**CORE JAVA NOTES BY HARSHAL CHAUDHARI**

**JavaSyllabus- Core Java Programming**

* Installing Java
* Installing Eclipse
* Variables
* Data Types
* Operators
* Conditional Statements-
* Loops
* Concept of class file
* What are methods?
  + Method Input Parameters
  + Method Return Type
  + Main Method
* Objects Creations in Java
  + Object and Object References
  + Call by reference and Value
* Constructors
  + Constructor Overloading
* Object-Oriented Programming in Java (OOPs)

§ Access Modifiers – Public/Private/Default/Protected

§ Importance of Final, Static, this, super keywords

* + Interface
    - Abstract class and concrete class
  + Inheritance
    - Usage of Inheritance in Selenium
    - Diamond ambiguity
  + Polymorphism
    - Overloading and Overriding Functions
  + Abstraction
    - Abstract class and concrete class
  + Interface
  + Encapsulation
  + Generalisation
* Casting in Java
* String class and functions/methods
* Arrays- for each loop
* Collection-SET, list, maps
* Exception Handling-throw, throws, try, catch, finally etc.

**Java Advantages**

1. **Beginner friendly**
2. **Simple & open source**
3. **Object oriented language**
4. **Secured language**
5. **Platform Independent**
6. **Most in demand**

ASCII values--->Unicode

128-->64k+

a-z, A-Z, 0-9, all symbols

1. **Deep community support-20lac java programer**

**Language Comparison**

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter** | **C** | **C++** | **Java** |
| **Security** | **Not secured** | **Secured** | **Highly Secured** |
| **Platform Independent** | **No** | **No** | **Yes** |
| **Language Design** | **Procedural oriented** | **Procedural oriented/obj oriented** | **Procedu**  **ral oriented/obj oriented** |
| **Example** | **OS, compilers,SQL,Database** | **Browsers-Chrome, Edge, Firefox, 3D games** | **Antivirus, Netflix, banking applns,** |

**History of Java**

* **Invented in 1995 - James Gosling- SunMicrosys- Oracle**
* **First name given to java language as Oak—> Java**
* **High level programming language→ Humans can read it.**
* **WORA- Write Once Run Anywhere**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Java Applications**

1. **Web application → websites→ Internet is required**

**Client- server**

1. **Standalone application-->Internet not required**

**Games, VLC, adobe reader, MS office**

1. **Enterprise Application**

**ERP, Banking application, CRM**

1. **Mobile Applications**
2. **All android & IOS applications**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Java Downloading**

**Java Versions**

**Starting Java 1.0 to 1.8, 9,......17**

**Java 1.8 == Java 8**

**Most popular versions of java are**

1. **Java 8 - free support**
2. **Java 11 - Chargeable support**

**Java Structure and Analogy with real life examples**

|  |  |  |
| --- | --- | --- |
| **Application- Eclipse** | **Country-India** | **university** |
| **Project** | **State- Maharashtra** | **Institute** |
| **Package** | **Dist- Nagar** | **Branch** |
| **Class** | **Tehsil-Akole** | **Class-4th year** |
| **Methods** | **AT-Akole Society** | **Div-A** |
| **Datamembers** | **Akash-Home** | **Akash-Roll No.05** |

**Methods/functions/tasks**

|  |  |  |
| --- | --- | --- |
|  | **Methods** | **Datamembers** |
|  | **Functions/tasks** | **Characteristics/Features** |
| **Person** | **Eat, sing, dance, sleep** | **Height, colour, weight, age** |
| **Online Payment** | **Payment ()**  **{**  **—-----Code of that method**  **}**  **//task-money transfer** | **Sender name, mobile no, Account Num, Balance** |
| **Mathematical Operation** | **Addition()**  **{**  **int z;**  **z=x+y; //z=10+20=30**  **Print z=30**  **}** | **int x=10**  **int y=20** |

**First Program in Java**

**package com.aspire.java.basics;**

**// I am trying to write my first java program**

**// start first program**

**/\* hi**

**hello**

**i am writing first program**

**\*/**

**public class Demo**

**{ //main method**

**public static void main(String[] args) //main(press Ctrl+spacebar)**

**{**

**System.out.println("Hello Team"); // syso(press ctrl+spacebar)**

**System.out.println("How are you");**

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

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

**System.out.println("Successfully executed my 1st program");**

**}**

**}**

**Output on console:**

**Hello Team**

**How are you**

**Fine**

**Close**

**Successfully executed my 1st program**

**Data Type**

**Data types are used to represent the type of data/information which we want to use in a java program.**

**Ex- number, name**

* **In java programming it is made mandatory to define/declare datatype before the declaration of variable**
* **Ex- int a=10;**

**int→datatype, a→variable**

**Types of DataTypes**

**A. Primitive DataTypes**

* **Memory size of Primitive data types is fixed**
* **All the primitive datatypes are keywords**
* **There are 8 types of primitive datatypes are available- byte, short, int, long, float, double, char, Boolean.**

**32GB=32\*1024MB**

**1GB=1024MB**

**1Mb=1024KB**

**1KB=1024 bytes=1024\*8= bits**

**1 byte= 8 bit**

**Bit→0 and 1**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** |
| * **Or -** | **0** | **0** | **0** | **0** | **0** | **0** | **1** |

**Storage Range= - (2)^(n-1) to (2)^(n-1) -1 n→ bits**

**For byte, n=8, Storage range= - 128 to 127 = total numbers=256**

1. **Integer Primitive DataTypes (Numeric+Non-decimal)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Data type** | **Size (in bytes)** | **Storage range** |
| **1** | **byte** | **1 byte** | **- 128 to 127** |
| **2** | **short** | **2 byte=2\*8=16 bit (n=16)** | **-32768 to 32767** |
| **3** | **int** | **4 byte (n=32)** | **-2,147,483,648 to 2,147,483,647** |
| **4** | **long (l)** | **8 byte** | **-9223372036854775808 to 9223372036854775807**  **And we need to add symbol ‘l’ after number** |

1. **Fractional DataTypes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr. No.** | **Data type** | **Size (in bytes)** | **Storage range before decimal** | **Storage range after decimal** |
| **5** | **float(f)** | **4 byte** | **-2,147,483,648 to 2,147,483,647**  **And we need to add symbol ‘f’ after number** | **6 to 7 decimals** |
| **6** | **double(d)** | **8 byte** | **-9223372036854775808 to 9223372036854775807**  **And we need to add symbol ‘d’ after number** | **15 decimal** |

1. **Character**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Data type** | **Size (in bytes)** | **Storage range** |
| **7** | **char** | **2 byte** | **-32768 to 32767** |

**UNICODE: (ASCII in older days)**

**ASCII values**

**A=65, B=66, C=67,.......**

**a=97, b=98, c=99,d=100,........**

**#=35**

**Char can store-**

**Eng. Alphabets (A-Z)= 26,a-z=26, numbers(0-9=10),Marathi=48;Telugu=56,**

**min=0, max=65536**

1. **Boolean Data Type**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Data type** | **Size (in bytes)** | **Storage range** |
| **8** | **Boolean** | **1 bit** | **true or false** |

**Boolean x=true;**

**Boolean y=false;**

**B. Non-Primitive DataTypes**

* **Memory size of Non-Primitive data types is not fixed/defined**
* **Non-primitive datatypes are not keywords**
* **Non-primitive data types are Identifiers**
* **Identifiers always starts with Capital letter**
* **Examples- String, Class, Arrays, Interface**

**Variables (Part-I)**

* **Variable are nothing but a piece of memory used to store some information**

**Ex- int i=10;**

* One variable can store one information at a single time
* Variable can be used for information reusability
* Steps to be followed to utilise variable in program-

1. Variable declaration
2. Variable Initialization
3. Variable Usage

**Rules for Variables-**

1. Uppercase(A-Z), lowercase(a-z), digits(0-9), Underscore (\_), dollar($)
2. First character must be letter
3. Blank spaces are not allowed
4. No java keywords can be used as variables
5. Variable are case sensitive

**Operators in Java**

**Operator** is nothing but a symbol which informs the compiler to perform specific mathematical operation or logical operations

**Types of Operators**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type of Operator** | **Operator symbol** | **Example**  **(int A=8, B=3, C=5)** | **Result** |
| **Arithmetic Operators** | **+** | **A+B** | **11** |
| **-** | **A-B** | **5** |
| **\*** | **A\*B** | **24** |
| **/** | **A/B** | **2** |
| **%**  **Remainder** | **A%B** | **2** |
| **Relational**  **Operators** | **<** | **A<B** | **false** |
| **<=** | **A<=B** | **false** |
| **>** | **A>B** | **true** |
| **>=** | **A>=B** | **true** |
| **==** | **A==B** | **false** |
| **!=**  **Not Equals** | **A!=B** | **true** |
| **Logical**  **Operators** | **&&**  **AND** | **(A<B) && (B>C)**   |  |  |  | | --- | --- | --- | | **T** | **T** | **T** | | **T** | **F** | **F** | | **F** | **T** | **F** | | **F** | **F** | **F** | | **False** |
| **||**  **OR** | **(A==B) || (B!=C)**   |  |  |  | | --- | --- | --- | | **T** | **T** | **T** | | **T** | **F** | **T** | | **F** | **T** | **T** | | **F** | **F** | **F** | | **True** |
| **!**  **NOT** | **! (A>B)** | **false** |
| **Assignment Operator** | **+=** | **A+=B → A=A+B** | **11** |
| **-=** | **A-=B → A=A-B** | **5** |
| **\*=** | **A\*=B → A=A\*B** | **24** |
| **/=** | **A/=B → A=A/B** | **2** |
| **%=** | **A%=5 → A=A%5** | **3** |
| **=a=b** | **Value of b will be assigned to a** |  |

**Conditional Statements/ Decision Making/ Selection**

|  |  |  |
| --- | --- | --- |
| **Type of statement** | **Syntax** | **Flowchart** |
| 1. **If** | **if(condition)**  **{**  **Statement(s)-if condition is true**  **}** |  |
| 1. **If\_else** | **if(condition)**  **{**  **Statement(s)-if condition is true**  **}**  **else**  **{**  **Statement(s)-if**  **condition is false**  **}** |  |
| 1. **If\_else If**   **Or**  **Else-if** | **if(condition 1)**  **{**  **Statement(s)-if condition 1 is true**  **}**  **else if (condition 2)**  **{**  **Statement(s)-if**  **condition is true**  **}**  **.**  **.**  **.**  **.**  **else**  **{**  **Statement(s)-if**  **condition is false**  **}** |  |
| 1. **Nested\_if** | **if(main condition)**  **{ Statement(s)- if main condition is true**  **if(sub condition)**  **{**  **Statement(s)- if condition is true**  **}**  **else**  **{**  **Statement(s)- if sub-condition is false**  **}**  **}**  **else**  **{**  **Statement(s)- if main condition is false**  **}** |  |
| 1. **Switch case** | **switch (key) {**  **case value1: break; case value2: break; case value3: break; default: break;**  **}** |  |

**Logical Loops**

**The logical loops in java are essential to execute a block of code multiple times.**

**Following loops are present in java-**

1. **For Loop**
2. **While Loop**
3. **Do-While Loop**
4. **For each Loop**

**All these types of loops execute a set of repeated statements as long as a given/specified condition remains true.**

**This particular condition is called a loop control.**

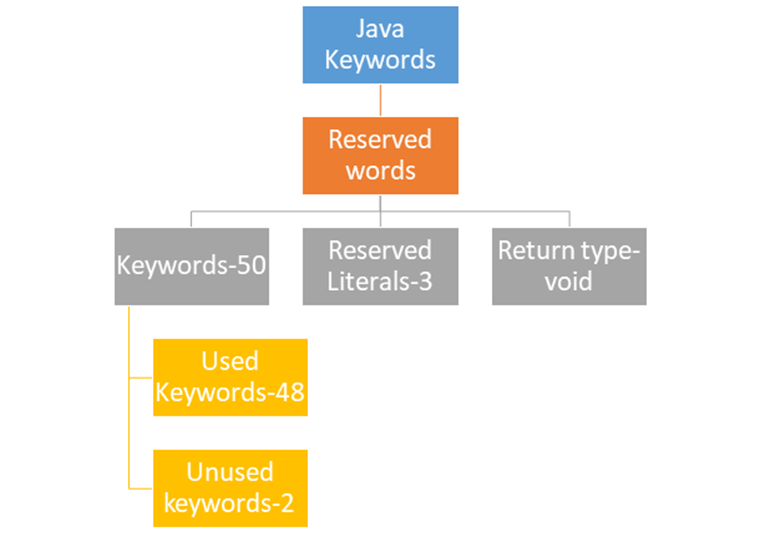
### **Q. Why do we need to use a loop ?**

**Whenever we want to execute a block of code several times then the looping concepts are preferred in Java language.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No** | **Loop** | **Syntax** | **Flow chart** |
| **1** | **For** | **//for( variable declarartion & initialization ; condition ; increment/decrement of value)**  **{**  **// body**  **}**  **//sequence of cursor movement**  **1. init**  **2. condition verify - 5,8**  **3. for loop block - 6,9**  **4. i/d (++/--) - 7,10**  **11. (in case of condition false) for loop exit** |  |
| **2** | **While** |  |  |
| **3** | **Do While** | **\*to repeat the statement block at least one time then use do-while loop** |  |
| **4** | **For each** | **Mostly used** | **Collection / Arrays**    **for(DataType localVariable:datasource)**  **{**  **}** |

**Java Keywords:**

**These are reserved words**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **Used Keywords-48** | |  | **Unused Keywords-2** |
| **data types-8** | **1) byte**  **2) short**  **3) int**  **4) long**  **5) float**  **6) double**  **7) char**  **8) boolean** |  | **1) goto- use is ban**  **2) const** |
| **flow control-11**  **(conditional statements & Loops)** | **1) if**  **2) else**  **3) switch**  **4) case**  **5) default**  **6) for**  **7) do**  **8) while**  **9) break**  **10) continue**  **11) return** |  | **Reserved Literals-3**  **(as a value)** |
| **1) true**  **2) false**  **3) null** |
| **Access modifiers (Access specifiers)-11** | **1) public**  **2) private**  **3) protected**  **4) static**  **5) final**  **6) abstract**  **7) synchronized**  **8) native**  **9) strictfp**  **10) transient**  **11) volatile** |  |  |
| **exception handling-6** | **1) try**  **2) catch**  **3) finally**  **4) throw**  **5) throws**  **6) assert** |  |  |
| **Class-6** | **1) class**  **2) package**  **3) import**  **4) extends**  **5) implements**  **6) interface** |  |  |
| **Object-4** | **1) new**  **2) instanceof**  **3) super**  **4) this** |  |  |

**Methods:**

* **Method is a block of code which only runs whenever it is called**
* **Methods are used to perform certain actions or tasks hence method are called as functions**
* **We can pass parameters as well as data into method**
* **Methods provide advantage of reusability of code- Write code once and use that same code repeatedly/many times.**

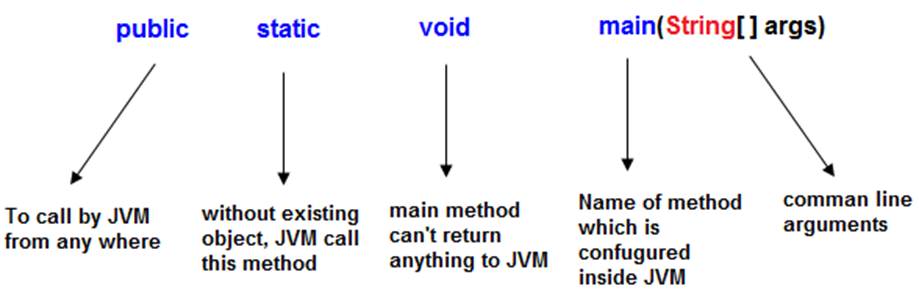
**Types of Method:**

**Main Method**

* **Main () method is the starting point from for any Java program, from**

**where the compiler starts the program execution.**

* **Hence, the compiler is required to call the main () method.**
* **Main method is required to run any java program**
* **Without main method we can't run any java program.**

****

## **Public**

* **public is a keyword in a java language**
* **whenever the public keyword is preceded by the main() method then the scope is available anywhere in the java environment.**
* **It means the main() method can be executed from anywhere in the java environment.**
* **main() method must be accessed by every java programmer and hence whose access specifier must be public**

## **Static**

* **static is a keyword in java**
* **Whenever static is preceded by any class properties then for that memory is allocated only once in the program.**
* **Static method is executed only once in the program.**
* **main() method of java executes only once throughout the java program execution and hence it declares must be static.**

## **Void**

* **void is a special datatype also known as no return type, whenever it is preceded by main() method that will be never return any value to the operating system.**
* **main() method of java is not returning any value and hence its return type must be void.**

## **String args[ ]**

**String args[ ] is a String array used to hold command line arguments in the form of String values.**

### **In case of main() method following changes are acceptable**

**1. We can declare String[] in any valid form.**

**· String[] args**

**· String args[]**

**· String []args**

**2. We can change the order of modifiers i.e Instead of**

**public static we can take static public**

**3. Instead of args we can take any valid java identifier.**

**public static void main(String a[])**

**Regular Method**

1. **Static Regular Method**
2. **Static method call from same class**

Method call: methodName();

1. **Static method call from different class**

Method call: ClassName.MethodName();

1. **Static method call from different package**

Method call:

IMPORT PACKAGE(// import PackageName.className);

ClassName.methodName();

1. **Non-static Regular Method/Instance Method**
2. **Non-Static method call from same class**

**Method call:**

**//create an object for the same class**

**//className objectName = new className();**

**objectName.methodName();**

1. **Non-Static method call from different class**

**Method call:**

**//create an object for Different Class**

**//className objectName = new className();**

**objectName.methodName();**

1. **Non-Static method call from different package**

**Method call:**

IMPORT PACKAGE(// import PackageName.className);

**//create an object for that class from Diff package**

**//className objectName = new className();**

**objectName.methodName();**

**Method without parameter(Zero parameter)-**All methods studied earlier belongs to this category

**Method with Parameters**

package com.aspire.java.Methods;

public class Parameterized\_Method

{

public static void main(String[] args)

{

subtraction(50,20);

subtraction(100,50);

subtraction(5000, 3000);

}

public static void subtraction(int a, int b)

{

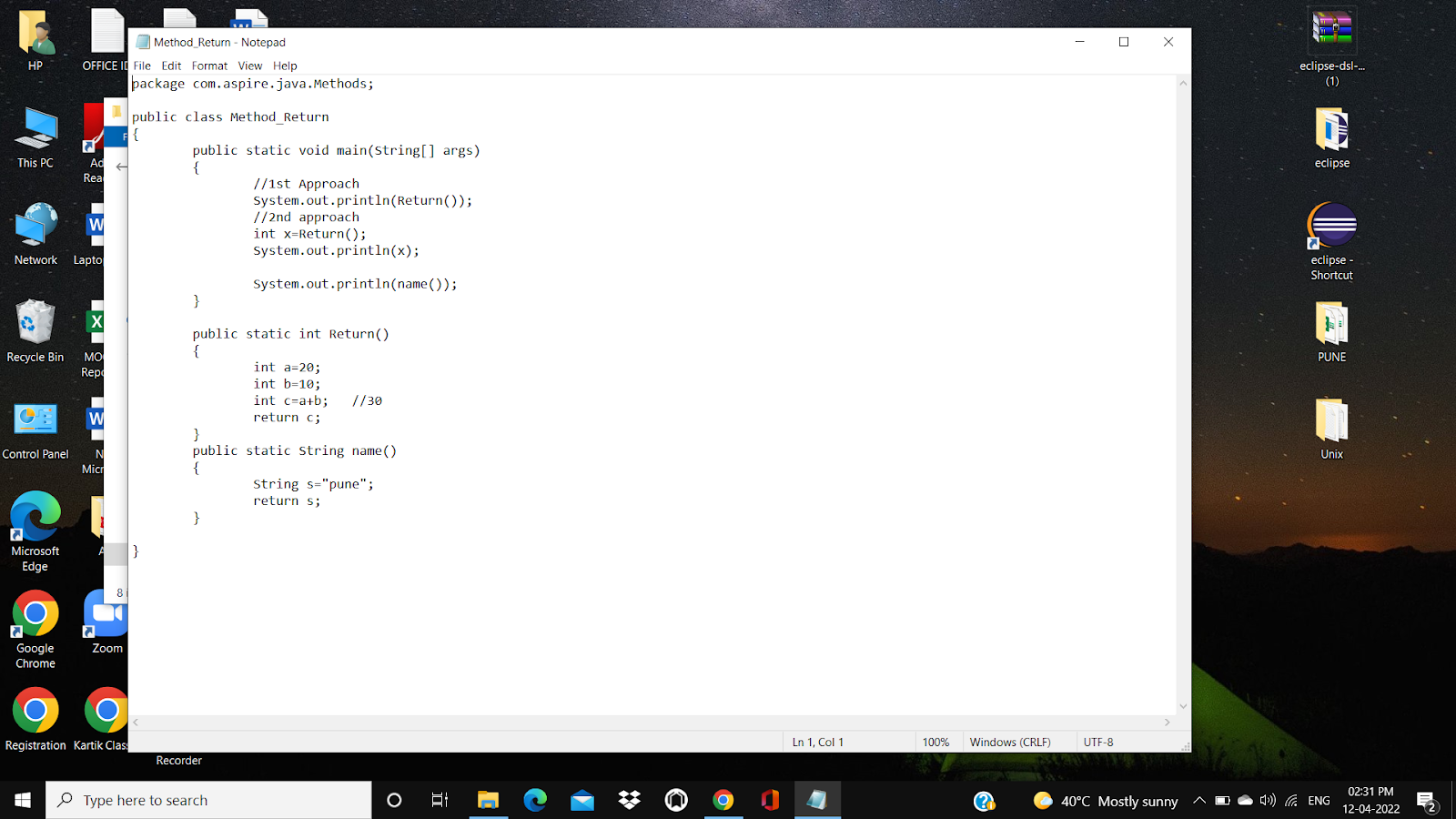
int c=a-b;

System.out.println(c);

}

}

**Method with Return Type**

****

**Constructor**

**●** A constructor in Java is a special member method which is used to initialise objects/instance variables/non-static members.

● The constructor is called when an object of a class is created.

Points to follow during constructor declaration:

● Constructor name must be same as class name

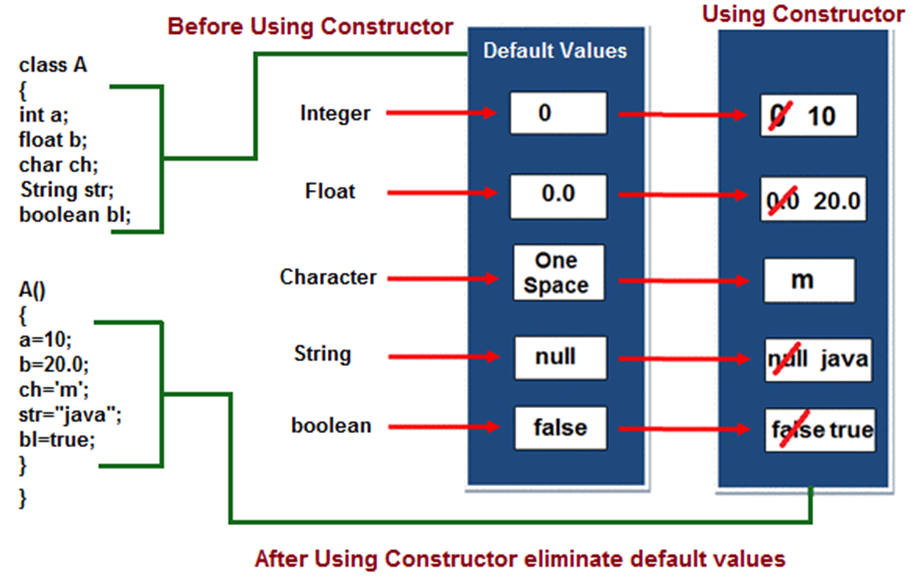
● Do not declare any return type for the constructor(like void,int, etc).

● Many number of constructor can be declared in a java class but constructor name should be the same as class name, but arguments/parameters should be different.

**Use of Constructor**

● To initialise instance data member/variable

● To copy/load non-static members of class into object --> when we create object of class



**Types of Constructor**

1. Default Constructor- we don't write, it is provided by compiler
2. User defined Constructor--we actually write in program

**1. Default Constructor-only one conditions apply**

● If Constructor is not declared in java class then at the time of compilation, compiler will provide Constructor for the class

● If the programmer has declared the constructor in the class then the compiler will not provide the default Constructor.

● The Constructor provided by compiler at the time of compilation is known as Default Constructor

**2. User defined Constructor-one or more in numbers**

● If a programmer is declaring a constructor in a Java class then it is considered to be a User defined constructor.

User defined Constructor are classified into 2 types

1. Without/zero parameter constructor

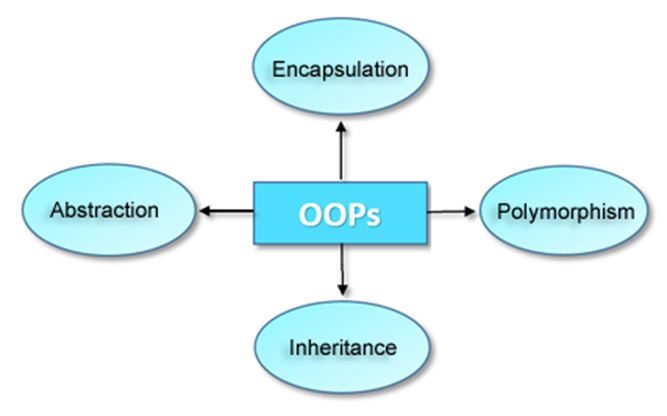
2. With parameter constructor

|  |  |  |
| --- | --- | --- |
|  | **Method** | **Constructor** |
| 1 | Method name can be any **user defined name** | Constructor name **must be same as class name** |
| 2 | Method should **have return type** | It should **not have any return type** (even void) |
| 3 | Method should be called explicitly either with object reference or class reference | It will be **called automatically** whenever object is created |
| 4 | Method is **not provided by the compiler** in any case. | The java **compiler provides a default constructor** if we do not have any constructor. |

**OOPs**

**Object Oriented Programing**

**OOPs is the methodology or structure with which we can design programs by using classes and objects.**

****

**1. Inheritance-The process of obtaining/acquiring features ( the data members and methods) from one class (Parent class/base class/main class) to another class (child class/derived/sub class) is known as inheritance.**

**2. Abstraction-Abstraction is the concept of exposing only the required essential characteristics and behaviour with respect to a context.**

**3. Encapsulation-Encapsulation is a process of wrapping data and methods in a single unit is called encapsulation.**

**4. Polymorphism-The process of representing one Form in multiple forms is known as Polymorphism**

**Features or Benefits of OOPs:**

* Suitable for large projects
* Better memory management
* It implements real time scenarios
* It is easy and simple to maintain and modify existing code
* Fairly efficient language
* Implementation details are hidden from other modules

**1. Inheritance**

* It is one of the Oops principles where one class acquires properties of another class with the help of 'extends' keywords is called Inheritance.
* The class from which properties are acquiring/inheriting is called super class/Parent class/base class/main class
* The class to where properties are inherited/delivered is called sub class/Child class/advance class/derived class
* Inheritance takes place between 2 or more than 2 classes.
* Inheritance is classified into 4 types:

**1. Single level Inheritance**

**2. Multilevel Inheritance**

**3. Multiple Inheritance**

**4. Hierarchical Inheritance**

**5. Hybrid**

**1. Single level Inheritance:**

* It is an operation where inheritance takes place between 2 classes.
* To perform single level inheritance only 2 classes are mandatory.

**2. Multilevel Inheritance:**

* Multilevel Inheritance takes place between 3 or more than 3 classes.

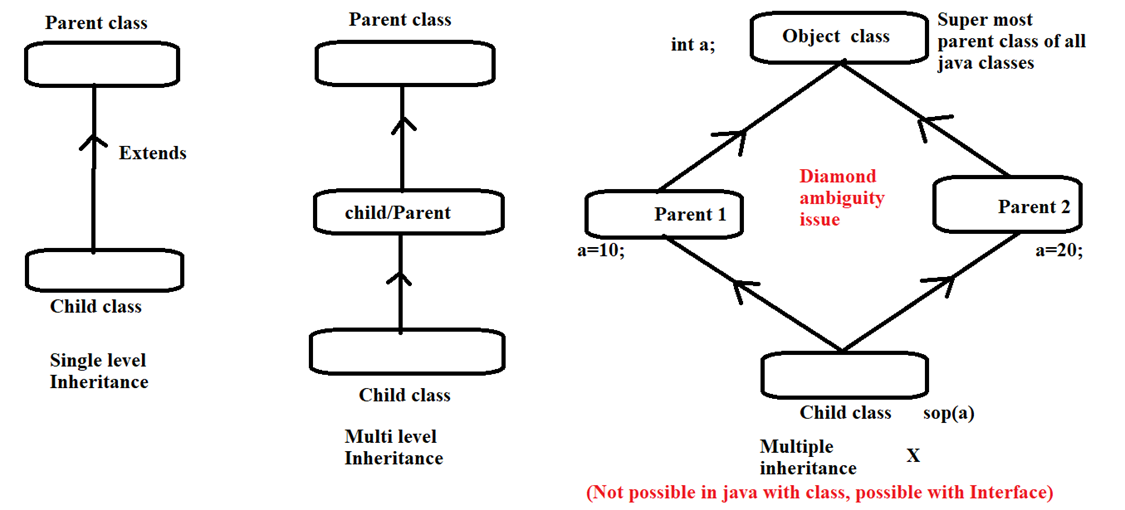
* In Multilevel Inheritance 1 sub class acquires properties of another super class & that class acquires properties of its another super class & phenomenon continuous.

**3. Multiple Inheritance:**

* One sub class acquiring properties of 2 super classes at the same time is known as Multiple Inheritance.
* Java doesn't support Multiple Inheritance using class because of the diamond ambiguity problem.

Note: **object class** is the super most class in java

**By using interfaces we can achieve Multiple Inheritance.**

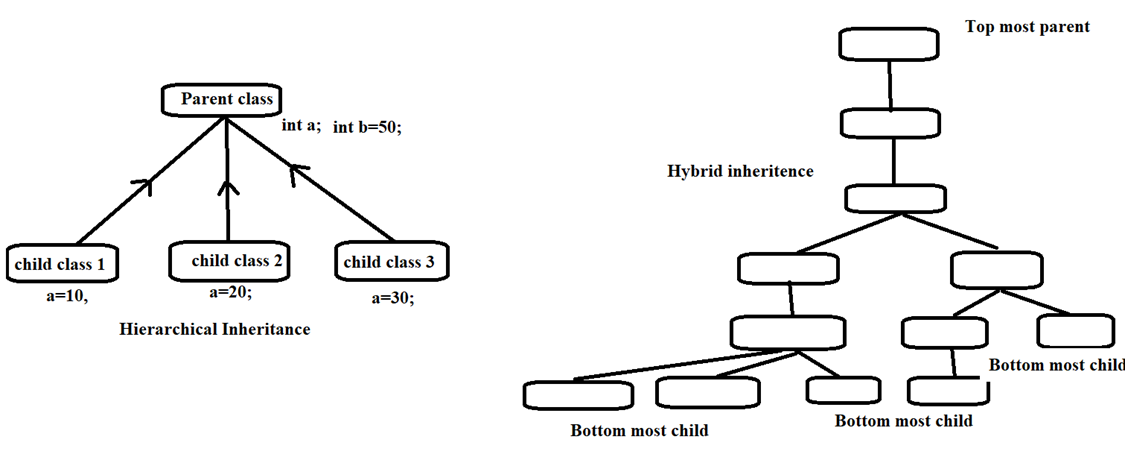
****

**4. Hierarchical Inheritance:**

* Multiple sub classes can acquire properties of 1 super class is known as hierarchical Inheritance.

**5. Hybrid**

Combination of above types



**Variables- Part II:**

**Types of Variables-**

1. **Local Variable**
2. **Global Variable**
3. **Static Variable**
4. **Instance Variable/ Non-static Variable**

**1. local variable**

* Creating variable inside method/block is known as local variable.
* Scope of local variable remains only within the method & they are temporary.

**2. global variable**

* Creating variable outside method/block is known as global variable.
* Scope of global variable remains throughout the class & they are permanent.

**3. class/static variable**

* Declaring the variable using a **static** keyword is known as class/static variable because to access a static variable class name is used.
* To access static variable from diff class we need to make use of below statement:
* Syntax:- classname.variablename;

**4. Instance/non-static variable**

* To access non-static variable we need to make use of below statement:

-Create object of class.

Syntax:- objectname.variablename;

|  |  |  |
| --- | --- | --- |
|  | Static data member-Global | Non static data member-Global |
| Static block/method | Yes | No |
| Non static Block/method | Yes | Yes |

**this and super keywords**

**this—> it is a keyword used to access global variable from same class**

**super—--> it is a keyword used to access global variable from super class/different class**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Static** | **Instance** | **Static in case of inheritance** | **Instance in case of inheritance** |
| **Global variable of same class** | directly | Create object & call by Object reference | this keyword | this keyword |
| **Global variable of different class** | Class reference | Create object of that class& call by Object reference | super keyword | super keyword |
| **Global variable of different package** | Import package and Class reference | Import package & call by Object reference |  |  |

**Access specifiers/modifier:**

**Access specifiers are used to represent scope of members of the class.**

**Access specifiers in Java are classified into 4 types as below-**

**1. private**

**2. default**

**3. protected**

**4. public**

**1. private:**

* If you declare any member of class as private then scope of that member remains only within the class
* It can't be accessed from other classes.

**2. default:**

* If you declare any member of class as default then scope of that member remains only within the package
* It can't be accessed from other packages.
* There is no keyword to represent the default access specifier.

**3. protected:**

* If you declare any member of class as protected then scope of that member remains only within the package that class which is present outside the package can access it by one condition ie. inheritance operation (child class ref)

**4. public:**

* If you declare any member of class as public then scope of that member remains throughout the project.

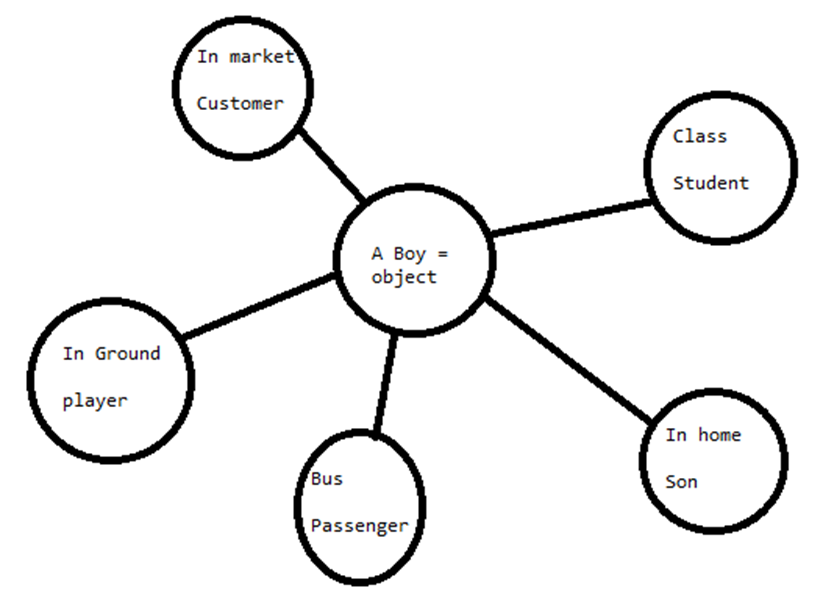
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **In same package** | | | | **In diff package** | | |
| **In same class** | **In diff class** | **In son class** | | **In diff class** | **In son class** | |
| **With parent class object ref.** | **With child class object ref.** | **With parent class object ref.** | **With child class object ref.** |
| **Private** | **Yes** | **No** | **No** | **No** | **NO** | **NO** | **No** |
| **default** | **Yes** | **Yes** | **Yes** | **Yes** | **NO** | **NO** | **NO** |
| **protected** | **Yes** | **Yes** | **Yes** | **Yes** | **NO** | **NO** | **Yes** |
| **public** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** | **Yes** |

**Polymorphism**

* It is one of the OOPs principle where one object shows different behaviours at different stages of the life cycle.
* Polymorphism is a Latin word where poly stands for many & morphism stands for forms.
* **In java Polymorphism is classified into 2 types:**

**1. Compile time Polymorphism**

**2. Runtime Polymorphism**

****

**1. Compile Time Polymorphism: early polymorhism/static polymorphism**

* **In Compile Time Polymorphism method declaration is going to get binded to its definition at compilation time, based on an argument known as compile time Polymorphism.**
* **As binding takes during compilation time only, so it is also known as early binding.**
* **Once binding is done, again rebinding can't be done, so it is called static binding.**

**Method overloading is an example of compile time Polymorphism.**

**2. Runtime Polymorphism: Late polymorphism/dynamic polymorphism**

* **In Runtime Polymorphism method declaration is going to get binded to its definition at Runtime, based on object creation is known as runtime Polymorphism.**
* **As binding takes during Runtime time, so it is also known as late binding.**
* **Once binding is done, again rebinding can be done, so it is called dynamic binding.**

**Method overriding is an example of Runtime Polymorphism.**

**Method overloading:**

**Declaring multiple method with same method name but with different argument in a same class is called method overloading**

**Creating multiple methods with same name but with different parameters or count of parameters or data type of parameters or different sequence/order of parameters**

**Minimum java class = 1 & incase inheritance multiple classes are allowed**

**Q. How to Overload a method?**

**By changing number of arguments or parameter**

**By changing the data type**

**1. Method name should be same**

**2. Parameters must be different**

**3. Count parameters must be different**

**4. Order of parameters is different**

**5. Minimum 1 class is sufficient**

**Q. Can we overload the main() method ?**

**Ans- Yes, We can overload the main() method.**

**Method overriding:**

* Acquiring super class method into sub class with the help of extends keyword & changing implementation/definition according to subclass specification is called method overriding
* Whenever the same method name is existing in both base class and derived class with same types of parameters or same order of parameters is known as method Overriding.
* **Without Inheritance method overriding is not possible.**
* Method Overriding is used to provide specific implementation of a method that is already provided by its super class.

**Method Overriding is used for Runtime Polymorphism**

* Minimum 2 classes with inheritance having methods with the same name, same return type, and same type of argument can be overridden.
* **The private, static and final methods can't be overridden as they are local to the class.**

**Q. Can we override the main() method ?**

**ANs- No because the main() is static method and we can not override static methods**

|  |  |  |
| --- | --- | --- |
|  | **Method overloading** | **Method overriding** |
| **Alternate name** | **Compile time/early/static polymorphism** | **Runtime/late/dynamic polymorphism** |
| **Methods** | **Multiple with same name** | **Parent class to child class**  **Method name should be same** |
| **Argument** | **Different (parameter or count or sequence )** | **May override**  **same** |
| **Where** | **Same class (Minimum 1 class)** | **Min.2 classes**  **parent & child** |
| **Keywords** | **—---** | **extends** |
| **Syntax** | **M1()**  **{……}**  **M1(int a, int b)**  **{…....}** | **Class A extends B** |
| **How to stop** | **We can not stop**  **(Make unique method names is only way to stop method overloading)** | **Declare methods final or static or private** |
| **For constructor** | **Overloadding of constructor is possible** | **Overridding of constructor is not possible** |
| **Main method** | **Yes** | **No because it is static** |

**##constructor name = class name**

**In each package a class is always unique**

**Abstraction**

Abstraction is one of the oops principles in java.

Abstraction means hiding the implementation code and providing only functionality to the end user is called abstraction.

the concept of exposing only the required essential characteristics and behaviour with respect to a context.

The scenario of Abstraction is "if customer is visiting or making use of any application then he should utilize functionality only & he should not feel any backend code processing"

**There are two ways to achieve abstraction in java**

**· Abstract class (0 to 100%)**

**· Interface (Achieve 100% abstraction)**

1. **Abstract class—-0% to 100%**
2. **Interface—------100%**

**Abstract Class:**

* **Abstract class is defined as a class which is declared with an "abstract" keyword.**

**Ex- abstract public class A**

* An Abstract class is also called as an incomplete class in which programmers can declare complete as well as incomplete methods.
* Programmers can declare incomplete methods as abstract methods, by declaring a keyword called "abstract" in front of the method.
* Abstract methods are also called Incomplete methods or undefined methods.
* object of abstract class can not be created, we need to make use of concrete class to create objects of abstract class

**Abstract class- We can create with 0-100% abstract methods**

**Abstract class has its own constructor.**

**Abstract methods don't have any block(definition).**

**Concrete class:**

It is defined as a class which provides definitions for all the abstract/ incomplete/undefined methods which are present in abstract class with the help of extends keyword(inheritance).

**Interface**

It is one of the oops principles.

It is used to achieve 100% abstraction.

Interface is used to declare only incomplete methods in it.

**Features of Interface:**

1. Methods declared inside an Interface are by default public & abstract.

2. Variables or data members declared in Interface are by default static and final.

3. Interface don’t have Constructor inside it.

4. We can not create object of Interface directly.

5. To create object of Interface programmers need to make use of the Implementation class.

6. Interfaces support multiple inheritance.

**Implementation class:**

It is defined as a class which provides definitions for all the abstract or incomplete methods those are present in interface with the help of implements keyword.

**Benefits of using Interface**

· We can achieve fully abstraction(100%).

· We can achieve multiple inheritance in java with interfaces

**Similarity between interface and class**

Whenever we compile any Interface program it generate .class file. That means the bytecode of an interface appears in a .class file.

**Difference between Interface compare to Class:**

* All methods in an interface are abstract.
* Interfaces can not contain instance fields.
* Interface only contains public static final variables.
* We can not instantiate an interface means can not create object
* Interface does not contain any constructors.
* Interfaces can not be extended by a class; it is implemented by a class
* Interfaces can extend multiple interfaces. It means interface support multiple inheritance

**Rules for implementation of an interface**

* A class can implement more than one interface at a time.
* A class can extend only one class, but implement many interfaces.
* An interface can extend another interface, similarly to the way that a class can extend another class.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | Parent | | | |
| 1 Parent class | More parent class | 1 parent interface | More parent interfaces |
| Child | 1 child class | extends | X | implements | implements |
| More child class | extends | X | implements | Implements |
| 1 child interface | X | X | extends | extends |
| More child interface | X | X | extends | extends |

**Comparison between Abstract class and Interface**

|  |  |  |
| --- | --- | --- |
| **Parameters/points** | **Abstract class** | **Interface** |
| **Constructor Presence** | Yes | No |
| **Methods** | Complete or abstract | All Abstract methods |
| **Definitions to abstract methods methods** | In concrete class | In Implementation class |
| **Where** | Min 2 classes  abstract class(parent) & concrete class (child) | Minimum 1 Interface & 1 class(implementation class) |
| **Keywords** | extends & abstract | implements & interface |
| **Syntax** | abstract Parent  Child extends Parent | Interface I  Class A implements I |

**Generalization:**

* Generalization is the process of extracting shared characteristics from two or more classes and combining them into a generalized superclass.
* Generalization files can be **normal java class or abstract class or Interface**, but only **Interface** is recommended.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Encapsulation**

* Encapsulation is a process of wrapping data and methods in a single unit.
* Encapsulation is achieved in java language by class concept.
* Encapsulation can be achieved using class keyword, state represents declaration of variables on attributes and behaviour represents operations in terms of method.

****

* To secure the data from other methods, when we make a data private then these data are only used within the same class, but these data are not accessible outside the class.

**Benefits of encapsulation**

* Provides abstraction between an object and its clients.
* Protects an object from unwanted access by clients.
* Example: A bank application forbids (restrict) a client to change an Account's balance

**Casting:**

Converting one type of information into another type is called casting

**In java casting is classified into 2 types:**

**1. Primitive casting-deals with primitive data type**

**2. Non-primitive casting-**

**1. Primitive casting:**

**●**  Converting one data type of information into another data type is called casting

**●**  Primitive casting is classified into 3 types:

1. implicit casting

2. explicit casting

3. boolean casting

**i. Implicit casting:**

**●**  Converting lower data type info into higher data type info is called implicit casting.

● Implicit casting is also called widening casting, where memory size goes on increasing,

**Eg.** byte a =10;

short b;

int c;

long d;

float e;

double f;

b=(short)a; // typecast a into short

b=a;

c=a; // c=(int) a; // typecast a into int

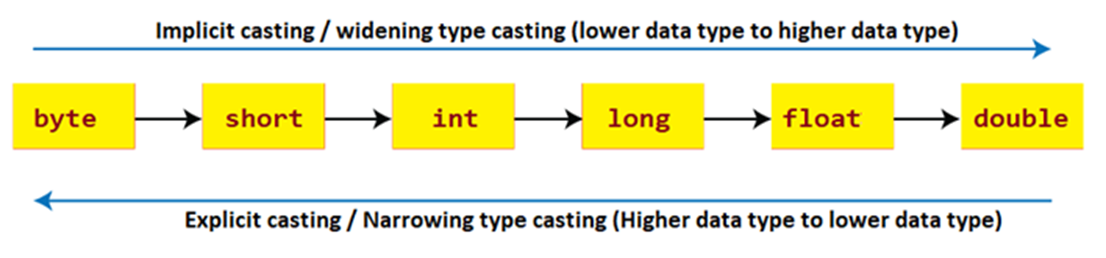
d=a;

e=a;

f=a;

c=b;

e=c;

****

**ii. Explicit casting:**

* Converting higher data type info into lower data type info is called explicit casting.
* Explicit casting is also called narrowing casting, where memory size goes on decreasing
* In explicit casting data loss may takes place

**Eg.**

**i**nt p=10; //130

byte q;

q=(byte)p; //typecasting p into byte

float r=21.756f;

int s;

s=(int)r; //typecasting r into int

**iii. Boolean casting:**

**●** Boolean casting is considered to be incompatible casting type, because Boolean data type is a unique type of data type where information is already pre-declared inside it.

● true/false

**2. Non-primitive**

**●** Converting one type of class into another type of class is called non-primitive.

●  **Non-primitive is classified into 2 types:**

**i. Up-casting**

**ii. Down-casting**

**i. Up casting:**

* Assigning subclass/child class property into superclass/parent class is called up-casting.
* Up-casting can be performed only after performing an inheritance operation
* After performing inheritance, the property which are present inside superclass/parent class comes into subclass/child class
* In the subclass programmer can declare new properties.
* At the time of up-casting the properties which are inherited from superclass are only eligible for the up-casting operation.
* The new properties which were declared inside subclass are not eligible for up-casting operation.

|  |  |  |
| --- | --- | --- |
|  | **methods** | **Global Variables** |
| **Child c=new Child();** | **Child class methods** |  |
| **Parent p =new Parent();** | **Parent class methods** |  |
| **Parent object=new Child();** | **Child class methods which are inherited from parent class** |  |

**How to read: creating the object of child class with reference to Parent class**

**Eg. webDriver driver = new chromeDriver()**

**ii. Down casting:**

**●** Assigning superclass property into subclass is called down-casting.

● Down-casting can be performed only after performing up-casting first.

**Array**

● Array is a collection of similar type of data

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

String [] b={ “Aspire”, “Institute”, “Pune”};

● It is fixed in size

● It is a collection of homogeneous data elements

- eg. all integers, or all strings

● It stores the value on the basis of the index value.

**Advantage**

**●** One variable can store multiple value

● Code Optimization

● Random access to any value in any index position

**Disadvantage → These disadvantages overcome in Collection**

**●**  Fix Size Limit

● Homogeneous data

**Types of Arrays**

1. **Single Dimensional Array**

1. int[] a;

2. int a[];

3. int []a;

1. **Multidimensional Array**

1. int[][] a;

2. int a[][];

3. int [][]a;

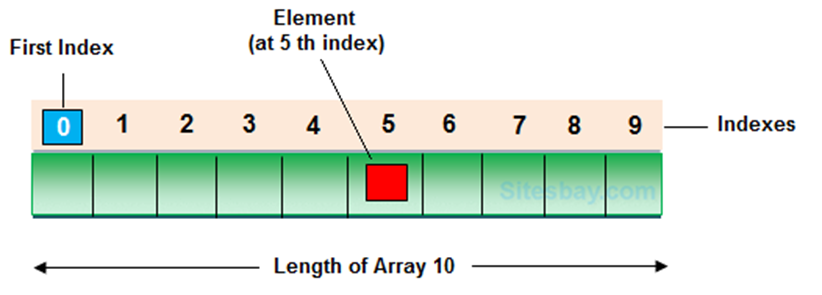
4. int[] a[];

5. int[] []a;

6. int []a[];

To specify the array size the allowed data types are **byte, short, int, char.**

The maximum allowed size of array in Java is 2,14,74,83,647



**Array length/size = highest index of array+1 OR**

**Highest array index=array size/length - 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **j=0** | **1** | **2** | **3** | **4** | **5** |
| **i=0** | **a[0][0]=11** | **a[0][1]=12** | **13** | **14** |  |  |
| **1** | **a[1][1]=15** | **a[1][1]=16** | **17** | **18** |  |  |
| **2** | **19** | **20** | **21** | **22** |  |  |
| **3** |  |  |  |  |  |  |
| **4** |  |  |  |  |  | **a[4][5]** |

**Array creation**

**1. String ar[] = new String[5]**

**ar[0]=**

**ar[1]=**

**ar[2]=**

**ar[3]=**

**ar[4]=**

**2. int ar []= {30,40,50,20,10};**

**3. ar.length --> Length/size of Array**

**4. Array index --> ar[i][j]**

**5. Arrays.sort(ar)**

**String**

1. String is a **non-primitive** data type, memory size is not fixed.

2. String is used to store collection of characters

3. String is a class present inside the "**java.lang**" package.

4. String class is the **final class** that can't be inherited to other classes.

5. At the time of String declaration, initialization, object creation takes place.

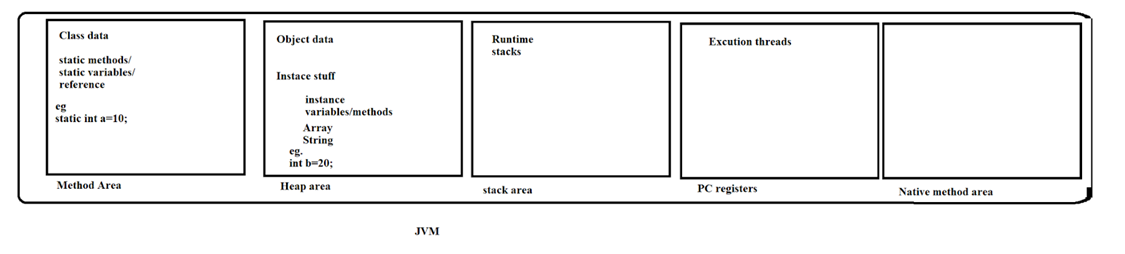
6.String objects are **immutable** in nature/can’t be changed.

7. Object creation of String can be done in 2 ways:

1. without using **new** keyword

2. Using **new** keyword

8. String objects are going to get stored inside the **String pool area** which is present inside the **heap area**.

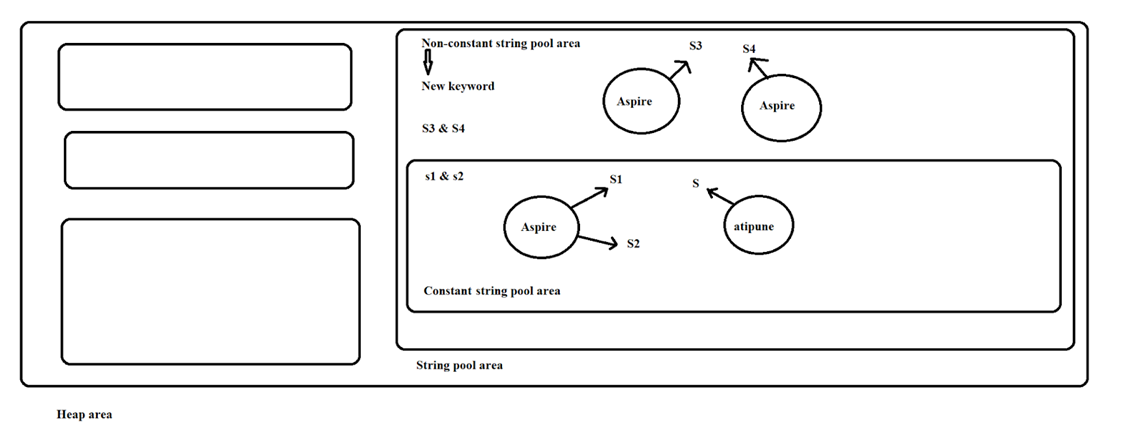
****

**String pool area:**

**It is used to store String objects.**

**It is classified into 2 areas:**

|  |  |
| --- | --- |
| **1. Constant pool area** | **2. Non-constant pool area** |
| 1.During object creation time if we don't make use of new keyword then Object creation takes place inside constant pool area.  2.Duplicate objects are not allowed inside constant pool area. | 1.During object creation time if you make use of new keyword then Object creation takes place inside non-constant pool area.    2. Duplicate objects are allowed inside non-constant pool area. |

****

**Methods/functions of String:**

1. toUpperCase();-

2. toLowerCase();

3. length()

4. contains()

5. equalsIgnoreCase()

6. equals(s1)

7. isEmpty()

8. charAt(3)

9. endsWith("")

10. startsWith("Institute")

11. substring(4)

12. substring(4, 7)

13. concat(s1)

14. indexOf('')

15. lastIndexOf('')

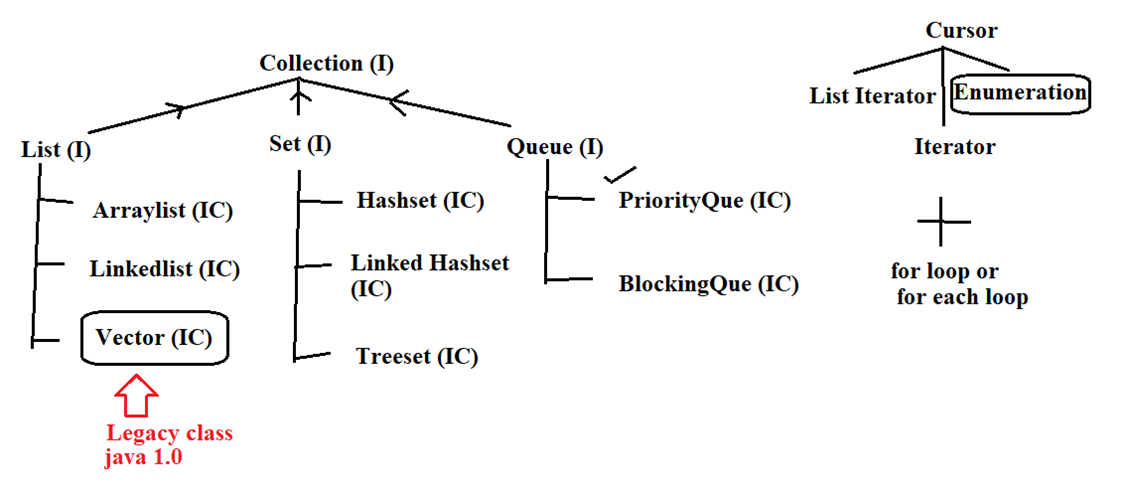
16. replace("", "")

17. String ar[] = s4.split("")

|  |  |  |  |
| --- | --- | --- | --- |
| **Case (Lower/Upper)** | **Relation / Comparison** | **Content** | **Operations** |
| **toUpperCase()** | **equalsIgnoreCase()** | **length()** | **substring(4)** |
| **toLowerCase()** | **equals(s1)** | **contains()** | **substring(4, 7)** |
|  |  | **isEmpty()** | **concat(s1)** |
|  |  | **charAt(3)** | **replace("", "")** |
|  |  | **endsWith("")** | **String ar[] = s4.split("")** |
|  |  | **startsWith("Institute")** |  |
|  |  | **indexOf('')** |  |
|  |  | **lastIndexOf('')** |  |

**Collection Framework**

It is a collection of collection classes in the java API. It can be used to handle the data structure in java language or collection classes in java API used to manage the data very efficiently like **inserting, deleting, updating, retrieving, sorting the data etc.**



1. List(I)

2. set(I)

3. queue(I)

**1. List(I)**

1. **Duplicate** are allowed in list

2. Allows **any number of null values**

3. Order of **insertion - maintain**

Types of List:

1. Arraylist (IC)

2. Vector (IC)------Legacy class

3. LinkedList (IC)

**2. Set(I)**

1. Doesn't allow duplicate

2. allow only **1 null value** (except treeset)

3. order of insertion - **random insertion (except linkedHashset)**

Types of Sets:

1. Hashset (IC)

2. LinkedHashset (IC)

3. Treeset (IC)

**3. queue(I)**

1. priority Queue (IC)a

**# Cursor**

1. Iterator-- all the collection object --universal cursor
2. List Iterator -- only for list interface type implementation classes --not universal cursor
3. enumeration -- Legacy class(vector) --not universal cursor

**1.Iterator-**- All the collection object --> Universal cursor

1)Enumeration and iterator only forward direction cursor not a backward.-- single directional cursor

2)In an iterator we can perform only read and remove operations, we can not perform replace and addition of new objects.

**2. list Iterator** -- only for list interface type impl classes --not universal cursor

1. using list iterator we can traverse a List in forward direction and backward direction--> bidirectional cursor

2. **we can perform read , remove , replace and addition of new objects.**

1. **enumeration**

1)Only applicable for legacy classes and hence it is not a universal cursor.

2)By using enumeration we can get only read access.

3)Enumeration and iterator only forward direction cursor not backward.--single directional cursor

**1. List (I):**

Types of List:

1. Arraylist (IC)

2. Vector (IC)------Legacy class

3. LinkedList (IC)

**1. Arraylist(IC)**

1. duplicate are allowed in Arraylist

2. allows any no of null values

3. order of insertion-maintain

4. Default capacity for Arraylist is 10

5. data structure: Resizable

6. Incremental capacity= (current capacity\*3/2)+1

7. best choice: retrieval operation (random access interface is implemented in Arraylist & vector)

8. worst choice: manipulation operation i.e. insertion in between Arraylist or delete ()

9. storage type: index

**2. Vector**

1. duplicate are allowed in vector

2. allows any no of null values

3. order of insertion-maintain

4. Default capacity for Vector is 10

5. data structure: doubly

6. Incremental capacity= current capacity\*2

7. best choice: retrieval operation (random access interface is implemented in Arraylist & vector)

8. worst choice: manipulation operation i.e. insertion in between Vector or delete ()

9. Vector is a legacy class.

10. storage type: index

**3. LinkList**

1. duplicate are allowed in Arraylist

2. allows any no of null values

3. order of insertion-maintain

4. no Default capacity in linkedlist

5. data structure: linear

7. best choice: manipulation operation i.e. insertion in between linkedlist or delete()

8. worst choice: retrieval operation

9. storage type: index

**2. Set(I)**

1. doesn't allow duplicate

2. allow only 1 null value(except treeset)

3. order of insertion-random insertion

**Types of sets:**

1. Hashset(IC)

2. LinkedHashset(IC)

3. Treeset(IC)

**1.Hashset:**

1. Doesn't allow duplicate values

2. Allow only 1 null value.

3. order of insertion-random insertion

4. no default capacity

5. DS: Hashtable

6. storage type: hashtable

7. best choice: To remove duplicate elements when order of insertion is not

Mandatory.

**2. LinkedHasSet:**

1. Doesn't allow duplicate values

2. Allow only 1 null value.

\*3. order of insertion-maintained

4. no default capacity

\*5. DS: Hybrid (liner+ hashtable)

6. storage type: hashtable

7.best choice: To remove duplicate elements when order of insertion is

Mandatory

**3. TreeSet:**

1. doesn't allow duplicate

2. null values: not allowed/ no null value

\*3. order of insertion- Ascending order.

4. no default capacity.

5. DS: Balanced tree

6. storage type: hashtable

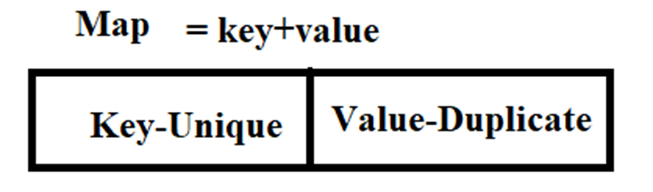
7.best choice: To remove duplicate elements when order of insertion

Ascending order.

**Note: we can store only homogeneous data in TreeSet**

Common program to compare all Set & List Classes of Collection

**Map/HashMap**



* Key-Unique
* Values-May be unique or duplicate
* Data type of key and value- May be same or different.
* Data storage- Ascending order as per key