Day - 1:

Software is a development process which converts the imaginaries into reality by writing comes set of programs.

![Diagram of Software分类]

Day - 2:

In IT we develop two types of applications. They are stand alone applications and distributed applications.

A stand alone application is one which runs in the context of local disk. All stand alone applications are not sharable. System software and application software comes under stand alone applications.

- **System software** is used for developing functionality of hardware devices. Examples are C and ALP (Assembly Language Programming).
- **Application software** is used for developing organizations oriented applications. This is also known as backend software’s. Examples are dbase, dbase III plus, FoxPro, oracle versions released till now.
- **Internet software** is used for developing distributed applications.

A distributed application is one which can be accessed across the globe. Distributed application is also one which runs in the contest of the World Wide Web. To develop distributed applications we must use client-server architecture.

In client-server architecture we must hare at least two programs they are **client program** and **server program**. A client program is one which always makes a request to get the service from the server. A server program is one which will do three operations receiving the request from client, processing the client request and sending the response to the client.

All the above three operations are performed by the server concurrently. In order to exchange the data between client and server we must use a protocol called http (hypertext transfer protocol).
Protocol is a set of values which are defined to exchange the data between client and server either locally or remotely.

In order to develop distributed applications, two software companies came forward whose names are Microsoft System and Sun Micro System.

Day - 3:

Microsoft System has developed a technology called DOT NET and Sun Micro System has developed a technology called JAVA. Both this technologies are called distributed technologies.

The technology DOT NET will run only on that operating system’s which are provided by Microsoft (as on today). Hence DOT NET technology is platform dependent technology. Whereas, the technology called JAVA will run on all operating system’s irrespective of their providers hence JAVA is called platform independent technology.

The slogan of JAVA is “Write One’s Reuse/Run Anywhere (WORA)”.
**DOT NET** technology is not freely downloadable where as **JAVA** freely downloadable. **DOT NET** supports Microsoft developed **design patterns** (general designed patterns are not supported) whereas **JAVA** supports all the **design patterns** which are available in IT.

**Design pattern** is a predefined and proved rule by third party industry experts to avoid the receiving problems which are occurring in software development.

**History of JAVA:**

**JAVA** is a **distributed technology** developed by **James Gosling**, **Patric Naugton**, etc., at **Sun Micro System** has released lot of rules for **JAVA** and those rules are implemented by **JavaSoft Inc, USA** (which is the software division of **Sun Micro System**) in the year 1990. The original name of **JAVA** is **OAK** (which is a tree name). In the year 1995, **OAK** was revised and developed software called **JAVA** (which is a coffee seed name).

**JAVA** released to the market in three categories **J2SE** (**JAVA** 2 Standard Edition), **J2EE** (**JAVA** 2 Enterprise Edition) and **J2ME** (**JAVA** 2 Micro/Mobile Edition).

i. **J2SE** is basically used for developing client side applications/programs.
   ii. **J2EE** is used for developing server side applications/programs.
   iii. **J2ME** is used for developing server side applications/programs.

If you exchange the data between client and server programs (**J2SE** and **J2EE**), by default **JAVA** is having on internal support with a protocol called **http**. **J2ME** is used for developing mobile applications and lower/system level applications. To develop **J2ME** applications we must use a protocol called **WAP** (Wireless Applications Protocol).

**Day - 4:**

**FEATURES of java:**

1. Simple
2. Platform independent
3. Architectural neutral
4. Portable
5. Multi threading
6. Distributed
7. Networked
8. Robust
9. Dynamic
10. Secured
11. High performance
12. Interpreted
13. Object Oriented Programming Language
1. **Simple**: JAVA is simple because of the following factors:
   
   i. JAVA is free from pointers hence we can achieve less development time and less execution time [whenever we write a JAVA program we write without pointers and internally it is converted into the equivalent pointer program].
   
   ii. Rich set of API (application protocol interface) is available to develop any complex application.

   iii. The software JAVA contains a program called **garbage collector** which is always used to collect unreferenced (unused) memory location for improving performance of a JAVA program. [Garbage collector is the system JAVA program which runs in the background along with regular JAVA program to collect unreferenced memory locations by running at periodical interval of times for improving performance of JAVA applications.]

   iv. JAVA contains user friendly syntax’s for developing JAVA applications.

2. **Platform Independent**:

   A program or technology is said to be platform independent if and only if which can run on all available operating systems.

   The languages like C, Cpp are treated as platform dependent languages since these languages are taking various amount of memory spaces on various operating systems [the operating system dos understands everything in the form of its native format called Mozart (MZ) whereas the operating system Unix understands everything in its negative format called embedded linking format (elf). When we write a C or Cpp program on dos operating and if we try to transfer that program to Unix operating system, we are unable to execute since the format of these operating systems are different and more over the C, Cpp software does not contain any special programs which converts one format of one operating system to another format of other operating system].

   The language like JAVA will have a common data types and the common memory spaces on all operating systems and the JAVA software contains the special programs which converts the format of one operating system to another format of other operating system. Hence JAVA language is treated as platform independent language.

**DAY - 5**:

JAVA language is also treated as server independent language since the server side program can run on any of the server which is available in the real world (web server or application server). JAVA can retrieve or store the data in any one of the data base product which is available in rest world irrespective of their vendors (developers) hence JAVA language is product independent language.

In order to deal with server side program from the client side, we can use C language client program, Cpp client program, DOT NET client program, etc. hence JAVA language is a simple, platform independent, server independent, data base/product independent and language independent programming language.
3. **Architectural Neutral:**

A language or technology is said to be *architectural neutral* which can run on any available processors in the real world. The languages like C, Cpp are treated as *architectural dependent*. The language like JAVA can run on any of the processor irrespective of their architecture and vendor.

![Diagram showing Matrix Multiplication with and without pointers.](image)

<table>
<thead>
<tr>
<th>With pointers</th>
<th>Without pointers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Development time more</td>
<td>1. Development time less</td>
</tr>
<tr>
<td>2. Execution time less</td>
<td>2. Execution time more</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Machine1</th>
<th>Machine2</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOS</td>
<td>UNIX</td>
</tr>
<tr>
<td>P3</td>
<td>AMD</td>
</tr>
<tr>
<td>JMK</td>
<td>J0K</td>
</tr>
</tbody>
</table>

**Portability = platform independent + architecture neutral**

4. **Portable:**

A portable language is one which can run on all operating systems and on all processors irrespective their architectures and providers. The languages like C, Cpp are treated as non-portable languages whereas the language JAVA is called *portable* language.
5. **Multi Threading:**

![Diagram of single instance and server with multiple flows]

**Definitions:**

1. A *flow of control* is known as a *thread*.
2. A *multi threaded program* is one in which there exists *multiple flow of controls* i.e., *threads*.
3. A program is said to be *multi threaded program* if and only if there exists *n* number of subprograms. For each and every sub-program there exists a separate *flow of control*. All such *flow of controls* are executing concurrently. Such *flow of controls* is known as *threads*. Such type of applications is known as *multi threading applications*.
4. The languages like C, Cpp are treated as *threads* as *single threaded modeling languages (STML)*. SMTL are those in which there exists *single flow of control*.
5. The languages like JAVA and DOT NET are treated as *multi threaded modeling languages (MTML)*. MTML are those in which there exist *multiple flows of controls*.
6. Whenever we write a JAVA program there exists by default *two threads*. They are *foreground/child thread* and *background/main/parent thread*.
7. A *foreground thread* is one which always executes user defined sub-programs. In a JAVA program there is a possibility of existing *n* number of *foreground threads*.
8. A *background thread* is one which always monitors the status of *foreground thread*. In each and every JAVA program there exists only one background thread.
9. Hence *background thread* will be created first and later *foreground thread* will be created.

6. **Distributed:**

A service is a said to be a *distributed* service which runs in *multiple servers* and that service can be accessed by *n* number of clients across the globe. In order to develop *distributed applications* we must require architecture called *trusted network* architecture. To develop these applications we require a technology called *J2EE*. *Distributed applications* are preferred by large scale organizations.
Day-7:

7. **Networked:**

In real world we have two types of networks. They are **un-trusted networks** and **trusted networks**.

**Un-trusted networks:**

A network is said to be **un-trusted network** in which there exists n number of inter connected non-autonomous architecture. **Un-trusted network** is also known as **LAN**. Using this network architecture, we develop centralized applications. A centralized application is one which runs on **single server** and it can be access in limited graces. In order to develop centralized applications we may use a technology called **J2SE** and these kinds of applications are preferred by small scale organization.

**Trusted network:**

A network is said to be **trusted network** in which there exists n number of inter connected autonomous architecture. **Trusted network** is also known as **WAN**. Using this network, we can develop distributed applications. A distributed application is one which runs on **multiple servers** and it can be access in unlimited graces. In order to develop distributed applications we may use
a technology called J2EE and these kinds of applications are preferred by large scale organization.

**Java is OBJECT ORIENTED PROGRAMMING language:**

In an IT we have two types of programming models (paradigms) are available. They are procedure oriented programming language and object oriented programming language.

If we represent the data using procedural oriented programming language then there is no security for the data which we represent. For example when we represent the data of a student in C language using structures concept, the student data can be accessed by all the functions which we write as a part of C program. If one of the functions manipulates or damages the data then we are loosing correction-less (integrity) of the data. Examples of procedure oriented programming languages are FORTRON, COBOL, PASCAL, BASIC, C, etc.

When we represent the data in object oriented programming language we get the security. Examples of object oriented programming languages are LISP, ADA, ALGOL, SMALLTALK, OBJECT COBOL, OBJECT PASCAL, Cpp, JAVA, DOT NET, etc. In order to say any language is an object oriented programming language it has to satisfy 8 principles of OOPs.

**OOPs Principles:**

1. Class.
2. Object.
3. Data Abstraction.
4. Data Encapsulation.
5. Inheritance.
6. Polymorphism.
7. Dynamic Binding.
Day - 8:

1. **CLASS**: “A class is a way of binding the data and associated methods in a single unit".

   Any JAVA program if we want to develop then that should be developed with respective class only i.e., without class there is no JAVA program.

   In object oriented programming's, generally we write two types of methods. They are member methods and non-member methods.

   - A member method is one which is comes under the scope of the class. In JAVA we use only member methods.
   - Non-member methods are those which are not comes under the scope of the class. JAVA does not allow non-member methods at all.

*Class* diagram for defining a *class*:

![Class Diagram](image)

**Syntax for defining a CLASS:**

```java
Class <clsname>
{
    Variable declaration;
    Methods definition;
}
```

Here, class is a keyword which is used for developing or creating user defined datatypes. Clsname represents a JAVA valid variable name and it is treated as name of the class. Class names are used for creating objects.

Class contains two parts namely variable declaration and method definitions. Variable declaration represents what type of data members which we use as a part of the class. Method definition represents the type of methods which we used as the path of the class to perform an operation.

By making use of the variables, which are declared inside the class? Every operation in JAVA must be defined with in the class only i.e., outside definition is not possible

Example: Define a class called a student..?

**Answer:**

```java
Class student
{
    Int stno;
    String stname;
```
Float marks;
String cname;
Int getnohoursstudy ()
{
    ........
    ........
}
String getgrade ()
{
    ........
    ........
}
} [;]—optional

Whenever we define a class there is no memory space for data members of the class. Memory space will be created for the data members of the class when we create object.

NOTE:
1. In JAVA memory space for the data members will be creating on heap memory (Dynamic memory).
2. Memory space for methods will be creating on stack memory (that too when we call the methods).
3. All constants of any JAVA program is available in associative memory (retrieving data from associative memory is negligible).
4. The class definition exists only one time but whose objects can exists many number of times i.e., a class is acting as a formula form.

Day - 9:

2. OBJECT: In order to store the data for the data members of the class, we must create an object.
   1. Instance (instance is a mechanism of allocating sufficient amount of memory space for data members of a class) of a class is known as an object.
   2. Class variable is known as an object.
   3. Grouped item (grouped item is a variable which allows us to store more than one value) is known as an object.
   4. Value form of a class is known as an object.
   5. Blue print of a class is known as an object.
   6. Logical runtime entity is known as an object.
   7. Real world entities are called as objects.

NOTE:
- JAVA always follows dynamic memory allocation but not static memory allocation.
- In order to create a memory space in JAVA we must use an operator called new. This new operator is known as dynamic memory allocation operator.
Syntax-1 for defining an OBJECT:

```java
<Clsname> objname = new <clsname ()>
```

*Clsname* represents name of the class. *Objname* represents JAVA valid variable name treated as object. *New* is called *dynamic memory allocation operator*.

*Clsname ()* represents *constructor*. The *new* operator will perform two standard actions. They are:

i. It allocates sufficient amount of memory space for the *data members* of the class.

ii. It takes an address of the class and stored in the left hand side variable of syntax-1.

Syntax-2 for defining an OBJECT:

```java
<Clsname> objname; //object declaration
Objname = new <clsname ()>; //object referencing
```

When an *object* is declared where value is null. Since, there is no memory space for *data members* of the class. When the *object* is referenced the value of the *object* is not null. Since, memory space is created for the *data members* of the class.
“The difference between class and object..?”

<table>
<thead>
<tr>
<th>Class</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) A class is a way of binding the data and</td>
<td>1) Class variable is known as an</td>
</tr>
<tr>
<td>associated methods in a single unit.</td>
<td>object.</td>
</tr>
<tr>
<td>2) Whenever we start executing a JAVA</td>
<td>2) After loading the class into</td>
</tr>
<tr>
<td>program, the class will be loaded into</td>
<td>main memory, objects can be</td>
</tr>
<tr>
<td>main memory with the help of class</td>
<td>created in n number.</td>
</tr>
<tr>
<td>loader subsystem (a part of JVM) only once.</td>
<td></td>
</tr>
<tr>
<td>3) When the class id defined there is no</td>
<td>3) When an object is created we</td>
</tr>
<tr>
<td>memory space for data members of a class.</td>
<td>get the memory space for data</td>
</tr>
<tr>
<td></td>
<td>members of the class.</td>
</tr>
</tbody>
</table>

3. **Data Abstraction:**

“The data abstraction is a mechanism of retrieving the essential details without dealing with background details”.

**Day - 10:**

In real world we have three levels of abstractions. They are physical level abstraction, conceptual/logical level abstraction and view level abstraction.

- Physical level abstraction is one; it always deals with physical organization architecture of the application. For example, in real world an application designing of any problem comes under physical level abstraction.

<table>
<thead>
<tr>
<th>Class</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) A class is</td>
<td>1) Class variable is known as</td>
</tr>
<tr>
<td>a way of</td>
<td>an object.</td>
</tr>
<tr>
<td>binding the</td>
<td></td>
</tr>
<tr>
<td>data and</td>
<td></td>
</tr>
<tr>
<td>associated</td>
<td></td>
</tr>
<tr>
<td>methods in a</td>
<td></td>
</tr>
<tr>
<td>single unit.</td>
<td></td>
</tr>
<tr>
<td>2) Whenever</td>
<td>2) After loading the class into</td>
</tr>
<tr>
<td>we start</td>
<td>main memory, objects can be</td>
</tr>
<tr>
<td>executing a</td>
<td>created in n number.</td>
</tr>
<tr>
<td>JAVA program,</td>
<td></td>
</tr>
<tr>
<td>the class will</td>
<td></td>
</tr>
<tr>
<td>be loaded into</td>
<td></td>
</tr>
<tr>
<td>main memory</td>
<td></td>
</tr>
<tr>
<td>with the help</td>
<td></td>
</tr>
<tr>
<td>of class</td>
<td></td>
</tr>
<tr>
<td>loader subsystem (a part of JVM) only once.</td>
<td></td>
</tr>
<tr>
<td>3) When the</td>
<td>3) When an object is created we</td>
</tr>
<tr>
<td>class id</td>
<td>get the memory space for data</td>
</tr>
<tr>
<td>defined there</td>
<td>members of the class.</td>
</tr>
<tr>
<td>is no memory</td>
<td></td>
</tr>
<tr>
<td>space for data</td>
<td></td>
</tr>
<tr>
<td>members of a</td>
<td></td>
</tr>
<tr>
<td>class.</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:**

In real world we have three levels of abstractions. They are physical level abstraction, conceptual/logical level abstraction and view level abstraction.

<table>
<thead>
<tr>
<th>Emp table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expno</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
• **Conceptual/logical level abstraction** is one it always deals with what kind of **data** we are entering **without dealing** with **physical architecture** of the application. For example, entering the data into the database, writing the coding and applying testing principle comes under **conceptual level abstraction**.

<table>
<thead>
<tr>
<th>Enrno</th>
<th>Ename</th>
<th>Sul</th>
</tr>
</thead>
<tbody>
<tr>
<td>143</td>
<td>suman</td>
<td>10000</td>
</tr>
<tr>
<td>144</td>
<td>kalyan</td>
<td>15000</td>
</tr>
</tbody>
</table>

• **View level abstraction** deals with what kind of **data** we are retrieving **without dealing** with both **conceptual level abstraction** and **physical level abstraction**. For example, retrieving the data from the **data base** in various combinations. All internet users come under **view level abstraction**.

4. **Data Encapsulation**:
   “Data encapsulation is the process of **wrapping up on data and associated methods in a single unit**”.
   - Data encapsulation is basically used for **achieving data/information hiding** i.e., security.
   - When we want to **send the data** from client to the server we must always send in the form of JAVA **object** only. Since, by default the JAVA object is in **encrypted** form (we should not send the data from client to the server in the form of fundamental data).

5. **Inheritance**:
   - **Inheritance** is the process of taking the features (**data members + methods**) from one **class** to another **class**.
   - The **class** which is **giving the features** is known as base/parent **class**.
   - The **class** which is **taking the features** is known as derived/child/sub **class**.
   - **Instance** is known as sub classing or derivation or extendable classes or reusability.

**Advantages of INHERITANCE**:
I. Application development time is very less.
II. Redundancy (repetition) of the code is reducing. Hence we can get less memory cost and consistent results.
III. Instrument cost towards the project is reduced.
IV. We can achieve the slogan write one’s reuse/run anywhere (WORA) of JAVA.

**Day - 11**:

**Types of INHERITANCES** (Reusable techniques):
Based on taking the features from **base class** to the **derived class**, in JAVA we have five types of inheritances. They are as follows:
i. **Single Inheritance:**
   *Single class* is one in which there exists *single base class* and *single derived class*.

   ![Diagram of Single Inheritance](image)

ii. **Multi Level Inheritance:**
   *Multi level inheritance* is one which there exist *single base class*, *single derived class* and *n number of intermediate base classes*.

   ![Diagram of Multi Level Inheritance](image)

   An *intermediate base class* is one, in *one context* it acts as *base class* and in *another context* it acts as *derived class*.

iii. **Hierarchical Inheritance:**
   *Hierarchical inheritance* is one in which there exists *single base class* and *n number of derived classes*.

   ![Diagram of Hierarchical Inheritance](image)

iv. **Multiple Inheritances:**
   *Multiple inheritance* is one in which there exists *n number of base classes* and *single derived classes*.

   ![Diagram of Multiple Inheritances](image)

   *Multiple inheritances* are *not supported* by JAVA through *classes* but it is *supported* by JAVA through the concept of *interfaces*. 
v. **Hybrid Inheritance:**

*Hybrid inheritance* = **combination** of any available *inheritances types.*

In the *combination,* one of the combinations is *multiple inheritances.*

*Hybrid inheritance* also will not be *supported* by JAVA through the concept of *classes* but it is supported through the concept of *interfaces.*

6. **Polymorphism:**

*Polymorphism* is a process of representing “one form in many forms”.

- In *object oriented programming’s,* we have two types of *polymorphism.* They are compile time *polymorphism* and run time *polymorphism*.
- JAVA does not support compile time *polymorphism* but JAVA supports only run time *polymorphism*.

In the above diagram we have a single sum method but it is defined in many forms hence that method is known as *polymorphic method.*

7. **Dynamic Binding:**

“Dynamic binding is a mechanism of binding an appropriate version of a derived class which is inherited from base class with base class object”.

• Every real time application will be developed with concept of polymorphism and executed with the concept of dynamic binding.

• Dynamic binding is basically used to reduce the amount of memory space for improving the performance of JAVA applications.

1) C1 O1 = new C1();
2) O1.sum();
3) O1 = new C2();
4) O1.sum();
5) O1 = new C3();
6) O1.sum();

In the line numbers 1, 3 and 5 the object oven contains address of c1, c2 and c3 classes respectively one at a time. Hence that object is known as polymorphic object. In the line numbers 2, 4 and 6 the statement O1.sum () is known as polymorphic statement.

NOTE:
• Function overriding = function heading is same but function definition is different.
• A function is said to be overloaded function if and only if function name is same but its signature (signature represents number of parameters, type of parameters and order of parameters) is different.

Day - 12:

8. Message Passing:
Definitions:
   i. Exchanging the data between multiple objects is known as message passing.
ii. *Message passing* is the process of **exchanging** the data between **two remote/local objects** for a period of time across the network (trusted) for generating **multiple requests** for obtaining **multiple responses** for performing a meaningful operation.

iii. *Message passing JAVA* is achieved through the concept of **methods**.
DATA TYPES in Java

“Data types are used for representing the data in main memory of the computer”.

In JAVA, we have eight data types which are organized in four groups. They are integer category data types, float category data types, character category data types and Boolean category data types.

1. Integer category data types:
   These are used to represent integer data. This category of data type contains four data types which are given in the following table:

<table>
<thead>
<tr>
<th>S. No</th>
<th>Data type</th>
<th>Size (bytes)</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Byte</td>
<td>1</td>
<td>+127 to -128</td>
</tr>
<tr>
<td>2</td>
<td>Short</td>
<td>2</td>
<td>+32767 to -32768</td>
</tr>
<tr>
<td>3</td>
<td>Int</td>
<td>4</td>
<td>+2147483647 to -2147483648</td>
</tr>
<tr>
<td>4</td>
<td>long</td>
<td>8</td>
<td>±9.223372036854775808 x 10^19</td>
</tr>
</tbody>
</table>

   Whatever the data type we use that should not exceed predefined value.

   NOTE: Range of any data type = (A) number of bits occupied by a data type

   Where, A = number of bits available in the language which is understand by computer i.e., 2 bits.

   For example:
   Range of byte = $2^8$
   = 1 to 256
   = 0 to 255
   = 0 to (255/2)
   = (127.5 - 0.5 = +127) (127.5 + 0.5 = -128)

2. Float category data types:
   Float category data types are used for representing the data in the form of scale, precision i.e., these category data types are used for representing float values. This category contains two data types; they are given in the following table:
Whenever we take any decimal constant directly in a JAVA program it is by default treated as highest data type in float category i.e., double.

3. **Character category data types:**
   - A character is an identifier which is enclosed within single quotes.
   - In JAVA to represent character data, we use a data type called char. This data type takes two bytes since it follows UNICODE character set.

   ![Character category data types](image)

<table>
<thead>
<tr>
<th>Data type</th>
<th>Size (bytes)</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Char</td>
<td>2</td>
<td>+32767 to -32768</td>
</tr>
</tbody>
</table>

   - JAVA is available in 18 international languages and it is following UNICODE character set.
   - UNICODE character set is one which contains all the characters which are available in 18 international languages and it contains 65536 characters.

**Day - 13:**

4. **Boolean category data types:**
   - Boolean category data type is used for representing logical values i.e., TRUE or FALSE values.
   - To represent logical values we use a keyword called Boolean.
   - This data type takes 0 bytes of memory space.

**NOTE:** All keywords in JAVA must be written in small letters only.
VARIABLES in java

“A variable is an identifier whose value will be changed during execution of the program”.

Rules for writing variables:

i. First letter must be an alphabet.
ii. The length of the variable should not exceed more than 32 characters.
iii. No special symbols are allowed except underscore.
iv. No keywords should use as variable names.

Types of variables in JAVA:

- Whenever we develop any JAVA program that will be developed with respect to class only.
- In a class we can use ‘n’ number of data members and ‘n’ number of methods.
- Generally in JAVA, we can use two types of data members or variables. They are instance/non-static variables and static variables.

INSTANCE/NON-STATIC VARIABLES

1) An instance variable is one whose memory space is creating each and every time whenever an object is created.

2) Programmatically instance variable declaration should not be preceded by keyword static.

3) Data type v1, v2…vn;

4) Instance variable must be accessed with respect to object name i.e., objname.varname;

5) Value of instance variable is not sharable.

6) Instance variable are also known as object level data members since they are dependent on objects.

STATIC VARIABLES

1) Static variables are whose memory space is creating only once when the class is loaded by class loader subsystem (a part of JVM) in the main memory irrespective of number of objects.

2) Programmatically static variable declaration must be preceded by keyword static.

3) Static data type v1, v2…vn;

4) Static variables must be accessed with respect to class name i.e., classname.varname;

5) Value of static variable is always recommended for sharable.

6) Static variable are also known as class level data members since they are dependent on classes.

CONSTANTS in java

“Constant is an identifier whose value cannot be changed during execution of the program”.

- In JAVA to make the identifiers are as constants, we use a keyword called final.
- Final is a keyword which is playing an important role in three levels. They are at variable level, at method level and at class level.
i. When we don’t want to change the value of the variable, then that variable must be declared as final.

**Syntax for FINAL VARIABLE INITIALIZATION:**
Final data type v1=val1, v2=val2 ... vn=valn;

**For example:**
Final int a=10;
   a=a+20;  //invalid
   a=30;   //invalid

ii. When the final variable is initialized, no more modifications or assignments are possible.

**Syntax for FINAL VARIABLE DECLARATION:**
Final data type v1, v2........vn;

**For example:**
Final int a;
   a=a+1;  //invalid
   a=10+2; //invalid
   a=400;  //valid for 1st time
   a=500;  //invalid

Whenever a final variable is declared first time assignment is possible and no more modification and further assignments are not possible. Hence, final variables cannot be modified.

**Day - 14:**

**PROGRAMMING BASIC’S**

**System.out.println (“”);**

- This statement is used for displaying the data or messages on to the console (monitor).
- Here, println is the predefined instance method of print stream class.
- To call this method we require an object called print stream class.
- The object of print stream class is called out is created as a static data member in system class (system is a predefined class).
- Hence to call the println method we must use the following statement:
  
  ```java
  System.out.println (“WELCOME TO JAVA”);
  ```

- **Print stream class** is a predefined class which contains nine overloaded instance println methods and nine overloaded instance print methods and whose prototypes are as follows:
  ```java
  Public void println (byte);
  Public void println (short);
  Public void println (int);
  Public void println (long);
  Public void println (float);
  Public void println (double);
  Public void println (char);
  ```
Day - 15:

For example 1:
Int a=40000;
System.out.println (a); //40000
System.out.println ("value of a=" + a); //value of a=40000
System.out.println (a + "is the value of a"); //40000 is the value of a

For example 2:
Int a=10, b=20, c;
C = a + b;
System.out.println (c); //30
System.out.println ("sum=" + c); //sum=30
System.out.println (c + "is the sum"); // 30 is the sum
System.out.println ("sum of" + a + "and" + b + "=" + c); //sum of 10 and 20 is 30

For example 3:
System.out.println ("WELCOME TO JAVA");

STRUCTURE of a java program

Package details:
class <clsname>
{
    Data member’s declaration;

    User defined methods;

    Public static void main (string k [])
    {
        Block of statements ();
    }
}

- Every program in JAVA must be developed with respect to class.
- Data member’s declaration represents the type of data members which we use as a part of the class.
• **User defined methods** represents the type of methods which we use as a part of the class to perform some meaningful operation by making use of the data members of class.

• Here **main** represents the name of the method where the program execution starts and **void** represents return type of **main method** which indicates **main method** does not return anything.

• Since **main method** is executing only once hence it must be static method. Since **main method** can be called/accessed by everybody and hence it belongs to public method.

• **Block of statements** represents the valid executable statements of JAVA which will call the user defined methods.

Write a JAVA program to display a message “welcome to JAVA”?

Answer:

```java
class First {
    public static void main (String k []) {
        System.out.println ("welcome to JAVA");
    }
}
```

**NOTE:**

• While giving the file name to a JAVA program we must give the file name as name of the class with an extension “.java” in which ever the class main method presents.

• Whenever we give println it will prints in the new line whereas print prints in the same line.
Write a JAVA program which illustrates the concept of instance methods and static methods?

**Answer:**

class Second {
    Void disp1 () {
        System.out.println ("disp1-instance");
    }
    Static void disp2 () {
        System.out.println ("disp2-static");
    }
    Public static void main (string k []) {
        System.out.println ("main-beginning");
        Second so = new Second ();
        so. disp1 ();
        Second. disp2 ();
        System.out.println ("main-end");
    }
} //second

**NOTE:**
1. **One static method can call another static method directly** provide both the static method belongs to same class.
2. **One instance method can call another instance method directly** provide both the static instance method belongs to same class.
Day - 16:

HUNGARIAN NOTATION:

Hungarian Notation is the naming convention followed by SUN (Stanford University Network) micro system to develop their predefined classes, interfaces, methods and data members.

Hungarian rule for CLASS or INTERFACE:

If a class object interface contains more than one word then we must write all the first letters must be capital.
For example:
System, NumberFormatException, ArrayIndexOutOfBoundsException

Hungarian rule for METHOD:

If a method name contains more than one word then first word letter is small and rest of the words first letters must be capital.
For example:
println (), actionPerformed (), adjustmentValueChanged ()

Hungarian rule for DATA MEMBERS:

All the data members are the predefined classes and interfaces must be represented used as capital letters.
For example:
PI, MAX_VALUE, and MIN_VALUE

All the data members in the predefined classes and interfaces are belongs to public static final XXX data members. XXX represents data type, variable name and variable value. Every final data member must belong to static data member but reverse may or may not be applicable.

Write a JAVA program to generate the multiplication table for the given number?
Answer:

class Mul //Business Logic Class (BLC)
{
    int n;
    void set (int x)
    {
        n=x;
    }
    void table ()
    {
        for (int i=1; i<=10; i++)
        {
            int res=n*i;
            System.out.println (n + “*” + i + “=” + result);
        }
    }
}
class MulDemo //Execution Logic Class (ELC)
{
Public static class main (string k [])
{
    Mul mo = new Mul ();
    mo.set (4);
    mo.table ();
}

NOTE:

• The data members of a class can be used in all the functions of the class.
• The variable which we use as a part of function heading is known as formal parameters and they can be used within the function only.
• The variables which we use as a part of function definition/body then those variables known as local variables and they can be used within its scope i.e., function definition.
• The variables which we use within the function call then those variables are known as actual parameters.

For example:
int n=4;
mo.set (n);
Where ’n’ is the actual parameter.

Day - 17:

The following table gives the details about those classes and methods used for converting storing data to fundamental data:

<table>
<thead>
<tr>
<th>DATA TYPES</th>
<th>WRAPPER CLASS</th>
<th>CONVERSION METHOD FROM STRING DATA TO FUNDAMENTAL DATA TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. byte</td>
<td>1. Byte</td>
<td>1. public static byte parseByte (string)</td>
</tr>
<tr>
<td>2. short</td>
<td>2. Short</td>
<td>2. public static short parseShort (string)</td>
</tr>
<tr>
<td>3. int</td>
<td>3. Integer</td>
<td>3. public static int parseInt (string)</td>
</tr>
<tr>
<td>4. long</td>
<td>4. Long</td>
<td>4. public static long parseLong (string)</td>
</tr>
<tr>
<td>5. float</td>
<td>5. Float</td>
<td>5. public static float parseFloat (string)</td>
</tr>
<tr>
<td>6. double</td>
<td>6. Double</td>
<td>6. public static double parseFloat (string)</td>
</tr>
<tr>
<td>7. char</td>
<td>7. Char</td>
<td>7. public static char parseChar (string)</td>
</tr>
<tr>
<td>8. boolean</td>
<td>8. Boolean</td>
<td>8. public static boolean parseBoolean (string)</td>
</tr>
</tbody>
</table>

Whenever we develop a JAVA program it is recommended to write ‘n’ number of business logic classes and single execution logic class. A business logic class is one which contains ‘n’ number of user defined methods in which we write business logic. Business logic is one which is provided by JAVA programmer according business rules (requirements) started by customer. Whatever data we represent in JAVA runtime environment it is by default treated as objects of string data. As a programmer when we start any JAVA program an object of string class is created depends on number of values we pass.
Wrapper classes are basically used for converting the string data into fundamental data type. Each and every wrapper class contains the following generalized parse methods.

```
public static Xxx parseXxx (String);
```

here, Xxx represents fundamental data type.

Length is an implicit attribute created by JVM to determine number of elements or size of the array.

For example:
```
int a [] = {10, 20, 30, 40};
System.out.println (a. length); //4
String s1 [] = {10, 20, 30};
System.out.println (s1.length); //3
```

class Data
{
    public static void main (String s [])
    {
        int noa=s.length;
        System.out.println ("NUMBER OF VALUES ="+noa);
        System.out.println (s [0] +" "+s [1]);
        int x=Integer.parseInt (s [0]); //8
        int y=Integer.parseInt (s [1]); //9
        int z=x+y;
        System.out.println ("SUM ="+z);
    }
}
```

8 and 9 lines used for converting string into int data type.

Write a JAVA program for printing the data which we pass from command prompt?

Answer:
```
class Print
{
    public static void main (String k [])
    {
        System.out.println ("NUMBER OF VALUES ="+k.length);
        for (int i=0; i<k.length; i++)
        {
            System.out.println (k[i] +" ");
        }
    }
}
```

Write a JAVA program which computes sum of two numbers by accepting the data from command prompt (DOS)?

Answer:
```
class Sum
{
    int a, b, c; //a,b,c are data members
    void assign (int x, int y) //x,y are formal parameters
    {
        a=x;
        b=y;
    }
}
```
void add() //add() is business logic method
{
    c = a + b;
}
void disp() //disp() is business logic method
{
    System.out.println("SUM OF "+a+" AND "+b+" = "+c);
}

class SumDemo
{
    public static void main(String k[])
    {
        int x = Integer.parseInt(k[0]);
        int y = Integer.parseInt(k[1]);
        Sum so = new Sum();
        so.assign(x, y);
        so.add();
        so.disp();
    }
}

NOTE: The data which we pass through command prompt is called command line arguments.

Write a JAVA program to check weather the given number is prime or not?
Answer:
class Prime
{
    int n;
    void set(int x)
    {
        n = x;
    }
    String decide()
    {
        int i;
        for (i=2; i<n; i++)
        {
            if (n%i==0)
            {
                break;
            }
        }
        if (i==n)
        {
            return "PRIME";
        }
        else
        {
            return "NOT"+"PRIME";
        }
    }
}
class PrimeDemo
```java
public static void main (String k [])
{
    int n=Integer.parseInt (k [0]);
    Prime po=new Prime ();
    po.set (n);
    String so=po.decide ();
    System.out.println (so);
}

Write a JAVA program which converts an ordinary number into roman number?

Answer:
class Roman
{
    int n;
    void set (int x)
    {
        n=x;
    }
    void convert ()
    {
        if (n<=0)
        {
            System.out.print ("NO ROMAN FOR THE GIVEN NUMBER");
        }
        else
        {
            while (n>=1000)
            {
                System.out.print ("M");
                n=n-1000;
            }
            if (n>=900)
            {
                System.out.print ("CM");
                n=n-900;
            }
            if (n>=500)
            {
                System.out.print ("D");
                n=n-500;
            }
            if (n>=400)
            {
                System.out.print ("CD");
                n=n-400;
            }
            while (n>=100)
            {
                System.out.print ("C");
                n=n-100;
            }
            if (n>=90)
            {
            
```
System.out.print("XC");
n=n-90;
}
if (n>=50)
{
    System.out.print("L");
n=n-50;
}
if (n>=40)
{
    System.out.print("XL");
n=n-40;
} while (n>=10)
{
    System.out.print("X");
n=n-10;
}
if (n>=9)
{
    System.out.print("IX");
n=n-9;
}
if (n>=5)
{
    System.out.print("V");
n=n-5;
}
if (n>=4)
{
    System.out.print("IV");
n=n-4;
} while (n>=1)
{
    System.out.print("I");
n=n-1;
}
System.out.println();
}
}
class RomanDemo
{
    public static void main (String k [])
    {
        int n=Integer.parseInt (k [0]);
        Roman r=new Roman ();
        r.set (n);
        r.convert ();
    }
}
Day - 18:

CONSTRUCTORS in java

A constructor is a special member method which will be called by the JVM implicitly (automatically) for placing user/programmer defined values instead of placing default values. Constructors are meant for initializing the object.

ADVANTAGES of constructors:
1. A constructor eliminates placing the default values.
2. A constructor eliminates calling the normal method implicitly.

RULES/PROPERTIES/CHARACTERISTICS of a constructor:
1. Constructor name must be similar to name of the class.
2. Constructor should not return any value even void also (if we write the return type for the constructor then that constructor will be treated as ordinary method).
3. Constructors should not be static since constructors will be called each and every time whenever an object is creating.
4. Constructor should not be private provided an object of one class is created in another class (constructor can be private provided an object of one class created in the same class).
5. Constructors will not be inherited at all.
6. Constructors are called automatically whenever an object is creating.

TYPES of constructors:
Based on creating objects in JAVA we have two types of constructors. They are default/parameter less/no argument constructor and parameterized constructor.

• A default constructor is one which will not take any parameters.

Syntax:
class <clsname>
{
   <clsname>() //default constructor
   {
      Block of statements;
      ......................;
      ......................;
   }
   ......................;
   ......................;
};

For example:
class Test
{
   int a, b;
   Test() {
      System.out.println("I AM FROM DEFAULT CONSTRUCTOR...");
      a=10;
      b=20;
   }
}
System.out.println("VALUE OF a = "+a);
System.out.println("VALUE OF b = "+b);
}
};
class TestDemo
{
    public static void main (String [] args)
    {
        Test t1=new Test();
    }
};

RULE-1:
Whenever we create an object only with default constructor, defining the default constructor is optional. If we are not defining default constructor of a class, then JVM will call automatically system defined default constructor (SDDC). If we define, JVM will call user/programmer defined default constructor (UDDC).

Day - 19:

• A parameterized constructor is one which takes some parameters.

Syntax:
class <classname>
{
    ................;
    ................;
    <classname> (list of parameters) //parameterized constructor
    {
        Block of statements (s);
    }
    ................;
    ................;
}

For example:
class Test
{
    int a, b;
    Test (int n1, int n2)
    {
        System.out.println("I AM FROM PARAMETER CONSTRUCTOR...");
        a=n1;
        b=n2;
        System.out.println("VALUE OF a = "+a);
        System.out.println("VALUE OF b = "+b);
    }
};
class TestDemo1
{
    public static void main (String k [])
    {

RULE-2:
Whenever we create an object using parameterized constructor, it is mandatory for the JAVA programmer to define parameterized constructor otherwise we will get compile time error.

- Overloaded constructor is one in which constructor name is similar but its signature is different. Signature represents number of parameters, type of parameters and order of parameters. Here, at least one thing must be differentiated.

For example:
Test t1=new Test (10, 20);
Test t2=new Test (10, 20, 30);
Test t3=new Test (10.5, 20.5);
Test t4=new Test (10, 20.5);
Test t5=new Test (10.5, 20);

RULE-3:
Whenever we define/create the objects with respect to both parameterized constructor and default constructor, it is mandatory for the JAVA programmer to define both the constructors.

NOTE:
When we define a class, that class can contain two categories of constructors they are single default constructor and ‘n’ number of parameterized constructors (overloaded constructors).

Write a JAVA program which illustrates the concept of default constructor, parameterized constructor and overloaded constructor?
Answer:
class Test
{
    int a, b;
    Test ()
    {
        System.out.println ("I AM FROM DEFAULT CONSTRUCTOR...");
        a=1;
        b=2;
        System.out.println ("VALUE OF a ="+a);
        System.out.println ("VALUE OF b ="+b);
    }
    Test (int x, int y)
    {
        System.out.println ("I AM FROM DOUBLE PARAMETERIZED CONSTRUCTOR...");
        a=x;
        b=y;
        System.out.println ("VALUE OF a ="+a);
        System.out.println ("VALUE OF b ="+b);
    }
    Test (int x)
}
J2SE (Core JAVA) Notes

{ 
    System.out.println ("I AM FROM SINGLE PARAMETERIZED CONSTRUCTOR..."); 
    a=x; 
    b=x; 
    System.out.println ("VALUE OF a ="+a); 
    System.out.println ("VALUE OF b ="+b); 
} 
Test (Test T) 
{ 
    System.out.println ("I AM FROM OBJECT PARAMETERIZED CONSTRUCTOR..."); 
    a=T.a; 
    b=T.b; 
    System.out.println ("VALUE OF a ="+a); 
    System.out.println ("VALUE OF b ="+b); 
} 
}; 
class TestDemo2 
{ 
    public static void main (String k []) 
    { 
        Test t1=new Test (); 
        Test t2=new Test (10, 20); 
        Test t3=new Test (1000); 
        Test t4=new Test (t1); 
    } 
};

NOTE: By default the parameter passing mechanism is call by reference.

‘this’: ‘this’ is an internal or implicit object created by JAVA for two purposes. They are 
  i. ‘this’ object is internally pointing to current class object. 
  i. Whenever the formal parameters and data members of the class are similar, to differentiate 
     the data members of the class from formal parameters, the data members of class must be 
     proceeded by ‘this’.

Day - 20:

this (): this () is used for calling current class default constructor from current class parameterized 
    constructors.

this (...): this (...) is used for calling current class parameterized constructor from other category 
    constructors of the same class.

For example:
class Test 
{ 
    int a, b; 
    Test () 
    { 
        this (10); //calling current class single parameterized constructor
System.out.println("I AM FROM DEFAULT CONSTRUCTOR...");
a=1;
b=2;
System.out.println("VALUE OF a =" +a);
System.out.println("VALUE OF b =" +b);
}
Test (int x) ---2
{
3-→
this (100, 200); // calling current class double parameterized constructor
System.out.println("I AM FROM SINGLE PARAMETERIZED CONSTRUCTOR...");
a=b=x;
System.out.println("VALUE OF a =" +a);
System.out.println("VALUE OF b =" +b);
}
Test (int a, int b) ---3
{
System.out.println("I AM FROM DEFAULT CONSTRUCTOR...");
this.a=a;
this.b=b;
System.out.println("VALUE OF a =" +this.a);
System.out.println("VALUE OF b =" +this.b);
System.out.println("VALUE OF a =" +a);
System.out.println("VALUE OF b =" +b);
}
};
class TestDemo3
{
    public static void main (String k [])
    {
1-→
    Test t1=new Test();
    }
};

Rule for ‘this’:
Whenever we use either this () or this (...) in the current class constructors, that statements must be used as first statement only.
The order of the output containing this () or this (...) will be in the reverse order of the input which we gave as inputs.
For more clarity refer the above program.
For example we need output as follows:
For example we need output as follows:
I AM FROM DEFAULT CONSTRUCTOR... ---1
VALUE OF a = 1
VALUE OF b = 2
I AM FROM SINGLE PARAMETERIZED CONSTRUCTOR... ---2
VALUE OF a = 100
VALUE OF b = 200
I AM FROM DOUBLE PARAMETERIZED CONSTRUCTOR... ---3
VALUE OF a = 10
VALUE OF b = 10

We must write in the following order as input:
Test (10); ---3
Test (100, 200); ---2
NOTE:
Whenever we refer the data members which are similar to formal parameters, the JVM gives first preference to formal parameters whereas whenever we write a keyword this before the variable name of a class then the JVM refers to data members of the class.

this methods are used for calling current class constructors.

NOTE:
• If any method called by an object then that object is known as source object.
• If we pass an object as a parameter to the method then that object is known as target object.

For example:
SOURCE OBJECT. METHOD NAME (TARGET OBJECT);

t1. display (t2); // written in main

In the definition of display method t1 data members are referred by this. Data member names (this. a & this. b) whereas t2 object data members are referred by formal object name. Data member names (T. a & T. b).

void display (Test T) //T is formal object member
{
    System.out.println ("VALUE OF a BELONGS TO DATA MEMBER ="+this.a);
    System.out.println ("VALUE OF b BELONGS TO DATA MEMBER ="+this.a);
    System.out.println ("VALUE OF a BELONGS TO FORMAL OBJECT MEMBER ="+T.a);
    System.out.println ("VALUE OF b BELONGS TO FORMAL OBJECT MEMBER ="+T.a);
}

Day - 21:

Write a JAVA program which computes sum of two objects by accepting the data from command prompt?
Answer:
class Test
{
    int a,b;
    Test ()
    {
        a=b=0;
    }
    Test (int a, int b)
    {
        this.a=a;
        this.b=b;
    }
    Test sum (Test T)
    {

Test T11=new Test ();
T11.a=this.a+T.a;
T11.b=this.b+T.b;
return (T11);
}

void display ()
{
    System.out.println ("VALUE OF a = "+a);
    System.out.println ("VALUE OF b = "+b);
}

};

class SumDemo1
{
    public static void main (String k[]) {
        int n1=Integer.parseInt (k[0]);
        int n2=Integer.parseInt (k[1]);
        int n3=Integer.parseInt (k[2]);
        int n4=Integer.parseInt (k[3]);
        Test t1=new Test (n1,n2);
        Test t2=new Test (n3,n4);
        Test t3=new Test ();
        // t3=t1+t2; invalid statement
        t3=t1.sum (t2);
        System.out.println ("t1 VALUES ARE AS FOLLOWS...");
        t1.display ();
        System.out.println ("t2 VALUES ARE AS FOLLOWS...");
        t2.display ();
        System.out.println ("t3 VALUES ARE AS FOLLOWS...");
        t3.display ();
    }
};

TYPES of RELATIONSHIPS in java

Based on reusing the data members from one class to another class in JAVA we have three types of relationships. They are is-a relationship, has-a relationship and uses-a relationship.

- **Is-a relationship** is one in which data members of one class is obtained into another class through the concept of inheritance.
- **Has-a relationship** is one in which an object of one class is created as a data member in another class.
- **Uses-a relationship** is one in which a method of one class is using an object of another class.

Inheritance is the technique which allows us to inherit the data members and methods from base class to derived class.

- **Base class** is one which always gives its features to derived classes.
- **Derived class** is one which always takes features from base class.

A Derived class is one which contains some of features of its own plus some of the data members from base class.
Syntax for **INHERITING** the features from **base class** to **derived class**:

```java
class <clsname-2> extends <clsname-1>
{
   Variable declaration;
   Method definition;
};
```

Here, `clsname-1` and `clsname-2` represents **derived class** and **base class** respectively.

**Extends** is a **keyword** which is **used for inheriting** the **data members** and **methods** from **base class** to the **derived class** and it **also improves functionality** of **derived class**.

**NOTE:**

- **Final classes cannot be inherited.**
- If the **base class** contains **private data members** then that type of **data members** will not be **inherited into derived class**.

Whenever we develop any inheritance application, it is always recommended to create an object of bottom most derived class. Since, bottom most derived class contains all the features from its super classes.

**Day - 22:**

- **One class can extend only one class** at a time. Since, JAVA does not support **multiple inheritance**.

Whenever we **inherit** the **base class members** into **derived class**, when we **creates** an **object of derived class**, JVM always **creates the memory space** for **base class members** first and **later memory space will be created** for **derived class members**.

**For example:**

```java
class c1;
{
   int a;
   void f1()
   {
      ........;
   }
};
class c2 extends c1
{
   int b;
   void f2()
   {
      ........;
   }
};
```
NOTE:
- Whatever the data members are coming from base class to the derived class, the base class members are logically declared in derived class, the base class methods are logically defined in derived class.
- Private data members and private methods of the base class will not be inherited at all.

Write a JAVA program computes sum of two numbers using inheritance?
Answer:
```java
class Bc {
    int a;
}
class Dc extends Bc {
    int b;
    void set (int x, int y) {
        a=x;
        b=y;
    }
    void sum () {
        System.out.println("SUM = "+(a+b));
    }
}
class InDemo {
    public static void main (String k []) {
        int n1=Integer.parseInt (k [0]);
        int n2=Integer.parseInt (k [1]);
        Dc do1=new Dc ();
        do1.set (n1, n2);
        do1.sum ();
    }
}
```

For every class in JAVA we have a super class called object class. The purpose of object class is that it provides garbage collector for collecting unreferenced memory locations from the derived classes.

‘Super’ keyword:
Super keyword is used for differentiating the base class features with derived class features. Super keyword is placing an important role in three places. They are at variable level, at method level and at constructor level.
- Super at variable level
  Whenever we inherit the base class members into derived class, there is a possibility that base class members are similar to derived class members.
  In order to distinguish the base class members with derived class members in the derived class, the base class members will be preceded by a keyword super.
Syntax for **super** at VARIABLE LEVEL:

super. base class member name

For example:
class Bc
{
    int a;
};
class Dc extends Bc
{
    int a;
    void set (int x, int y)
    {
        super.a=x;
        a=y; // by default 'a' is preceded with 'this.' since 'this.' represents current class
    }
    void sum ()
    {
        System.out.println ("SUM = "+(super.a+a));
    }
};
class InDemo1
{
    public static void main (String k [])
    {
        int n1=Integer.parseInt (k[0]);
        int n2=Integer.parseInt (k[1]);
        Dc dol=new Dc ();
        dol.set (n1, n2);
        dol.sum ();
    }
};

- **Super at method level**

Whenever we inherit the **base class methods** into the **derived class**, there is a **possibility** that **base class methods are similar to derived methods**.

To differentiate the **base class methods** with **derived class methods** in the **derived class**, the **base class methods must be preceded by a keyword super**.

Syntax for **super** at method level: super. base class method name

For example:
class Bc
{
    void display ()
    {
        System.out.println ("BASE CLASS - DISPLAY...");
    }
};
class Dc extends Bc
{
    void display ()
    {
super.display (); //refers to base class display method
System.out.println ("DERIVED CLASS - DISPLAY...");
}
}
class InDemo2
{
    public static void main (String k [])
    {
        Dc dol=new Dc ();
        dol.display ();
    }
}

**Day-23:**

- **Super at constructor level**
  
  Whenever we develop any inheritance application, we use to create always object of bottom most derived class. When we create an object of bottom most derived class, it in turns calls its immediate super class default constructor and it in turns calls its top most super class default constructor. Therefore, in JAVA environment, constructors will be called always from bottom to top and the execution starts from top to bottom.

  Consider the following multi level inheritance:

  ![Multi Level Inheritance Diagram](image)

  *Super()* is used for calling super class default constructor from default constructor or from parameterized constructor of derived class. It is optional.

  *Super (...)* is used for calling super class parameterized constructor either from default constructor or from parameterized constructor of derived class. It is always mandatory.

**RULES:**

Whenever we use either super () or super (...) in derived class constructors they must be used as first statement.
1. Whenever we want to call default constructor of base class from default constructor of derived class using super () in default constructor of derived class is optional.

For example:
```java
class Bc {
    Bc () {
        System.out.println ("I AM FROM BASE CLASS...");
    }
}
class Ibc extends Bc {
    Ibc () {
        System.out.println ("I AM FROM INTERMEDIATE BASE CLASS...");
    }
}
class Dc extends Ibc {
    Dc () {
        super (); //optional
        System.out.println ("I AM FROM DERIVED CLASS...");
    }
}
class InDemo3 {
    public static void main (String k []) {
        Dc o1=new Dc ();
    }
}
```

2. Whenever we want to call the super class parameterized class from parameterized class of the derived class using super (...) in parameterized class of derived class is mandatory.
For example:

```java
class C1
{
    int a;
    C1 (int a)
    {
        System.out.println ("PARAMETERIZED CONSTRUCTOR - C1");
        this.a=a;
        System.out.println ("VALUE OF a = " +a);
    }
};
class C2 extends C1
{
    int b;
    C2 (int a, int b)
    {
        super (a);
        System.out.println ("PARAMETERIZED CONSTRUCTOR - C2");
        this.b=b;
        System.out.println ("VALUE OF b = " +b);
    }
};
class InDemo4
{
    public static void main (String k [])
    {
        C2 o2=new C2 (10, 20);
    }
};
```

3. Whenever we want to call default constructor of base class from parameterized class of derived class using `super()` in parameterized class of derived class is optional.

For example:

```java
class C1
{
    int a;
    C1 ()
    {
        System.out.println ("PARAMETERIZED CONSTRUCTOR - C1");
        this.a=a;
        System.out.println ("VALUE OF a = " +a);
    }
};
class C2 extends C1
{
    int b;
    C2 (int b)
    {
        super (); //optional
        System.out.println ("PARAMETERIZED CONSTRUCTOR - C2");
        this.b=b;
        System.out.println ("VALUE OF b = " +b);
    }
};
```
class InDemo5
{
    public static void main (String k [])
    {
        C2 o2=new C2 (20);
    }
};

4. Whenever we want to call parameterized class of base class from default constructor of derived class using super (...) in default constructor of derived class is mandatory.

For example:
class C1
{
    int a;
    C1 (int a)
    {
        System.out.println ("PARAMETERIZED CONSTRUCTOR - C1");
        this.a=a;
        System.out.println ("VALUE OF a = "+a);
    }
};
class C2 extends C1
{
    int b;
    C2 ()
    {
        super (10);
        System.out.println ("DEFAULT CONSTRUCTOR - C2");
        this.b=20;
        System.out.println ("VALUE OF b = "+b);
    }
};
class InDemo6
{
    public static void main (String k [])
    {
        C2 o2=new C2 ();
    }
};

Day - 24:

Best example for the above given rules:
class Bc
{
    Bc ()
    {
        System.out.println ("BASE CLASS - DEFAULT CONSTRUCTOR");
    }
    Bc (int x)
    {
        this ();
    }
System.out.println("BASE CLASS - PARAMETERIZED CONSTRUCTOR");
);
class Ibc extends Bc {
    Ibc () {
        super (10);
        System.out.println("INTERMEDIATE BASE CLASS - DEFAULT CONSTRUCTOR");
    }
    Ibc (int x) {
        this ();
        System.out.println("INTERMEDIATE BASE CLASS - PARAMETERIZED CONSTRUCTOR");
    }
};
class Dc extends Ibc {
    Dc () {
        this (10);
        System.out.println("DERIVED CLASS - DEFAULT CONSTRUCTOR");
    }
    Dc (int x) {
        super (10);
        System.out.println("DERIVED CLASS - PARAMETERIZED CONSTRUCTOR");
    }
};
class StDemo {
    public static void main (String k []) {
        Dc do1=new Dc ();
    }
}

ABSTRACT CLASSES:

In JAVA we have two types of classes. They are concrete classes and abstract classes.

- A concrete class is one which contains fully defined methods. Defined methods are also known as implemented or concrete methods. With respect to concrete class, we can create an object of that class directly.

For example:
class C1 {
    int a,b;
    void f1 () {
        ...........
        ...........
    }
    void f2 () {

To call the above method:
C1 O1=new C1 ();
O1.f1 ();
O1.f2 ();

- An **abstract class** is one which **contains some defined methods and some undefined methods**. **Undefined methods** are also known as **unimplemented or abstract methods**. **Abstract method** is one which **does not contain any definition**. To make the method as **abstract** we have to use a keyword called **abstract before the function declaration**.

**Syntax for ABSTRACT CLASS:**
abstract return_type method_name (method parameters if any);

**For example:**
Abstract void sum ();

The abstract methods make us to understand what a method can do but it does not give how the method can be implemented. Every abstract method belongs to a class under class is known as abstract class, to make the class as abstract we use a keyword called abstract before the class specification.

**Syntax for ABSTRACT CLASS:**
abstract class <clsname>  
{     Abstract return_type method_name (method parameters if any);  
}  

**For example:**
abstract class Op  
{     abstract void sum ();  
}  

**Day - 25:**

With respect to **abstract class** we cannot create an object direct but we can create indirectly. An object abstract class is equal to an object of that class which extends that abstract class.

**For example:**
class CC extends AC  
{     ..........;
AC Ao=new CC (); //invalid
AC Ao=new CC ();
or
AC Ao;
Ao=new CC ();

Write a JAVA program for computing sum of two integers and floats using abstract classes?

Answer:
abstract class Op
{
    abstract void sum ();
};
class isum extends Op
{
    void sum ()
    {
        int a,b,c;
        a=10;
        b=20;
        c=a+b;
        System.out.println ("INT VALUE = "+c);
    }
};
class fsum extends Op
{
    void sum ()
    {
        float f1,f2,f3;
        f1=10.26f;
        f2=20.32f;
        f3=f1+f2;
        System.out.println ("FLOAT VALUE = "+f3);
    }
};
class AbDemo
{
    public static void main (String k [])
    {
        // Op o1=new Op (); invalid
        Op o2;
o2=new isum ();
o2.sum ();
o2=new fsum ();
o2.sum ();
    }
};

Abstract classes should not be final, since, they are always reusable. Abstract classes are basically used to implement polymorphism; we use the concept of dynamic binding. Hence, abstract classes, polymorphism and dynamic binding will improve the performance of JAVA J2EE applications by reducing amount of memory space.
Whenever we inherit ‘n’ number of abstract methods from abstract base class to derived class, if the derived class defines all ‘n’ number of abstract methods then the derived class is concrete class. If the derived class is not defining at least one abstract method out of ‘n’ abstract methods then the derived class is known as abstract derived class and to make that class abstract, we use a keyword called abstract.

An abstract base class is one which contains physical representation of abstract methods. An abstract derived class is one which contains logical declaration of abstract methods which are inherited from abstract base class.

**Day - 26:**

Implement the above diagram by using abstract class’s polymorphism and dynamic binding.

**Answer:**

```java
abstract class C1
{
    abstract void f1 ();
    abstract void f2 ();
};
abstract class C2 extends C1
{
    void f1 ()
    {
        System.out.println("f1-C2-original");
    }
};
class C3 extends C2
{
    void f1 ()
    {
        super.f1 ();
        System.out.println("f1-C3-OVERRIDDEN");
    }
    void f2 ()
```
class AbDemo1
{
    public static void main (String k [])
    {
        C3 o3=new C3 ();
        o3.f1 ();
        o3.f2 ();
        o3.f3 ();
        // C2 o2=new C2 (); invalid
        C2 o2=new C3 ();
        o2.f1 ();
        o2.f2 ();
        // o2.f3 (); invalid
        C1 o1=new C3 (); // or o2
        o1.f1 ();
        o1.f2 ();
        // o1.f3 ();
    }
}

{ System.out.println ("f2-C3"); }
void f3 ()
{ System.out.println ("f3-C3-SPECIALLY DEFINED"); }

Output:
With respect to Concrete Class:
C3 o3=new C3 ();
o3.f1 (); // f1 - overridden - C3
o3.f2 (); // f2 - C3
o3.f3 (); // f3 - defined in - C3

With respect to Abstract Derived Class:
C2 o2=new C2 (); // invalid
C2 o2=new C3 ();
o2.f1 (); // f1 - overridden - C3
o2.f2 (); // f2 - C3
o2.f3 (); // invalid

With respect to Abstract Base Class:
C1 o1;
o1=o2; // it mean., new C3 ()
o1.f1 (); // f1 - overridden - C3
o1.f2 (); // f2 - C3
o1.f3 (); // invalid

An object of either concrete base class or abstract base class contains the details about those methods which are available in that class only but this object (concrete base class or abstract base class) does not contains details of those methods which are specially defined in derived class's.
Application:
Write a JAVA program to display the fonts of the system?

Answer:
import java.awt.GraphicsEnvironment;
class Fonts
{
    public static void main (String k [])
    {
        GraphicsEnvironment
        ge=GraphicsEnvironment.getLocalGraphicsEnvironment ();
        String s []=ge.getAvailableFontFamilyNames ();
        System.out.println ("NUMBER OF FONTS = "+s.length);
        for (int i=0; i<s.length; i++)
        {
            System.out.println (s [i]);
        }
    }
};

A factory method is one whose return type is similar to name of the class where it presents.
For example:
getLocalGraphicsEnvironment ();

RULES for factory method:
1. The return type of the factory method must be similar to name of the class where it presents.
2. Every factory method must be static (so that we can call with respect to name of the class).
3. Every factory method must be public.

Factory methods are used for creating an object without using new operator. Every predefined abstract class contains at least one factory method for creating an object of abstract class.

Whenever we define a concrete class, that concrete class also can be made it as abstract and it is always further reusable or extendable by further classes.

When we define only concrete class we may extend or we may not extend by derived classes.