## INTRODUCTION OF JAVA 1

* + 1. Platform independent
    2. Open source
    3. Multi-threading
    4. More secure
    5. portable

## CORE JAVA TOPICS 4

* + 1. Oops concept
    2. Control statement and looping
    3. Arrays
    4. String
    5. Collections
    6. Exceptions
    7. Constructor

## INTRODUCTION OF OOPS 6

* + 1. Class
    2. Object
    3. Method
    4. Encapsulation
    5. Inheritance
    6. Polymorphism
    7. Abstraction

## ENCAPSULATION 7

1. **INHERITANCE 7**
   * 1. Single inheritance
     2. Multilevel inheritance
     3. Multiple inheritance
     4. Hybrid inheritance
     5. Hierarchical inheritance

## ACCESS SPECIFIER 9

* + 1. Public
    2. Private
    3. Protected
    4. Default

## DATA TYPES AND WRAPPER CLASS 10

* + 1. Data types
    2. Wrapper class

## CONTROL STATEMENT 12

* + 1. If
    2. If-else
    3. If-else-if

## LOOPING 13

* + 1. For
    2. While
    3. Do-while

## POLYMORPHISM 17

* + 1. Method overloading
    2. Method overriding

## ABSTRACTION… 20

* + 1. Abstract class
    2. Interface

## ARRAYS… 22

1. **STRING 25**
   * 1. String functions
     2. Literal string
     3. Non-literal string
     4. Mutable string
     5. Immutable string

## COLLECTIONS… 26

* + 1. List
    2. Set
    3. Map

## EXCEPTION… 59

* + 1. Unchecked exception
    2. Checked exception
    3. Exception handling

## CONSTRUCTOR 73

1. **CORE JAVA INTERVIEW QUESTIONS AND ANSWERS… 79**

## CORE JAVA INTERVIEW PROGRAMS 92

1. **JDBC CONNECTIONS 98**

**JAVA INTRODUCTION**

* + Java is a simple programing language
  + Writing, compilation and debugging a program is very easy in java
  + It helps to create reusable code
  + Java has more features,

1. Platform independent
2. Open source
3. Multithreading
4. More secure
5. Portable

## Platform independent

* + During the compilation the java program converts into byte code
  + Using byte code we can run the application to any platform such as windows, mac, linux, etc

## Open source:

* + A program in which source code is available to the general public for use and/or modification from its original design at free of cost is called open source

## Multithreading:

* + Java supports multithreading
  + It enables a program to perform several task simultaneously

## More secure:

* + It provides the virtual firewall between the application and the computer
  + So it’s doesn't grant unauthorized access

## Portable:

* + "Write once run anywhere"
  + Java code written in one machine can run on another machine

## CORE JAVA

Syllabus:

* 1. Oops concept
  2. Control statement/looping
  3. Arrays
  4. String
  5. Exceptions
  6. Collections

## Terminology:

1. JDK
2. JRE
3. JVM

## JDK:

* Java Development Kit
* If run any applications we need JDK have to installed
* JDK versions: 1.0 to 1.9
* Mostly V1.8 is used now

## JRE:

* Java Runtime Environment
* It is a pre-defined. class files (i.e.) library files

## JVM:

* Java Virtual Machine
* It is mainly used to allocate the memory and compiling

## TOOLS:

* 1. Notepad
  2. Net bean
  3. Eclipse
  4. J Developer-oracle
  5. RAD-IBM
* Nowadays we mostly used eclipse (75% of the people using).
* Versions of eclipse:
  + Juno
  + Kepler
  + Luna
  + Mars
  + Neon

## OOPS CONCEPT:

* Object Oriented Programing Structure
* OOPS is a method of implementation in which programs are organized as collection of objects, class and methods

Oops principles are

1. Class
2. Method
3. Object
4. Abstraction
5. Encapsulation
6. Inheritance
7. Polymorphism

## CLASS:

* Class is nothing but collection of methods or collection of objects.
  + Project name : Should be in Pascal notation
  + Pascal notation : Each word of the first letter should be in capital
  + src - Source file
  + Class name: Pascal notation
  + Package creation: ex, org.cts.scope-All small letters

## Syntax:

(First type class name and click ctrl +space)

**public class** Bank {

} // Bank is a class name

Public-Access specifier

## METHOD:

* Set of action to be performed

Method name: camel notation

Camel notation: First word should be small after every word of the first letter should be capital **Syntax:**

**public void** dummy() {

// Here dummy is a method name

}

Main Method:

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

}

Main method type main and click ctrl +space

## OBJECT:

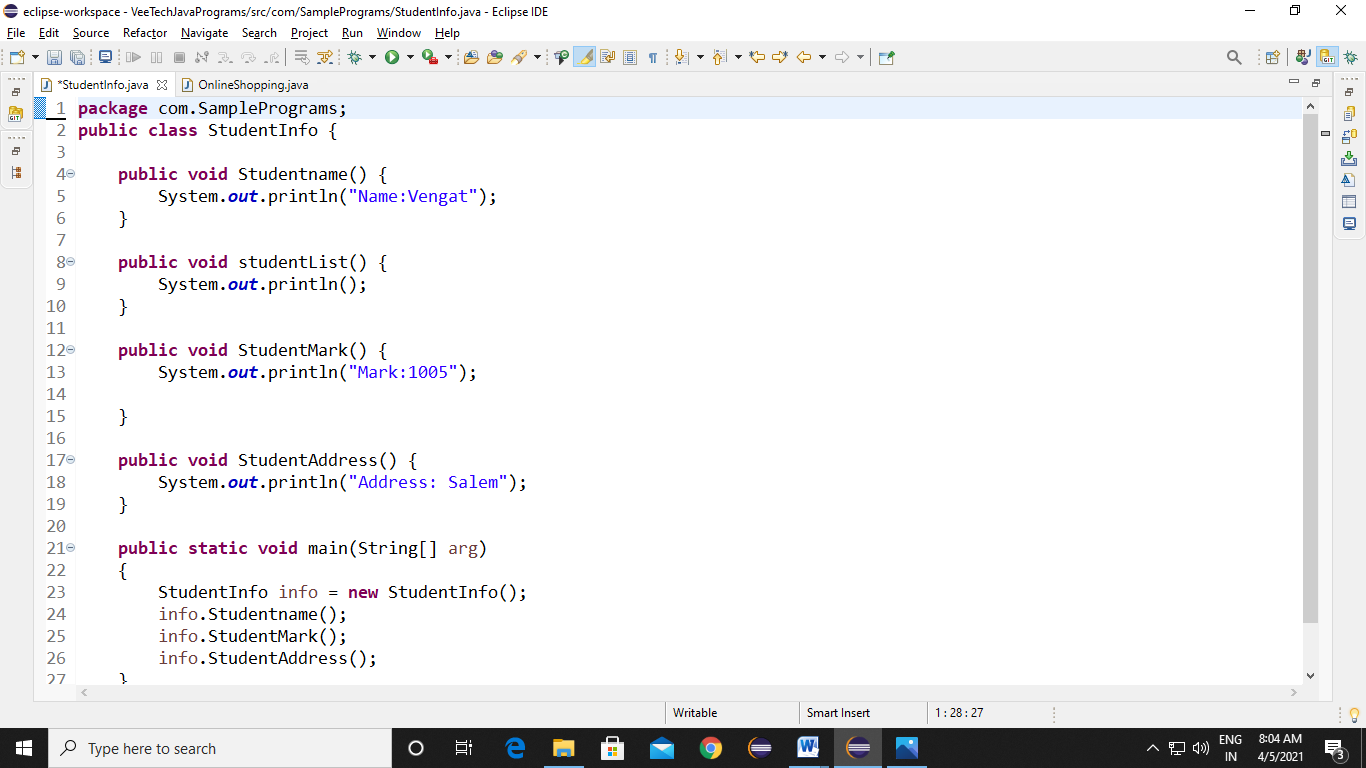
* Run time memory allocation
* Using object we call the any methods

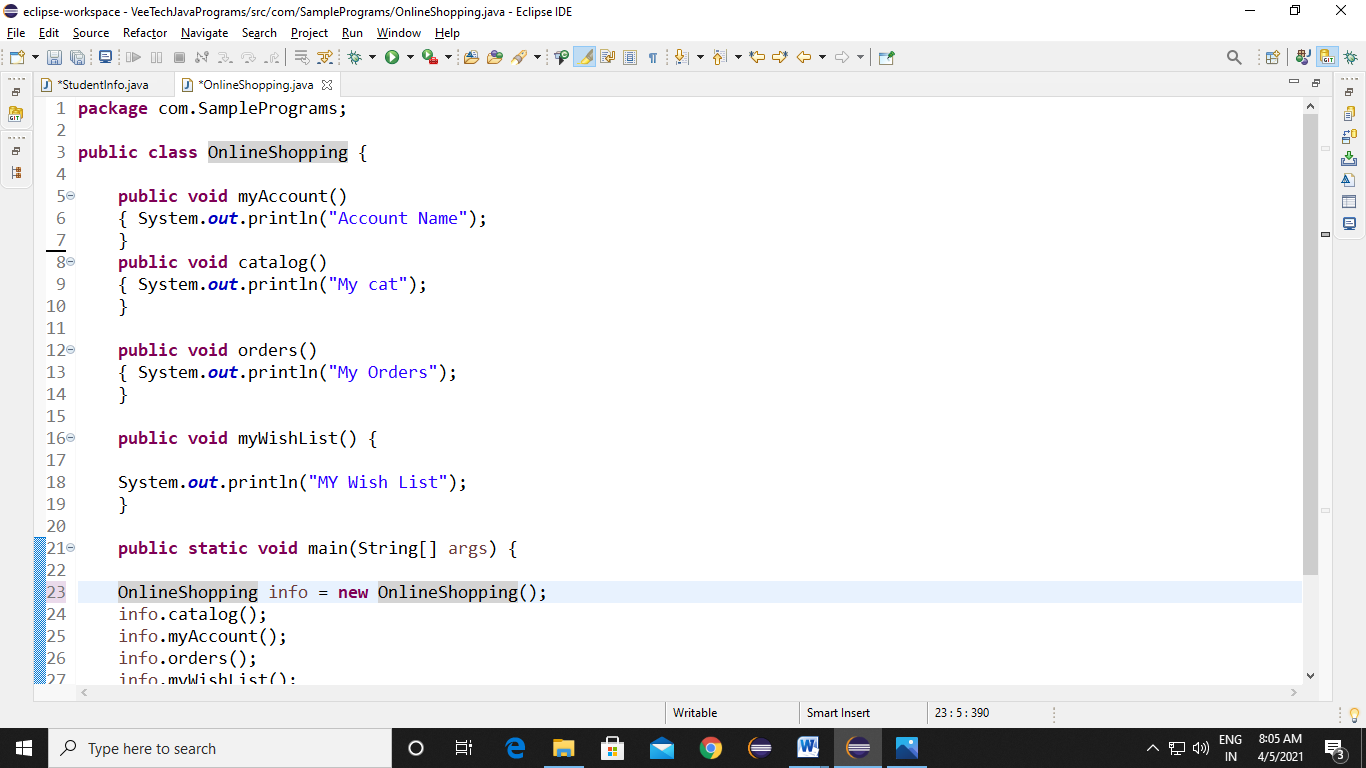
Syntax:

(Class name) (Object name) =new (Class name) ();

* Alignment ctrl + shift+ F
* Run  ctrl +F11

## Example program:





## Heap Memory:

* Object are stored in heap memory

RAM JVM Heap memory

* To reduce object memory we go for inheritance

## ENCAPSULATION

* Structure of creating folders

## INHERITANCE:

* We can access one class property into another class using 'extend' keyword and reusuable purpose

Child class Sub class Parent class Super class **Types:**

* 1. Single Inheritance
  2. Multilevel Inheritance
  3. Multiple Inheritances
  4. Hybrid Inheritance
  5. Hierarchical Inheritance

## Single Inheritance :

* + One parent class is directly support into one child class using extend keyword

Child Class Parent class

## Multilevel Inheritance:

* + One child class and more than one parent class

Child Class

Parent Class

Grand parent

## Multiple Inheritance:

Parent class

Child class

Parent class

* + More than one parent class parallely support into one child class but it won't suport in java because

1. Priority problem
2. Compilation error/syntax error

(i.e) if both parent class having same method name it will get priority problem so it doesn;t work in java

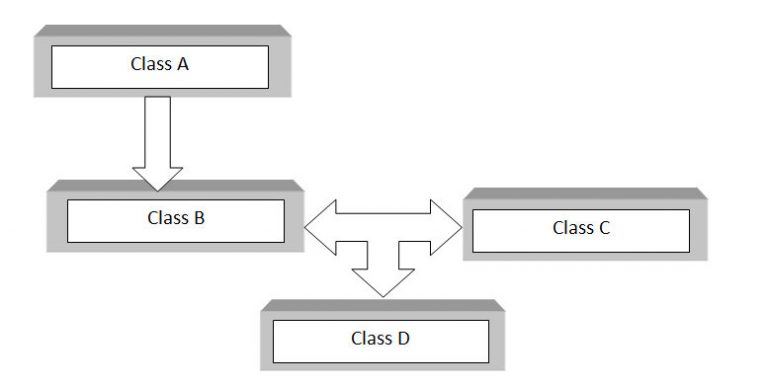
Parent class child class  parent class

test () test()

* + test () is a method name, it present in both parent class, so its get priority problem

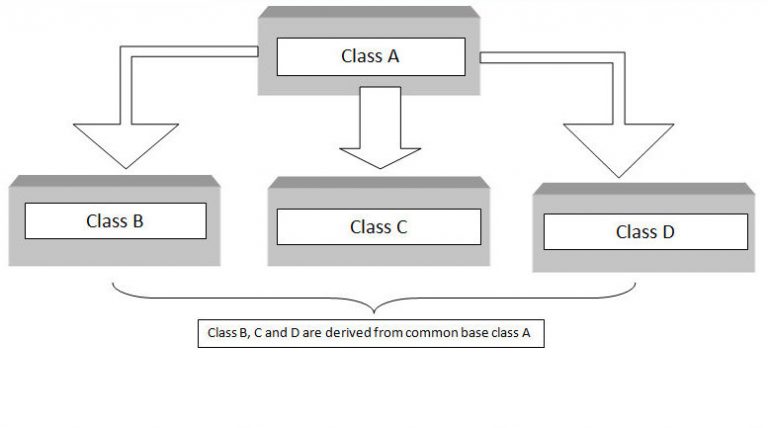
## Hybrid Inheritance:

* + It is a combination of single and multiple inheritance



## Hierarchical Inheritance:

* + One parent class and more than one child class



**ACCESS SPECIFIER:**

1. Public
2. Protected
3. Default
4. Private
5. Public:
   * It is global level access( same package + different package)
6. Private:
   * It is a class level access
7. Default:
   * Package level access
   * Without access specifier within the package we can access

Ex,

Public static public- access specifier (need to mention public) Private static private- access specifier (need to mention) Static default- access specifier (need not to mention default)

1. Protected:

Inside package + outside Package (Extends)

**DATA TYPES:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data types** | **Size** | **Wrapper Class** | **Default value** |
| byte | 1 | Byte | 0 |
| short | 2 | Short | 0 |
| int | 4 | Integer | 0 |
| long | 8 | Long | 0 |
| float | 4 | Float | 0.0 |
| double | 8 | Double | 0.0 |
| boolean | - | Boolean | false |
| char | - | Character | - |
| String | - | String | null |

* + To find range: formula

-2^n-1 to +2^n-1 - 1

For **byte**,

1 byte =8 bits So n=8 Apply

-128 to + 127

This is a byte range

## Ex:

**public class** ByteInfo {

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

}

}

## Long:

**byte** num=12; System.***out***.println(num);

symbol 'l'

long n= 123467l( need to enter l finally)

**Float** Symbol-‘f’ float f=10.06f **Double:**

No need to enter d

## char:

Character is any number, any alphabet or any special character char= 'A'single quotation

## String:

String = "klaou8778k" double quotation

## Boolean:

Boolean is a true or false value boolean b1=true;

boolean b2=false;

## Default package of java:

* java.lang
* ctrl+2+L this shortcut is used to find the data type Syntax: to get the values from the user
* bytenextByte();
* shortnextShort();
* intnextInt();
* longnextLong();
* floatnextFloat();
* doublenextDouble();
* charnext().charAt(0);
* Stringnext();
* StringnextLine();
* booleannextBoolean();

## String:

* nextLine() is used to include space String.s=sc.nextLine();

**WRAPPER CLASS:**

* Classes of data types is called wrapper class
* It is used to convert any data type into object
* All classes and wrapper classes default value is Null

**CONTROL STATEMENT:**

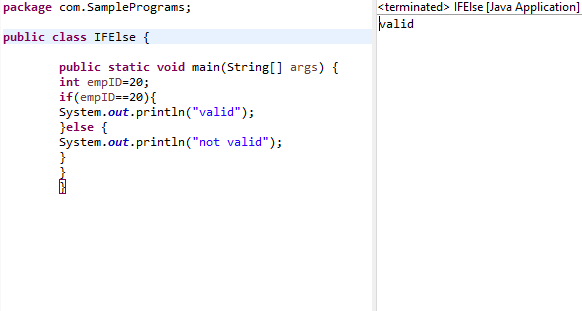
1. if
2. if.else
3. else.if

variable name camel notation Difference between "=" and "=="

 = is used to assigning the value

 == is used for condition checking

## Example Program:



**Output** valid

* More than one condition we use for
  1. logical &&,// logical && check first condition if its fail it doesn't check

second

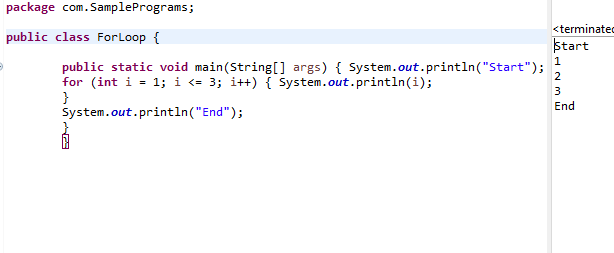
* 1. Bitwise &,/ bitwise & is check both condition
* So logical && is better than bitwise

## LOOPING:

1. for
2. while
3. do.while

## For:

**Example Program:**



**INNERFORLOOP:**

## Break:

* It will exit from the current loop

## Example Program:

**public class** InnerForLoop {

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

**for** (**int** i = 1; i <= 10; i++) {

**if** (i == 5) {

## break;

}

}}

## Output:

1

2

3

4

}

System.***out***.println(i);

## Continue:

* It will skip the particular iteration

## Example Program:

**public class** InnerForLoop {

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

**for** (**int** i = 1; i <= 10; i++) {

**if** (i == 5) {

**continue**;

}

System.***out***.println(i);

}

}

}

## Output

1

2

3

4

6

7

8

9

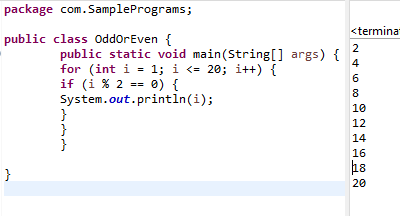
10

## Basic programs using conditional statements:

EVEN NUMBER AND ODD NUMBER:

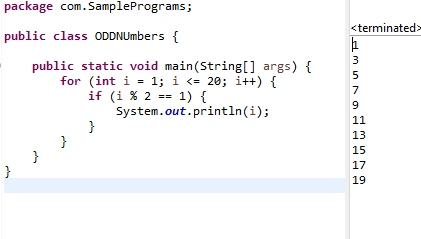
## To print Even num:

**Example Program:**



## To print Odd:

**Example Program:**



## POLYMORPHISM:

* Poly-many
* Morphism-forms
* Taking more than one forms is called polymorphism or one task completed by many ways

It has 2 types,

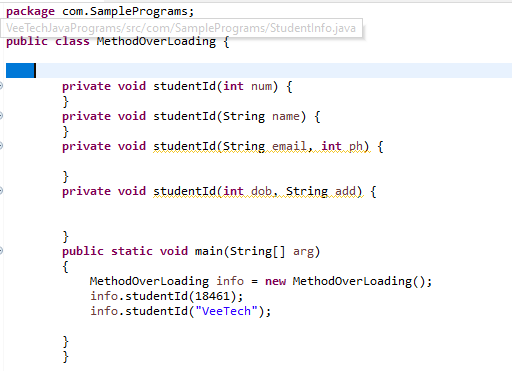
1.Method overloading(static binding/compile time polymorphism) 2.Method overriding(dynamic binding/run time polymorphism)

## Method overloading:

Class-same Method-same Argument-differ

* + In a same class method name is same and the argument is different is called method overloading
  + the argument is depends on
    - data types
    - data types count
    - data type order

**Example Program**:



* + In the same method the argument can't use int and byte because int &byte both are numbers. so it doesn't work.
  + public void employeeID(int num, byte num2) is not correct

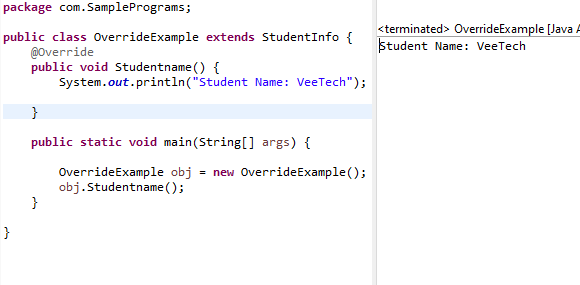
## Method overriding:

Class name-differ(using extends) Method-same

Argument- same

* + In a different class , the method name should be same and argument name should be same is called overriding

## Example Program:



## Example Program:

## Static KeyWords:

Marriage **public class** Employee{ **public static void** addNum(){ System.***out***.println("Hello");

}

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

*addNum*(); // dont want to create object

}

}

**Output**: Hello

* + If its different class we have to use class name(i.e)

**sub class**:

**public class** Employee{ **public static void** addNum(){ system.out.println("Hello");

}}

**super class**:

**public class** sample{

**public static void** main(string[] args){ Employee.addNum();

}}

**Output:** Hello

**ABSTRACTION:**

* + Hiding the implementation part is called abstraction
  + it has 2 types,

1.Partially abstraction(abstract class) 2.Fully abstraction(interface)

## Partially Abstraction(Abstract class):

* + It will support abstract method and non-abstract method.
  + We can’t create object for abstract class because in the method signature we didn't mention any business logic. so
  + In abstract method, we only mention abstract signature, won't create business logic
  + It have 2 class, abstract class(sub class) and super class. we create object and business logic only in super class, won't create in abstract class

## Example Program:

abstract class

**public abstract class** Bank {

**abstract void** saving(); //method signature

**abstract void** current(); **abstract void** salary(); **abstract void** joint(); **public void** branchDetails(){

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

}

}

super class

**public class** AxisBank **extends** Bank {

**public void** saving() { // method signature System.***out***.println("saving regular"); // business logic

}

**public void** current() { System.***out***.println("current");

}

**public void** salary() { System.***out***.println("salary");

}

**public void** joint() { System.***out***.println("joint");

}

**public static void** main(String[] args) { AxisBank info = **new** AxisBank(); info.branchDetails(); info.salary();

info.saving();

}}

## Output:

chennai

salary

saving regular

## INTERFACE/FULLY ABSTRACTION;

* + It will support only abstract method, won't support non abstract method
  + In interface "public abstract" is default. we no need to mention
  + It using implements keywords

## Example Program:1

interface

**public interface** Bank { **abstract void** saving(); **abstract void** current(); **abstract void** salary(); **abstract void** joint();

**public void** branchDetails();

}

super class

**public class** AxisBank **implements** Bank {

**public void** saving() { System.***out***.println("saving regular");

}

**public void** current() { System.***out***.println("current");

}

**public void** salary() { System.***out***.println("salary");

}

**public void** joint() { System.***out***.println("joint");

}

**public void** branchDetails() { System.***out***.println("chennai");

}

**public static void** main(String[] args) { AxisBank info = **new** AxisBank(); info.branchDetails(); info.salary();

info.saving();

}

}

**Output:** chennai salary

saving regular

 multiple inheritance its won’t support in java but using interface its support

 here we have to create 2 interface(super class) and one sub class(normal). In the sub class we implement both interface

## Example Program:2

interface

**public interface** AxisBank {

**public void** test();

}

**public interface** HdfcBank {

**public void** test();

}

sub **class**(normal **class**)

**public class** Bank **implements** AxisBank, HdfcBank{ @Override

**public void** test() {

// **TODO** Autogenerated method stub

}

}

## Difference between abstract class and interface

Abstract class:

* It is partially abstraction
* It support both abstract method and non-abstract method
* It’s using "extends" keyword
* Here "public abstract" have to mention
* We can use whatever access specifier we want

Interface:

* It is fully abstraction
* It support only abstract method
* It’s using "implement" keyword
* "public Abstract" is default. no need to mention
* Here we use only public( access specifier)

## ARRAYS:

* Collection of similar data
* The value are stored based on index
* The index will start 0 to n1

Syntax:

int num[]=new num[5]

Here,

int  data type num  variable []  Array

5 Array length

* It takes 0 to 4(i.e) 0 to n-1, n=5

## Example Program:

**public class** BasicArray {

**public static void** main(String[] args) { **int** num[]=**new int**[5]; System.***out***.println(num[2]);

}

}

**Output**: 0

* If we didn't assign any value, it will takes the default value of data types(int)
* Default value of int is 0

## Example Program:

**public class** BasicArray {

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

**int** num[]=**new int**[5]; num[0]=10;

num[1]=20; num[2]=30; num[3]=40; num[4]=50;

System.***out***.println(num[2]);

}

}

**Output**: 30

 Overwrite the value:

**public class** BasicArray {

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

**int** num[]=**new int**[5]; num[0]=10;

num[1]=20; num[2]=30; num[3]=40; num[4]=50; num[2]=300;

System.***out***.println(num[2]);

}}

## Output:

300

 If we overwrite the value, it takes last one

## To find array length:

**public class** BasicArray {

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

**int** num[]=**new int**[5]; num[0]=10;

num[1]=20; num[2]=30; num[3]=40; num[4]=50; num[2]=300;

**int** len=num.length; System.***out***.println(len);

}}

## Output:

5

## Using for loop:

**public class** BasicArray {

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

**int** num[]=**new int**[5]; num[0]=10;

num[1]=20; num[2]=30; num[3]=40; num[4]=50; num[2]=300;

**for**(**int** i=0;i<num.length;i++) System.***out***.println(num[i]);

}

}

## Output:

10

20

300

40

50

## Enhanced for loop:

syntax:

for(int k:num) System.out.println(k); **Example Program:**

**public class** BasicArray {

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

**int** num[]=**new int**[5]; num[0]=10;

num[1]=20; num[2]=30; num[3]=40; num[4]=50;

}

}

## Output:

10

20

300

40

50

num[2]=300;

**for**(**int** k:num) System.***out***.println(k);

* In this enhanced for loop, have no condition checking and value assign
* It is very fast compare to normal for loop

## Advantage of array:

* In a single variable we can store multiple values

## Disadvantage of arrays:

* It support only similar data types
* It is a fixed size
* Memory wastage is high
* To overcome these we go for collections

## STRING:

* Collections of charactor or word enclosed with double quotes

Basic Topics:

* String function
* Mutable string
* Immutable string

## Example Program:

**public class** StringBasic {

**public static void** main(String[] args) { String s1="Raman"; System.***out***.println(s1);

}

}

**Output**: Raman

**Some Basic Methods:**

**charAt():**

* + It is used to print the particular character

## Example Program:

**public class** StringBasic {

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

String s1="Raman"; System.***out***.println(s1); **char** ch = s1.charAt(2); System.***out***.println(ch);

}

}

## Output:

Raman n

* 2 takes as 0 to 2 (i.e) 0 1 2> v e n

## Equqls():

* + equals is a method is used to check our string index is true or false

## Example Program:

**public class** StringBasic {

**public static void** main(String[] args) { String s1 = "Raman"; System.***out***.println(s1);

**boolean** b = s1.equals("Raman"); System.***out***.println(b);

**boolean** b1 = s1.equals("Raman"); System.***out***.println(b1);}}

## Output:

Raman true false

* + b1 is false because equals() is case sensitive

## Equalsignorecase():

* + It is like a equals() method but it is not case sensitive

## Example Program:

**public class** StringBasic {

**public static void** main(String[] args) { String s1 = "Raman"; System.***out***.println(s1);

**boolean** b = s1.equals("Raman"); System.***out***.println(b);

**boolean** b1 = s1.equalsIgnoreCase("Raman"); System.***out***.println(b1);

}}

**Output:** Raman true

true

**contains():**

* + Contains() is a method , is used to check the particular character or word in the string

## Example Program:

**public class** StringBasic {

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

String s1 = "Hello welcome to java class"; System.***out***.println(s1);

**boolean** b = s1.contains("welcome"); System.***out***.println(b);

}

}

## Output:

Hello welcome to java class true

* + If we check other than the string index, it shows false

## Example Program:

**public class** StringBasic {

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

String s1 = "Hello welcome to java class"; System.***out***.println(s1);

**boolean** b = s1.contains("welcome"); System.***out***.println(b);

**boolean** b1 = s1.contains("hai"); System.***out***.println(b1);

}

}

## Output:

Hello welcome to java class true

false

## split():

* + split() is a method, is used to split the string by space or character or word or whatever

## Example Program:

**public class** StringBasic {

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

String s1 = "Hello welcome to java class";

String[] x = s1.split(" "); // here we split by space System.***out***.println(s1.length()); System.***out***.println(x.length);

String[] x1 = s1.split("o"); // here we split by "o" System.***out***.println(s1.length()); System.***out***.println(x1.length);

}

}

**Output:**

27 //this whole string length s1

5 // this is after splitting by spacex

27 //this whole string length s1

4 // this is after splitting by "o"x1

## For loop:

**Example Program:**

**public class** StringBasic {

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

String s1 = "Hello welcome to java class";

String[] x = s1.split(" "); // here we split by space

**for**(**int** i=0;i<x.length;i++){ System.***out***.println(x[i]);

}

}

}

## Output:

Hello welcome to

java class

## Enhanced for loop:

**public class** StringBasic {

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

String s1 = "Hello welcome to java class";

String[] x = s1.split(" "); // here we split by space

**for** (String k : x) { System.***out***.println(k);

}

}

}

## Output:

Hello welcome to

java class

**toUpperCase() and toLowerCase():**

* toUpperCase() is used to convert the string into uppercase
* toLowerCase() is used to convert the string into lowercase

## Example Program:

**public class** StringBasic {

**public static void** main(String[] args) { String s1 = "Hello";

String m = s1.toLowerCase(); // to convert lowercase System.***out***.println(m);

String m1 = s1.toUpperCase(); // to convert upper System.***out***.println(m1);

}

}

## Output:

hello HELLO

## subString();

* It is used to print from, which character we want in the string index

## Example Program:

**public class** StringBasic {

**public static void** main(String[] args) { String s1 = "Hello java";

String m = s1.substring(2);



// upto

}}

**Output:** llo java llo

**indexOf():**

System.***out***.println(m);

String m1 = s1.substring(2, 6); System.***out***.println(m1);

* It is used to print the position of the character in the string
* If it is available means, its print the relevant position
* But if the character is not available , it will print "-1"
* As well as , if multiple same character is have, it takes first one position

## Example Program:

**public class** StringBasic {

**public static void** main(String[] args) { String s1 = "Hello java";

**int** m = s1.indexOf("o"); // "o", to print o position System.***out***.println(m);

**int** m1 = s1.indexOf("b"); // "b" is not in the string, so it

print "-1"

System.***out***.println(m1);

**int** m2 = s1.indexOf("a"); // multiple character "a", it takes first one

System.***out***.println(m2);

}

}

**Output:**

4

-1

7

## lastIndexof():

* If multiple same character , it takes last one

## Example Program:

**public class** StringBasic {

**public static void** main(String[] args) { String s1 = "Hello java";

**int** m = s1.indexOf("o"); // "o", to print o position System.***out***.println(m);

**int** m1 = s1.indexOf("b"); // "b" is not in the string, so it

print "1"

System.***out***.println(m1);

**int** m2 = s1.indexOf("a"); // multiple character "a", it

takes first one

System.***out***.println(m2);

**int** m3 = s1.lastIndexOf("a"); // multiple character "a", it takes last one

System.***out***.println(m3);

}

}

## Output:

4

1

7

9

## replace():

* replace() is a method ,it is used to replace the index character or word

## Example Program:

**public class** StringBasic {

**public static void** main(String[] args) { String s1 = "Hello world";

String m = s1.replace("world", "java"); // to replace world

to java

System.***out***.println(m);

}

}

**Output:**

Hello java

## Example Program:

**public class** StringBasic {

**public static void** main(String[] args) { String s1 = "This is manual Testing";

String m = s1.replace("manual", "Automation"); // to replace manual to Automation

System.***out***.println(m);

}

}

## Output:

This is Automation Testing

## isEmpty():

* It is used to check the index length is zero or not,
* If its zero, its true otherwise false

## Example Program:

**public class** StringBasic {

**public static void** main(String[] args) { String s1 = "";

**boolean** m = s1.isEmpty();// here index is empty, so its true System.***out***.println(m);

String s2 = " ";

**boolean** m1 = s2.isEmpty();

// here index is not empty because space included System.***out***.println(m1); // space is also a one character

}

}

## Output:

true false

## startsWith() and endsWith():

* It is used to check the index starts with particular word or character
* As well as ends with

## Example Program:

**public class** StringBasic {

**public static void** main(String[] args) { String s1 = "Hello java class"; **boolean** m = s1.startsWith("Hello"); System.***out***.println(m);

**boolean** m1 = s1.endsWith("class");

System.***out***.println(m1);

}

}

**Output:**

true true

**ASCII value:**

* To find ascii value

## Example Program:

**public class** Dummy {

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

**char** ch='M';

**int** x=ch; System.***out***.println(x);

}

}

**Output**: 77

* Every character have one ASCII value
* A-Z 65 to 90
* A-z 97 to 122
* 0-9 48 to 57
* remaining special characters

## compareTo():

* It is a method, it is used to compare the character based on ASCII value

## Example Program:

**public class** Dummy {

**public static void** main(String[] args) { String s="A";

**int** i = s.compareTo("A"); System.***out***.println(i);

}

}

## Output:

0

Here, A ASCII value is 65, so 65-65=0

## Example Program:

**public class** Dummy {

**public static void** main(String[] args) { String s="A";

**int** i = s.compareTo("B"); System.***out***.println(i);

}}

**Output:**

1

Here, 65-66=1

* If we use many character, it will compare only first differing character

## Example Program:

**public class** Dummy {

**public static void** main(String[] args) { String s="ABCD";

**int** i = s.compareTo("ABFK "); System.***out***.println(i);

}

}

## Output:

3 // 6770=3

* If it is different length and same accuration, the output will be based on length

## Example Program:

**public class** Dummy {

**public static void** main(String[] args) { String s="ABCD";

**int** i = s.compareTo("AB"); System.***out***.println(i);

}

}

## Output:

2

Here, ABCD length is 4 AB2, 42=2

* If different length and different accuration , it will compare the first differing character

## Example Program:

**public class** Dummy {

**public static void** main(String[] args) { String s="ABCD";

**int** i = s.compareTo("ACLK"); System.***out***.println(i);

}

}

**Output:**

1

Here, 6667=1, BC=1

**Literal String:**

* It’s stored inside the heap memory (string pool or string constant).
* It will share the memory if same value (duplicate value)

**Non-literal string:**

* Its stored in the heap memory.
* Its create a new memory every time even if its duplicate value(same value)

## Example Program:

**public class** StringBasic {

**public static void** main(String[] args) { String s1 = "Raman";

String s2 = "Raman"; // literal string(same value so its share the memory)

System.***out***.println(System.*identityHashCode*(s1)); System.***out***.println(System.*identityHashCode*(s2)); String x1=**new** String("Raman");

String x2=**new** String("Raman");// non literal string( its won't share, create new memory

System.***out***.println(System.*identityHashCode*(x1)); System.***out***.println(System.*identityHashCode*(x2));

}

}

* identityHashcode() is used to print the reference value(storage reference)

## Output:

31168322 // literal string share the memory if same value 31168322

17225372

5433634 // but non literal won't share

## Immutable string:

* + We can store more duplicate value in same memory
  + We can't change the value in memory
  + In concord nation, it’s have to create new memory

## Mutable string:

* + we can't store duplicate value in same memory
  + we can change the value in memory
  + In concord nation, its takes same memory

## Example Program:

**public class** StringBasic {

**public static void** main(String[] args) { String s1 = "Raman";

String s2 = "prabu"; // mutable string System.***out***.println("Immutable string"); System.***out***.println(System.*identityHashCode*(s1)); System.***out***.println(System.*identityHashCode*(s2)); String r = s1.concat(s2); System.***out***.println(r); System.***out***.println(System.*identityHashCode*(r));

StringBuffer x1=**new** StringBuffer("Raman");

StringBuffer x2=**new** StringBuffer("prabu");// mutable string System.***out***.println("mutable string"); System.***out***.println(System.*identityHashCode*(x1)); System.***out***.println(System.*identityHashCode*(x2)); x1.append(x2);

System.***out***.println(x1); System.***out***.println(System.*identityHashCode*(x1));

}

}

## Output:

Immutable string 31168322

17225372

Ramanprabu

5433634 // here it takes new memory for concordinattion mutable string

2430287

17689166

Ramanprabu

2430287 // but here it takes x1 memory

## COLLECTIONS:

Why we go for collections:

* + It will support dissimilar data types.
  + It is dynamic memory allocation
  + No memory wastage like array

It has 3 types,

1. List
2. Set
3. Map

## List :( Interface)

* + ArrayList(class)
  + LinkedArrayList(c)
  + vector(c)

1. **Set:(Interface)**
   * Hashset(c)
   * Linked hashset(c)
   * Treeset(c)
2. **Map:(Interface)**
   * Hashmap(c)
   * Linked hashmap(c)
   * Treemap(c)
   * Hashtable(c)
   * concurrent hashmap(C)

**List:**

**ArrayList:**

Syntax:

List ex=new ArrayList(); Here,

Listinterface exobject name ArrayList() class **Example Program:**

**public class** ArList {

**public static void** main(String[] args) { List ex=**new** ArrayList(); ex.add(10); ex.add(10000000000000000l);

ex.add(10.12f);

ex.add("Hai");

ex.add("A"); ex.add(**true**); System.***out***.println(ex);

}

}

**Output:**

[10, 10000000000000000, 10.12, Hai, A, true]

* add() is a method, it is used to insert a value.
* Arraylist will display the output based on the insertion order

## Generics:

* It will support particular datatypes or object only
* It is a features of jdk 1.5
* In the generics, we can mention only wrapper class
* < >- This is generic symbol, is used to define the particular datatype
* If we need integer datatype,

syntax:

List<Integer> ex=new ArrayList<Integer>();

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex=**new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(40);

ex.add(50); System.***out***.println(ex);

}

}

Output:

[10, 20, 30, 40, 40, 50]

* List allows the duplicate value
* ArrayList print in a insertion order

## size():

* size is a method, it is used to find the size of the ArrayList

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex=**new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(40);

ex.add(50);

**int** i = ex.size(); System.***out***.println(i);

}}

## Output:

6

## get():

* get() is a method , it is used to print the particular value

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex=**new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(40);

ex.add(50);

**int** x = ex.get(3); System.***out***.println(x);

}

}

**Output:**

40

* it takes the value from 0(i.e) 0 1 2 3 > 10 20 30 40

**For loop:**

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex=**new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(40);

ex.add(50);

**for**(**int** i=0;i<ex.size();i++){ System.***out***.println(ex.get(i));

}

}

}

## Output:

10

20

30

40

40

50

## Enhanced for loop:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex=**new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(40);

ex.add(50);

**for**(Integer k:ex){ System.***out***.println(k);

}

}

}

**Output:**

10

20

30

40

40

50

**Remove()**:

* remove is a method, it is used to remove the particular index value
* If we remove the particur index value, index order will not change
* After that the index value move to forward

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex = **new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(40);

ex.add(50);

ex.remove(3); System.***out***.println(ex);

}

}

## Output:

[10, 20, 30, 40, 50]

* In this output, index order is not change
* But the values moved to forward

## Index based add():

* It is used to add the value based on the index

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex = **new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(40);

ex.add(50);

ex.add(2,100); System.***out***.println(ex);

}

}

**Output:**

[10, 20, 100, 30, 40, 40, 50]

* In this o/p , if we insert one value based on index, after all the index value move to backward

**set();**

* set is a method, it is used to replace the value but index and value order will not change

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex = **new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(40);

ex.add(50);

ex.set(2,100); System.***out***.println(ex);

}

}

## Output:

[10, 20, 100, 40, 40, 50]

## contains():

* contains() is a method it is used to check the particular value or object

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex = **new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(40);

ex.add(50);

**boolean** x = ex.contains(30); System.***out***.println(x); **boolean** y = ex.contains(100); System.***out***.println(y);

}

}

**Output:**

true false **clear():**

* clear is a method it is used to clear the all index value

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex = **new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(40);

ex.add(50);

System.***out***.println(ex);// before clearing ex.clear();

System.***out***.println(ex);// after clearing

}

}

Output:

[10, 20, 30, 40, 40, 50] []

## indexof():

* indexof() is a method, it is used to print the position of the list

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex = **new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(40);

ex.add(50);

**int** x = ex.indexOf(30); System.***out***.println(x);

}

}

**Output:**

2

**Lastindexof():**

* It is used to print the position from the last

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex = **new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(40);

ex.add(50);

**int** x = ex.lastIndexOf(40); System.***out***.println(x);

}

}

## Output:

4

## addAll():

* addAll() is a method, it is used to copy from one list to another list

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex = **new** ArrayList<Integer>(); List<Integer> ex1 = **new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(40);

ex.add(50); System.***out***.println(ex);

System.***out***.println(ex1);// before addAll ex1.addAll(ex);

System.***out***.println(ex); System.***out***.println(ex1);// After addAll

}

}

**Output:**

[10, 20, 30, 40, 40, 50] []

[10, 20, 30, 40, 40, 50]

[10, 20, 30, 40, 40, 50]

## removeAll():

* removeAll() is a method , it is used to compare the both list and remove all the list1 values in list 2

## (i.e)

list2=list2-list1

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex = **new** ArrayList<Integer>(); List<Integer> ex1 = **new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(50); ex1.addAll(ex); ex.add(100);

ex.add(200);

ex.add(300); ex1.add(1000); ex1.add(50); ex1.add(2000); ex1.add(3000); System.***out***.println(ex); System.***out***.println(ex1); ex1.removeAll(ex); System.***out***.println(ex1);

}

}

## Output:

[10, 20, 30, 40, 50, 100, 200, 300]

[10, 20, 30, 40, 50, 1000, 50, 2000, 3000]

[1000, 2000, 3000]

* If we go for removeAll method, here ex1.removeAll(ex), ex1 compare to ex and remove all ex values in the ex1.

## retainAll():

* retainAll() is a method, it is used to compare both list and print the common values

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex = **new** ArrayList<Integer>(); List<Integer> ex1 = **new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(50); ex1.addAll(ex); ex.add(100);

ex.add(200);

ex.add(300); ex1.add(1000); ex1.add(2000); ex1.add(3000); System.***out***.println(ex); System.***out***.println(ex1); ex1.retainAll(ex); System.***out***.println(ex1);

}

}

## Output:

[10, 20, 30, 40, 50, 100, 200, 300]

[10, 20, 30, 40, 50, 1000, 2000, 3000]

[10, 20, 30, 40, 50]

## LinkedList:

**systax:**

List<Integer> ex = new LinkedList<Integer>();

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex = **new** LinkedList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(50); System.***out***.println(ex);

}

}

## Output:

[10, 20, 30, 40, 50]

* it will also print in insertion order.

**Vector:**

**syntax:**

List<Integer> ex = new Vector<Integer>();

## Example Program:

**public class** ArList {

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

List<Integer> ex = **new** Vector<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(50); System.***out***.println(ex);

}

}

## Output:

[10, 20, 30, 40, 50]

* It will also print the same insertion order.
* in all the arraylist methods, will also support in LinkedList and Vector

## ArrayList: Worst case

* In ArrayList deletion and insertion is a worst one because if we delete/insert one index value after all the index move to forward/backward.
* It makes performance issue.

## ArrayList: Best case

* In Arraylist retrieve/searching is a best one
* For ex we have 100 index is there, if we going to print 60th value, we can easily search

## LinkedList: Best case

* Insertion and deletion is a best one because
* Here all values based on the seperate nodes. so, here we can easily delete/insert one value(i.e) if we delete one value, the next node will join to the previous one

## LinkedList: Worst case

* Searching/retrieving is a worst
* For ex, if we have 100 nodes, we have to print 90th node value, it will pass to all the previous nodes and comes to first and then it will print.
* It’s makes performance issue

## Difference between ArrayList and Vector: ArrayList:

* Asynchronize
* It is not a thread safe

## Vector:

* Synchronize
* Thread safe

Here,

Synchronize one by one(thread safe) Asynchronizeparalally(not thread safe)

**Example:** ticket booking,

If one ticket is having,10 people is booking at a same time, what happen , the one person only booked the ticket. because its a synchronize process. it allows one by one.

List<Integer> ex = new ArrayList<Integer>(); List<Integer> ex = new LinkedList<Integer>(); List<Integer> ex = new Vector<Integer>(); **here we can write these in different way,**

ArrayList<Integer> ex = new ArrayList<Integer>(); LinkedList<Integer> ex = new LinkedList<Integer>(); Vector<Integer> ex = new Vector<Integer>();

## User defined Array list:

* Here we can use our own data type

**Pojo class :(client old java object/model class/bean class)**

* In class level if we use private , even we can access in another class.
* If we use private in class 2, right click in class 2source  generate getters and setters
* Using this methods we can access in another class
* This method is called pojo class

## Class 1:

**import** java.util.ArrayList;

**import** java.util.List;

**public class** Employee **extends** New {

**public static void** main(String[] args) { List<Employee> emp=**new** ArrayList<Employee>(); Employee E1=**new** Employee();

E1.setId(12); E1.setName("Raman");

E1.setEmail("[Raman123@gmail.com](mailto:vengat123@gmail.com)"); Employee E2=**new** Employee(); E2.setId(13);

E2.setName("mohan"); E2.setEmail("[mohan123@gmail.com](mailto:mohan123@gmail.com)"); Employee E3=**new** Employee(); E3.setId(14);

E3.setName("vel"); E3.setEmail("[vel123@gmail.com"](mailto:vel123@gmail.com)); emp.add(E1);

emp.add(E2);

emp.add(E3);

**for** (Employee x : emp) { System.***out***.println(x.getId()); System.***out***.println(x.getName()); System.***out***.println(x.getEmail());

}

}

}

**Class 2:**

**public class** New {

**private int** id; **private** String name; **private** String email; **public int** getId() {

**return** id;

}

**public void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public void** setName(String name) {

**this**.name = name;

}

**public** String getEmail() {

**return** email;

}

**public void** setEmail(String email) {

**this**.email = email;

}

}

## Output:

12

Raman [Raman123@gmail.com](mailto:vengat123@gmail.com) 13

mohan [mohan123@gmail.com](mailto:mohan123@gmail.com) 14

vel [vel123@gmail.com](mailto:vel123@gmail.com)

**List:**

In the list we have to know these points,

* It is all insertion order
* It allows duplicate value
* It is index based

**Set:**

* It ignore the duplicate value
* It is value based

## Hashset:

* It will print random order

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { Set<Integer> ex = **new** HashSet<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(50);

ex.add(50); System.***out***.println(ex);

}

}

## Output:

[50, 20, 40, 10, 30] // random order and ignore duplicate value

* It will allows one Null value and won't allow duplicate NULL

## LinkedHashset:

* Insertion order

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { Set<Integer> ex = **new** LinkedHashSet<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(50);

ex.add(50); System.***out***.println(ex);

}

}

**Output:**

[10, 20, 30, 40, 50] // insertion order

* It will also allows one Null value and won't allow duplicate NULL

**TreeSet:**

* Ascending order

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { Set<Integer> ex = **new** TreeSet<Integer>(); ex.add(20);

ex.add(10);

ex.add(30);

ex.add(50);

ex.add(40);

ex.add(50); System.***out***.println(ex);

}

}

## Output:

[10, 20, 30, 40, 50]

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { Set<String> ex = **new** TreeSet<String>(); ex.add("Ramesh");

ex.add("babu");

ex.add("Vasu");

ex.add("10000"); System.***out***.println(ex);

}

}

## Output:

[10000, Ramesh, Vasu, babu] Here,

* It will print ascending order
* Ascending order based on the ASCII value

## (i.e)

* 1ASCII value is 49
* RASCII value is 82
* VASCII value is 86
* bASCII value is 98

[49,82,86,98][10000, Ramesh, Vasu, babu] this is a way to print ascending order.

* TreeSet won't allow single Null value

**Set:**

* It is not maintaining any order(i.e)
* Hashset random order
* LinkedHashsetinsertion order
* Treeset ascending order
* It is value based

**remove():**

* remove is a method , it is used to remove particular value

**public class** ArList {

**public static void** main(String[] args) { Set<Integer> ex = **new** TreeSet<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(50);

ex.add(50); ex.remove(40); System.***out***.println(ex);

}

}

## Output:

[10, 20, 30, 50]

* Normal for loop is not work here because it is not index based, it is value based

## Enhanced for loop:

**public class** ArList {

**public static void** main(String[] args) { Set<Integer> ex = **new** TreeSet<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(50);

ex.add(50);

**for**(**int** i:ex){ System.***out***.println(i);

}

}

}

**Output:**

10

20

30

40

50

* All wrapper class default value is Null as well as all class default value is Null

**Null:**

* Null is a undefined/unknown/unassigned value
* Null is won't create any memory
* So Treeset will give exception in compile time if we use Null

## Difference between List and Set:

**List:**

* It is all insertion order
* It allows duplicate value
* It is index based

## Set:

* It is not maintaining any order(i.e)

Hashset random order LinkedHashsetinsertion order Treeset ascending order

* It is value based
* It ignores duplicate value

we can copy the values from List to set as well as set to list

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex=**new** ArrayList(); Set<Integer> ex1 = **new** TreeSet<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(50);

ex.add(50);

ex.add(10); ex1.addAll(ex); System.***out***.println(ex); System.***out***.println(ex1);

}}

## Output:

[10, 20, 30, 40, 50, 50, 10]

[10, 20, 30, 40, 50]

Here, set ignore the duplicate value

* we can find the duplicate count using size() method

## Example Program:

**public class** ArList {

**public static void** main(String[] args) { List<Integer> ex = **new** ArrayList();

Set<Integer> ex1 = **new** TreeSet<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40);

ex.add(50);

ex.add(50);

ex.add(10); ex1.addAll(ex); System.out.println(ex); System.out.println(ex1);

**int** i = ex.size() ex1.size(); System.out.println(i);

}

}

## Output:

[10, 20, 30, 40, 50, 50, 10]

[10, 20, 30, 40, 50]

2

Here 2 duplicate value is there

## Map:

* It is key and value pair
* Here key+value is a one entry
* Key ignore the duplicate value and value allow the duplicate

## Hashmap:

* It is a random order(based on key)

## Example Program:

**public class** ArList {

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

Map<Integer, String> ex = **new** HashMap<Integer, String>(); ex.put(10, "Java");

ex.put(20, "Java");

ex.put(30, "sql");

ex.put(40, ".net");

ex.put(50, "sales");

ex.put(50, "fire"); System.***out***.println(ex);

}

}

**Output:**

{50=fire, 20=Java, 40=.net, 10=Java, 30=sql}

* If duplicate key is there, it takes the last one
* Key will allows the only one Null
* Value allow the duplicate null

**Linked Hashmap:**

* Insertion order( based on key)
* Key will allows the only one Null
* Value allow the duplicate null

Map<Integer, String> ex = new LinkedHashMap<Integer, String>()

## TreeMap:

* Ascending order(based on key)
* Key won't allow Null(even single null)
* Value allow the duplicate null

Map<Integer, String> ex = new TreeMap<Integer, String>()

## Hashtable:

* Random order
* Both key and values are ignore the Null

Map<Integer, String> ex = new Hashtable<Integer, String>()

## concurrent hashmap:

* Random order
* Both key and values are ignore the Null

Map<Integer, String> ex = new ConcurrentHashMap<Integer, String>()

**Difference between HashMap and HashTable:**

**HashMap:**

* Key allows single null
* Asynchronies(not thread safe)

**Hashtable:**

* Key and value won't allow null
* Synchronize(thread safe)

## Some Methods:

**get():**

* It is a method, it is used to print the value based on key

## Example Program:

**public class** ArList {

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

Map<Integer, String> ex = **new** HashMap<Integer, String>(); ex.put(10, "Java");

ex.put(20, "Java");

ex.put(30, "sql");

ex.put(40, ".net");

ex.put(50, "sales");

ex.put(50, "fire"); String s=ex.get(40); System.***out***.println(s);

}

}

**Output:**

.net

**keyset():**

* It is a method, it is used to seperate the key

## Example Program:

**public class** ArList {

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

Map<Integer, String> ex = **new** HashMap<Integer, String>(); ex.put(10, "Java");

ex.put(20, "Java");

ex.put(30, "sql");

ex.put(40, ".net");

ex.put(50, "sales");

ex.put(50, "fire"); Set<Integer> s = ex.keySet(); System.***out***.println(s);

}

}

## Output:

[50, 20, 40, 10, 30]

## Value():

* It is a method, it is used to seperate the value

## Example Program:

**public class** ArList {

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

Map<Integer, String> ex = **new** HashMap<Integer, String>(); ex.put(10, "Java");

ex.put(20, "Java");

ex.put(30, "sql");

ex.put(40, ".net");

ex.put(50, "sales");

ex.put(50, "fire"); Collection<String> s = ex.values(); System.***out***.println(s);

}

}

**Output:**

[fire, Java, .net, Java, sql]

**entryset():**

* It is used to iterate the map

## Example Program:

**public class** ArList {

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

Map<Integer, String> ex = **new** HashMap<Integer, String>(); ex.put(10, "Java");

ex.put(20, "Java");

ex.put(30, "sql");

ex.put(40, ".net");

ex.put(50, "sales");

ex.put(50, "fire");

Set<Entry<Integer, String>> s = ex.entrySet();

**for**(Entry<Integer, String> x:s){ System.***out***.println(x);

}

}

}

## Output:

50=fire 20=Java 40=.net 10=Java 30=sql

* We can print key and value seperately

## Example Program:

**public class** ArList {

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

Map<Integer, String> ex = **new** HashMap<Integer, String>(); ex.put(10, "Java");

ex.put(20, "Java");

ex.put(30, "sql");

ex.put(40, ".net");

ex.put(50, "sales");

ex.put(50, "fire");

Set<Entry<Integer, String>> s = ex.entrySet();

**for**(Entry<Integer, String> x:s){ System.***out***.println(x.getKey()); System.***out***.println(x.getValue());

}

}

}

**Output:**

50

fire 20

Java 40

.net 10

Java 30

Sql

## EXCEPTION:

* Exception is like a error, the program will terminated that line itself

## Example Program:

**public class** Exception {

**public static void** main(String[] args) { System.***out***.println("Start"); System.***out***.println("1"); System.***out***.println("2"); System.***out***.println("3"); System.***out***.println(10/0); System.***out***.println("4"); System.***out***.println("5"); System.***out***.println("End");

}

}

## Output:

Start 1

2

3

Exception in thread "main" java.lang.ArithmeticException: / by zero at org.exception.Exception.main(Exception.java:9)

* This is exception, if we getting error in run time , the program will be terminated from that line
* Here, java:9 is 9th line only we getting exception

**Throwable:**

* Super class of exception  throwable
* Exception
* Error

**Exception:**

1. Unchecked exception(Run time exception)
2. Checked exception(Compile time exception)

## Unchecked exception:

1. ArithmaticException
2. NullPointException
3. InputMismatchException
4. ArrayIndexOutOfBoundExcepion
5. StringIndexOutOfBoundExcepion
6. IndexOutOfBoundExcepion
7. NumberFormatException

## Checked exception:

1. IOException
2. SQLException
3. FileNotFoundException
4. ClassNotFoundException

## ArithmaticException:

* + If we are trying to give any number divided by zero, we get Arithmatic exception.

## Example Program:

**public class** Exception {

**public static void** main(String[] args) { System.***out***.println("Start"); System.***out***.println("1"); System.***out***.println("2"); System.***out***.println("3"); System.***out***.println(10/0); System.***out***.println("4"); System.***out***.println("5"); System.***out***.println("End");

}

}

## Output:

Start 1

2

3

Exception in thread "main" java.lang.ArithmeticException: / by zero at org.exception.Exception.main(Exception.java:9)

1. **NullPointException:**
   * If we give Null in the string, it will throw the Null point exception. Because default value of string is Null.

## Example Program:

**public class** Exception {

**public static void** main(String[] args) { String s= **null**; System.***out***.println(s.length());

}

}

**Output:**

Exception in thread "main" java.lang.NullPointerException at org.exception.Exception.main(Exception.java:6)

## InputMismatchException:

* + If we getting input from the user, the user need to give integer input but the user trying to input string value , at this this we get input mismatch exception

## Example Program:

**public class** Exception {

**public static void** main(String[] args) { Scanner sc=**new** Scanner(System.***in***); System.***out***.println("PLs enter value"); **int** i=sc.nextInt(); System.***out***.println(i);

}

}

## Output:

PLs enter value hai

Exception in thread "main" java.util.InputMismatchException at java.util.Scanner.throwFor(Unknown Source)

at java.util.Scanner.next(Unknown Source)

at java.util.Scanner.nextInt(Unknown Source) at java.util.Scanner.nextInt(Unknown Source)

at org.exception.Exception.main(Exception.java:9)

1. **ArrayIndexOutOfBoundExcepion:**
   * In particular array, the index value is not available it will throw Array index of bound exception.

## Example Program:

**public class** Exception {

**public static void** main(String[] args) { **int** num[]=**new int**[4]; System.***out***.println(num[5]);

}

}

**Output:**

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5 at org.exception.Exception.main(Exception.java:8)

1. **StringIndexOutOfBoundExcepion:**
   * In particular String, the index value is not available it will throw String index Out of bound exception.

## Example Program:

**public class** Exception {

**public static void** main(String[] args) { String s="Java";

**char** c = s.charAt(10); System.***out***.println(c);

}

}

**Output:**

Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: 10

at java.lang.String.charAt(Unknown Source)

at org.exception.Exception.main(Exception.java:8)

1. **IndexOutOfBoundExcepion:**
   * In a list, the index value is not available it will throw index out of bound exception.

## Example Program:

**public class** Exception {

**public static void** main(String[] args) { List<Integer> ex = **new** ArrayList<Integer>(); ex.add(10);

ex.add(20);

ex.add(30);

ex.add(40); System.***out***.println(ex.get(3)); System.***out***.println(ex.get(10));

}

}

**Output:**

40

Exception in thread "main" java.lang.IndexOutOfBoundsException: Index: 10, Size: 4 at java.util.ArrayList.rangeCheck(Unknown Source)

at java.util.ArrayList.get(Unknown Source)

at org.exception.Exception.main(Exception.java:16)

1. **NumberFormatException**:
   * if we give numbers in the string, we can convert the data type into integer. but if we give num and char combination in the string, we can't convert to integer.
   * if we trying to convert, it will throw number format exception

## Example Program:

**public class** Exception {

**public static void** main(String[] args) { String s="1234"; System.***out***.println(s+5);// string +5 **int** i = Integer.*parseInt*(s); System.***out***.println(i+5);// Integer +5

String s1="123Abc45"; **int** j = Integer.*parseInt*(s1); System.***out***.println(j+5);

}

}

## Output:

12345

1239

Exception in thread "main" java.lang.NumberFormatException: For input string: "123Abc45" at java.lang.NumberFormatException.forInputString(Unknown Source)

at java.lang.Integer.parseInt(Unknown Source) at java.lang.Integer.parseInt(Unknown Source)

at org.exception.Exception.main(Exception.java:13)

## Exception Handling:

1. Try
2. Catch
3. Finally
4. Throw
5. Throws

## Errors:

1. Network error
2. JVM crack
3. out of memory
4. stack overflow

**Try and catch:**

* + If we get exception, try will throw the exception and catch will catch the exception

## Example Program:

**public class** Exception {

**public static void** main(String[] args) { System.***out***.println("start"); System.***out***.println("1"); System.***out***.println("2"); System.***out***.println("3");

## try {

System.***out***.println(10/0);}

**catch**(ArithmeticException e){ System.***out***.println("dont/by zero");

}

System.***out***.println("4"); System.***out***.println("5"); System.***out***.println("end");

}

}

## Output:

start 1

2

3

dont/by zero 4

5

end

* + Here we can enter the same exception or super class of the exception

## (i.e)

* + Super class of the all unexpected exception is run time exception/exception
  + Super class of exception throwable
  + So we can use runtime exception/exception/throwable instead of the particular exception

**Example Program:** we can use throwable try {

System.out.println(10/0);} catch(Throwable e){ System.out.println("dont/by zero");

}

## instead of

try {

System.out.println(10/0);} catch(ArithmeticException e){

System.out.println("dont/by zero");

}

**Finally:**

* + finally will execute always whether the exception through or not
  + We can give the combination like trycatchfinally, we can't reverse/interchange
  + If we give tryfinally, again it will show the exception

## Example Program:

**public class** Exception {

**public static void** main(String[] args) { System.***out***.println("start"); System.***out***.println("1"); System.***out***.println("2"); System.***out***.println("3");

## try {

System.***out***.println(10/0);}

**catch**(ArithmeticException e){ System.***out***.println("dont/by zero");

}**finally**{

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

}

System.***out***.println("4"); System.***out***.println("5"); System.***out***.println("end");}}

## Output:

start 1

2

3

dont/by zero final

4

5

end

* + Even if exception not through, finally will print

## Example program:

**public class** Exception {

**public static void** main(String[] args) { System.***out***.println("start"); System.***out***.println("1"); System.***out***.println("2"); System.***out***.println("3");

## try {

System.***out***.println(10/0);}

**catch**(NullPointerException e){ System.***out***.println("dont/by zero");

}**finally**{

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

}

System.***out***.println("4"); System.***out***.println("5"); System.***out***.println("end");

}

}

**Output:**

start 1

2

3

final

Exception in thread "main" java.lang.ArithmeticException: / by zero at org.exception.Exception.main(Exception.java:11)

* + In between try ,catch and finally, we won't write any logics
  + In one try block we can use n-number of catch blocks but we can't repeat the same exception
  + In one try block we can handle only one exception

## Example Program:

**public class** Exception {

**public static void** main(String[] args) { System.***out***.println("start"); System.***out***.println("1"); System.***out***.println("2"); System.***out***.println("3");

## try {

System.***out***.println(10/0);}

**catch**(NullPointerException e){ System.***out***.println("null point");

}

**catch**(ArithmeticException e) { System.***out***.println("dont/by zero");

}

**finally**{

}

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

System.***out***.println("4"); System.***out***.println("5"); System.***out***.println("end");

}

}

**Output:**

start

1

2

3

dont/by zero final

4

5

end

* + In a try block, one catch we can use same excepion and another catch we use throwable exception
  + At this time, it will through the first one if it is match, will print. if it is not correct will throw the second
  + throwable is the super class of all exception
  + In more than one catch block, we can use like sub class and super class combination. But we can't use reverse

sub class ArithmaticException, NullPointException,.....

super classThrowable/Exception

* + if we give Super class and sub class combination, it will give compilation error

## Example Program:

**public class** Exception {

**public static void** main(String[] args) { System.***out***.println("start"); System.***out***.println("1"); System.***out***.println("2"); System.***out***.println("3");

## try {

System.***out***.println(10/0);}

**catch**(NullPointerException e){ System.***out***.println("null point");

}

**catch**(Throwable e) { System.***out***.println("dont/by zero");

}

**finally**{

}

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

System.***out***.println("4"); System.***out***.println("5"); System.***out***.println("end");

}

}

**Output:**

start

1

2

3

dont/by zero final

4

5

end

## Example Program:

try {

System.out.println(10/0);} catch(Throwable e) { System.out.println("dont/by zero");

}

catch(NullPointerException e){ System.out.println("null point");

}

finally{ System.out.println("final");

}

* + If we give like above, will get compile time exception/error because we can't reverse
  + In one try block, we can write only one finally block

## Inner try:

* + If we use inner try, it will print inner catch, inner finally and outer finally.
  + But one try block handle one exception only, even if we use inner try also
  + If main try have no exception, it will through inner try. in that inner try if catch exception is wrong, it will goes and print outer finally

## Example Program:

**public class** Exception {

**public static void** main(String[] args) { System.***out***.println("start"); System.***out***.println("1"); System.***out***.println("2"); System.***out***.println("3");

## try {

System.***out***.println(10/0);

## try {

String s=**null**; System.***out***.println(s.charAt(0));

}**catch**(NullPointerException e){ System.***out***.println("inner catch");

}**finally**{

System.***out***.println("inner finally");

}

}

**catch**(ArithmeticException e) { System.***out***.println("dont/by zero");

}

**finally**{

}

System.***out***.println("outer finaly");

System.***out***.println("4"); System.***out***.println("5"); System.***out***.println("end");

}

}

## Output:

start 1

2

3

dont/by zero outer finaly 4

5

end

## Example Program:

**public class** Exception {

**public static void** main(String[] args) { System.***out***.println("start"); System.***out***.println("1"); System.***out***.println("2"); System.***out***.println("3");

## try {

System.***out***.println(10/2);

## try {

String s=**null**; System.***out***.println(s.charAt(0));

}**catch**(NullPointerException e){ System.***out***.println("inner catch");

}**finally**{

System.***out***.println("inner finally");

}

}

**catch**(ArithmeticException e) { System.***out***.println("dont/by zero");

}

**finally**{

System.***out***.println("outer finaly");

}

System.***out***.println("4"); System.***out***.println("5"); System.***out***.println("end");

}

}

**Output:**

start 1

2

3

5

inner catch inner finally outer finaly 4

5

end

## Throw and Throws:

**Throw:**

* + Throw is a keyword, we can through any exception inside the method
  + At a time we can throw only one exception

## Throws:

* + Throws is a keyword, it is used to declare the exception(in method level)
  + At a time we can declare more than one exception

## Example Program:

**public class** Exception {

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

InterruptedException, ArithmeticException,IOException{

*info*();

}

**private static void** info() **throws** IOException { System.out.println("hello");

**throw new** IOException();

}

* + If we try to throws the compile time exception in any method, we must handle it in compile time

## Example Program:

**public class** Exception {

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

InterruptedException, ArithmeticException,IOException{

*info*();

}

**private static void** info() **throws** IOException { System.out.println("hello");

**throw new** IOException();

}

**Constructor:**

* + Class name and constructor name must be same.
  + It doesn’t have any return type.
  + We don’t want to call constructor which is creating object itself.
  + It will automatically invoke the default constructor.
  + It will support in method overloading but won’t support in method overriding

## Example :

**public class** Const1 {

**public** Const1() {

System.***out***.println("i am in constructor");

}

**public** Const1(**int** a) {

System.***out***.println("1 argument int");

}

**private void** Const1(**int** i) { System.***out***.println(i);

}

**public static void** main(String[] args) { Const1 c = **new** Const1(); Const1 c1 = **new** Const1(10);

c.Const1(10);

}

}

Here,

public Const1() Non- Argument based constructor public Const1(int a)  Argument based constructor

* + In argument base constructor we have to pass the argument in object Const1 c1=new Const1(10);
  + In non-argument base constructor we don’t want to pass any argument Const1 c=new Const1();

**Output:**

i am in constructor

1 argument int

10

## This:

* It is a keyword.
* If we use ‘this’ argument , we can pass without object
* This argument we must use only in first statement

## Example 1:

public class Const1 { public Const1() {

this(10);

System.out.println("i am in constructor");

}

public Const1(int a){

System.out.println("1 argument int");

}

public static void main(String[] args) {

Const1 c=new Const1();

}

}

## sOutput:

1 argument int

i am in constructor

## EXAMPLE 2:

**public class sample { public sample(){**

## this(10);

**System.out.println("i am in constructor");**

## }

**public sample(int i) {**

## this(23.23f);

**System.out.println("am integer constructor");**

## }

**public sample(float f) {**

## System.out.println("am float constructor");

**}**

## public static void main(String[] args) {

**sample s=new sample();**

## }

**}**

**Output:**

am float constructor am integer constructor i am in constructor

* If we trying to declare any variable without value in class level. It will print the default value

**public class** sample {

**int** id;

* Inside the method, we must initialize the local variable otherwise we get compile time error

**public class** sample {

**int** id=10;

**private void** num() {

**int** id=100; System.**out**.println(id);

}

## Output: 100

* If same variable name in local level and class level, it will give first preference is local level

## This():

**Class level:**

* + It will refer the class level variable value

## Example:

**public class** sample {

**int** id=100;

**private void** num() {

System.**out**.println(id);

}

**public static void** main(String[] args) { sample s=**new** sample(); s.num();

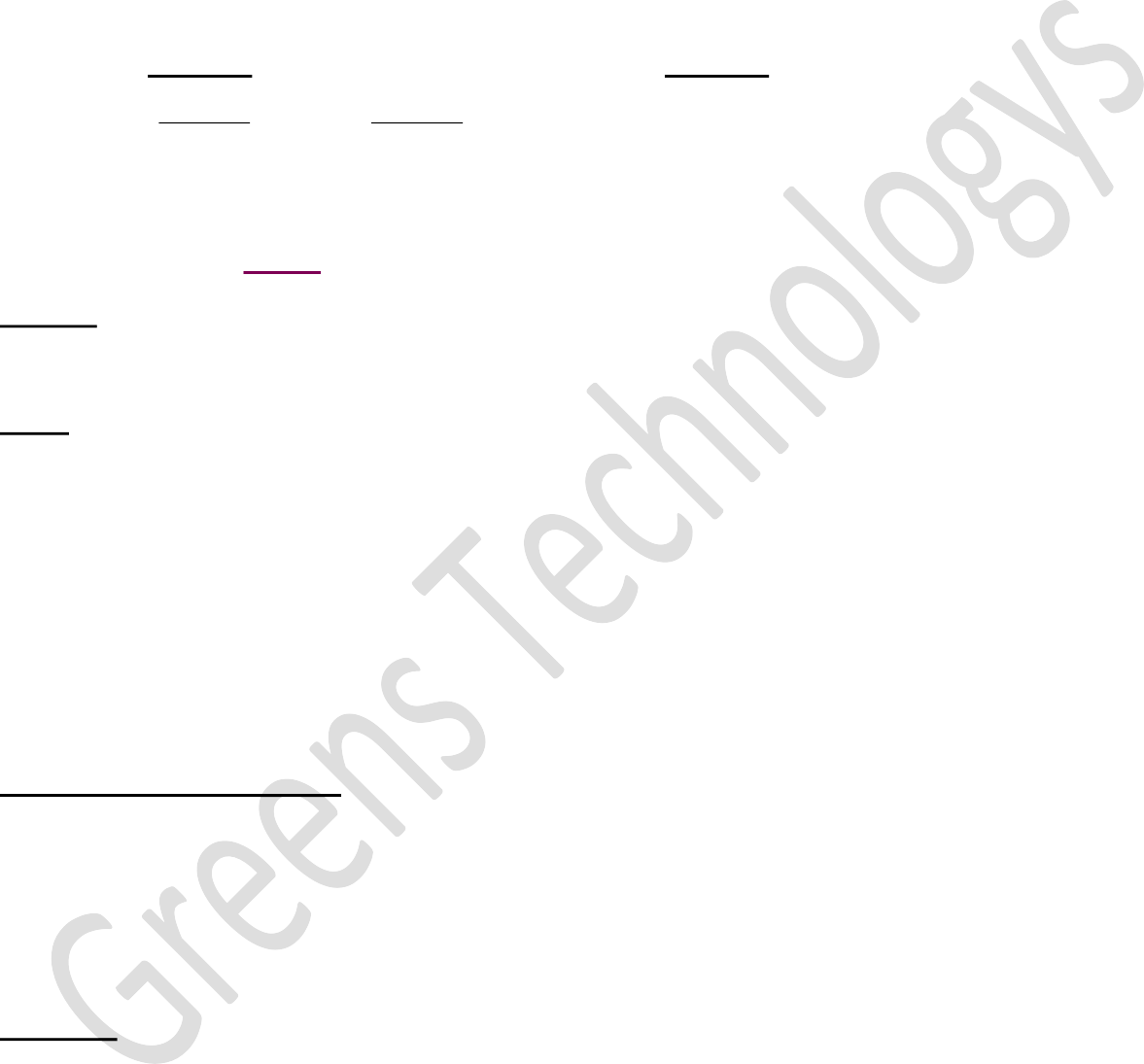
}}

**Output:**

100

**Method level access of this();**

**Super :**

* It will refer the parent class level variable value

## Class 1: Class 2:

**public class** sample **extends** Const1 {

**int** id=100;

**private void** num() {

**int** id=10; System.***out***.println(**super**.id);

## }Output:

**publ**3**i**0**c static void** main(String[] args) {

# sample s=**new** sample();

s**F**.n**in**um**al:**();

**public class** Const1 {

**int** id =30;

# }

}  It’s a keyword.

} If we use final before the variable, we can’t overwrite.

* If we trying to overwrite it show compile time error.
* As well as if we use final before the method/class, we can’t extend.
* We can use final in three ways,
  + Variable level
  + Method level
  + Class level

## Variable level using final:

* We can overwrite the value of variable on variable declaration.
* If we final, the value of variable can’t be overwrite.

## Example:

**Without using final: With using final**

**public class** sample {

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

**int** a=10; a=20;

# System.***out***.println(a);

}

# }

Output:

**public class** sample {

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

**final int** a=10; a=20;

# System.***out***.println(a);

}

# }

output:

## Method level using final(): Class 1:

**public class** sample {

**public final void** example() {

}

}

## Class 2:

**public class** Const1 **extends** sample {

**public void** example(){ // we get compile time error here, because if we use final in method level(parent class), we can't use again

}

}

* We get compile time error, because we using final in method level
* We can’t over ride while using final

## Class level :

* + If we are using final in class level , we can’t extends to anywhere

**public final class** sample {

**}**

**JAVA INTERVIEW QUESTIONS:**

### What is the difference between JDK, JRE and JVM?

* + **JDK**: Java Development Kit
    - JDK is a software, it contains JRE and JVM
    - If we run any applications, JDK have to installed
  + **JRE**: Java Runtime Environment
    - It is a predefined .class files(i.e.) library files
  + **JVM**: Java Virtual Machine
    - It is mainly used to allocate the memory and compiling

1. ***What is meant by class, method and objects?***

## CLASS:

* + - Class is nothing but collection of methods or collection of objects.

## METHOD:

* + - A set of action to be performed

## OBJECT:

* + - Run time memory allocation
    - Using object we can call the any methods

### What is meant by Encapsulation?

* + Structure of creating folder is called encapsulation

### What is the use of inheritance and its types?

* + Inheritance is used to reduce the object memory
  + We can access one class property into another class using 'extend' keyword is called inheritance
  + Reusable purpose
  + It has 5 types
  1. Single Inheritance
  2. Multilevel Inheritance
  3. Multiple Inheritance
  4. Hybrid Inheritance
  5. Hierarchical Inheritance

## Single Inheritance:

* + One parent class is directly support into one child class using extend keyword

## Multilevel Inheritance:

* + More than one parent class support into one child class using extends keyword

## Multiple Inheritance:

* + More than one parent class parallely support into one child class but it won't support in java because
    - Priority problem
    - Compilation error/syntax error

(i.e.) if both parent class having same method name it gets priority problem so it doesn't work in java

* + but multiple inheritance support in java using interface

## Hybrid inheritance:

* + It’s a combination of single and multiple inheritance

## Hierarchical Inheritance:

* + One parent class directly support into more than one child class

### What is meant by java? why we go for java??

* 1. Java is a simple programing language
  2. Writing, compilation and debugging a program is very easy in java
  3. It helps to create reusable code
  4. Java has more features,
     1. platform independent
     2. open source
     3. multithreading
     4. more secure
     5. portable

## Platform independent:

* During the compilation the java program converts into byte code
* Using byte code we can run the application to any platform such as windows, mac, Linux.Etc.

## Open source:

* A program in which source code is available to the general public for use and/or modification from its original design at free of cost is called open source

## Multithreading:

* Java supports multithreading
* It enables a program to perform several task simultaneously

## More secure:

* It provides the virtual firewall between the application and the computer
* So it’s doesn't grant unauthorized access

## Portable:

* + "Write once Run anywhere"
  + Java code written in one machine can run on another machine

### What is meant by garbage collection??

* + Automatic De- allocation of objects is called garbage collection.

### What is the difference between while and do while?

* + While:
    - Entry level condition checking
  + Do. While:
    - Exit level condition checking
    - Even if condition fails, it print one time

### What is the difference between break and continue??

* + Break:
    - It exit from the current loop
  + Continue:
    - It will skip the particular iteration

### What is the use of polymorphism??

* + Polymany
  + Morphismforms
  + Taking more than one forms is called polymorphism
  + One task is completed by many ways

1. ***What is the difference between method overloading and method over hiding?***

## Method overloading :( static binding/compile time polymorphism)

* + In a same class method name is same and the argument is different is called method overloading
  + The argument is depends on
* data types
* data types count
* data type order

## Method overriding :( dynamic binding/run time polymorphism)

* + In a different class , the method name should be same and argument name should be same is called overriding

### What is meant by abstraction??

* + Hiding the implementation part is called abstraction
  + It has 2 types,

1. Partially abstraction(abstract class)
2. Fully abstraction(interface)
3. ***What is the difference between abstract class and interface??***

## Abstract class:

* It is partially abstraction
* It support both abstract method and non-abstract method
* Its using "extends" keyword
* Here "public abstract" have to mention
* We can use whatever access specifier we want

## Interface:

* It is fully abstraction
* It support only abstract method
* It’s using "implement" keyword
* "Public Abstract" is default. no need to mention
* Here only use public( access specifier)

### What is the super class of java??

* + Object

### What is default package of java??

* + java.lang

### Define heap memory???

* + The objects are stored in the heap memory

### What is meant by wrapper class and uses??

* + Classes of data types is called wrapper class
  + It is used to convert any data types into objects

### What is meant by string????

* + Collections of character or word enclosed with double quotes is called string

1. ***What is the difference between literal and non-literal string???***

## Literal String:

* + Its stored inside the heap memory(string pool or string constant).
  + It share the memory if same value (duplicate value)

## Non literal string:

* + It’s stored in the heap memory.
  + its create a new memory every time even if its duplicate value(same value)

### Define oops concept,

* Object Oriented Programing Structure
* OOPS is a method of implementation in which programs are organized as collection of objects, class and methods
* Oops principles are

1. Class
2. Method
3. Object
4. Abstraction
5. Encapsulation
6. Inheritance
7. Polymorphism
8. ***What is meant by interface??***

## Interface:

* + It will support only abstract method, won't support non abstract method
  + In interface "public abstract" is default. we no need to mention
  + It using implements keywords
  + It is fully abstraction

1. ***Define abstraction:***

## ABSTRACTION:

* Hiding the implementation part is called abstraction
* It has 2 types,

1. Partially abstraction(abstract class)
2. Fully abstraction(interface)

## 1. Partially Abstraction(Abstract class):

### variable??

* Long-8

### What is the size of short variable??

* Short-2

### What is the size of int variable??

* Int-4

1. ***Define access specifier & its types??***

## ACCESS SPECIFIER:

* 1. Public
  2. Protected
  3. Default
  4. Private

## Public:

* + It is global level access( same package + different package)

## Private:

* + It is a class level access

## Default :

* + Package level access
  + Without access specifier within the package we can access

EX,

* public static public acces specifier( need to mention public)
* private static private access specifier( need to mention)
* static default access secifier( need not)

## Protected:

* + Inside package + outside Package( Extends)
  + we can access inside package and outside package using extend keyword

1. ***What is meant by polymprphism???***

## POLYMORPHISM:

* + Poly-many
  + Morphism-forms
  + Taking more than one forms is called polymorphism.
  + One task is completed by many ways
  + It has 2 types,
  1. Method overloading (static binding/compile time polymorphism)
  2. Method overriding (dynamic binding/run time polymorphism)

## Method overloading:

* + In a same class method name is same and the argument is different is called method overloading
  + The argument is depends on
    - Data types
    - Data types count
    - Data type order

## Method overriding:

* + In a different class , the method name should be same and argument name should be same is called overriding

### What is array??

* Collection of similar data
* The values are stored based on index
* The index will start from 0 to n-1

1. ***What is difference between hashmap and hashtable?***

## HashMap:

* Key allows single null
* Asynchronies(not thread safe)

## Hashtable:

* Key and value won't allow null
* Synchronize(thread safe)

1. ***What is the difference between linked list and array list?***

## ArrayList: Worst case

* In ArrayList deletion and insertion is a worst one because if we delete/insert one index value after the entire index move to forward/backward.
* It makes performance issue.

## ArrayList: Best case

* In arraylist retrieve/searching is a best one
* For ex we have 100 index is there, if we going to print 60th value, we can easily search

## LinkedList: Best case

* + Insertion and deletion is a best one because
  + Here all values based on the separate nodes. So, here we can easily delete/insert one value(i.e.) if we delete one value, the next node will join to the previous one

## LinkedList: Worst case

* + Searching/retrieving is a worst
  + For ex, if we have 100 nodes, we have to print 90th node value, it’s communicate all the previous nodes and comes to first and then it will print.
  + It’s makes performance issue

### Can we able to write any logic in interface?

* + No. In class only we can write

1. ***What is the difference between mutable and immutable string?***

## Immutable string:

* + We can store more duplicate value in same memory
  + We can't change the value in memory
  + In concord nation, we have to create new memory

## Mutable string:

* + We can't store duplicate value in same memory
  + We can change the value in memory
  + In concord nation, its takes same memory

1. ***What is the difference between thread safe and non- thread safe?***

## Thread safe:

* + synchronize we can access one by one
  + ex. ticket booking

## Non-thread safe:

* + Asynchronies parallaly we can access

### List will allow duplicate value?

* + Yes. it will allow

1. ***What are all coding standards available in java? Where we use it?***

## Coding standards:

1. Pascal notation
2. Camel notation
   * Pascal notation: Each word of first letter should be in capital
   * Camel notation : First word should be small after every word of the first letter should be capital
   * Pascal notation:
   1. Project name
   2. Class name
   * Camel notation :
3. Method name
4. Variable name

### What is NullPointException?

* + If we give Null in the string, it will throw the Null point exception. Because default value of string is Null.

1. ***What is meant by List and set?***

## List:

* + It is all insertion order
  + It allows duplicate value
  + It is index based

## Set:

* + It is not maintaining any order(i.e.)
* Hashset Random order
* LinkedHashsetInsertion order
* Treeset Ascending order
  + It is value based
  + it will not allow duplicate value

### How will you iterate map?

* + Using entrySet() method we can iterate the map

1. ***What is the difference between ArrayList and Vector ?***

## ArrayList:

* + Asynchronies
  + It is not a thread safe

## Vector:

* + Synchronize
  + Thread safe

Here,

Synchronize One by one (thread safe) Asynchronies Paralally(not thread safe)

Ex, ticket booking,

### Define map?

* It is key and value pair
* Here key + value is a one entry
* Key ignore the duplicate value and value allow the duplicate
* It has 5 types
* Hashmap(c)
* Linked hashmap(c)
* Treemap(c)
* Hashtable(c)
* Concurrent hashmap(C)

### Define generics?

* + It will support particular datatypes or object only
  + It is a one of the features of JDK 1.5
  + In the generics, we can mention only wrapper class
  + <> it is a generic symbol. it is used to define the particular data type
  + If we need integer data type,
  + Syntax:

List<Integer> ex=new ArrayList<Integer>();

1. ***What is the difference between throw and throws?***

## Throw:

* Throw is a keyword, we can through any exception inside the method
* At a time we can throw only one exception

## Throws:

* Throws is a keyword, it is used to declare the exception(in method level)
* At a time we can declare more than one exception

1. ***What is the difference between hashset,linked hashset and treeset***

## Hashset:

* + It will print random order
  + It will allow single Null value but won’t allow duplicate Null

## Linked hashset:

* + It will print insertion order
  + It will allow single Null value but won’t allow duplicate Null

## Treeset:

* + It will print ascending order
  + Treeset won’t allow Null value

### How many null values allow in treeset?

* + Treeset won’t allow Null value

### What is the super class of all exceptions?

* + Throwable

### What is the difference between equal and double equals?

 =  It is used to assigning the value

 ==  It is used for condition checking

1. ***What is the difference between retain all and remove all?***

## removeAll():

* + removeAll() is a method , it is used to compare the both list and remove all the list1 values in list 2

## (i.e)

list2 = list2-list1 or a = a-b

## retainAll():

* + retainAll() is a method, it is used to compare both list and print the common values

### How to create object for interface and abstract class?

* + We won’t create object for interface and abstract class.

1. ***What are the advantages and dis advantages of arrays?***

## Advantage of array:

* + In a single variable we can store multiple values

## Disadvantage of arrays:

* + It support only similar data types
  + It is a fixed size
  + Memory wastage is high
  + To overcome these we go for collections

1. ***What is the difference between normal class and abstract class***

## Abstract class:

* It will support abstract method and non-abstract method
* We won’t create object for abstract class
* We won’t write any business logic in abstract method

## Class:

* It support only in non-abstract method
* We can create object for class

1. ***Difference between final and finally?***

## Final Finally

1. A final class variable whose value cannot be changed.
2. A final is declared as class level, they cannot be inherited.
3. If final is declared as method level, they cannot be override.
4. It’s a block of statement that definitely executes after the try catch block.
5. If try block fails means, the final block will executes once.

### 51. How to access one class property into another class?

2 ways we can access

1. by using extends keyword(inheritance)
2. By creating object

**CORE JAVA PROGRAMS**

1. **Sum of odd number(1 to 100):**

**public class** SumofOddNum {

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

**int** count = 0;

**for** (**int** i = 1; i <= 100; i++) {

**if** (i % 2 == 1) {

count = count + i;

}

}

System.***out***.println(count);

}

}

### Output:

2500

## Sum of even number(1 to 100):

**public class** SumofEvenNum {

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

**int** count = 0;

**for** (**int** i = 1; i <= 100; i++) {

**if** (i % 2 == 0) {

count = count + i;

}

}

System.***out***.println(count);

}

}

### Output:

2550

## Count of odd number(1 to 100):

**public class** CountOfOddNumber {

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

**int** count = 0;

**for** (**int** i = 1; i <= 100; i++) {

**if** (i % 2 == 1) {

count = count + 1;

}

}

System.***out***.println(count);

}

}

### Output:

50

## Count of even number(1 to 100):

**public class** CountOfEvenNumbers {

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

**int** count = 0;

**for** (**int** i = 1; i <= 100; i++) {

**if** (i % 2 == 0) {

count = count + 1;

}

}

System.***out***.println(count);

}

}

### Output:

50

## Factorial number:

**public class** FactorialNumbers {

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

**int** count = 1;

**for** (**int** i = 1; i <= 8; i++) { count = count \* i;

}

System.***out***.println(count);

}

}

### Output:

40320

## Fibonacci series:

**public class** Fibanacci {

**public static void** main(String[] args) { **int** a = 0, b = 1; System.***out***.println(a); System.***out***.println(b);

**for** (**int** i = 2; i <= 10; i++) {

**int** c = a + b;

### Output:

0

1

1

2

3

5

8

13

21

34

}}}

System.***out***.println(c); a = b;

b = c;

## To find even/odd number:

**public class** EvenoddNumber {

**public static void** main(String[] args) { Scanner e = **new** Scanner(System.***in***); System.***out***.println("Enter a Number"); **int** n = e.nextInt();

**if** (n % 2 == 0) {

System.***out***.println("Even number");

} **else** {

System.***out***.println("Odd number");

}

}

}

### Output:

Enter a Number 121

Odd number

## Swapping numbers using third variable:

**public class** SwappingNumbersWithVar {

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

**int** a, b, c;

Scanner sw = **new** Scanner(System.***in***); System.***out***.println("The numbers are"); a = sw.nextInt();

b = sw.nextInt(); c = a;

a = b; b = c;

System.***out***.println("swapping numbers are"); System.***out***.println(a); System.***out***.println(b);

}

}

### Output:

The numbers are 25

45

swapping numbers are 45

25

## Swapping numbers without using third variable:

**public class** SwappingNumWithoutVar {

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

**int** a, b;

Scanner sw = **new** Scanner(System.***in***); System.***out***.println("The numbers are"); a = sw.nextInt();

b = sw.nextInt(); a = a + b;

b = a - b;

a = a - b;

System.***out***.println("swapping numbers are"); System.***out***.println(a); System.***out***.println(b);

}

}

### Output:

The numbers are 20

40

Swapping numbers are 40

20

## Reverse the number:

**public class** Reversenumber {

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

Scanner rn = **new** Scanner(System.***in***); System.***out***.println("Enter a number"); **int** n = rn.nextInt();

**int** a, i = 0, j = 0;

a = n;

**while** (a > 0) {

i = a % 10;

j = (j \* 10) + i; a = a / 10;

}

System.***out***.println("Reverse number is=" + j);

}

}

### Output:

Enter a number 12345

Reverse number is=54321

## To check palindrome number:

**public class** PolyndromeNumberCheck {

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

**int** n, a, i = 0, j = 0;

Scanner an = **new** Scanner(System.***in***); System.***out***.println("Enter a number"); n = an.nextInt();

a = n;

**while** (a > 0) {

i = a % 10;

j = (j \* 10) + i; a = a / 10;

}

**if** (n == j) {

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

} **else** {

System.***out***.println("Not palindrome Number");

}

}

}

### Output:

Enter a number 141

palindrome

## To print palindrome number(1 to 100):

**public class** PolyndromeNumbers {

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

**for** (**int** n = 1; n <= 100; n++) {

**int** a, i = 0, j = 0; a = n;

**while** (a > 0) {

i = a % 10;

j = (j \* 10) + i; a = a / 10;

}

**if** (n == j) {

System.***out***.println(n);

}

}

}

}

### Output:

1

2

3

4

5

6

7

8

9

11

22

33

44

55

66

77

88

99

## To count palindrome number(1 to 1000):

**public class** palindromeNum {

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

**int** c = 0;

**for** (**int** n = 1; n <= 1000; n++) {

**int** a, i = 0, j = 0; a = n;

**while** (a > 0) {

i = a % 10;

j = (j \* 10) + i; a = a / 10;

}

**if** (n == j) {

c++;

}

}

System.***out***.println(c);

}

}

### Output:

108

## To check Armstrong number:

**public class** ArmstrongNumberCheck {

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

**int** n, a, i = 0, j = 0;

Scanner an = **new** Scanner(System.***in***); System.***out***.println("Enter a number"); n = an.nextInt();

a = n;

**while** (a > 0) {

i = a % 10;

j = j + (i \* i \* i); a = a / 10;

}

**if** (n == j) {

System.***out***.println("Armstrong number");

} **else** {

System.***out***.println("Not armstrong Number");

}

}

}

### Output:

Enter a number 153

Armstrong number

## To print Armstrong number(1 to 1000):

**public class** ArmstrongNumbers {

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

**for** (**int** n = 1; n <= 1000; n++) {

**int** a, i = 0, j = 0; a = n;

**while** (a > 0) {

i = a % 10;

j = j + (i \* i \* i); a = a / 10;

}

**if** (n == j) {

System.***out***.println(n);

}

}

}}

### Output:

1

153

370

371

407

## To count Armstrong number(1 to 1000):

**public class** ArmsrongNumberCount {

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

**int** c = 0;

**for** (**int** n = 1; n <= 1000; n++) {

**int** a, i = 0, j = 0; a = n;

**while** (a > 0) {

i = a % 10;

j = j + (i \* i \* i); a = a / 10;

}

**if** (n == j) {

c++;

}

}

System.***out***.println(c);

}

}

### Output:

5

## Triangle program:

**public class** Triangle {

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

**for** (**int** i = 1; i <= 5; i++) {

**for** (**int** j = 1; j <= i; j++) {

System.***out***.print("\*");

}

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

}

}

}

### Output:

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

## Reverse triangle:

**public class** ReverseTriangle {

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

**for** (**int** i = 1; i <= 5; i++) {

**for** (**int** j = 5; j >= i; j--) {

System.***out***.print("\*");

}

}

### Output:

}

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

}

\*\*\*\*\*

\*\*\*\*

\*\*\*

\*\*

\*

## To check prime number:

**public class** PrimeNumberChecking {

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

**int** n;

Scanner input = **new** Scanner(System.***in***); System.***out***.println("enter the number");

n = input.nextInt();

**int** count = 0;

**for** (**int** i = 2; i <= n / 2; i++) {

**if** (n % i == 0) {

count = 1;

}

}

**if** (count == 0) {

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

} **else** {

System.***out***.println("not prime");

}

}

}

### Output:

enter the number 17

prime

enter the number 21

not prime

## To print prime number(1 to 10):

**public class** PrimeNumber {

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

**int** count;

**for** (**int** i = 1; i <= 10; i++) { count = 0;

**for** (**int** j = 2; j <= i / 2; j++) {

**if** (i % j == 0) {

count++;

}

}

**if** (count == 0) {

System.***out***.println(i);

}

}

}

}

### Output:

1

2

3

5

7

## To count prime number(1 to 100):

x**public class** PrimeNumberCount {

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

**int** count, c = 0;

**for** (**int** i = 1; i <= 100; i++) { count = 0;

**for** (**int** j = 2; j <= i / 2; j++) {

**if** (i % j == 0) {

count++;

}

}

**if** (count == 0) {

c++;

}

}

System.***out***.println(c);

}

}

### Output:

26

## Student grade:

**public class** StudentsGrade {

**public static void** main(String[] args) { Scanner sm = **new** Scanner(System.***in***); System.***out***.println("enter value:"); **int** n = sm.nextInt();

**if** (100 >= n && n >= 90) { System.***out***.println("S grade");

} **else if** (89 >= n && n >= 80) { System.***out***.println("A grade");

} **else if** (79 >= n && n >= 70) { System.***out***.println("B grade");

} **else if** (69 >= n && n >= 60) { System.***out***.println("C grade");

}

**else if** (59 >= n) { System.***out***.println("Fail");

}

}

}

### Output:

enter value:

67

C grade enter value:

55

Fail

## Multiplication table:

**public class** MultiflicationTable {

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

**int** n, j;

Scanner mt = **new** Scanner(System.***in***); System.***out***.println("Enter the Table"); n = mt.nextInt(); System.***out***.println("table upto");

j = mt.nextInt();

**for** (**int** i = 1; i <= j; i++) {

**int** c = n \* i;

System.***out***.println(i + "\*" + n + "=" + c);

}

}

}

### Output:

Enter the Table 6

table upto 10

1\*6=6

2\*6=12

3\*6=18

4\*6=24

5\*6=30

6\*6=36

7\*6=42

8\*6=48

9\*6=54

10\*6=60

## Biggest of 4 numbers:

**public class** BiggestNumberUsingif {

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

**int** a, b, c, d;

Scanner bn = **new** Scanner(System.***in***); System.***out***.println("The four numbers are"); a = bn.nextInt();

b = bn.nextInt(); c = bn.nextInt(); d = bn.nextInt();

**if** (a > b && a > c && a > d) {

System.***out***.println("the biggest number is=" + a);

} **else if** (b > a && b > c && b > d) { System.***out***.println("the biggest number is=" + b);

} **else if** (c > a && c > b && c > d) { System.***out***.println("the biggest number is=" + c);

} **else** {

System.***out***.println("the biggest number is=" + d);

}

}

}

### Output:

The four numbers are 23

45

56

22

The biggest number is=56

## Find vowels and non-vowels count

**public class** VowelsCount {

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

String a = "welcome";

**int** vowels = 0;

**int** nonVowels = 0;

**for** (**int** i = 0; i < a.length(); i++) {

**char** ch = a.charAt(i);

**if** (ch == 'a' || ch == 'A' || ch == 'e' || ch == 'E'

|| ch == 'i' ch == 'u'

|| ch == 'I' || ch == 'o' || ch == 'O' ||

|| ch == 'U') { vowels++;

} **else** { nonVowels++;

}

}

System.***out***.println(vowels); System.***out***.println(nonVowels);

}

}

### Output:

3

4

## Ascending order using array:

**public class** Ascending {

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

{

**int** n, temp;

Scanner s = **new** Scanner(System.***in***);

System.***out***.print("Enter no. of elements you want in array:"); n = s.nextInt();

**int** a[] = **new int**[n]; System.***out***.println("Enter all the numbers:"); **for** (**int** i = 0; i < n; i++)

{

a[i] = s.nextInt();

}

**for** (**int** i = 0; i < n; i++)

{

**for** (**int** j = i + 1; j < n; j++)

{

**if** (a[i] > a[j])

{

temp = a[i]; a[i] = a[j]; a[j] = temp;

}

}

}

System.***out***.print("Ascending Order:");

**for** (**int** i = 0; i < n - 1; i++)

{

System.***out***.print(a[i] + ",");

}

System.***out***.print(a[n - 1]);

} }

***Output:***

## Enter no. of elements you want in array:6 Enter all the numbers:

**10**

## 20

**100**

## 40

**200**

## 60

**Ascending Order: 10,20,40,60,100,200**

## 27. Descending order using array:

**public class** DescendingOrder {

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

**int** n, temp;

Scanner s = **new** Scanner(System.***in***); System.***out***.print("Enter no. of elements you want in

array:");

n = s.nextInt();

**int** a[] = **new int**[n]; System.***out***.println("Enter all the elements:"); **for** (**int** i = 0; i < n; i++) {

a[i] = s.nextInt();

}

**for** (**int** i = 0; i < n; i++) {

**for** (**int** j = i + 1; j < n; j++) {

**if** (a[i] > a[j]) {

temp = a[i]; a[i] = a[j]; a[j] = temp;

}

}

}

System.***out***.print("Descending Order:");

**for** (**int** i = n - 1; i > 0; i--) {

System.***out***.print(a[i] + ",");

}

System.***out***.print(a[0]);

}

}

***Output:***

## Enter no. of elements you want in array:6 Enter all the elements:

**10**

## 20

**100**

## 40

**200**

## 60

**Descending Order:200,100,60,40,20,10**

## Second minimum number:

**public class** SecondMinimumNumber {

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

**int** a[]={-12,45,-23,64,-100,24};

**for**(**int** i=0;i<a.length;i++){

**for**(**int** j=i+1;j<a.length;j++){ **int** temp=0; **if**(a[i]<a[j]){

temp=a[j]; a[j]=a[i]; a[i]=temp;

}

}

}

**for**(**int** k=0;k<a.length;k++){ System.***out***.println(a[k]);

a[a.length-2] );

}

}

System.***out***.println("The Second minimum number is" +

}

### Output:

64

45

24

-12

-23

-100

The Second minimum number is-23

## First maximum number :

**public class** FirstLargest {

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

**int** a[]={-12,45,-23,64,-100,24};

**for**(**int** i=0;i<a.length;i++){

**for**(**int** j=i+1;j<a.length;j++){ **int** temp=0; **if**(a[i]<a[j]){

temp=a[j]; a[j]=a[i]; a[i]=temp;

}

}

}

**for**(**int** k=0;k<a.length;k++){ System.***out***.println(a[k]);

a[a.length-6] );

}

}

### Output:

64

45

}

System.***out***.println("The First maximum number is" +

24

-12

-23

-100

The First maximum number is64

## Third maximum number:

**public class** ThirdLarge {

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

**int** a[]={-12,45,-23,64,-100,24};

**for**(**int** i=0;i<a.length;i++){

**for**(**int** j=i+1;j<a.length;j++){ **int** temp=0; **if**(a[i]<a[j]){

temp=a[j]; a[j]=a[i]; a[i]=temp;

}

}

}

**for**(**int** k=0;k<a.length;k++){ System.***out***.println(a[k]);

}

System.***out***.println("The Third maximum number is" + a[a.length-4] ); }}

### Output:

64

45

24

-12

-23

-100

The Third maximum number is24

## Count the Small ,Caps, number and Special character in string:

**package** org.room.assign4;

**public class** LettersCount {

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

String s = "Hi Welcome To Java Classes Tommorow At 2.00

p.m!!";

**int** count = 0;

**int** count1 = 0; **int** count2 = 0; **int** count3 = 0;

**for** (**int** i = 0; i < s.length(); i++) {

**if** (s.charAt(i) >= 'a' && s.charAt(i) <= 'z') {

count++;

} **else if** (s.charAt(i) >= 'A' && s.charAt(i) <= 'Z') {

count1++;

} **else if** (s.charAt(i) >= '0' && s.charAt(i) <= '9') {

count2++;

} **else** {

count3++;

}

count3);

}

}

System.***out***.println("total no of small letters:" + count); System.***out***.println("total no of capital letters:" + count1); System.***out***.println("total no of digits :" + count2); System.***out***.println("total no of special characters:" +

}

### Output:

total no of small letters:27 total no of capital letters:7 total no of digits :3

total no of special characters:12

## Count of given number:

**package** org.room.assign4; **import** java.util.Scanner; **public class** CountOfGivenNum {

**public static void** main(String[] args) { **int** n,i=0; System.***out***.println("enter a no");

Scanner get=**new** Scanner(System.***in***); n=get.nextInt();

**while**(n>0)

{

n=n/10; i++;

}

System.***out***.println("no of digits present:"+i);

}

}

### Output:

enter a no 5267546

no of digits present:7

## Palindrome string:

**package** org.room.assign4;

**import** java.util.Scanner;

**public class** PoyindromeString {

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

{

String original, reverse = ""; Scanner in = **new** Scanner(System.***in***);

System.***out***.println("Enter a string to check if it is a palindrome");

original = in.nextLine();

**int** length = original.length();

**for** ( **int** i = length - 1; i >= 0; i-- ) reverse = reverse + original.charAt(i);

**if** (original.equals(reverse)) System.***out***.println("Entered string is a palindrome.");

**else**

System.***out***.println("Entered string is not a

palindrome.");

}

}

### Output:

Enter a string to check if it is a palindrome madam

Entered string is a palindrome.

## Reverse the String:

**package** org.room.assign4;

**import** java.util.Scanner;

**public class** ReverseString {

**public static void** main(String args[]) { String original, reverse = ""; Scanner in = **new** Scanner(System.***in***);

System.***out***.println("Enter a string to reverse"); original = in.nextLine();

**int** length = original.length();

**for** (**int** i = length - 1; i >= 0; i--) reverse = reverse + original.charAt(i);

System.***out***.println("Reverse of entered string is: " +

reverse);

}

}

### Output:

Enter a string to reverse welcome

Reverse of entered string is: emoclew

## Triangle with stars:

**package** org.room.assign4;

**public class** Triangle1 {

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

**for** (**int** i = 1; i <= 5; i++) {

**for** (**int** j = 1; j <= 5 - i; j++) { System.***out***.print(" ");

}

**for** (**int** k = 1; k <= i; k++) { System.***out***.print("\* ");

### Output:

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \* \* \*

}

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

## Sum of give num:

**package** org.room.assign4;

**public class** SumOfGivenNum {

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

**int** m, n, sum = 0; m = 12345;

**while** (m > 0) {

n = m % 10;

sum = sum + n; m = m / 10;

}

System.***out***.println("Sum of Digits:" + sum);

}

}

### Output:

Sum of Digits:15

## Count of each word in the string:

**public class** Count {

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

{

String s = "Raman ram"; String[] s1 = s.split(" ");

HashMap<String, Integer> emp = **new** HashMap<String, Integer>();

**for** (String c : s1) {

**if** (emp.containsKey(c)) { **int** x = emp.get(c); emp.put(c, x + 1);

} **else** {

emp.put(c, 1);

}

}

System.***out***.println(emp);

}

}

}

### Output:

{Raman=1, ram=1}

## Count of each character in the string:

**public class** ReverseString {

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

{

String s = "Ramanram";

HashMap<Character, Integer> emp = **new** HashMap<Character, Integer>();

**char**[] ch = s.toCharArray();

**for** (**char** c : ch) {

**if** (emp.containsKey(c)) { **int** x = emp.get(c); emp.put(c, x + 1);

} **else** {

emp.put(c, 1);

}

}

System.***out***.println(emp);

}

}

}

### Output:

{a=2, r=1, t=1, e=1, v=1, g=1, m=1, n=1}

## Assume a string “welcome to Polaris” remove space and print the string. public class Dummy {

**public static void** main(String[] args) { String s="Welcome to Polaris";

String x = s.replace(" ", ""); System.***out***.println(x);

}

}

## Output:

WelcometoPolaris

## Write a program to split and then reverse a string.

**Reverse the string:**

**public class** ReverseString {

**public static void** main(String args[]) { String original, reverse = "";

Scanner in = **new** Scanner(System.***in***); System.***out***.println("Enter a string to reverse"); original = in.nextLine();

**int** length = original.length();

**for** (**int** i = length - 1; i >= 0; i--) { reverse = reverse + original.charAt(i); }

System.***out***.println("Reverse of entered string is: " + reverse);

}

}

## Output:

Enter a string to reverse HELLO

Reverse of entered string is: OLLEH

## Split:

**public class** StringBasic {

**public static void** main(String[] args) { String s1 = "Hello welcome to java class";

String[] x = s1.split(" "); // here we split by space **for**(**int** i=0;i<x.length;i++){ System.***out***.println(x[i]);

}

}

}

## Output:

Hello welcome to

java class

## Construct the triangle

|  |  |  |
| --- | --- | --- |
|  | 9 |  |
| 8 9 8 |
| 7 8 9 8 7 |
| 6 | 7 8 9 8 7 | 6 |
| 5 6 | 7 8 9 8 7 | 6 5 |
| 4 5 6 | 7 8 9 8 7 | 6 5 4 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 | 8 | 7 | 6 | 5 | 4 | 3 |
| 2 3 | 4 | 5 | 6 7 | | 8 | 9 | 8 7 6 | | | 5 | 4 | 3 2 |
| 1 2 3 | 4 | 5 | 6 7 | | 8 | 9 | 8 7 6 | | | 5 | 4 | 3 2 1 |

**Program:**

**public class** ReverseString {

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

{

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("How Many Rows You Want In Your Pyramid?");

**int** noOfRows = sc.nextInt();

**int** rowCount = 1;

System.***out***.println("Here Is Your Pyramid");

**for** (**int** i = noOfRows; i >= 1; i--)

{

//Printing i\*2 spaces at the beginning of each row

**for** (**int** j = 1; j <= i\*2; j++)

{

System.***out***.print(" ");

}

//Printing j where j value will be from i to noOfRows

**for** (**int** j = i; j <= noOfRows; j++)

{

System.***out***.print(j+" ");

}

//Printing j where j value will be from noOfRows-1 to i

**for** (**int** j = noOfRows-1; j >= i; j--)

{

System.***out***.print(j+" ");

}

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

//Incrementing the rowCount

rowCount++;

}

}

}

## Output:

How Many Rows You Want In Your Pyramid? 9

Here Is Your Pyramid

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | 9 |  |  |  | | | | |
| 8 | 9 | 8 |
|  | | | 7 | 8 | 9 | 8 | 7 | | |  | |
| 6 | | | 7 | 8 | 9 | 8 | 7 6 | | |  | |
| 5 6 | | | 7 | 8 | 9 | 8 | 7 6 5 | | |  | |
| 4 5 6 | | | 7 | 8 | 9 | 8 | 7 6 5 | | | 4 | |
| 3 4 | 5 | 6 | 7 8 | | 9 | 8 7 | | 6 | 5 | 4 3 |  |
| 2 3 4 | 5 | 6 | 7 8 | | 9 | 8 7 | | 6 | 5 | 4 3 2 |  |
| 1 2 3 4 | 5 | 6 | 7 8 | | 9 | 8 7 | | 6 | 5 | 4 3 2 | 1 |

## Write a program to find sum of each digit in the given number using recursion?

**Program:**

**public class** MyNumberSumRec {

**int** sum = 0;

**public int** getNumberSum(**int** number){

**if**(number == 0){

**return** sum;

} **else** {

sum += (number%10); getNumberSum(number/10);

}

**return** sum;

}

**public static void** main(String a[]){ MyNumberSumRec a = **new** MyNumberSumRec();

System.***out***.println("Sum is: "+a.getNumberSum(5678));

}

}

## Output:

Sum is: 26

## Longest substring without repeating characters

|  |  |  |
| --- | --- | --- |
| **INPUT** |  | **OUTPUT** |
| **java2novice** | **=** | **a2novice** |
| **java\_language\_is\_sweet** | **=** | **uage\_is** |
| **java\_java\_java\_java** | **=** | **va\_j, \_jav** |
| **abcabcbb**  **program:** | **=** | **bca, abc, cab** |

**public class** MyLongestSubstr {

**private** Set<String> subStrList = **new** HashSet<String>();

**private int** finalSubStrSize = 0;

**public** Set<String> getLongestSubstr(String input){

//reset instance variables subStrList.clear();

finalSubStrSize = 0;

// have a boolean flag on each character ascii value

**boolean**[] flag = **new boolean**[256]; **int** j = 0;

**char**[] inputCharArr = input.toCharArray();

**for** (**int** i = 0; i < inputCharArr.length; i++) {

**char** c = inputCharArr[i];

**if** (flag[c]) { extractSubString(inputCharArr,j,i); **for** (**int** k = j; k < i; k++) {

**if** (inputCharArr[k] == c) { j = k + 1;

## break;

}

flag[inputCharArr[k]] = **false**;

}

} **else** {

flag[c] = **true**;

}

}

extractSubString(inputCharArr,j,inputCharArr.length);

**return** subStrList;

}

**private** String extractSubString(**char**[] inputArr, **int** start, **int** end){

StringBuilder sb = **new** StringBuilder();

**for**(**int** i=start;i<end;i++){ sb.append(inputArr[i]);

}

String subStr = sb.toString();

**if**(subStr.length() > finalSubStrSize){ finalSubStrSize = subStr.length(); subStrList.clear(); subStrList.add(subStr);

} **else if**(subStr.length() == finalSubStrSize){ subStrList.add(subStr);

}

**return** sb.toString();

}

**public static void** main(String a[]){ MyLongestSubstr mls = **new** MyLongestSubstr();

System.***out***.println(mls.getLongestSubstr("java2novice")); System.***out***.println(mls.getLongestSubstr("java\_language\_is\_sweet"));

System.***out***.println(mls.getLongestSubstr("java\_java\_java\_java")); System.***out***.println(mls.getLongestSubstr("abcabcbb"));

}

}

## Output :

[a2novice] [uage\_is] [va\_j, \_jav] [bca, abc, cab]

## Kth largest or smallest element in an array

**Example : if given array is [1,3,12,19,13,2,15] and you are asked for the 3rd largest element i.e., k=3 then your program should print 13**

## Program:

**public class** ThirdLarge {

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

**int** a[]={1,3,12,19,13,2,15};

**for**(**int** i=0;i<a.length;i++){

**for**(**int** j=i+1;j<a.length;j++){ **int** temp=0; **if**(a[i]<a[j]){

temp=a[j]; a[j]=a[i]; a[i]=temp;

}

}

}

**for**(**int** k=0;k<a.length;k++){ System.***out***.println(a[k]);

}

System.***out***.println("The Third maximum number is :" + a[a.length-5] );

}

}

## Output:

19

15

13

12

3

2

1

The Third maximum number is :13

## Armstrong number: Program:

**public class** ArmstrongNumberCheck {

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

**int** n, a, i = 0, j = 0;

Scanner an = **new** Scanner(System.***in***); System.***out***.println("Enter a number"); n = an.nextInt();

a = n;

**while** (a > 0) {

i = a % 10;

j = j + (i \* i \* i); a = a / 10;

}

**if** (n == j) {

System.***out***.println("Armstrong number");

} **else** {

System.***out***.println("Not armstrong Number");

}

}

}

Output :

Enter a number 371

Armstrong number

## Write a program to remove duplicates from sorted array Input : 2,3,6,6,9,10,10,10,12,12

**Output : 2,3,6,9,10,12**

## Program:

**public class** MyDuplicateElements {

**public static int**[] removeDuplicates(**int**[] input){

**int** j = 0;

**int** i = 1;

//return if the array length is less than 2

**if**(input.length < 2){

**return** input;

}

**while**(i < input.length){

**if**(input[i] == input[j]){ i++;

}**else**{

input[++j] = input[i++];

}

}

**int**[] output = **new int**[j+1];

**for**(**int** k=0; k<output.length; k++){ output[k] = input[k];

}

**return** output;

}

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

**int**[] input1 = {2,3,6,6,8,9,10,10,10,12,12};

**int**[] output = *removeDuplicates*(input1);

**for**(**int** i:output){ System.***out***.print(i+" ");

}

}

}

## Output:

2 3 6 8 9 10 12

## Binary search Program:

**public class** MyBinarySearch {

**public int** binarySearch(**int**[] inputArr, **int** key) {

**int** start = 0;

**int** end = inputArr.length - 1;

**while** (start <= end) {

**int** mid = (start + end) / 2;

**if** (key == inputArr[mid]) {

**return** mid;

}

**if** (key < inputArr[mid]) { end = mid - 1;

} **else** {

start = mid + 1;

}

}

**return** -1;

}

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

MyBinarySearch mbs = **new** MyBinarySearch();

**int**[] arr = {2, 4, 6, 8, 10, 12, 14, 16};

System.***out***.println("Key 14's position: "+mbs.binarySearch(arr, 14));

**int**[] arr1 = {6,34,78,123,432,900};

System.***out***.println("Key 432's position: "+mbs.binarySearch(arr1, 432));

}

}

## Output:

Key 14's position: 6 Key 432's position: 4

## Butterfly shuffle: Program:

import java.util.ArrayList; import java.util.Scanner;

public class SampleTest {

public static void main(String[] args) { int a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };

int len = a.length / 2;

for (int i = len - 1; i >= 0; i--) { System.out.println(a[i]);

}

for (int i = a.length - 1; i >= len; i--) {

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

}

}

}

## Output:

**5**

## 4

**3**

## 2

**1**

## 0

**9**

## 8

**7**

## 6