.K..with....eno....101...marks....  78  77  66  77  88  .....Manish..with....eno....102...marks....  78  77  66  .....Lakshman..with....eno....103...marks....  78  77  66 |

What is a method?

|  |
| --- |
| A method is a function written in a class. |

What is the starting point of the program?

|  |
| --- |
| 1. Main method 2. In Java main method should be a static method 3. In Java main method return type should be only void.   Note: In Java you should declare the main method as public and static |

How to read total line at a time from keyboard?

|  |
| --- |
| readLine() method of DataInputStream class  (or)  readLine() method of BufferedReader class |

Reading data from keyboard by using readLine() method of DataInputStream class?

|  |
| --- |
| * readLine() method is a non-static method existed in DataInputStream class, so we have to call it by using object of DataInputStream class * readLine() method returns string only.. |

How to read the data from keyboard?

|  |
| --- |
| By using readLine () method we can read the data see the below example  import java.io.DataInputStream;  import java.io.IOException;  public class Eleven {      public static void main(String[] args) throws IOException      {          //Creating object for DataInputStream class          DataInputStream dis=new DataInputStream(System.in);          System.out.print("Enter your name:\t");          String name=dis.readLine();  //  name <-  = <- "sambasivarao" <- readLine()  <- sambasivarao <- keyboard          System.out.println("Hi "+name);        }  }  Output: |

**Operators**

What is an operator?

|  |
| --- |
| Operator is a symbol, which is used to perform an operation.  **Different types of operators supported by Java**  Ex: +, -, \*, /, %, (arithmetic operators)  Ex: <,>,<=,>=,==,!=,instanceof (relational operators  Ex: &&,||, ! (logical operators)  Ex: &,|,^,<<,>>, ~, >>>(Bitwise Operators)(zero fill right shift operator)  Ex: = (assignment operator)  Ex: new (memory allocation operator), . (dot is called as memory access operator), () type cast operator  Ex: ++,--, +=,-=,\*=,/=,%= (short cut operators) |

Example on Arithmetic Operators

|  |
| --- |
| import java.io.DataInputStream;  import java.io.IOException;  public class Eleven {      public static void main(String[] args) throws IOException      {          DataInputStream dis=new DataInputStream(System.in);          System.out.print("Enter int:\t");          String s1=dis.readLine();  // a <-  = <- "10" <- readLine()  <- 10 <- keyboard          int a=Integer.parseInt(s1);          System.err.print("Enter another int value:\t");          String s2=dis.readLine();          int b=Integer.parseInt(s2);          System.out.println(a+b);          System.out.println(a-b);          System.out.println(a\*b);          System.out.println(a/b);          System.out.println(a%b);      }  }  Output:  Enter int: 10  Enter another int value: 2  12  8  20  5  0 |

What is the drawback of readLine() of DataInputStream class? And why it is deprecated?

|  |
| --- |
| * by using this method, we can read only the characters which are in a range of 0 to 255 (ASCII character set). * this method can’t read other than ASCII character set (i.e. it can’t read all the characters of Unicode character set). * The method has been deprecated since Java 1.1 because it does not properly convert bytes to characters in a way that supports internationalization. * readLine() reads input as a **byte stream** and does not properly decode characters according to the character encoding (e.g., UTF-8). This can lead to **incorrect character conversions**, especially for non-ASCII characters. |

readLine() method of BufferedReader class

|  |
| --- |
| * 1. by using this method, we can read the characters which are in Unicode character set. |

Example on usage of readLine() method of BufferedReader class

|  |
| --- |
| import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  public class Eleven {      public static void main(String[] args) throws IOException      {          InputStreamReader isr=new InputStreamReader(System.in);          BufferedReader br=new BufferedReader(isr);          System.out.print("Enter int:\t");          String s1=br.readLine();  // a <-  = <- "10" <- readLine()  <- 10 <- keyboard          int a=Integer.parseInt(s1);          System.err.print("Enter another int value:\t");          String s2=br.readLine();          int b=Integer.parseInt(s2);          System.out.format("%d + %d = %d%n",a,b,a+b);          System.out.format("%d - %d = %d%n",a,b,a-b);          System.out.format("%d \* %d = %d%n",a,b,a\*b);          System.out.format("%d / %d = %d%n",a,b,a/b);          System.out.format("%d %% %d = %d%n",a,b,a%b);        }  }  **Output:**  Enter int: 12  Enter another int value: 3  12 + 3 = 15  12 - 3 = 9  12 \* 3 = 36  12 / 3 = 4  12 % 3 = 0 |

Where the parseInt() method is existed?

|  |
| --- |
| * It is existed in Integer class * It takes string as an argument and converts it into int value and returns that int value |

Relational Operators

|  |
| --- |
| Relational Operators are used to compare 2 values(operands), after comparison these operators gives us a Boolean value(true/false) |

Example on relational operators

|  |
| --- |
| import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  public class Twelve  {      public static void main(String[] args) throws IOException       {          InputStreamReader isr = new InputStreamReader(System.in);          BufferedReader br = new BufferedReader(isr);          System.out.print("Enter a number :\t");          int i = Integer.parseInt(br.readLine());          System.out.print("Enter another number :\t");          int j = Integer.parseInt(br.readLine());          System.out.println("i:\t "+i);          System.out.printf("j:\t%d %n",j);          // test case-1          //i=10          //j=5          boolean b1= i>j; //b1=true          boolean b2=i>=j; // (i>j or i==j)  b2=true          boolean b3=i<j;  //b3=false          boolean b4= i<=j; //i<j or i==j  b4=false          boolean b5=i==j; //b5=false          boolean b6=i!=j; //b6=true          System.out.println("i>j:\t"+b1);          System.out.println("i>=j:\t"+b2);          System.out.println("i<j:\t"+b3);          System.out.println("i<=j:\t"+b4);          System.out.println("i==j:\t"+b5);          System.out.println("i!=j:\t"+b6);      }  }  Output:  Enter a number : 10  Enter another number : 5  i: 10  j: 5  i>j: true  i>=j: true  i<j: false  i<=j: false  i==j: false  i!=j: true |

**What is an operand?**

|  |
| --- |
| It is a value on which the operator performs an operation.  Ex: 10+20 Here 10,20 are operands and + is an operator |

**Logical Operators (Boolean operators)**

|  |
| --- |
| * 1. Logical operators are used to perform operation on Boolean values only   2. These operators gives us Boolean values as a result. * Logical and operator (&&): it gives us true if both operands are true, otherwise false * Logical or operator (||):it gives us true if any one or both operands are true, otherwise false * Logical not operator (!): it returns true if operands is false vice versa. |

How parseBoolean() method works?

|  |
| --- |
| * 1. It converts “TRUE” to true and returns it   2. It converts “true” to true and returns it   3. If any string other than “true” (case-insensitive) is passed then It returns false |

Example on parseBoolean() method which returns true

|  |
| --- |
| public class Fourteen  {      public static void main(String[] args) {          System.err.println(Boolean.parseBoolean("true"));          System.err.println(Boolean.parseBoolean("truE"));          System.err.println(Boolean.parseBoolean("trUE"));          System.err.println(Boolean.parseBoolean("True"));          System.err.println(Boolean.parseBoolean("TRUE"));      }  }  Output:  true  true  true  true  true |

Example on parseBoolean() method which returns false

|  |
| --- |
| public class Fourteen  {      public static void main(String[] args) {          System.err.println(Boolean.parseBoolean("false"));          System.err.println(Boolean.parseBoolean("madhu"));          System.err.println(Boolean.parseBoolean("1234"));          System.err.println(Boolean.parseBoolean("Flaws"));          System.err.println(Boolean.parseBoolean("FALSE"));          System.err.println(Boolean.parseBoolean("TOKKALEY"));          System.err.println(Boolean.parseBoolean("1"));          System.err.println(Boolean.parseBoolean("C"));      }  }  Output:  false  false  false  false  false  false  false  false |

Example on logical operators

|  |
| --- |
| import java.io.BufferedReader;  import java.io.InputStreamReader;  public class Thirteen {      public static void main(String[] args) throws Exception      {          InputStreamReader isr=new InputStreamReader(System.in);          BufferedReader br=new BufferedReader(isr);          System.out.print("Enter a boolean value:\t");          boolean a = Boolean.parseBoolean(br.readLine());  //"true"          System.out.print("Enter another boolean value:\t");          boolean b = Boolean.parseBoolean(br.readLine());  //"true"          System.out.println("a:\t"+a);          System.out.println("b:\t"+b);          System.out.println("a&&b:\t"+(a&&b));          System.out.println("a||b:\t"+(a||b));          System.out.println("!a:\t"+!a);          System.out.println("!b:\t"+!b);        }  }  Output:  Enter a boolean value: madhu  Enter another boolean value: TrUe  a: false  b: true  a&&b: false  a||b: true  !a: true  !b: false |

How to read a single character (within a range of 0 to 255)?

|  |
| --- |
| public class Fifteen {      public static void main(String[] args) throws Exception      {          System.out.println("Enter any character:\t");          int n=System.in.read();  //it reads only single character within range of 0 t 255 and returns it’s Unicode value( or ascii value)          System.out.println("n:\t"+n);      }  }  Output:  Enter any character:  A  n: 65 |

Does JVM is able to display Telugu letter అ(see the below example)

|  |
| --- |
| import java.io.BufferedReader;  import java.io.InputStreamReader;  public class Fifteen {      public static void main(String[] args) throws Exception      {          char ch1='\u0C05'; //in ch1 i am storing telugu letter అ it's unicode value is 3077          System.out.println("int(ch1):\t"+(int)ch1);          System.out.println("int:\t"+ch1);      }  }  **Output:**  int(ch1): 3077  int: ? |

Can we declare a local variable as static?

|  |
| --- |
| No, local variables can’t be either static or instance |

What is a literal?

|  |
| --- |
| 1. It is a value used in a program   Different types of literals supported by Java   1. Integer literal (10,20) 2. Float literal (10.0,2.0) 3. String literal(“madhu”,”123”) 4. Boolean literal(true,false) 5. Character literal (‘a’,’1’) |

What is a token?

|  |
| --- |
| Each and every individual unit in a program is called as token.  The list of tokens supported by Java   1. Keywords 2. Identifiers 3. Literals 4. Operators 5. Separators 6. Special characters 7. Comments |

What is a comment?

|  |
| --- |
| A **comment** in Java is a piece of text in the code that is ignored by the compiler and is used to provide explanations, notes, or documentation for better code readability. Java supports three types of comments:  **1. Single-line Comment (//)**   * Used for short, single-line explanations. * Anything after // on the same line is ignored by the compiler.   **2. Multi-line Comment (/\* ... \*/)**   * Used for longer explanations spanning multiple lines.   **3. Javadoc Comment (/\*\* ... \*/)**   * Used for generating documentation using the **Javadoc** tool. * Typically used for describing classes, methods, and parameters. |

Example on documentation comments

Comments.java

|  |
| --- |
| package pack1; //it is a package statement: by using it i am creating pack1 package  /\*  importing all the static methods and fields of Integer class,  so that we can use them direclty without using class name.   \*/  import static java.lang.Integer.\*;  //this is a main method class  public class Comments {      /\*\*       main() method is the starting point of the program       \*/      public static void main(String[] args)      {                  System.out.println("Comments example...");      }  } |

MyMath.java

|  |
| --- |
| package pack1;  /\*\*  it is a class which contains methods to perform  arithmetic operations. it is existed in pack1 package   \*/  public class MyMath {      public int add(int a,int b){          return a+b;      }      public int sub(int a,int b){          return a-b;      }      public int multi(int a,int b){          return a\*b;      }  } |

How to generate documentation for the above programs(MyMath.java and Comments.java)

|  |
| --- |
| We have to use Javadoc tool to generate documentation |

How to see the documentation generated Javadoc command?

|  |
| --- |
| Click on index.html file to see the documentation  If you run the index.html (just double click it to run) page then you can see the documentation like below |

What is String in “String args[]”?

|  |
| --- |
| It is a pre-defined class existed in java.lang package |

What is a System in “System.out.println()”?

|  |
| --- |
| It is a pre-defined class existed in java.lang package |

What is a println() in “System.out.println()”?

|  |
| --- |
| It is a pre-defined method |

What is a reference variable?

|  |
| --- |
| If we declare a variable by using class name as a data type. Then those variables are called as reference variables (like pointer variables in C/C++). |

What is a reference data type?

|  |
| --- |
| It is a data type which is used to declare reference variables. |

Conditional statements allow the program to make decisions based on certain conditions. The main conditional statements available in Java are:

* + 1. If statement
    2. If…else… statement
    3. Else…if. Ladder (if..else..if) (if..else if…else)
    4. Nested if statement
    5. Switch statement
    6. Ternary operator
    7. While
    8. for
    9. Do..while
    10. Foreach

If statement?

|  |
| --- |
| If you want to execute one or more statements whenever a condition is true then we write if statement.  Syntax-1: if (condition)  Single-statement-1;  Syntax-2: if (condition){  Stmt-1;  Stmt-2;  …………  Stmt-n;  } |

Example on if statement(condition)

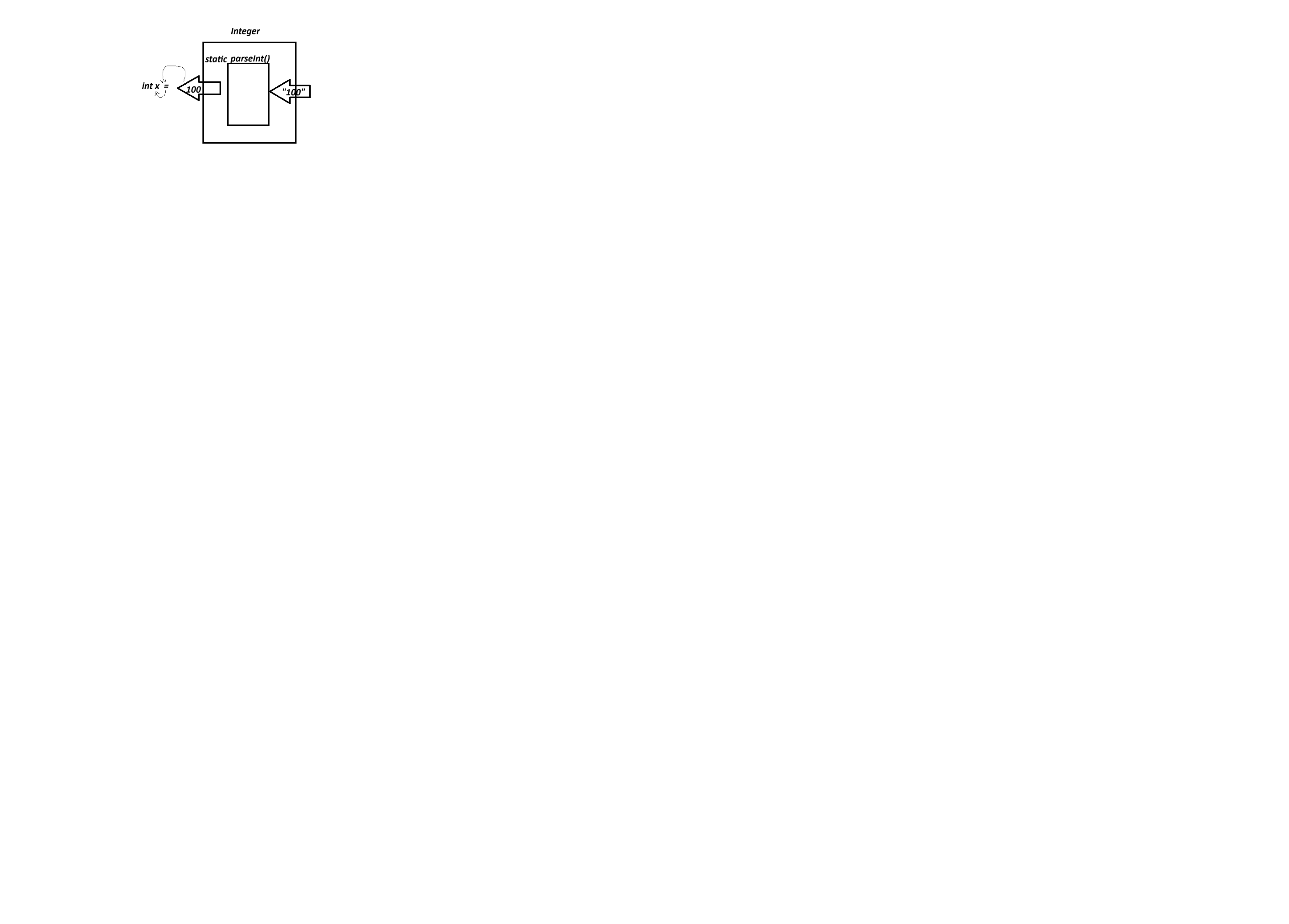
|  |
| --- |
| import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  public class SeventeenIfDemo {      public static void main(String[] args) throws IOException {          InputStreamReader isr=new InputStreamReader(System.in);          BufferedReader br= new BufferedReader(isr);          System.out.print("Enter a value:\t");          String s1=br.readLine();          int x=Integer.parseInt(s1);          System.out.print("Enter another value:\t");          String s2=br.readLine();          int y=Integer.parseInt(s2);          if (x>y){              System.out.println("true");          }      }  }  Output: |

What is Integer in Integer.parseInt() statement?

|  |
| --- |
| It is a pre-defined class existed in java.lang package |

What is parseInt() in Integer.parseInt() statement?

|  |
| --- |
| * It is a pre-defined method existed in Integer class * This method takes string as an argument and returns int(primitive int) value. |



how we can call the static method?

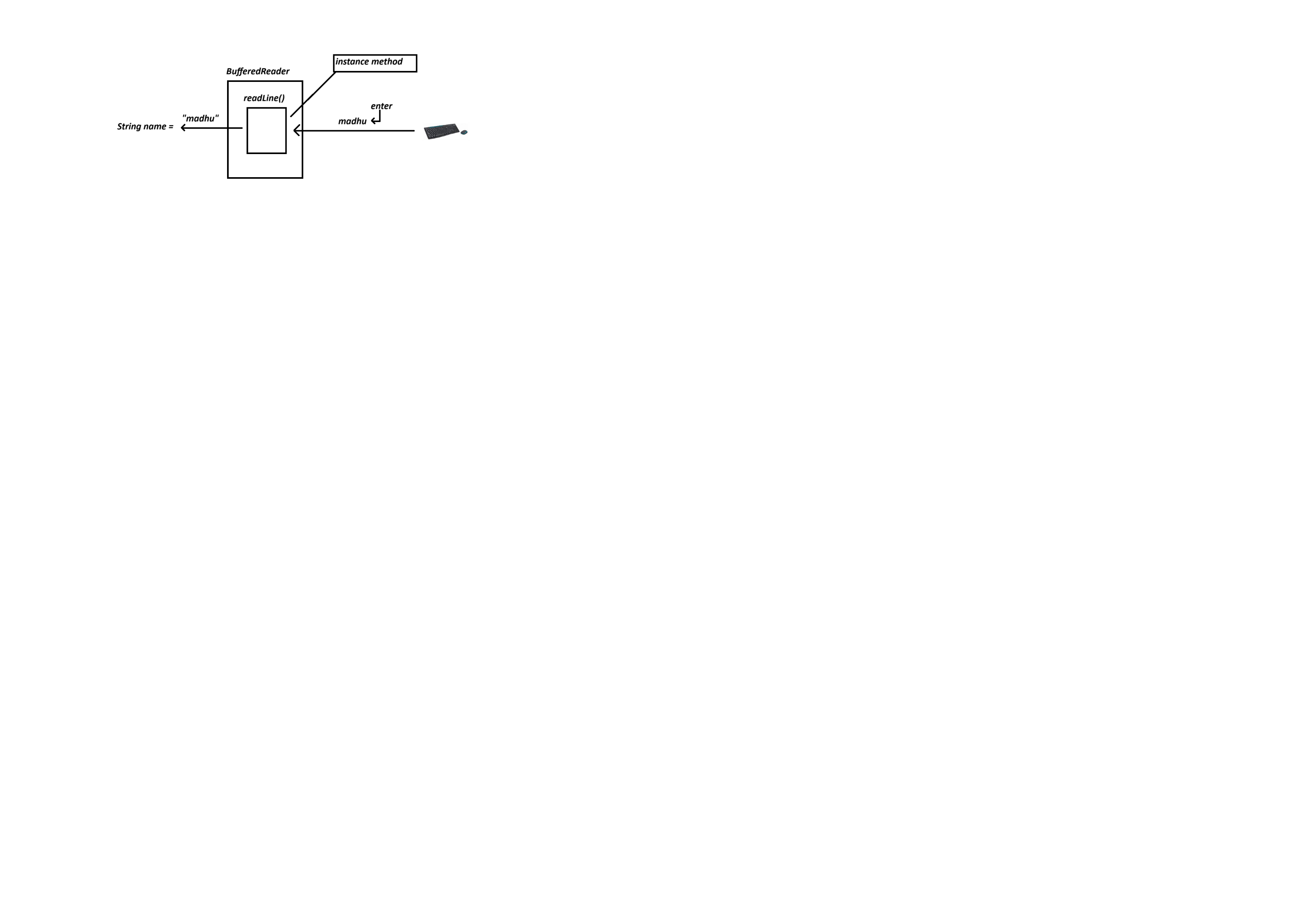
|  |
| --- |
| We can call the static method by using class name or a reference variable name(don’t bother whether it contains address of the object or null)  Ex: int x=Integer.parseInt(“100:); |

Why we write a class?

|  |
| --- |
| 1. We write a class to create objects |

What is the job of readLine() method?

|  |
| --- |
| * 1. It waits in the terminal (console monitor) to take the input from the keyboard   2. Whenever you press the enter after typing something then readLine() method takes it and returns as a string |



How to call instance method?

|  |
| --- |
| * + 1. We can call instance method by using object only     2. We can all instance method by using reference variable which must contain object address. |

Why NullPointerException will raise during program execution?

|  |
| --- |
| If you call instance method by using reference variable which contains null, then we will get this exception. |

If…else…

|  |
| --- |
| import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  public class Eighteen {      public static void main(String[] args) throws IOException      {          InputStreamReader isr=new InputStreamReader(System.in);          BufferedReader br=new BufferedReader(isr);          System.out.print("Enter a value:\t");          float x=Float.parseFloat(br.readLine());          System.out.print("Enter anothor value:\t");          float y=Float.parseFloat(br.readLine());          if (x > y)              System.out.printf("%.2f is Bigger Number",x);          else              System.out.printf("%.2f is Bigger Number",y);      }  }  **Output:**  Enter a value: 10.90  Enter anothor value: 11.00  11.00 is Bigger Number |

Else…if.. ladder and switch statement

|  |
| --- |
| If you want to execution one option among many then we can use either else…if ladder or switch statement. |

What is static import?

|  |
| --- |
| It is used to import static variables and static methods of a class into our program. If we import them then we can use them directly without using class name. |

What is the scope and life of local variable?

|  |
| --- |
| * + - 1. Life means: time between creation and destruction of a variable       2. Local variables are created whenever the declaration statement is executed by JVM and destroyed from stack frame whenever the block where declared execution is completed.       3. Scope: in which block or blocks it is accessible is called as the scope of a variable.       4. Local variables are accessible only within the block where these variables are declared. |

Example on else..if... ladder usage to performing arithmetic operation based on the option you have given

|  |
| --- |
| import java.io.BufferedReader;  import java.io.InputStreamReader;  import static java.lang.Integer.parseInt;  public class Nineteen {      public static void main(String[] args) throws Exception      {          BufferedReader br=new BufferedReader(new InputStreamReader(System.in));          System.out.println("1.add");          System.out.println("2.sub");          System.out.println("3.multiply");          System.out.println("4.div");          System.out.println("5.modulus");          System.out.println("Option Please:\t");          int opt=parseInt(br.readLine());          if(opt>=1 && opt<=5)          {              System.out.print("Enter first int:\t");              int a=parseInt(br.readLine());              System.out.print("Enter second int:\t");              int b=parseInt(br.readLine());              if(opt==1)                  System.err.printf("%d + %d = %d",a,b,a+b);              else if(opt==2)                  System.err.printf("%d - %d = %d",a,b,a-b);              else if(opt==3)                  System.err.printf("%d \* %d = %d",a,b,a\*b);              else if(opt==4)                  System.err.printf("%d / %d = %d",a,b,a/b);              else                  System.err.printf("%d %% %d = %d",a,b,a%b);          }else{              System.out.println("Given Option is Invalid");          }      }  }  **Output:**  1.add  2.sub  3.multiply  4.div  5.modulus  Option Please:  10  Given Option is Invalid |

Example on usage of switch

|  |
| --- |
| import java.io.BufferedReader;  import java.io.InputStreamReader;  import static java.lang.Integer.parseInt;  public class Nineteen {      public static void main(String[] args) throws Exception      {          BufferedReader br=new BufferedReader(new InputStreamReader(System.in));          System.out.println("1.add");          System.out.println("2.sub");          System.out.println("3.multiply");          System.out.println("4.div");          System.out.println("5.modulus");          System.out.println("Option Please:\t");          int opt=parseInt(br.readLine());          if(opt>=1 && opt<=5)          {              System.out.print("Enter first int:\t");              int a=parseInt(br.readLine());              System.out.print("Enter second int:\t");              int b=parseInt(br.readLine());              switch (opt) {                  case 1:                      System.err.printf("%d + %d = %d",a,b,a+b);                      break;                  case 2:                      System.err.printf("%d - %d = %d",a,b,a-b);                      break;                  case 3:                      System.err.printf("%d \* %d = %d",a,b,a\*b);                      break;                  case 4:                      System.err.printf("%d / %d = %d",a,b,a/b);                      break;                  case 5:                      System.err.printf("%d %% %d = %d",a,b,a%b);              }          }else{              System.out.println("Given Option is Invalid");          }      }  }  Output:  1.add  2.sub  3.multiply  4.div  5.modulus  Option Please:  2  Enter first int: 10  Enter second int: 2  10 - 2 = 8 |

Switch Statement (with advanced features) without break statement but behaves like old switch statement

|  |
| --- |
| import java.io.BufferedReader;  import java.io.InputStreamReader;  import static java.lang.Integer.parseInt;  public class Nineteen {      public static void main(String[] args) throws Exception      {          BufferedReader br=new BufferedReader(new InputStreamReader(System.in));          System.out.println("1.add");          System.out.println("2.sub");          System.out.println("3.multiply");          System.out.println("4.div");          System.out.println("5.modulus");          System.out.println("Option Please:\t");          int opt=parseInt(br.readLine());          System.out.print("Enter first int:\t");          int a=parseInt(br.readLine());          System.out.print("Enter second int:\t");          int b=parseInt(br.readLine());          switch (opt) {                  case 1->System.err.printf("%d + %d = %d",a,b,a+b);                  case 2->System.err.printf("%d - %d = %d",a,b,a-b);                  case 3->System.err.printf("%d \* %d = %d",a,b,a\*b);                  case 4->System.err.printf("%d / %d = %d",a,b,a/b);                  case 5->System.err.printf("%d %% %d = %d",a,b,a%b);                  default->System.out.println("Invalid Option..");              }      }  }  Output:  1.add  2.sub  3.multiply  4.div  5.modulus  Option Please:  1  Enter first int: 10  Enter second int: 2  10 + 2 = 12 |

Switch expression

|  |
| --- |
| A switch statement can return a value. These kind of switch statements are called as switch expressions  Note: every case must yield a value or thow an exception and it must contain default case |

Example on switch expression

|  |
| --- |
| import java.io.BufferedReader;  import java.io.InputStreamReader;  import static java.lang.Integer.parseInt;  public class Nineteen {      public static void main(String[] args) throws Exception      {          BufferedReader br=new BufferedReader(new InputStreamReader(System.in));          System.out.println("1.add");          System.out.println("2.sub");          System.out.println("3.multiply");          System.out.println("4.div");          System.out.println("5.modulus");          System.out.println("Option Please:\t");          int opt=parseInt(br.readLine());          System.out.print("Enter first int:\t");          int a=parseInt(br.readLine());          System.out.print("Enter second int:\t");          int b=parseInt(br.readLine());          int result=switch (opt) {                  case 1->{                      int c=a+b;                      System.out.println("chachinodaaa...");                      yield c;}                  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);      }  }  Output:  1.add  2.sub  3.multiply  4.div  5.modulus  Option Please:  1  Enter first int: 10  Enter second int: 2  chachinodaaa...  Result: 12 |

**Looping Statements (or) Iterative Statements**

|  |
| --- |
| **If we want to execute one or more statements repeatedly until the given condition becomes false.**  **Iterative statements are**   * **while** * **do...while…** * **for** * **foreach** |

**While loop**

|  |
| --- |
| **Syntax-1: while(condition)**  **stmt-1**  **Syntax-2: while(condition){**  **stmt-1;**  **stmt-2;**  **…………**  **stmt-n;**  **}** |

**Basic example on while loop**

|  |
| --- |
| **public class TwentyWhileDemo1 {**  **public static void main(String[] args) {**  **/\***  **1. declare a variable and intialize it with a starting value**  **Task: print welcome 10 times**  **1,2,3,4,5,6,7,8,9,10**  **2. use that variable in the condition**  **\*/**  **int i=1; //i=1,2,3**  **int n=10; //n=1**  **while(i<=n){**  **System.out.println("welcome");**  **i++;  //i=i+1**  **}**  **System.out.println("Iyyipayee..");**  **}**  **}**  **Output:**  **welcome**  **welcome**  **welcome**  **welcome**  **welcome**  **welcome**  **welcome**  **welcome**  **welcome**  **welcome**  **Iyyipayee..** |

**Example to print 1 to 10 natural numbers**

|  |
| --- |
| **public class TwentyWhileDemo1 {**  **public static void main(String[] args) {**  **/\***  **1. declare a variable and intialize it with a starting value**  **Task: print welcome 10 times**  **1,2,3,4,5,6,7,8,9,10**  **2. use that variable in the condition (Note: write the condition in such a way that it should become false once the goal is reached)**  **3.  update the value of variable**  **\*/**  **int i=1; //i=1,2,3**  **int n=10; //n=1**  **while(i<=n){**  **System.out.println(i);**  **i++;  //i=i+1**  **}**  **System.out.println("Iyyipayee..");**  **}**  **}**  **Output:**  **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** |

**Program to print 1 to 10 even numbers**

|  |
| --- |
| **public class TwentyWhileDemo1 {**  **public static void main(String[] args) {**  **/\***  **1. declare a variable and intialize it with a starting value**  **Task: print welcome 10 times**  **1,2,3,4,5,6,7,8,9,10**  **2. use that variable in the condition (Note: write the condition in such a way that it should become false once the goal is reached)**  **3.  update the value of variable**  **\*/**  **int i=2; //i=2,4**  **int n=10; //n=1**  **while(i<=n){**  **System.out.println(i);**  **i=i+2;**  **}**  **System.out.println("Iyyipayee..");**  **}**  **}**  **Output:**  **2**  **4**  **6**  **8**  **10**  **Iyyipayee..** |

**Example to print 1 to 10 odd numbers**

|  |
| --- |
| **public class TwentyWhileDemo1 {**  **public static void main(String[] args) {**  **/\***  **1. declare a variable and intialize it with a starting value**  **Task: print welcome 10 times**  **1,2,3,4,5,6,7,8,9,10**  **2. use that variable in the condition (Note: write the condition in such a way that it should become false once the goal is reached)**  **3.  update the value of variable**  **\*/**  **int i=1; //i=1,3....**  **int n=10; //**  **while(i<=n){**  **System.out.println(i);**  **i=i+2;**  **}**  **System.out.println("Iyyipayee..");**  **}**  **}**  **Output:**  **1**  **3**  **5**  **7**  **9**  **Iyyipayee..** |

**Program to print factors of a given number**

|  |
| --- |
| **public class TwentyWhileDemo1 {**  **public static void main(String[] args) {**  **int i=1; //i=1,2,3...**  **int n=10; //**  **while(i<=n){**  **if(n%i ==0)**  **System.out.println(i);**  **i++;**  **}**  **System.out.println("Iyyipayee..");**  **}**  **}**  **Output:**  1  2  5  10  Iyyipayee.. |

**Another example on while loop**

|  |
| --- |
| **import java.io.BufferedReader;**  **import java.io.InputStreamReader;**  **import static java.lang.Integer.parseInt;;**  **public class TwentyTwoLoop**  **{**  **static int count(int n)**  **{**  **//test-1: n=123,12,1**  **int count=0; //count=0,1,2,3**  **while (n>0)**  **{**  **n=n/10;**  **count++;**  **}**  **return count;**  **}**  **public static void main(String[] args) throws Exception**  **{**  **BufferedReader br=new BufferedReader(new InputStreamReader(System.in));**  **System.out.println("Enter n value:\t");**  **int n=parseInt(br.readLine());**  **int c=count(n);**  **System.out.println("count:\t"+c);**  **int d=0;**  **int sum=0;**  **while(n>0)**  **{**  **d=n%10;**  **n=n/10;**  **System.out.println();**  **sum=sum+(int)Math.pow(d,c);**  **}**  **System.out.println(sum);**  **}**  **}**  **Output:**  **Enter n value:**  **1634**  **count: 4**  **1634** |

**For loop**

|  |
| --- |
| **Syntax-1: for(initialization;condition;updation)**  **Stmt-1**  **Syntax-2: for(initialization;condition;updation){**  **Stmt-1**  **……….**  **Stmt-n**    **}** |

**Example on for loop**

|  |
| --- |
| **public class ForDemo {**  **public static void main(String[] args)**  **{**  **int n=10;**  **//i=1,2**  **System.out.println("for loop");**  **for(int i=1;i<=n;i++)**  **{   System.out.println(i);**  **}**  **}**  **}**  **Output:**  **for loop**  **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** |

**Another example on for loop**

|  |
| --- |
| **public class ForDemo {**  **public static void main(String[] args)**  **{**  **int n=10;**  **int i=1;**  **//i=1,2**  **System.out.println("for loop");**  **for(;i<=n;)**  **{   System.out.println(i);**  **i++;**  **}**  **}**  **}**  **Output:**  **for loop**  **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10** |

**Do…while…**

|  |
| --- |
| In do…while… we will enter into the loop first, after that condition checking will be done.  Syntax-1: do{  Stmt-1;  …………;  Stmt-n;  }while(condition); |

**Example on do…while… and switch statements**

|  |
| --- |
| import java.io.BufferedReader;  import java.io.InputStreamReader;  import static java.lang.Integer.parseInt;  public class ForDemo  {      public static void main(String[] args) throws Exception      {          String opt3="";          System.out.println("BackBenchers...");          BufferedReader br=new BufferedReader(new InputStreamReader(System.in));          do{              System.out.println("1. Veg");              System.out.println("2. Non-veg");              System.out.print("Sir! Veg(1) or Non-veg(2)?:\t");              int opt1=parseInt(br.readLine());              switch (opt1) {                  case 1->{                      float plate\_meal=150;                      float panner\_biryani=350;                      float vegFriedRice=200;                      System.out.println("Veg-Menu");                      System.out.printf("1. Meals - %.2f %n",plate\_meal);                      System.out.printf("2. Panner Biryani - ( %.2f ) %n",panner\_biryani);                      System.out.printf("3. Veg-fried rice - ( %.2f ) %n",vegFriedRice);                      System.out.print("Order please!");                      int opt2=parseInt(br.readLine());                      int plate\_count=0;                      switch (opt2) {                          case 1->{                              System.out.print("How plates sir!:\t");                              plate\_count=parseInt(br.readLine());                              System.out.printf("Billing Amount:\t%.2f",(plate\_count\*plate\_meal));                          }                          case 2->{                              System.out.print("How plates sir!:\t");                              plate\_count=parseInt(br.readLine());                              System.out.printf("Billing Amount:\t%.2f",(plate\_count\*panner\_biryani));                          }                          case 3->{                              System.out.print("How plates sir!:\t");                              plate\_count=parseInt(br.readLine());                              System.out.printf("Billing Amount:\t%.2f",(plate\_count\*vegFriedRice));                          }                          default->{                              System.out.println("Not Available Sir!");                          }                      }                  }                  case 2->{                      float plate\_nonVegMeal=250;                      float chicken\_biryani=450;                      float eggFriedRice=250;                      System.out.println("Non-Veg-Menu");                      System.out.printf("1. Non-Veg-Meals - %.2f %n",plate\_nonVegMeal);                      System.out.printf("2. Chicken-Biryani - ( %.2f ) %n",chicken\_biryani);                      System.out.printf("3. Egg-Fried-Rice - ( %.2f ) %n",eggFriedRice);                      System.out.print("Order please!");                      int opt2=parseInt(br.readLine());                      int plate\_count=0;                      switch (opt2) {                          case 1->{                              System.out.print("How plates sir!:\t");                              plate\_count=parseInt(br.readLine());                              System.out.printf("Billing Amount:\t%.2f",(plate\_count\*plate\_nonVegMeal));                          }                          case 2->{                              System.out.print("How plates sir!:\t");                              plate\_count=parseInt(br.readLine());                              System.out.printf("Billing Amount:\t%.2f",(plate\_count\*chicken\_biryani));                          }                          case 3->{                              System.out.print("How plates sir!:\t");                              plate\_count=parseInt(br.readLine());                              System.out.printf("Billing Amount:\t%.2f",(plate\_count\*eggFriedRice));                          }                          default->{                              System.out.println("Not Available");                          }                      }                  }                  default->{                      System.out.println("Sorry sir! Not Available!");                  }              }              System.out.println("\nInka yemanna kavala sir!(yes/no)");              opt3=br.readLine();          }while(  opt3.equals("yes") );          System.out.println("Thank u sir! Visit Again!");      }  } |

Nested loops

|  |
| --- |
| We can write a loop within another loop, it is called as nested loops  Syntax:  for(initialization;condition;update){  stmt-1;  stmt-2;  for(initialization;condition;update){  stmt-1;  ………..  stmt-n;  }  Stmt-3;  ………..  Stmt-n;  } |

Example to print the below pattern

|  |
| --- |
| 1 1 2 3 4 5  2 1 2 3 4 5  3 1 2 3 4 5  4 1 2 3 4 5  5 1 2 3 4 5  ...end...  Program:  public class NestedForDemo {      public static void main(String[] args)      {          for(int i=1;i<=5;i++)          {              System.out.printf("%-8d",i);              for(int j=1;j<=5;j++)              {                  System.out.printf("%-3d",j);              }              System.out.println();          }          System.out.println("...end...");      }  }  /\*   1       1  2  3  4  5  \_   2       1  2  3  4  5  \_   3       1  2  3  4  5  \_   4       1  2  3  4  5  \_   5       1  2  3  4  5  \_   ...end...     \*/ |

Example to print the below pattern

|  |
| --- |
| 1 1  2 1 2  3 1 2 3  4 1 2 3 4  5 1 2 3 4 5  ...end...  Program:  public class NestedForDemo {      public static void main(String[] args)      {          for(int i=1;i<=5;i++)          {              System.out.printf("%-8d",i);              for(int j=1;j<=i;j++)              {                  System.out.printf("%-3d",j);              }              System.out.println();          }          System.out.println("...end...");      }  } |

Program to print the below pattern

|  |
| --- |
| \*  \* \*  \* \* \*  \* \* \* \*  \* \* \* \* \*  ...end...  Program  public class NestedForDemo {      public static void main(String[] args)      {          for(int i=1;i<=5;i++)          {              for(int j=1;j<=i;j++)              {                  System.out.printf("%-3s","\*");              }              System.out.println();          }          System.out.println("...end...");      }  } |

Write a program to print the below pattern

|  |
| --- |
| 1  2 2  3 3 3  4 4 4 4  5 5 5 5 5  ...end...  Program:  public class NestedForDemo {      public static void main(String[] args)      {          for(int i=1;i<=5;i++)          {              for(int j=1;j<=i;j++)              {                  System.out.printf("%-3d",i);              }              System.out.println();          }          System.out.println("...end...");      }  } |

Write a program to print the pyramid shape like below

|  |
| --- |
| Program  public class Pattern  {      public static void main(String[] args)      {          int n=5;          //n=5          //i=1,2,3,4,5,6          for(int i=1;i<=n;i++)          {              //j=1              for(int j=1;j<=(n-i);j++)              System.out.printf(" ");              //k=1,2,3              for(int k=1;k<=i;k++)                  System.out.printf("%-2s","\*");                System.out.println();          }      }  } |

Write a program to print the below pattern

|  |
| --- |
| Program  public class NestedLoop {      public static void main(String[] args)      {          int n=4;          int k=1;          for(int i=1;i<=n;i++)          {              for(int j=1;j<=i;j++)              {                  System.out.printf("%-2d",k);                  k++;              }              System.out.println();          }      }  }  /\*  1  23  456  78910  \*/ |

Write a program to print the below pattern

|  |
| --- |
| public class NestedLoop {      public static void main(String[] args)      {          int n=6;          int k=1;          for(int i=1;i<=n;i++)          {              for(int j=1;j<=i;j++)              {                  System.out.printf("%-2d",k);                  if(k==1)                      k=0;                  else                      k=1;              }              System.out.println();          }      }  } |

Write a program to print the below pattern

|  |
| --- |
| public class NestedLoop {      public static void main(String[] args)      {          int n=6;          int k=1;          for(int i=1;i<=n;i++)          {              for(int j=1;j<=i;j++)              {                  if ((i+j)%2==0)                      System.out.printf("%-2d",1);                  else                      System.out.printf("%-2d",0);                }              System.out.println();          }      }  } |

Write a program to print the below pattern



|  |
| --- |
| public class NestedLoop {      public static void main(String[] args)      {          int n=4;          //i=1          //n=4          for(int i=1;i<=n;i++)          {              for(int j=1;j<=n;j++)              {                  //j=1,2,3,4                  if (i==j || j== n-(i-1))                      System.out.printf("%s","\*");                  else                      System.out.print("   ");              }              System.out.println();          }      }  } |

**Arrays**

What is an array?

|  |
| --- |
| * 1. Till now to store values we declare variables. And memory allocation for the variables will be done in different addresses. So memory allocation is not contiguous. To allocate memory contiguously we use arrays.      * 1. Array is a collection similar data element (data items) stored at contiguous memory locations and shared by a common name.   2. After array creation the default values will be stored in each block.   Advantages of array   * + 1. Accessing/modifying elements is fast     2. We can get/access or modify the elements randomly   **Dis-advantages**   * + - 1. Array size is fixed |

How to create an array?

|  |
| --- |
| We have different ways to create an array in Java? |

In which memory arrays will be created?

|  |
| --- |
| Arrays will be created in heap memory |

What will be considered as array address?

|  |
| --- |
| First block address will be considered as array address |

What is an index?

|  |
| --- |
| * + - 1. It is a number given to each and every block of an array. And it always starts with zero       2. By using index, we can access elements faster |

Example-1: Basic example on 1D array

|  |
| --- |
| public class ArrayDemo1  {      public static void main(String[] args)      {         int[] arr=new int[5];         //arr=1000         System.out.println(arr);         System.out.println(arr[0]);         System.out.println(arr[1]);         System.out.println(arr[2]);         System.out.println(arr[3]);         System.out.println(arr[4]);      }  }  Output:  [I@5caf905d  0  0  0  0  0 |

Types of arrays?

|  |
| --- |
| There are 2 types of arrays   * 1D array: it is an array, which is created with single subscript and it contains list of elements of the same type stored in a single row. * 2D array: collection of 1D arrays   (or)  A two-dimensional array is an array of arrays, which is commonly used to represent matrices or tables. |

Which class object will be created if we create an array in Java?

|  |
| --- |
| The Java compiler creates an instance of a class that extends java.lang.Object and implements java.lang.Cloneable and java.io.Serializable.  **The class created:**   * Java does **not** explicitly create a named class for the array, but internally, it is represented as a **special class** that extends Object. * The **runtime type** of the array is int[], which is a subclass of Object. |

Example to see the class of the array object?

|  |
| --- |
| public class ArrayDemo1  {      public static void main(String[] args)      {         int[] arr=new int[5];         float[] farr=new float[10];         System.out.println("arr is the object of class called:\t"+arr.getClass());         System.out.println("farr is the object of class called:\t"+farr.getClass());         String s1="array";         System.out.println("s1 is the object of a class called :\t"+s1.getClass());      }  }  Output:  arr is the object of class called: class [I  farr is the object of class called: class [F  s1 is the object of a class called : class java.lang.String |

**ArrayDemo2.java**

|  |
| --- |
|  |

**What is length of an array?**

|  |
| --- |
| * It is an instance variable existed in every array. Which is used to find the length of an array. |

**How to access static variable or static method?**

|  |
| --- |
| * 1. We can access static variable (or) static method by using either class name or a ref-variable |

**How to access instance variable or instance method?**

|  |
| --- |
| * We can access instance variable or instance method by using object only. |

When we will get NullPointerException?

|  |
| --- |
| * 1. We will get NullPointerException when we call instance method or instance variable by using reference variable which is pointing to null. |

Example on getting NullPointerException

|  |
| --- |
| import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  import static java.lang.Integer.parseInt;  public class ArrayDemo1  {      int a; //instance variable      static int s; //static variable      public static void main(String[] args) throws IOException      {            ArrayDemo1 ad1=null;          System.out.println(ad1.s);          System.out.println(ArrayDemo1.s);        System.out.println(ad1.a);      }  }  Run:  0  0  Exception in thread "main" java.lang.NullPointerException: Cannot read field "a" because "<local1>" is null  at ArrayDemo1.main(ArrayDemo1.java:16) |

Example to get NullPointerException, while accessing instance variable length of an array

|  |
| --- |
| import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  import static java.lang.Integer.parseInt;  public class ArrayDemo1  {      int a; //instance variable      static int s; //static variable      public static void main(String[] args) throws IOException      {          int[] arr=null;          System.out.println("size of array:\t"+arr.length);      }  }  Run:  Exception in thread "main" java.lang.NullPointerException: Cannot read the array length because "<local1>" is null  at ArrayDemo1.main(ArrayDemo1.java:12) |

Example to print the elements of an array using for loop

|  |
| --- |
| import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  import static java.lang.Integer.parseInt;  public class ArrayDemo1  {      int a; //instance variable      static int s; //static variable      public static void main(String[] args) throws IOException      {          int[] arr=null;          //System.out.println("size of array:\t"+arr.length);          BufferedReader br=new BufferedReader(new InputStreamReader(System.in));          System.out.println("Enter array size:\t");          int n=parseInt(br.readLine());          arr=new int[n];          System.out.println("size of array:\t"+arr.length);          System.out.println("Elements in an array:\t");          //n=5          //0,1,2,3,4          //i=0,1          for(int i=0;i<arr.length;i++)          {   System.out.println(arr[i]);          }      }  } |

Example to get elements from an array using while loop

|  |
| --- |
| import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  import static java.lang.Integer.parseInt;  public class ArrayDemo1  {      int a; //instance variable      static int s; //static variable      public static void main(String[] args) throws IOException      {          int[] arr=null;          //System.out.println("size of array:\t"+arr.length);          BufferedReader br=new BufferedReader(new InputStreamReader(System.in));          System.out.println("Enter array size:\t");          int n=parseInt(br.readLine());          arr=new int[n];          System.out.println("size of array:\t"+arr.length);          System.out.println("Elements in an array:\t");          //n=5          //0,1,2,3,4          //i=0,1          int i=0;          while(i<arr.length)          {   System.out.println(arr[i]);              i++;          }      }  }  Output:  Enter array size:  10  size of array: 10  Elements in an array:  0  0  0  0  0  0  0  0  0  0 |

Example to get ArrayIndexOutOfBoundsException(Samba)

|  |
| --- |
| import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStreamReader;  import static java.lang.Integer.parseInt;  public class ArrayDemo1  {      int a; //instance variable      static int s; //static variable      public static void main(String[] args) throws IOException      {          int[] arr=null;          //System.out.println("size of array:\t"+arr.length);          BufferedReader br=new BufferedReader(new InputStreamReader(System.in));          System.out.println("Enter array size:\t");          int n=parseInt(br.readLine());          arr=new int[n];          System.out.println("size of array:\t"+arr.length);          System.out.println("Elements in an array:\t");          int i=0;          do          {              System.out.println(arr[i]);              i++;          }while(i<arr.length);        }  }  Output:  Enter array size:  0  size of array: 0  Elements in an array:  Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0  at ArrayDemo1.main(ArrayDemo1.java:22) |

Scanner

|  |
| --- |
| * It is a pre-defined class existed in java.util package * This class has many instance methods which are used to get the data from either keyboard (standard input device) or a file. |

Storing and getting elements from an array by using Scanner class method nextInt()

|  |
| --- |
| import java.util.Scanner;  import static java.lang.System.in;  public class ArrayDemo1  {      int a; //instance variable      static int s; //static variable      public static void main(String[] args)      {          int[] arr=null;          //System.out.println("size of array:\t"+arr.length);          Scanner scan=new Scanner(in);          System.out.println("Enter array size:\t");          int n=scan.nextInt();          arr=new int[n];          System.out.println("size of array:\t"+arr.length);          for(int i=0;i<arr.length;i++)          {   System.out.printf("anniyaa elementu enter seyyiraa index  %d  lo:\t",i);              arr[i]=scan.nextInt();          }          System.out.println("Elements in an array:\t");          for(int i=0;i<arr.length;i++)          {             System.out.println(arr[i]);          }      }  }  Output:  Enter array size:  5  size of array: 5  anniyaa elementu enter seyyiraa index 0 lo: 10  anniyaa elementu enter seyyiraa index 1 lo: 20  anniyaa elementu enter seyyiraa index 2 lo: 30  anniyaa elementu enter seyyiraa index 3 lo: 40  anniyaa elementu enter seyyiraa index 4 lo: 50  Elements in an array:  10  20  30  40  50 |

Initializing array during creation itself

|  |
| --- |
| Syntax-1: int [] arr={10,20,30,40,50};  Syntax-2: int[] arr=new int[]{10,20,30,40,50};  At the time of reference variable declaration, the subscript can be placed either side of the variable.  Ex-1: int[] arr;  Ex-2: int [] arr;  Ex-3: int []arr;  Ex-4: int arr[]; |

Reading and writing array elements by using user defined methods

|  |
| --- |
| import java.util.Scanner;  import static java.lang.System.in;  public class ArrayDemo1  {      static void display(int[] arr)      {  //arr=1000          System.out.println("Elements in an array:\t");          for(int i=0;i<arr.length;i++)          {   System.out.println(arr[i]);          }      }      static void read(int arr[])      {   //arr=1000          Scanner scan=new Scanner(in);          System.out.println("size of array:\t"+arr.length);          for(int i=0;i<arr.length;i++)          {   System.out.printf("anniyaa elementu enter seyyiraa index  %d  lo:\t",i);              arr[i]=scan.nextInt();          }      }      public static void main(String[] args)      {          //System.out.println("size of array:\t"+arr.length);          Scanner scan=new Scanner(in);          System.out.println("Enter array size:\t");          int n=scan.nextInt();          int[] arr=new int[n];          read(arr);          display(arr);  //display(1000)          int[] arr2=new int[3];          read(arr2);      }  }  Output:  Enter array size:  3  size of array: 3  anniyaa elementu enter seyyiraa index 0 lo: 100  anniyaa elementu enter seyyiraa index 1 lo: 200  anniyaa elementu enter seyyiraa index 2 lo: 300  Elements in an array:  100  200  300  size of array: 3 |

Foreach

|  |
| --- |
| For each is used to get the elements from a collection object or an array. |

Example to get the elements from an array by using foreach

|  |
| --- |
| import java.util.Scanner;  import static java.lang.System.in;  public class ArrayDemo1  {      static void display(int[] arr)      {          //arr=1000          System.out.println("Elements in an array:\t");          for(int ele:arr)          {   System.out.println(ele);          }      }      static void read(int arr[])      {   //arr=1000          Scanner scan=new Scanner(in);          System.out.println("size of array:\t"+arr.length);          for(int i=0;i<arr.length;i++)          {   System.out.printf("anniyaa elementu enter seyyiraa index  %d  lo:\t",i);              arr[i]=scan.nextInt();          }      }      public static void main(String[] args)      {          //System.out.println("size of array:\t"+arr.length);          Scanner scan=new Scanner(in);          System.out.println("Enter array size:\t");          int n=scan.nextInt();          int[] arr=new int[n];          read(arr);          display(arr);  //display(1000)      }  }  **Output:**  Enter array size:  5  size of array: 5  anniyaa elementu enter seyyiraa index 0 lo: 10  anniyaa elementu enter seyyiraa index 1 lo: 20  anniyaa elementu enter seyyiraa index 2 lo: 30  anniyaa elementu enter seyyiraa index 3 lo: 40  anniyaa elementu enter seyyiraa index 4 lo: 50  Elements in an array:  10  20  30  40  50 |

What is an anonymous array?

|  |
| --- |
| It is an array which is created without name and without specifying size. |

Example to find whether the given element is found or not

|  |
| --- |
| import java.util.Scanner;  import static java.lang.System.\*;  public class ArrayDemo2  {      static void display(int[] arr)      {          boolean flag=false;          Scanner scan=new Scanner(System.in);          //arr=1000          System.out.println("Enter element to find:\t");          int ele\_to\_find=scan.nextInt();          for (int ele : arr) {              if(ele==ele\_to\_find){                  flag=true;                  break;              }          }          if(flag==true)          System.out.println("Element found");          else          System.out.println("Element not found");      }      public static void main(String[] args) {            display( new int[]{1,8,2,9,5,3,6,7} );      }  }  **Output:**  Enter element to find:  5  Element found |

**Two Dimensional Arrays**

|  |
| --- |
| * 1. A two-dimensional array is an array which contains collection of single dimensional arrays.   2. It is an array of arrays |

**Syntax to create a two-dimensional array**

|  |
| --- |
| Syntax-1: data\_type[][] arr=new data\_type[row\_size][col\_size];  Syntax-2: data\_type arr2[][]=new data\_type[1D arrays count][size of each 1D array]; |

Example on Two dimensional arrays

|  |
| --- |
| public class TDA1 {      public static void main(String[] args)      {          int arr[][]=new int[2][3];          System.out.println("arr.length:\t"+arr.length);          System.out.println("arr[0]:\t"+arr[0]);          System.out.println("arr[1]:\t"+arr[1]);          System.out.println("arr[0].length:\t"+arr[0].length);          System.out.println("arr[1].length:\t"+arr[1].length);          for(int i=0;i<arr.length;i++){              for(int j=0;j<arr[i].length;j++){                  System.out.printf("[%d,%d]:\t%d %n",i,j,arr[i][j]);              }              System.out.println();          }      }  }  **Output:**  arr.length: 2  arr[0]: [I@1fb3ebeb  arr[1]: [I@548c4f57  arr[0].length: 3  arr[1].length: 3  [0,0]: 0  [0,1]: 0  [0,2]: 0  [1,0]: 0  [1,1]: 0  [1,2]: 0 |

Example on 2D array

|  |
| --- |
| import java.util.Scanner;  public class TDA1 {      public static void main(String[] args)      {          int arr[][]=new int[2][3];          Scanner scan=new Scanner(System.in);          //i=0,1,2          //arr.length  =  2          for(int i=0;i<arr.length;i++){              System.out.printf("Enter elements into row  %d:\n",i);              //j=0,1,2,3              //arr[0].length=3              //arr[1].length=3              for(int j=0;j<arr[i].length;j++){                  arr[i][j]=scan.nextInt();              }          }          //i=0          //arr.length=2          for(int i=0;i<arr.length;i++){              //j=0              //arr[0].length=3              for(int j=0;j<arr[i].length;j++){                  System.out.printf("[%d,%d]:\t%d %n",i,j,arr[i][j]);              }              System.out.println();          }      }  }  Output:  Enter elements into row 0:  1  2  3  Enter elements into row 1:  4  5  6  [0,0]: 1  [0,1]: 2  [0,2]: 3  [1,0]: 4  [1,1]: 5  [1,2]: 6 |

Example to get the elements from 2D array using for each

|  |
| --- |
| import java.util.Scanner;  public class TDA1 {      public static void main(String[] args)      {          int arr[][]=new int[2][3];          Scanner scan=new Scanner(System.in);          //i=0,1,2          //arr.length  =  2          for(int i=0;i<arr.length;i++){              System.out.printf("Enter elements into row  %d:\n",i);              //j=0,1,2,3              //arr[0].length=3              //arr[1].length=3              for(int j=0;j<arr[i].length;j++){                  arr[i][j]=scan.nextInt();              }          }          //i=0          //arr.length=2          for(int[] sda:arr){              for(int ele:sda){                  System.out.printf("%-5d",ele);              }              System.out.println();          }          System.out.println("aapararai.....");      }  }  /\*   10\_\_\_20\_\_\_30\_\_\_   40\_\_\_50\_\_\_60\_\_\_   \_   \*/ |

When we will get array constants only be used in initializers?

|  |
| --- |
| See the below image    If you try to assign array constants after array initialization, we will get the above error. |

**Jagged (or) Ragged Array**

|  |
| --- |
| A **jagged array** is an array of arrays where the sub-arrays can have different lengths. Unlike a two-dimensional array (which has fixed row and column sizes), a jagged array allows each row to have a different number of elements.  **Key Characteristics of Jagged Arrays:**   * It is an array of arrays. * Each inner array can have a different size. * It is useful for scenarios where a rectangular structure is not needed. * In languages like Java and C#, jagged arrays are implemented as an array of references to other arrays.     Example on Jagged Array  public class ArrayDemo3 {      public static void main(String[] args)      {          //here 1st subscript represents rows  and subscripts represents columns count of each row          int[][] arr=new  int[3][];          System.out.println(arr.length);          System.out.println(arr[0]);          System.out.println(arr[1]);          System.out.println(arr[2]);          //System.out.println(arr[0].length);  //length is a variable existed in every array and it is instance variable            arr[0]=new int[]{10};          int arr2[]={10,20};          arr[1]=new int[]{10,20};          arr[2]=new int[]{10,20,30};          System.out.println("Elements existed in Jagged array");          for(int[] sda:arr)          {              //sda=1000,2000              for(int ele:sda)              {                  System.out.print(ele+"\t");              }              System.out.println();          }          System.out.println("ayipayeee...");      }  }  **Output:**  3  null  null  null  Elements existed in Jagged array  10  10 20  10 20 30  ayipayeee... |

**What is matrix addition?**

|  |
| --- |
|  |

**Example on matrix addition**

**What is matrix multiplication?**

|  |
| --- |
| Matrix multiplication is a method of finding the product of two matrices to get the result as one matrix. It is a type of binary operation.  **Rules for Matrix Multiplication:**   * If A is an 𝑚×𝑛 matrix and B is an n×p matrix, then the resulting matrix C will be an m×p matrix. * The number of columns in the first matrix (A) must be equal to the number of rows in the second matrix (B). |

**Frequently asked questions in matrix multiplication**

|  |
| --- |
| **What is matrix multiplication?**  Matrix multiplication is a method of finding the product of two matrices to get the result as one matrix. It is a type of binary operation.  Q2  **How to multiply two given matrices?**  To multiply one matrix with another, we need to check first, if the number of columns of the first matrix is equal to the number of rows of the second matrix. Now multiply each element of the column of the first matrix with each element of rows of the second matrix and add them all. We need to do the dot product of columns and rows here.  Q3  **What is the result of multiplication of (2**×**3) matrix and (3**×**3) matrix?**  The result of multiplication of (2×3) matrix and (3×3) matrix will be 2×3 matrix only.  Q4  **How to multiply 3**×**3 matrix?**  Multiply each row of the first matrix with each column of the second matrix and add all to get the first element. Similarly, multiply and add the elements of the two matrices, column and row-wise, to get the elements of the product of two 3×3 matrices.  Q5  **How do we find the multiplication of two matrices?**  If A is a m×n matrix and B is a p×q matrix, then the multiplication of A and B is denoted by dot matrix, such as: C = AB Thus, C will be an m×q matrix. |

**Example on matrix multiplication**

|  |
| --- |
| import static java.lang.System.out;  import static java.lang.System.in;  import java.util.Scanner;  class MatrixMultiplication  {      /\*\*       this method takes the 2d array(matix) as an arugument       later it takes the elements from the keyboard and stores them in to array.       \*/      static void readElements(int[][] arr)      {  //arr=5000          Scanner scan=new Scanner(in);          out.printf("Enter Elements into %d X %d matrix%n ",arr.length,arr[0].length);          //arr.length=2          for(int i=0;i<arr.length;i++)          {   //i=0,1              //j=0              for(int j=0;j<arr[i].length;j++)              {   //j=0,1,2,3                  //arr[0].length=3                  //arr[1].length=3                  out.printf("Enter Element at arr[%d][%d] matrix:\t ",i,j);                  arr[i][j]=scan.nextInt();              }          }      }      static void display(int arr[][])      {          for(int[] sda:arr)          {              for(int ele:sda)              {                  System.out.printf("%-5d",ele);              }              System.out.println();          }      }      public static void main(String[] args)      {              int fmr=2,fmc=3;              int smr=3,smc=3;              int[][] m1=new int[2][3];              int[][] m2=new int[3][3];              int[][] m3=null;;              if(fmc==smr){                  System.out.printf("Resultant matix will be %d X %d%n",fmr,smc);                  readElements(m1);                  readElements(m2);                  m3=multiply(m1,m2);                  System.out.println("Resultant matrix");                  display(m3);              }      }      static int [][] multiply(int[][] m1,int[][] m2)      {          int [][] m3=new int[m1.length][m2[0].length];          //i=0,1          //m1.length=2          for(int i=0;i<m1.length;i++)          {              //j=0,1              //m2[0].length=3              for(int j=0;j<m2[0].length;j++)              {                  int sum=0;                  //sum=0,4,24,66                  for(int k=0;k<m2.length;k++)                  {                      //k=0                      //m2.length=3                      sum=sum+ m1[i][k]\*m2[k][j] ;                  }                  m3[i][j]=sum;              }          }          return m3;      }  }  **Output:**  Resultant matix will be 2 X 3  Enter Elements into 2 X 3 matrix  Enter Element at arr[0][0] matrix: 1  Enter Element at arr[0][1] matrix: 2  Enter Element at arr[0][2] matrix: 3  Enter Element at arr[1][0] matrix: 4  Enter Element at arr[1][1] matrix: 5  Enter Element at arr[1][2] matrix: 6  Enter Elements into 3 X 3 matrix  Enter Element at arr[0][0] matrix: 7  Enter Element at arr[0][1] matrix: 8  Enter Element at arr[0][2] matrix: 9  Enter Element at arr[1][0] matrix: 1  Enter Element at arr[1][1] matrix: 2  Enter Element at arr[1][2] matrix: 3  Enter Element at arr[2][0] matrix: 4  Enter Element at arr[2][1] matrix: 5  Enter Element at arr[2][2] matrix: 6  Resultant matrix  21 27 33  57 72 87 |

**Command line arguments**

|  |
| --- |
| Command line arguments are arguments, which are given to the main method at the time of giving command to run the program like below.  java First 10 20 madhu 10.40   * 1. in the above command 10,20, madhu and 10.40 are command line arguments   2. these arguments are taken by JVM and place them in a string array and passes that array to the main method as an argument. |

What the JVM should pass to the main method as an argument?

|  |
| --- |
| Single dimensional string type array address |

Example on command line arguments

|  |
| --- |
| public class CommandLineArgs {      public static void main(String[] args) {          System.out.println(args.length);          for(String ele:args)          {   System.out.println(ele);          }      }  }  **Output:** |

Performing Addition operation on command line arguments?

|  |
| --- |
| import static java.lang.Double.parseDouble;  public class CommandLineArgs {      public static void main(String[] args) {         if(args.length==2)         {          double d1=parseDouble(args[0]);          double d2=parseDouble(args[1]);          System.out.println(d1+d2);         }else{          System.out.println("expecting only two command line args...");         }      }  }  Output: |

**OOP’s concepts**

|  |
| --- |
| **Java is called as an Object-Oriented Programming Language**   * + 1. class     2. object     3. Data abstraction & encapsulation     4. Polymorphism     5. Inheritance |

**In general, what is an object**

|  |
| --- |
| Real world physically existed anything Is called as an object in general |

What is a class?

|  |
| --- |
| * A class is a blue print of objects * A class is an abstract data type or a user-defined data type which contains fields(variables) and methods. * A class is a prototype/template of objects |

Why we write a class?

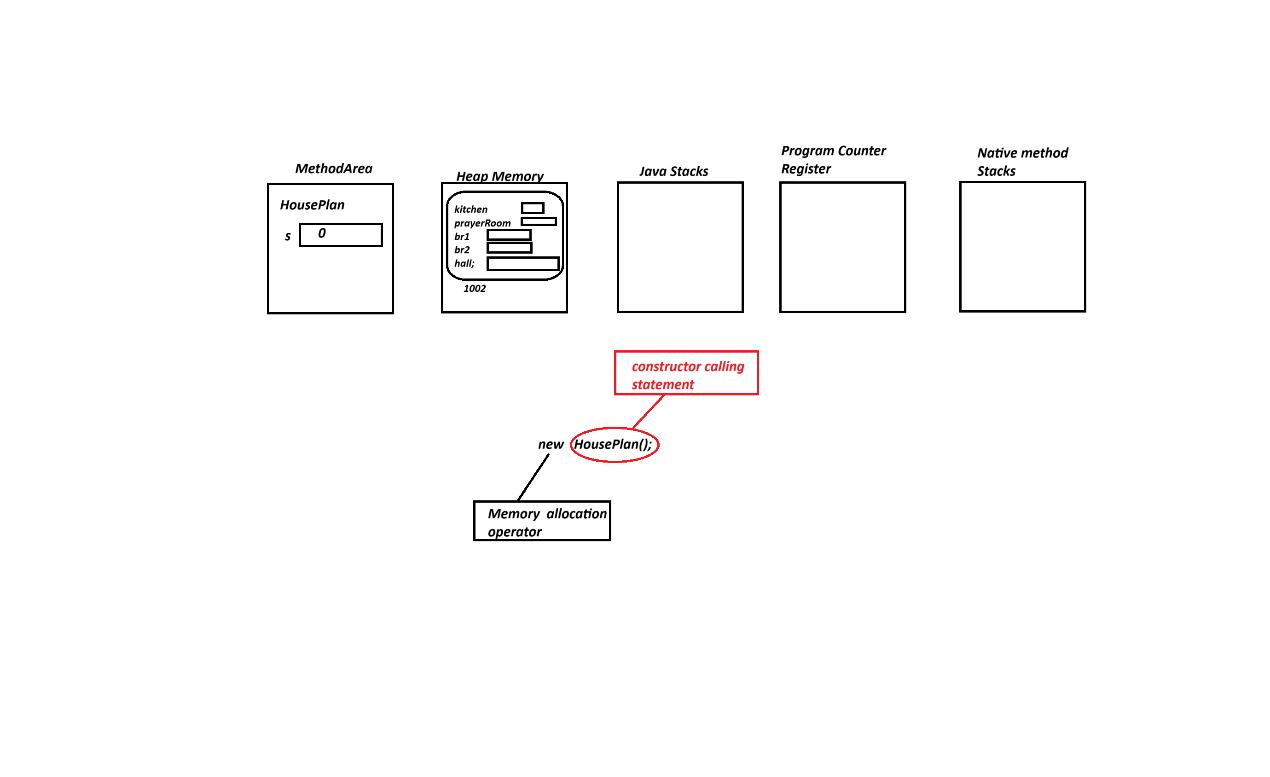
|  |
| --- |
| We write a class to create objects. |

How to create an object?

|  |
| --- |
| We can create an object in 5 ways  Method Usage   * 1. new Keyword Most common and simple way   2. Class.forName() Dynamic class loading   3. clone() Creates copy of existing object   4. Deserialization Restores object from stream   5. Factory Method Common in Design Patterns |

Creating object by using new operator

|  |
| --- |
| * + 1. During object creation the class will be loaded first, if already it is not existed in method area.     2. During class loading if that class contains static fields then memory allocation will be done for those variables/fields in method area itself. And if those variables are not initialized during declaration, then class loader assigns default values to them.     3. Now JVM creates object i.e. it allocates memory for the instance fields of a class in heap memory. and then constructor will be invoked and it assigns default values to the instance fields if those are not initialized during declaration.     4. After constructor execution completes, it returns (gives us) object address     5. Now we have to store the address in a reference variable by using assignment operator   Note: object type and reference variable type must be same |



How many times a class will be loaded during program execution?

|  |
| --- |
| Only once |

Who loads the classes?

|  |
| --- |
| Class loader |

What is constructor?

|  |
| --- |
| It is a special method which has same name as class name and doesn’t contain return type. |

What is object initialization?

|  |
| --- |
| Assigning values to the instance fields of an object. |

What is dot(.)?

|  |
| --- |
| Memory access operator |

Example on class and object

|  |
| --- |
| class HousePlan{      static int s;  //static field      byte kitchen;  //1 byte memory      short prayerRoom; //2 bytes memory      int br1,br2;      //4 bytes each      long hall;        //8 bytes memory      //special method called constructor      HousePlan(){          // kitchen=0;          // prayerRoom=0;          // br1=0;          // br2=0;          // hall=0;          System.out.println("saradaaaki.....");      }  }  public class ClassObjDemo1 {      public static void main(String[] args) {          HousePlan p1;          HousePlan krishnaNilayam=new HousePlan();          //krishnaNilayam = 1002          System.out.println(krishnaNilayam.kitchen);          System.out.println(krishnaNilayam.prayerRoom);          System.out.println(krishnaNilayam.br1);          System.out.println(krishnaNilayam.br2);          System.out.println(krishnaNilayam.hall);      }  }  Output:  saradaaaki.....  0  0  0  0  0 |

What is the use of constructor?

|  |
| --- |
| We write a constructor to initialize the object, at the time of object creation |

What is an object

|  |
| --- |
| * Instance of a class (or) run time entity |

Another example on class and object

|  |
| --- |
| class One  {      static int s; //static field      int a,b;  //instance fields      One(){          a=100;          b=200;          System.out.println("non-parameterized constructor...");      }      void display() //r1      {          System.out.println(a);  //System.out.println(r1.a);          System.out.println(b);  //System.out.println(r1.b);      }  }  public class ClassObjDemo2 {      public static void main(String[] args) {          One r1=new One();          //r1=1002          //System.out.println(r1.a);          //System.out.println(r1.b);          r1.display();      }  }  **Output:**  non-parameterized constructor...  100  200 |

What is an instance method?

|  |
| --- |
| It is a method written in a class and not declared as static |

Types of constructors?

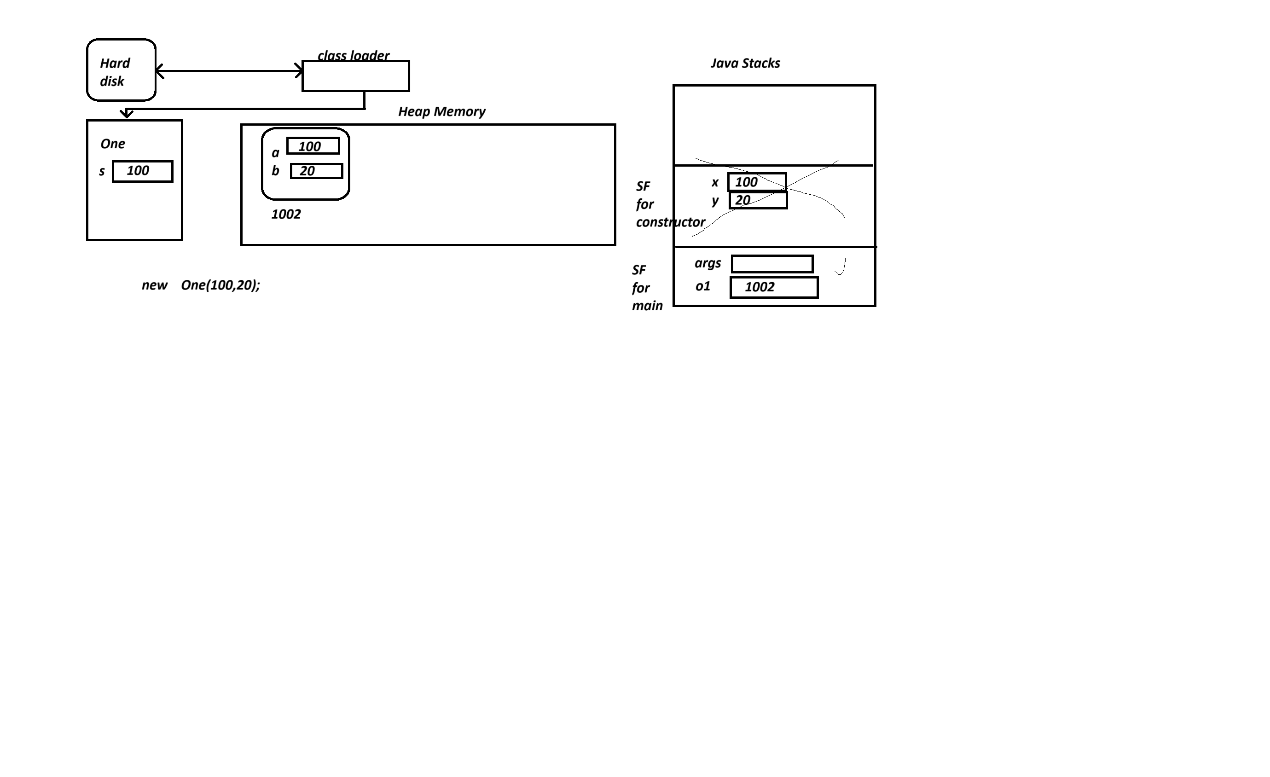
|  |
| --- |
| There are 2 types of constructors available in Java   * 1. Default constructors (non-parameterized constructors).   2. Parameterized constructors. |

What is static block?

|  |
| --- |
| It is a block, which has no name and defined by using a keyword called static. |

When the static block will be executed?

|  |
| --- |
| During class loading after memory allocation is done for static variables. |



Example on parameterized constructor and static block

|  |
| --- |
| class One  {  static int s; //method araa  static{  s=100;  System.out.println("static block...is invoked...");  }  int a,b; //heap memory  //Here x,y are local variables(parameters)  //for x,y memory allocation will be done in stack memory  One(int x,int y){  a=x; //this.a=x;  b=y; //this.b=y;  System.out.println("parameterized constructor..");  }  void display(){  System.out.println("instance method");  }  static void get(){  System.out.println("saradaki it's fun...");  }  }  class ClassObjDemo3  {  public static void main(String[] args) {  One o1;  o1=new One(100,20);  o1.display();  One.get();  }  } |

How to access instance variables in a block which contains local variables with same name?

|  |
| --- |
| * We can’t use instance variables in a block if it contains local variables with same name directly * We can access instance variables by using ‘this’ keyword in a block if it contains local variables with same name. |

What is the use of this keyword?

|  |
| --- |
| * 1. It is a keyword or a pointer which points to the present object.   2. If you want to assign local variable value to the instance variable whenever both names are same then we have to use ‘this’ keyword.   3. By using this keyword, we can call a constructor in another constructor of the same class. |

Why we give local and instance variables names same?

|  |
| --- |
| To increase the readability in the program |

What is instance block?

|  |
| --- |
| * It is a non-static block, which doesn’t contain any name * If you want to place one or more statements in every constructor of the class to execute during every object creation, then you need to place them(statements) in an instance block so that those statements will be placed in every constructor. |

Example on usage of instance block

|  |
| --- |
| class One  {      static int s; //method area      //static block      static{          s=100;          System.out.println("static block...is invoked...");      }      int a,b;  //heap memory      {          System.out.println("yetraa bulliraju..... appadalu ...vadiyalu ayipoyaaayaa...");      }      One(int a){          this.a=a;          System.out.println("single param constructor...");      }      One(){          a=0;          b=0;          System.out.println("non-parameterized constructor..is invoked");      }      One(int a,int b)      {              this.a=a;    //this.a=x;              this.b=b;    //this.b=y;              System.out.println("parameterized constructor..");      }      void display(){          System.out.println("instance method");      }      static void get(){          System.out.println("saradaki it's fun...");      }  }  class ClassObjDemo3  {      public static void main(String[] args) {          One o1=new One(100,20);  //o1=1002          One o2=new One(1000);  //o2=2002          One o3=new One();  //o3=3002      }  }  Output:  static block...is invoked...  yetraa bulliraju..... appadalu ...vadiyalu ayipoyaaayaa...  parameterized constructor..  yetraa bulliraju..... appadalu ...vadiyalu ayipoyaaayaa...  single param constructor...  yetraa bulliraju..... appadalu ...vadiyalu ayipoyaaayaa...  non-parameterized constructor..is invoked |

Can a construct call another constructor of the same class?

|  |
| --- |
| * 1. Yes, we can do it by using this keyword |

Can we call more than one constructor in another constructor of the same class?

|  |
| --- |
| No, in Java there is a rule called a constructor calling statement must be the first statement In a constructor. |

Why the main method is static?

|  |
| --- |
| Because main method is invoked by the JVM by using class name. |

Example on calling constructor in another constructor

|  |
| --- |
| class One  {      int a,b;      One(int a)      {          this.a=a;  //this.a=100          System.out.println("single parameterized constructor...");      }      One()      {   System.err.println("non-parameterized....");      }      One(int a,int b)      {//a=10, b=20          this(a);          System.err.println("parameterized....");          this.b=b;      }      void display() //o2=2002      {          System.out.println("object state....");          System.out.println("a:\t"+a);  //System.out.println("a:\t"+o1.a);          System.out.println("b:\t"+b); //System.out.println("b:\t"+o1.b);      }  }  public class ClassObjDemo4 {      public static void main(String[] args) {          One o1=new One(10,20);          o1.display();          One o2=new One(100);          o2.display();      }  }  **Output:**  single parameterized constructor...  parameterized....  object state....  a: 10  b: 20  single parameterized constructor...  object state....  a: 100  b: 0 |

**What is static variable?**

|  |
| --- |
| * 1. It is a variable which is declared within the class, outside the method or constructor and should be declared by using a modifier(keyword) called static.   2. It is also called as class variable |

**How to access static variable?**

|  |
| --- |
| * By using class name or reference variable name (it doesn’t matter whether it contains reference(address) or null). |

**When a class will be loaded into method area?**

|  |
| --- |
| * 1. During accessing static members (fields or methods) of a class.   2. During object creation (if already not loaded).   3. Programmer also can tell the class loader |

**Example on when the class will be loaded?**

|  |
| --- |
| class One  {      static int s;      static{          System.out.println("class One Static block.....");      }  }  class Second{      static{          System.out.println("class Second static block");      }  }  class Three{      int a;      static{          System.out.println("class Three static block");          int x=100;      }  }  public class ClassObjectDemo5 {      public static void main(String[] args) throws Exception      {          One o1; //class will not be loaded          One o2=null; //class will not be loaded          System.out.println(o2.s); //during this statement execution class will be loaded          Class.forName("Second");          new Three();          Class.forName("Three"); //will not be loaded because already loaded during object creation(above statement)      }  }  **Output:**  class One Static block.....  0  class Second static block  class Three static block |

**When we get illegal forward reference error?**

|  |
| --- |
| * If you try to access static variable in static block which is declared after static block. Then we will get illegal forward reference error. * If you try to access instance variable in instance block which is declared after instance block. Then we will get illegal forward reference error. |

**Example on illegal forward reference error**

|  |
| --- |
| **class One**  **{**  **static{**  **System.out.println("static block of One class");**  **System.out.println("s:\t"+s);  //illegal forward reference error**  **}**  **static int s;**  **{**  **System.out.println(a); //illegal forward reference error**  **}**  **void display()**  **{   System.out.println(a);**  **}**  **int a;**  **}**  **public class ClassObjDemo6 {**  **public static void main(String[] args) throws Exception{**  **Class.forName("One");**  **}**  **}** |

Types of class loaders

|  |
| --- |
| A **ClassLoader** in Java is a part of the Java Runtime Environment (JRE) responsible for dynamically loading Java classes into the Java Virtual Machine (JVM) during runtime. This mechanism allows Java to load classes as needed, rather than loading all classes at once.  **Types of ClassLoaders**  In Java, **Class Loaders** are responsible for dynamically loading classes into the JVM during runtime. Java provides several types of class loaders, each with a specific purpose.  **1. Bootstrap ClassLoader**   * It is the **parent of all Class Loaders**. * Loads **core Java classes** from the **<JAVA\_HOME>/lib** directory (like java.lang.\*, java.util.\*). * Written in **native code (C/C++)** and is a part of the JVM.   **2. Extension ClassLoader (Platform ClassLoader in Java 9+)**   * Loads classes from the **lib/ext** directory or any directory specified by the **java.ext.dirs** system property. * Loads **standard extension classes** like **javax.crypto** and others.   **3. System ClassLoader (Application ClassLoader)**   * Loads classes from the **classpath** (directories and JAR files specified in the **CLASSPATH** environment variable or -cp option). * Loads user-defined application classes.   **4. Custom ClassLoaders**   * Developers can create their own class loaders by extending the **ClassLoader** class. * Used in frameworks like Spring, Hibernate, and web servers to load classes dynamically. |

What class loader is used by forName() method?

|  |
| --- |
| It totally depends on the class which you want to load. If you are loading pre-defined class then bootstrap class loader will be used by, if you are loading user-defined class then application(system) class loader will be used. |

**How Parent Delegation Works:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| When a class needs to be loaded:   1. The current ClassLoader first delegates the request to its parent. 2. The parent keeps delegating to its own parent. 3. This process continues until the **Bootstrap ClassLoader** is reached. 4. If the parent cannot find the class, the current class loader loads it.   **Key Points:**   | **Class Loader** | **Priority** | **Example Classes** | | --- | --- | --- | | Bootstrap | Highest | java.lang.String, java.util.ArrayList | | Extension | High | javax.crypto.\*, javax.sound.\* | | System | Medium | User-defined classes | | Custom | Lowest | Custom dynamic classes | |

Can we override (re-write) pre-defined classes?

|  |
| --- |
| * 1. No, bootstrap class loader doesn’t allow   2. If you override a pre-defined class then the pre-defined can’t be loaded by bootstrap class loader. |

Example on overriding System class

|  |
| --- |
| class System{      static{          java.lang.System.out.println("user --defined System class");      }  }  public class Demo {   public static void main(String[] args)throws Exception {      System.out.println();   }  }  **Output:**  Demo.java:8: error: cannot find symbol  System.out.println();  ^  symbol: variable out  location: class System  1 error |

When we get can not reference a variable without super type constructor is called?

|  |
| --- |
| class One{      int a,b;      //non-parameterized constructor      One(){          this(a,b); //we will get error: cannot reference a before supertype constructor has been called          System.out.println("non-parameterized constructor..");      }      One(int a,int b){          System.out.println("Parameterized constructor..");          this.a=a;          this.b=b;      }  }  public class Demo2 {      public static void main(String[] args) {          new One();      }  }  **Output:** |

**Types of Conversions**

Type conversions in java are divided into **13 categories** those are listed and briefly discussed below.

* Identity conversions
* Widening primitive conversions
* Narrowing primitive conversions
* Widening and Narrowing Primitive conversion
* Widening reference conversions
* Narrowing reference conversions
* Boxing conversions
* Un-boxing conversions
* Unchecked conversions
* Capture conversions
* String conversions
* Value set conversions
* Forbidden Conversion

**Type Casting**

|  |
| --- |
| * Process of converting one type of value to another type is called as type casting   Type casting is of 2 types   * 1. Implicit type casting (widening primitive conversion)   2. Explicit ty pe casting (Narrowing primitive conversion) |

**Identity conversion**

|  |
| --- |
| In Java, identity conversion refers to a type of conversion where no actual change happens to the data type — the source type and the target type are exactly the same. |

**Primitive data types are divided into 2 categories based on sign**

* + 1. **Signed data types (byte, short,int,long,float and double)**
    2. **Unsigned data types (char, boolean)**

**Can I store a -ve value in character type variable?**

|  |
| --- |
| If you try to store -ve value in a char type variable directly, then you will get a compile time error called “incompatible types: possible lossy convertsion from int to char” |

**if you try to assign int value to char type variable directly, then what range of int value it accepts without explicit type casting?**

|  |
| --- |
| 65535 |

**Why implicit type casting is not performing if you assign byte type to char**

|  |
| --- |
| Byte type may contain -ve value which is not fit into char type variable. So explicit type casting is need. |

**If we assign integer literal to char type variable does it support implicit or explicit type casting?**

|  |
| --- |
| If integer literal is within the character range then it support implicit type casting otherwise explicit. |

**What is the range of character type variable?**

|  |
| --- |
| 0 to 65535  (or)  0 to 216 – 1 |

**Widening Primitive Conversion**

This conversion is also called as implicit or automatic conversion, because developer no need to use typecast operator explicitly.

A widening primitive conversion does not lose information. We have 19 specific conversions on primitive types those are called as widening primitive conversions.

* byte to short, int, long, float, or double
* short to int, long, float, or double
* char to int, long, float, or double
* int to long, float, or double
* long to float or double
* float to double

**Note:** while executing a widening primitive conversion runtime error will not occur.

**Example on widening primitive conversion**

|  |
| --- |
| public class Demo4 {      public static void main(String[] args) {          byte b=100;          short s=b;  //implict type casting          System.out.println("b:\t"+b);          System.out.println("s:\t"+s);          long l=Long.MAX\_VALUE;          float f=l;  //implicit          System.out.println("l:\t"+l);          System.out.println("f:\t"+f);          char ch1='A';          int i=ch1;//implicit          System.out.println("l:\t"+i);          System.out.println("ch1:\t"+ch1);          /\*  Eventhough the float type variable allocates only 4 bytes memory, in that memory we can store   “3.4028235E38”  this much amount of value in exponential notation, which means maximum we can store “340282350000000000000000000000000000000.00” value. So we can store long value in float value that means.  \*/      }  }  **Output:**  b: 100  s: 100  l: 9223372036854775807  f: 9.223372E18  l: 65  ch1: A |

**Explicit type casting (Narrowing primitive conversion)**

|  |
| --- |
| **Narrowing Primitive Conversion**  A narrowing primitive conversion may lose information. We have 22 specific conversions on primitive types those are called as narrowing primitive conversions.   * *short to byte or char* * *char to byte or short* * *int to byte, short, or char* * *long to byte, short, char, or int* * *float to byte, short, char, int, or long* * *double to byte, short, char, int, long, or float*   **Note:** while compilation compiler displays an error, if we don’t use the typecast operator. While executing a narrowing primitive conversion runtime error will not occur.  **Note:** there may be a chance of loss of data. |

**When we will get incompatible types error**

|  |
| --- |
| public class Demo4 {      public static void main(String[] args) {          short s=100;          char ch1=s;          // System.out.println("l:\t"+i);          // System.out.println("ch1:\t"+ch1);      }  }  output: |

**Same as above example but I am performing explicit type casting**

|  |
| --- |
| public class Demo4 {      public static void main(String[] args) {          short s=65;          char ch1=(char)s;          System.out.println("s:\t"+s);          System.out.println("ch1:\t"+ch1);      }  }  **Output:**  s: 65  ch1: A |

Can we assign a Boolean value to int type variable

|  |
| --- |
| public class Check {      public static void main(String[] args) {          boolean b=false;          int i1=true;          int i2=b;      }  }  Output: |

Can I assign a value type variable to reference variables and vice versa?

|  |
| --- |
| No, we can’t |

Example on assigning value type variable to reference variable

|  |
| --- |
| public class Check {      public static void main(String[] args) {          char ch1='A';          String s1=ch1;      }  }  **Output:** |

**Short cut operators**

|  |
| --- |
| * += * -= * \*= * /= * %= * ++ * -- |

Example on usage of increment operator

|  |
| --- |
| public class Increment {      public static void main(String[] args) {              int a=10; //a=10              System.err.printf("a:%d%n",a);              a++;  //a= a + 1  //a=11              System.err.printf("a:%d%n",a);              ++a;  //a=a+1              System.err.printf("a:%d%n",a);      }  }  Output:  **a:10**  **a:11**  **a:12** |

Example on usage of pre increment and post increment

|  |
| --- |
| public class Increment {      public static void main(String[] args) {              int a=10; //a=10              int b=a++; //1.assign value of a to b, 2.increment a              //1. b=10              //2. a=a+1  a=11              System.out.printf("a:\t%d%n",a);              System.out.printf("b:\t%d%n",b);              int x=10;  //x=10              int y=++x; //2.assigning x value to y, 1.incrementing x by 1              //1. x=x+1  x=11              //2. y=a   y=11              System.out.printf("x:\t%d%n",x);              System.out.printf("x:\t%d%n",y);      }  }  **Output:**  a: 11  b: 10  x: 11  x: 11 |

IncDec.java

|  |
| --- |
| /\*  Ajay: 4 2 4 4  Laxman: 2 2 4 2  Bala:   2 2 4 2  Prasanna:  4 2 4 4  Manish: 4 2 4 4  Madhu: 4 2 4 2   \*/  class One  {      static int a=2;      int b=a++;      int c=++a;      static int d=a;      //One(){}      void display()      {          System.out.printf("a:\t%d%n",a);          System.out.printf("b:\t%d%n",b);          System.out.printf("c:\t%d%n",c);          System.out.printf("d:\t%d%n",d);      }  }  public class IncDec {      public static void main(String[] args) {          One o1=new One();          o1.display();      }  }  Output:  a: 4  b: 2  c: 4  d: 2 |

IncDec.java

|  |
| --- |
| class One  {   static int a=2;      int b=a++;      int c=++a;      static int d=a;      //One(){}      void display()      {          System.out.printf("a:\t%d%n",a);          System.out.printf("b:\t%d%n",b);          System.out.printf("c:\t%d%n",c);          System.out.printf("d:\t%d%n",d);      }  }  public class IncDec {      public static void main(String[] args) {          One o1=new One();          o1.display();          One o2=new One();          o2.display();      }  }  **Output:**  a: 4  b: 2  c: 4  d: 2  a: 6  b: 4  c: 6  d: 2 |

**Operator Precedence in Java**

**Operator Precedence Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Precedence Level** | **Operators** | **Associativity** | **Description** |
| **1 (Highest)** | **++ -- () [] .** | **Left to Right** | **Postfix (Increment, Array Access, Dot Operator)** |
| **2** | **++ -- + - ~ !** | **Right to Left** | **Unary Operators (Prefix Increment/Decrement, Unary Plus/Minus, Bitwise NOT, Logical NOT)** |
| **3** | **\* / %** | **Left to Right** | **Multiplication, Division, Modulus** |
| **4** | **+ -** | **Left to Right** | **Addition, Subtraction** |
| **5** | **<< >> >>>** | **Left to Right** | **Bitwise Shift Operators** |
| **6** | **< <= > >= instanceof** | **Left to Right** | **Relational Operators** |
| **7** | **== !=** | **Left to Right** | **Equality Operators** |
| **8** | **&** | **Left to Right** | **Bitwise AND** |
| **9** | **^** | **Left to Right** | **Bitwise XOR** |
| **10** | **`** | **`** | **Left to Right** |
| **11** | **&&** | **Left to Right** | **Logical AND** |
| **12** | **`** |  | **`** |
| **13** | **? :** | **Right to Left** | **Ternary (Conditional)** |
| **14** | **= += -= \*= /= %=** | **Right to Left** | **Assignment Operators** |
| **15 (Lowest)** | **,** | **Left to Right** | **Comma Operator** |

**Example on increment and decrement**

|  |
| --- |
| public class Operarai {      public static void main(String[] args) {          int a=5;          int b=4;          //a=5,6          //b=4,3,2          int r=a++ + ++a - --b + b--;          //int r=5 + 7 - 3 + 3;          System.out.println("a:\t"+a);          System.out.println("b:\t"+b);          System.out.println("r:\t"+r);      }  }  Output:  a: 7  b: 2  r: 12 |

**Another example**

|  |
| --- |
| public class Operarai  {      public static void main(String[] args)      {          int a=2,b=5;          int r=--b - --a + b++ / a-- \* a;          //r= 4 - 1 + 4 / 1 \* 0          System.out.println("a:\t"+a); //a=0          System.out.println("b:\t"+b); //b=5          System.out.println("r:\t"+r); //r=3      }  }  **Output:**  a: 0  b: 5  r: 3 |

**Another example on increment and decrement operator usage**

|  |
| --- |
| public class Operarai  {      public static void main(String[] args)      {          int x = 10, y = 5, z;          z = x++ - --y + ++x - y-- + x--;          //z = 10 - 4 + 12 - 4 + 12;          System.out.println("x = " + x);          System.out.println("y = " + y);          System.out.println("z = " + z);      }  }  Output:  x = 11  y = 3  z = 26 |

**Method Overloading**

What is method overloading?

|  |
| --- |
| We can write a method with same name with different signature in a class. It is called as method overloading. |

**What is method overloadings**

|  |
| --- |
| class MyMath{      //Signature: add(int,int)      int add(int a,int b){          return a+b;      }      //add(int,float)      float add(int a,float b){          return a+b;      }      //add(float,int)      double add(float a,int b){          return a+b;      }      //add(float,float)      float add(float a,float b){          return a+b;      }      //add(int,int,int)      long add(int a,int b,int c){          return a+b+c;      }      //add(int,double)      double add(int a,double b){          return a+b;      }  }  public class MethodOverloading {      public static void main(String[] args) {          MyMath mm=new MyMath();          int r1=mm.add(10, 20);         //r1=30          float r2=mm.add(10.50f,20.50f);          int r3=(int)mm.add(10,20,30);          double r4=mm.add(10,33.30);          System.out.println("r1:\t"+r1);          System.out.println("r2:\t"+r2);          System.out.println("r3:\t"+r3);          System.out.println("r4:\t"+r4);      }  }  **Output:**  r1: 30  r2: 31.0  r3: 60  r4: 43.3 |

**Polymorphism**

|  |
| --- |
| * 1. Here poly means many and morphs means forms   2. If we write a method which behaves differently in different situations then it is called as polymorphism. |

**Inheritance**

**What is inheritance?**

|  |
| --- |
| Process of including members (fields and methods) of one class into another class, is called as inheritance. |

**What is object state?**

|  |
| --- |
| The data existed in an object is called as an object state |

**What is base class?**

|  |
| --- |
| If a class is being inherited by another(child) class then it is called as base or parent or super class. |

**What is child class?**

|  |
| --- |
| It is a class which inherits the base class  Child class is also called as derived or sub class. |

**How to inherit a Base class into Child class?**

|  |
| --- |
| By using extends keyword we can inherit a base class into child class |

**InheritanceDemo1.java**

|  |
| --- |
| class Base{      int a,b;      void display(){          System.out.println("object state...");          System.out.println("a:\t"+a);          System.out.println("b:\t"+b);      }  }  class Child extends Base{  }  public class InheritanceDemo1 {      public static void main(String[] args) {          Child c1=new Child();          c1.display();      }  }  **Output:**  object state...  a: 0  b: 0 |

**Forms of inheritance**

* Simple (single-level) inheritance
* Multiple inheritance

**Other forms of inheritance (extended forms of simple or multiple inheritance)**

* 1. Multilevel inheritance (extension to simple inheritance)
  2. Hierarchical inheritance (extension to simple inheritance)
  3. Hybrid inheritance (combination of one or more forms)

**Simple or single level inheritance**

|  |
| --- |
| If a class inherits a single base class at a time, then it is called as single level inheritance. |

**Note:** every child class constructor, must contains super() class constructor calling statement as a first statement.

**Note:** If base class has only parameterized constructor, then we have to call it explicitly in child constructor.

Example on simple inheritance(calling base class parameterized constructor)

|  |
| --- |
| class Base{      static int s1;      int a,b;      static{          s1=100;          System.out.println("Base class is loaded");      }      Base(int a,int b){          //a=10, b=20          this.a=a;          this.b=b;          System.out.println("Parameterized constructor...of supser class");      }      void display(){          System.out.println("object state...");          System.out.println("a:\t"+a);          System.out.println("b:\t"+b);      }  }  class Child extends Base{      static int s2;      int c,d;      Child(){          //super();  //super class non-parameterized constructor          super(10,20);          c=3;          d=4;          System.out.println("child class constructor");      }      static{          System.out.println("Child class is loaded");          s2=200;      }      void display2()      {   System.out.println("object state...");          System.out.println("c:\t"+c);          System.out.println("d:\t"+d);      }  }  public class InheritanceDemo1 {      public static void main(String[] args) {              Child c1=new Child();              c1.display();              c1.display2();      }  }  **Output:**  Base class is loaded  Child class is loaded  Parameterized constructor...of supser class  child class constructor  object state...  a: 10  b: 20  object state...  c: 3  d: 4 |

What is the use of super?

|  |
| --- |
| * It is a keyword which is used to call the super class constructor in a child constructor. * We can call overridden method in child class by using super keyword. * This represents present object and super represent super class object. |

What is method overriding?

|  |
| --- |
| If we write base class instance method in child class again, then it is called as method overriding. |

After overriding the base class instance method in child class, can I call the base class method by using child class object?

|  |
| --- |
| No, we can’t |

Access modifiers

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| You're absolutely correct! According to **OCJP (Oracle Certified Java Programmer)** standards, there are technically **only 3 access modifiers** in Java:   1. public 2. protected 3. private   However, there are **4 levels of access control**:  **access control**:   |  |  |  | | --- | --- | --- | | **Access Level** | **Modifier** | **Description** | | Public | Public | Accessible in any package. | | Protected | protected | Accessible within the **same package** and **subclasses (even in different packages)** through inheritance. | | Default (Package-Private) | *(No Modifier)* | Accessible **only within the same package**. | | Private | Private | Accessible **only within the same class**. |   **🔥 Key Points for OCJP Certification:**   * **Only 3 access modifiers** are available (public, protected, and private). * The **default (package-private)** access level is not a modifier — it's the absence of any modifier. * Classes can have:   + public   + Default (Package-Private) * Inner classes, Methods, variables, and constructors can have all **3 access modifiers** + **default**. |

What are the rules we have to follow at the time of overriding a method?

**Rules to Follow During Method Overriding:**

|  |  |
| --- | --- |
| **Rule** | **Description** |
| 1. Method Name | The **method name** must be **exactly the same** as in the parent class. |
| 2. Method Signature | The **parameter list** must be **the same** (same type, order, and number of parameters). |
| 3. Return Type | The **return type** must be **the same** or **Covariant Type** (Child class return type is allowed from Java 5 onwards). |
| 4. Access Modifier | - The access modifier **cannot be more restrictive**. ✅ public → public ✅ protected → protected or public ❌ public → protected |
| 5. Static Methods | **Static methods cannot be overridden** (It is called **method hiding** instead). |
| 6. Final Methods | **Final methods cannot be overridden** because they are fixed by the parent class. |
| 7. Private Methods | **Private methods cannot be overridden** because they are not inherited. |
| 8. Exception Handling | The overriding method **cannot throw broader or new checked exceptions** but can throw narrower exceptions. |
| 9. Constructors | **Constructors cannot be overridden**. |
| 10. Synchronization | Synchronization modifier is **ignored** during overriding. |

What is method signature?

|  |
| --- |
| Method signature is a combination of method name and parameter list(parameter type, order and count)s |

Class of p1 package

First.java

|  |
| --- |
| package p1;  public class First {      private int pri=10;              int def=20;      protected int pro=30;      public int pub=40;      public void display(){          System.out.println("p1.First(Same class) display....");          System.out.println("pri:\t"+pri);          System.out.println("def:\t"+def);          System.out.println("pro:\t"+pro);          System.out.println("pub:\t"+pub);      }  } |

Second.java

|  |
| --- |
| package p1;  public class Second{        public void display(){          First f=new First();          System.out.println("p1.Second(Other Same class) display....");          //System.out.println("pri:\t"+f.pri);          System.out.println("def:\t"+f.def);          System.out.println("pro:\t"+f.pro);          System.out.println("pub:\t"+f.pub);      }  } |

Three.java

|  |
| --- |
| package p1;  import p1.First;  public class Three extends First{      // private int pri=10; no it won't inherit      //        int def=20; inherited      // protected int pro=30; inherited      // public int pub=40; inherited      public void display(){          System.out.println("p1.Three(Child class) display....");          //System.out.println("pri:\t"+pri);          System.out.println("def:\t"+def);          System.out.println("pro:\t"+pro);          System.out.println("pub:\t"+pub);      }  } |

P2 package classes

Four.java

|  |
| --- |
| package p2;  import p1.First;  public class Four extends First{      //pri:   no      //def:   no      //pro:   yes      //pub:   yes      public void display(){          System.out.println("p2.Four(Sub class) display....");          //System.out.println("pri:\t"+pri);          //System.out.println("def:\t"+def);          System.out.println("pro:\t"+pro);          System.out.println("pub:\t"+pub);      }  } |

Five.java

|  |
| --- |
| package p2;  public class Five extends Four{      //pri:  no      //def:  no      //pro:  yes      //pub:  yes      public void display(){          System.out.println("p2.Five(SubSub class) display....");          //System.out.println("pri:\t"+pri);          //System.out.println("def:\t"+def);          System.out.println("pro:\t"+pro);          System.out.println("pub:\t"+pub);      }  } |

Sixth.java

|  |
| --- |
| package p2;  public class Sixth {      public void display(){          p1.First f=new p1.First();          System.out.println("p2.Sixth(Other class) display....");          //System.out.println("pri:\t"+f.pri);          //System.out.println("def:\t"+f.def);          //System.out.println("pro:\t"+f.pro);          System.out.println("pub:\t"+f.pub);      }  } |

Main Method class

Main.java

|  |
| --- |
| import p1.First;  import p1.Second;  import p1.Three;  import p2.Five;  import p2.Four;  import p2.Sixth;  public class Main {      public static void main(String[] args) {          //p1.First f=new p1.First();          First f=new First();          Second s=new Second();          Three t=new Three();          Four fr=new Four();          Five fv=new Five();          Sixth st=new Sixth();          f.display();          s.display();          t.display();          fr.display();          fv.display();          st.display();      }  }  Output:  p1.First(Same class) display....  pri: 10  def: 20  pro: 30  pub: 40  p1.Second(Other Same class) display....  def: 20  pro: 30  pub: 40  p1.Three(Child class) display....  def: 20  pro: 30  pub: 40  p2.Four(Sub class) display....  pro: 30  pub: 40  p2.Five(SubSub class) display....  pro: 30  pub: 40  p2.Sixth(Other class) display....  pub: 40 |

Why we override a method?

|  |
| --- |
| If we want base class instance method in child class, with different implementation then we override a method. |

Example on Method Overriding

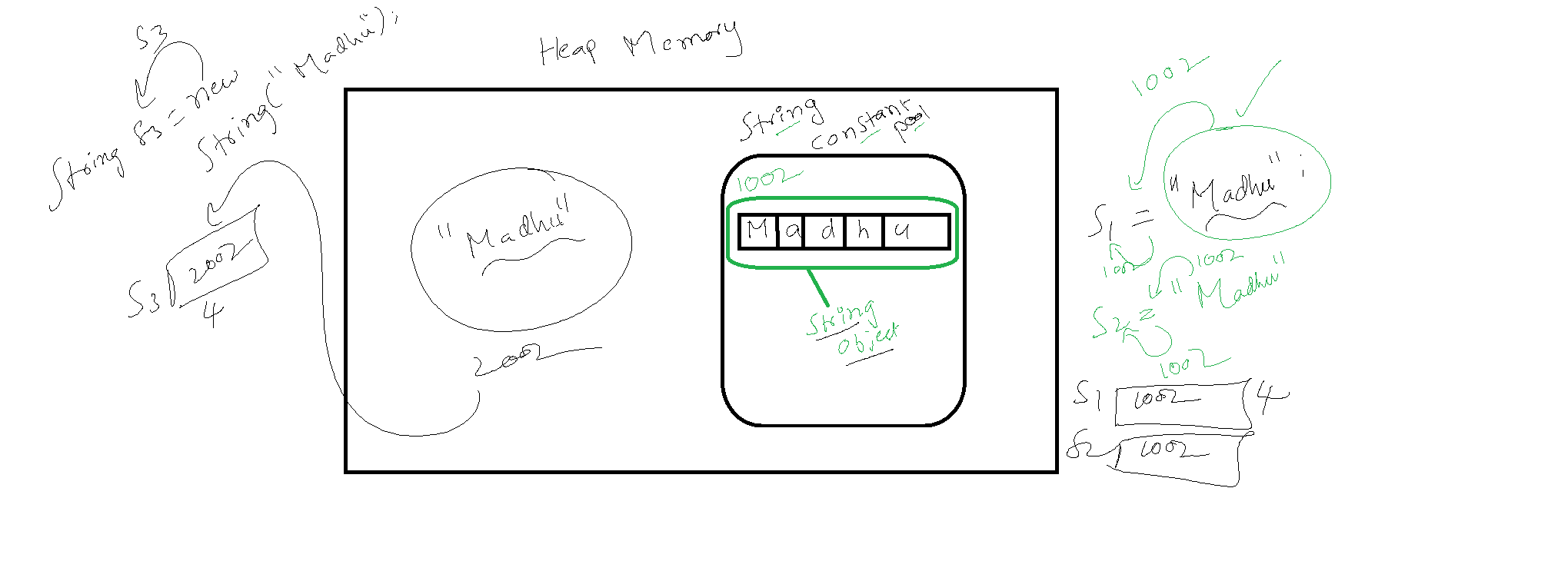
|  |
| --- |
| class Bird{      String name,color,breed,food;      Bird(String name,String color,String breed,String food){          this.name=name;          this.color=color;          this.breed=breed;          this.food=food;      };      void fly(){          System.out.println(name+" can fly");      }      void eat(){          System.out.println(name+" eats  "+food);      }      void sing(){          System.out.println(name+" can't sing ");      }      void talk(){          System.out.println(name+" can't talk ");      }  }  class Parrot extends Bird{      Parrot(String name,String color,String breed,String food){          super(name,color,breed,food);      }      @Override      void talk(){          System.out.println(name+" can talk ");      }  }  class Ostritch extends Bird{       Ostritch(String name,String color,String breed,String food){          super(name,color,breed,food);      }      @Override      void fly(){          System.out.println(name+" can't fly");      }  }  public class MethodOverride {      public static void main(String[] args) {          Parrot p=new Parrot("Parrot","Green With Red beek","Indina Parrot","Seeds, Guava etc..");          p.fly();          p.eat();          p.sing();          p.talk();          Ostritch o=new Ostritch("Ostritch", "White & Black", "African Ostritch","Lizards, seeds, etc..");          o.fly();          o.eat();          o.sing();          o.talk();      }  }  Output:  Parrot can fly  Parrot eats Seeds, Guava etc..  Parrot can't sing  Parrot can talk  Ostritch can't fly  Ostritch eats Lizards, seeds, etc..  Ostritch can't sing  Ostritch can't talk |

What is the use of @Override annotation?

|  |
| --- |
| It is used to inform the compiler to verify whether we are overriding a method correctly or not. |

**How the memory allocation will be done for String literals?**

|  |
| --- |
| String name1=”Madhu”;  For the above statement a new object will be created in string constant pool, if string object with same content is not present already and returns its address.  String name2=”Madhu”;  If the above statement executes old object address will be assigned to name2. i.e.. name1 and name2 contains same address.  String name3=new String(“Madhu”);  Here JVM creates new object in heap memory (not in string constant pool). Without checking about old object with same content. I.e.. name1 and name2 contains same address but name3 contains different address. |



**Example:**

|  |
| --- |
| public class StringDemo1 {      public static void main(String[] args) {          String s1="Madhu";          String s2="Madhu";          String s3=new String("Madhu");          System.out.println(s1==s2);          System.out.println(s1==s3);          System.out.println(s1.equals(s3));      }  }  Output:  **true**  **false**  **true** |

**Multilevel Inheritance**

|  |
| --- |
| * 1. If a child class is inherited by another child class it is called as multilevel inheritance   2. It is an extension to single level |

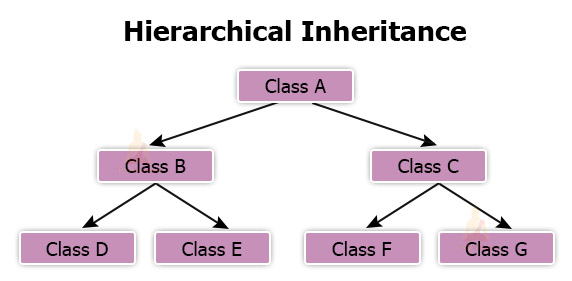
**What is the advantage of inheritance?**

|  |
| --- |
| * + 1. Re-usability |

|  |
| --- |
| class Base{      int a;      Base(int a){          this.a=a;      }      void display(){          System.out.printf("a:\t%d%n",a);      }  }  class Child extends Base{      Child(int a,int b){          super(a);          this.b=b;      }      int b;      void display(){          super.display();          System.out.printf("b:\t%d%n",b);      }  }  class SubChild extends Child{      SubChild(int a,int b,int c){          super(a,b);          this.c=c;      }      int c;      void display(){          super.display();          System.out.printf("c:\t%d%n",c);      }  }  public class MultilevelInher {      public static void main(String[] args) {          SubChild sb1=new SubChild(100, 200,300);          sb1.display();      }  }  Output:  a: 100  b: 200  c: 300 |

**What is hierarchical inheritance**

|  |
| --- |
| If a base class is inherited by more than one sub class then it is called as hierarchical inheritance. |



**Example on hierarchical inheritance**

|  |
| --- |
| class Bird{      String name,color,breed,food;      Bird(String name,String color,String breed,String food){          this.name=name;          this.color=color;          this.breed=breed;          this.food=food;      };      void fly(){          System.out.println(name+" can fly");      }      void eat(){          System.out.println(name+" eats  "+food);      }      void sing(){          System.out.println(name+" can't sing ");      }      void talk(){          System.out.println(name+" can't talk ");      }  }  class Parrot extends Bird{      Parrot(String name,String color,String breed,String food){          super(name,color,breed,food);      }      @Override      void talk(){          System.out.println(name+" can talk ");      }  }  class Ostritch extends Bird{       Ostritch(String name,String color,String breed,String food){          super(name,color,breed,food);      }      @Override      void fly(){          System.out.println(name+" can't fly");      }  }  public class MethodOverride {      public static void main(String[] args) {          Parrot p=new Parrot("Parrot","Green With Red beek","Indina Parrot","Seeds, Guava etc..");          p.fly();          p.eat();          p.sing();          p.talk();          Ostritch o=new Ostritch("Ostritch", "White & Black", "African Ostritch","Lizards, seeds, etc..");          o.fly();          o.eat();          o.sing();          o.talk();      }  }  Output:  Parrot can fly  Parrot eats Seeds, Guava etc..  Parrot can't sing  Parrot can talk  Ostritch can't fly  Ostritch eats Lizards, seeds, etc..  Ostritch can't sing  Ostritch can't talk |

**Relationships**

* 1. **Has-a-Relationship**
     1. Weak-has-a relationship
     2. Strong-has-a relationship
  2. **Is-a-Relationship**

Has-A-Relationship

|  |
| --- |
| If an object contains another object it is called as has a relationship  Example: Engine is an object and Car is an object, here the relationship is Car has engine. |

When the objects will be destroyed from heap memory?

|  |
| --- |
| Whenever the reference count becomes zero then those objects will be destroyed from heap by GC (Garbage Collector). |

Can we invoke Garbage Collector explicitly?

|  |
| --- |
| Yes, we can do it by calling gc() method of System class.  Note: gc() is a static method existed in System class. |

Example on Weak Has-A-Relation

|  |
| --- |
| class Engine{      private String engineNo;      Engine(String engineNo){          //engineNo=3002          this.engineNo=engineNo;      }      public void finalize(){          System.out.println("Engine object kooda destroy iyyindeee...");      }  }  class Car1{      String name,model;      Engine engine;      Car1(){          //name=null;model=null;engine=null;          System.out.println("Car1 constructor is invoked");      }      public void finalize(){          System.out.println("Car1 object is destroyed");      }  }  public class HasARelationship {      public static void main(String[] args) {              Engine engine=new Engine("F8DN1234567");              for(int i=1;i<=1;i++){                  Car1 car1=new Car1();                  car1.engine=engine;              }      }  }  Output:  Car1 constructor is invoked  Car1 object is destroyed |

Example on Strong Has A Relation

|  |
| --- |
| class Engine{      private String engineNo;      Engine(String engineNo){          //engineNo=3002          this.engineNo=engineNo;      }      public void finalize(){          System.out.println("Engine object kooda destroy iyyindeee...");      }  }  class Car2{      Engine engine;      Car2(){          engine=new Engine("L15A34567890");      }      public void finalize(){          System.out.println("Car2 object is destroyed");      }  }  public class HasARelationship {      public static void main(String[] args) {              //System.gc();              for(int i=1;i<=1;i++){                  Car2 car2=new Car2();              }              System.gc();      }  }  Output:  Engine object kooda destroy iyyindeee...  Car2 object is destroyed |

What are accessor methods and mutator methods?

|  |
| --- |
| Instance methods are divided into 2 categories   * + 1. Accessor methods     2. Mutator methods |

Example on Accessor and mutator methods

|  |
| --- |
| class One  {      private int a,b;      //setA() is a mutator method      void setA(int a){          this.a=a;      }      //setB() is a mutator method      void setB(int b){          this.b=b;      }      //getA() and getB() methods are accessor methods      int getA()  //o1=1002      {          return a;  //return o1.a      }      int getB(){          return b;      }  }  public class AccessMutatorMethods {      public static void main(String[] args) {          One o1=new One();          o1.setA(10);          o1.setB(20);          int x=o1.getA();          System.out.printf("x=%d%n",x);          System.out.printf("y=%d%n",o1.getB());      }  }  Output:  x=10  y=20 |

Static & Dynamic Polymorphism

What is static binding?

|  |
| --- |
| * At the time of compilation, if compiler decides what should be accessed at runtime it is called as compile time binding or static binding. * It is also called as compile time binding or early binding   (married to sister’s daughter) |

What is run time binding?

|  |
| --- |
| * 1. At the time program execution JVM decides what should be accessed or executed. It is called as runtime binding.   2. Runtime binding is also called as dynamic or late binding   (love marriage) |

What is upcasting?

|  |
| --- |
| * 1. Process of assigning child class object to Base type reference variable is called as upcasting   2. It is also called as widening reference conversion |

What is down casting (narrowing reference conversion)?

|  |
| --- |
| Process of assigning Base type reference variable to Child type reference variable is called as down casting. |

Example on UpDownCasting

|  |
| --- |
| class Base{}  class Child extends Base{}  public class StaticDynaBind1{      public static void main(String[] args) {          //upcasting (Widening reference conversion)          Base b1=new Child();          //Downcasting          Child c1=(Child)b1;        }  }  Note: we can’t assign Base class object to Child type reference variables, if you do this we will get ClassCastException at run time. |

Example on compile time and runtime binding as well as dynamic polymorphism?

|  |
| --- |
| class Base{      static int s=100;      int i=200;      void display(){          System.out.println("Instance method of Base");      }      static void get(){          System.out.println("Static method of Base");      }  }  class Child extends Base{      static int s=1000;      int i=2000;      @Override      void display(){          System.out.println("Instance method of Child");      }      static void get(){          System.out.println("Static method of Child");      }  }  public class StaticDynaBind1{      public static void main(String[] args) {          Base b1=new Child();          System.out.println(b1.s);          System.out.println(b1.i);          b1.get();          b1.display();          /\*          q1) is it instance method(yes)          q2) is it done upcasting(yes)          q3) is it overridden(yes)          Now display method of Child class will be invoked by JVM          \*/      }  }  Output:  100  200  Static method of Base  Instance method of Child |

What Is dynamic polymorphism?

|  |
| --- |
| At the time implementing polymorphism, if dynamic binding is performed then it is called as dynamic polymorphism. |

What Is static polymorphism?

|  |
| --- |
| At the time implementing polymorphism, if static binding is performed then it is called as static polymorphism. |

Example on static or compile time polymorphism?

|  |
| --- |
| class MyMath{      int add(int a,int b){          return a+b;      }      int add(int a,int b,int c){          return a+b+c;      }      float add(float a,int b){          return a+b;      }  }  class MeraMath extends MyMath{}  class NaaMath extends MyMath{      @Override      int add(int a,int b){          return a\*2+b\*2;      }  }  public class MethodOverloading {     public static void main(String[] args) {          MyMath mm=new MyMath();          System.out.println( mm.add(10,2)  );          System.out.println( mm.add(10,2,5)  );          System.out.println( mm.add(10.50f,2)  );          MyMath mm2= new MeraMath();//upcasting          System.out.println( mm2.add(10,2)  );          NaaMath nm=new NaaMath();          System.out.println( nm.add(10, 5) );     }  }  Output:  12  17  12.5  12  30 |

What is a jar (Java Application Archive) file?

|  |
| --- |
| A **JAR (Java Applicaion Archive) file** is a packaged file format used in Java to bundle multiple files into a single archive. It is essentially a **compressed file (ZIP format)** that can contain Java class files, metadata, configuration files, and resources like images, sound files, or libraries. The .jar extension represents these files. |

How to create a jar file?

|  |
| --- |
| By using the tool called jar we can create jar files  Example: |

Can I use the classes and packages existed in jar file in our system without extracting it?

|  |
| --- |
| Yes, we can but you have to set the classpath for that jar file. |

How to set the class path?

|  |
| --- |
| * + 1. Open the edit the system environment variables from windows search box     2. Click on Environment Variables button     3. Click on new button      * 1. Give classpath as variable name and path of the jar file along with it’s name and put the semicolon at the end of the path (see the above image)   2. Ok -> ok -> OK ->   3. DONE   Note: Now you can use the classes and packages existed in myapi.jar file anywhere within the computer  Tokkaley.java  import p1.First;  public class Tokkaley  {  public static void main(String args[])  {  new First().display();  }  }    I have written the above program in documents folder and I am able to access the p1 package here which is existed in myapi.jar file. |

Can I use the classes and packages existed in programs folder anywhere within the computer without creating .jar file?

|  |
| --- |
| Yes we can use then if you set the classpath for the programs folder. (see the below image)      Open the new command prompt and Try to access classes and packages like below |

What is a classpath?

|  |
| --- |
| * It is an environment variable which is used to set the path for .class files and packages, so that we can access them from anywhere within the system. |

What is temporary class path?

|  |
| --- |
| If you set the classpath within a command prompt, the classes and packages can be used only through that command prompt. If you close the command prompt then class will be gone.    Here %classpath% represents old classpath |

* 1. Streams
  2. Reading data from a file by using Scanner class methods
  3. Customized class loader
  4. Inheritance
  5. Instance of operator
  6. Strings and string handling