Software Testing Notes

Automation testing (java+selenium)

**Day 1 -**

Java installation procedure:

Step1: Command to check java version: java -version

step2: check operating system:

Step2: how to download java

search download java 1.8-->click on oracle.com-->Java SE Development Kit 8u281--> click on 32/64 related file to download java

Step3: install downloaded java file

Step4: Copy java path

open C drive-->program files-->java-->jdk/jre(jdk prefered)-->bin folder-->copy bin folder path

Step5: Set java path

right click on This Pc/my computer-->properties-->advanced system setting-->envirnment variable-->user variable--> check for "path" variable

--Case1: already path variable exist -->click on path variable-->edit-->new-->ctrl+ V(Paste) --> ok--ok

--Case2: no path variable--> click on new --> enter variable name i.e. -"path"--> variable value -ctrl+ V(Paste)--> ok-->ok

--> check for java version

Error msg: version 1.80 \_261 of the jvm is not suitable for this product version 11 or greater is required --> install java 11 or install eclipse old version

how to check PC/laptop configuration: Right click on This PC/ My Computer --> Properties--> System type

**Day 2**

Diff versions eclipse IDE:

versions:

oxygen

neon

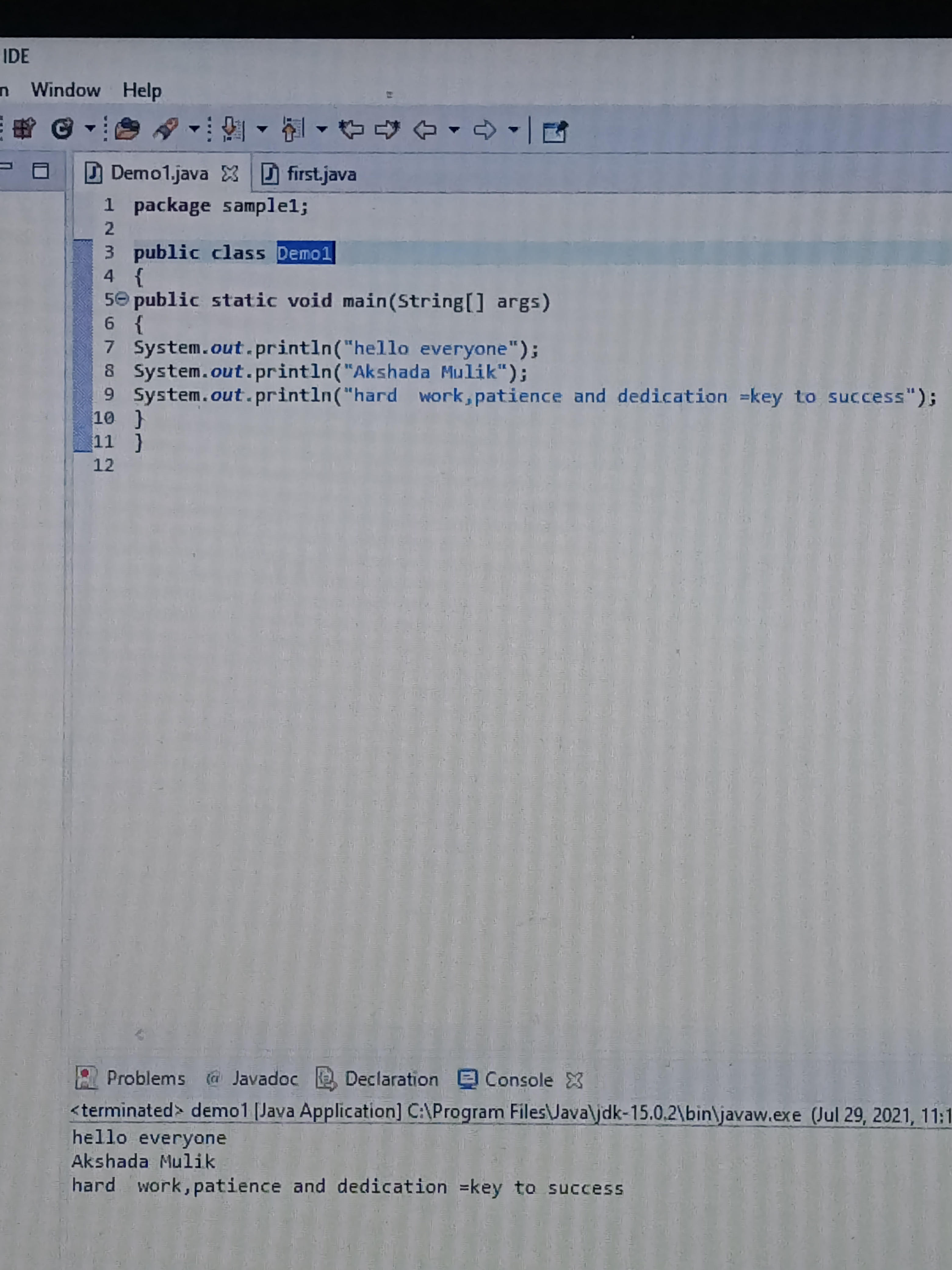
marsh

photon

eclipse installation procedure:

--search download eclipse oxygen-->click on https://www.eclipse.org/-->MORE DOWNLOADS-->Eclipse 2020-12 (4.14)-->Eclipse IDE for Java and DSL Developers-->Windows x86\_64--> eclipse-dsl-2020-12-R-win32-x86\_64.zip

--open downloads folder --> unzip eclipse file ---> open eclipse folder--> double click on eclipse application file (blue colour icon)--> it should ask for workspace path--> keep as it is(default path)-->select checkbox-->launch--> welcome page



Day 3

java 1st program:

Step1: create java project--> file--> new--> java project--> enter project name--> finish

Step2: create java package--> right click on project name/Src folder-->new--> package--> enter package name--> finish

Step3: create java class-->right click on package name-->new-->class-->enter class name--> finish

1 java project--> multiple packages-->

1 packages--> multiple classes

1 class--> multiple method--> main method

main method --> multiple printing statements --> to print messages

Shortcuts in eclipse:

1. main method: "main"+control+space

2. Printing statement: "syso"+control+space

**Day 4**

package Variables;

public class sample2

{

public static void main(String[] args) {

//1. Variable declaration

String sname; //dataType variableName

int rollnum;

char grade;

float per;

//2. Variable Initialization(Assigning or Inserting value)

sname = "Kanchan"; //variableName = "variable value"

rollnum = 100; //variableName = variable value

grade = 'A'; //variableName = 'variable value'

per = 65.5f; //variableName = variablevalue f

//3. Usage

System.out.println(sname);

System.out.println(rollnum);

System.out.println(grade);

System.out.println(per);

}

}

1.Variables:

Variables are nothing but piece of memory use to store information.

one variable can store 1 information at a time.

Variables also used in information reusability.

To utilise variables in java programing language we need to follow below steps:

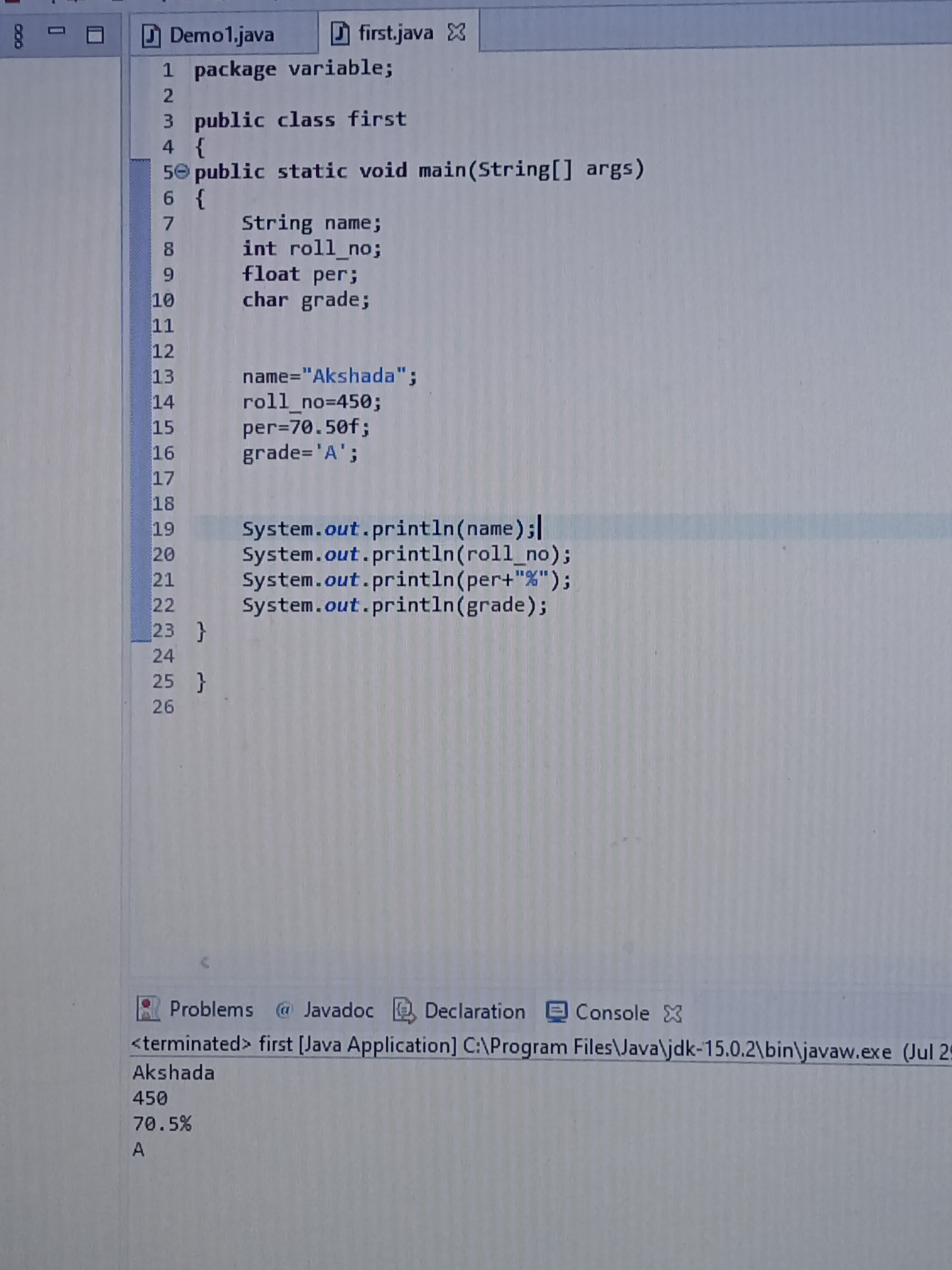
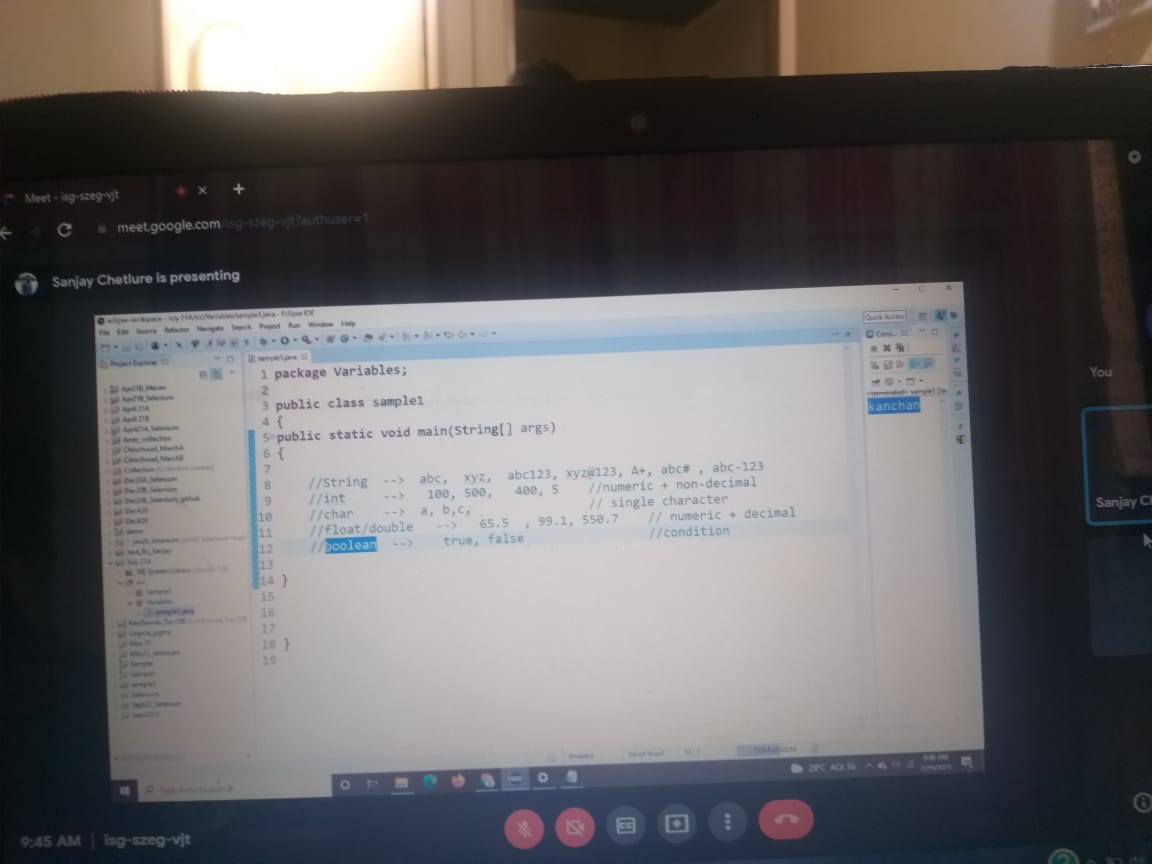
1. Variable declaration (Allocating/Reserving memory)

2. Variable Initialization(Assigning or Inserting value)

3. Usage

Note:- According to all programming language dealing with information directly is not a good practice

to overcome this variables are introduced.



package Variables;

public class sample2

{

public static void main(String[] args)

{

//1. Variable declaration

String sname; //dataType variableName

int rollnum;

char grade;

float per;

//2. Variable Initialization(Assigning or Inserting value)

sname = "Kanchan"; //variableName = "variable value"

rollnum = 100; //variableName = variable value

grade = 'A'; //variableName = 'variable value'

per = 65.5f; //variableName = variablevalue f

//3. Usage

System.out.println("Student name: "+sname);

System.out.println("student rollNum: "+rollnum);

System.out.println("student grade: "+grade);

System.out.println("student percentage: "+ per +" %");

}

}

Data Types:

Data type are used to represent type of data or information which we are going to use in our java program.

In java programming it is mandatory to declare datatype before declaration of variable.

In java datatypes are classified into two types :

1. Primitive datatype.

2. Non-primitive datatype.

1.Primitive datatype:

There are 8 type of primitive datatypes.

all the primitive datatypes are keywords.

\* Memory size of primitive datatype are fix.

The types of primitive datatype are:

Note:- keyword starts with lower case

syntax: datatype variablename;

1.(Numeric + Non-decimal):-

Ex: 80,85,10,..etc

Data Type Size

1. byte 1 byte

2. short 2 bytes

3. int 4 bytes

4. long 8 bytes

2.(Numeric + decimal):-

Ex: 22.5,22.8,6.4....

5. float 4 byte

6. double 8 byte

3.Character :-

Ex: A,B,X,Z.

7. char 2 byte

---------------------------------------

4.Conditional:-

Ex: true,false.

8. boolean 1 bit

---------------------------------------------------------

2. Non-primitive datatype:

There are 2 types of non primitive datatypes .

all the Non primitive datatypes are identifiers.

\* Memory size of non primitive datatype is not defined or not fix.

Note: Identifier starts with capital letter.

e.g. String, className

[03/08, 9:02 am] +91 99754 04785:

Methods:

A method is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as functions.

Why use methods? To reuse code: define the code once, and use it many times.

1. main method

In any Java program, the main() method is the starting point from where compiler starts program execution.

So, the compiler needs to call the main() method.

without main method we can't run any java program.

2. Regular method

1. static regular method

1. static method call from same class --> methodname();

2. static method call from diffrent/another class -->className.methodname();

2. non- static regular method

3. non-static method call from same class --> create object of same class

4. non-static method call from diffrent/another class --> create object of diff class

Note: At the time of program execution main method is going to get executed automatically,

where as regular methods are not going to get executed automatically.

At the time of program execution priority is sceduled for main method only.

To call a regular method we need to make call method call from main method,

until unless if the method call is not made regular method will not get executed.

regular methods can be called multiple times.

[03/08, 9:02 am] +91 99754 04785: package Methods;

public class demo1 {

//1. static regular method call from same class --> methodname();

public static void main(String[] args) //main method-->manager

{

System.out.println("main method started");

m1(); // methodname(); //static regular method call from same class

m2();

System.out.println("main method ended");

}

// static ->regular method

public static void m1() //regular method-->emp

{

System.out.println("running static regular method from same class: m1");

}

public static void m2() {

System.out.println("running static regular method from same class: m2");

}

}

[03/08, 9:02 am] +91 99754 04785: package Methods;

public class demo2 {

//2. static method call from diffrent/another class -->className.methodname()

public static void main(String[] args) {

System.out.println("main method started");

demo3.m3(); // className.methodname(); //static method call from diff class

demo3.m4();

System.out.println("main method ended");

}

}

[03/08, 9:02 am] +91 99754 04785: package Methods;

public class demo3 {

// static ->regular method

public static void m3()

{

System.out.println("running static regular method from diff class: m3");

}

// static ->regular method

public static void m4()

{

System.out.println("running static regular method from diff class: m4");

}

}

[04/08, 8:58 am] +91 99754 04785: package Methods;

public class demo4 {

//3. non-static method call from same class --> create object of same class

public static void main(String[] args)

{

System.out.println("main method started");

//classname objectName = new classname();

demo4 d = new demo4(); //object creation

d.m5(); //objectName.methodname();

d.m6();

System.out.println("main method ended");

}

//non-static regular method

public void m5()

{

System.out.println("running non-static regular method from same class: m5");

}

//non-static regular method

public void m6()

{

System.out.println("running non-static regular method from same class: m6");

}

}

[04/08, 8:58 am] +91 99754 04785: package Methods;

public class demo5

{

//4. non-static method call from diffrent/another class --> create object of diff class

public static void main(String[] args)

{

System.out.println("main method started");

//classname objectName = new classname();

demo6 d6=new demo6(); //object creation

d6.m7();

d6.m8();

System.out.println("main method ended");

}

}

[04/08, 8:58 am] +91 99754 04785: package Methods;

public class demo6 {

//non-static regular method

public void m7()

{

System.out.println("running non-static regular method from diff class: m7");

}

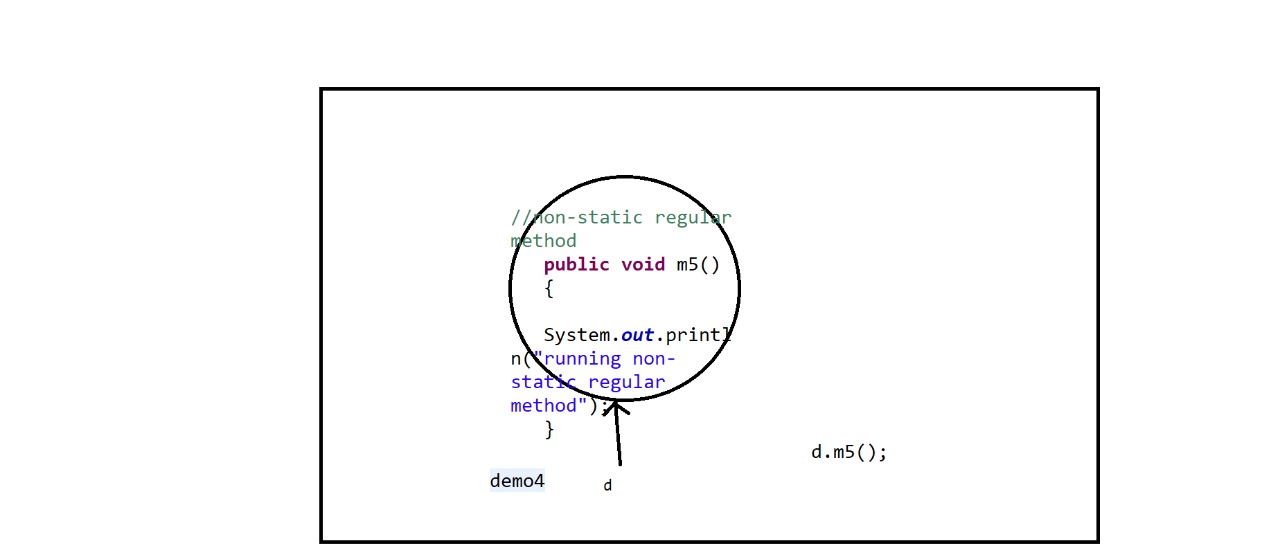
//non-static regular method

public void m8()

{

System.out.println("running non-static regular method from diff class: m8");

}



}

class Methods

{

public static void main(String []args)

{

System.out.println("main method started");

m1();

demo.m2();

Methods a= new Methods();

a.m3();

demo b= new demo();

b.m4();

System.out.println("main method ended");

}

public static void m1()

{

System.out.println("running static regular method from same class m1");

}

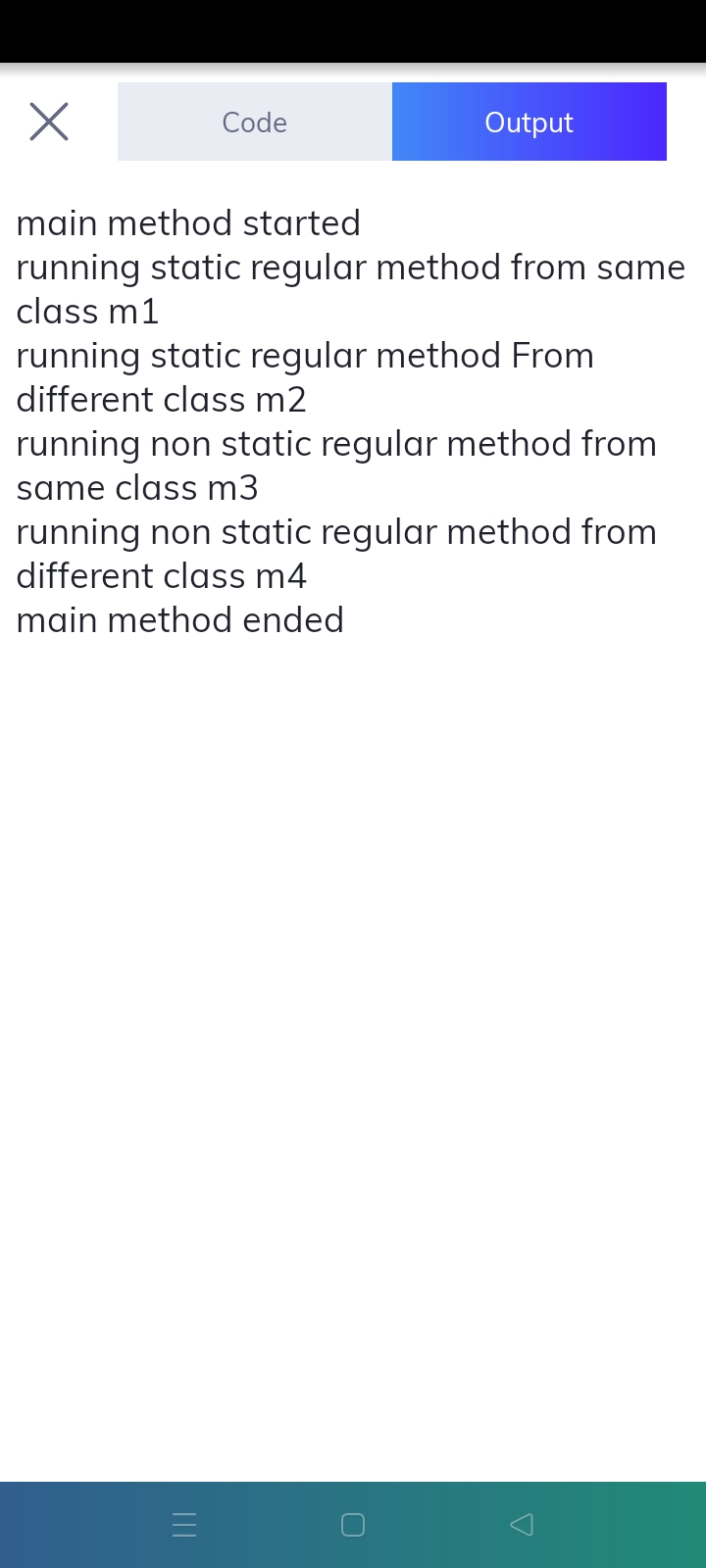
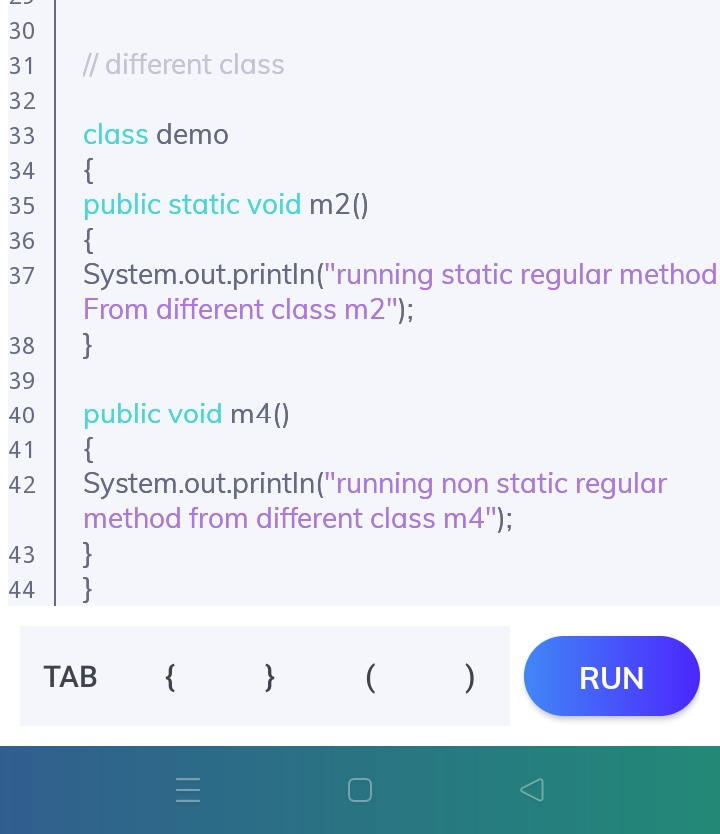
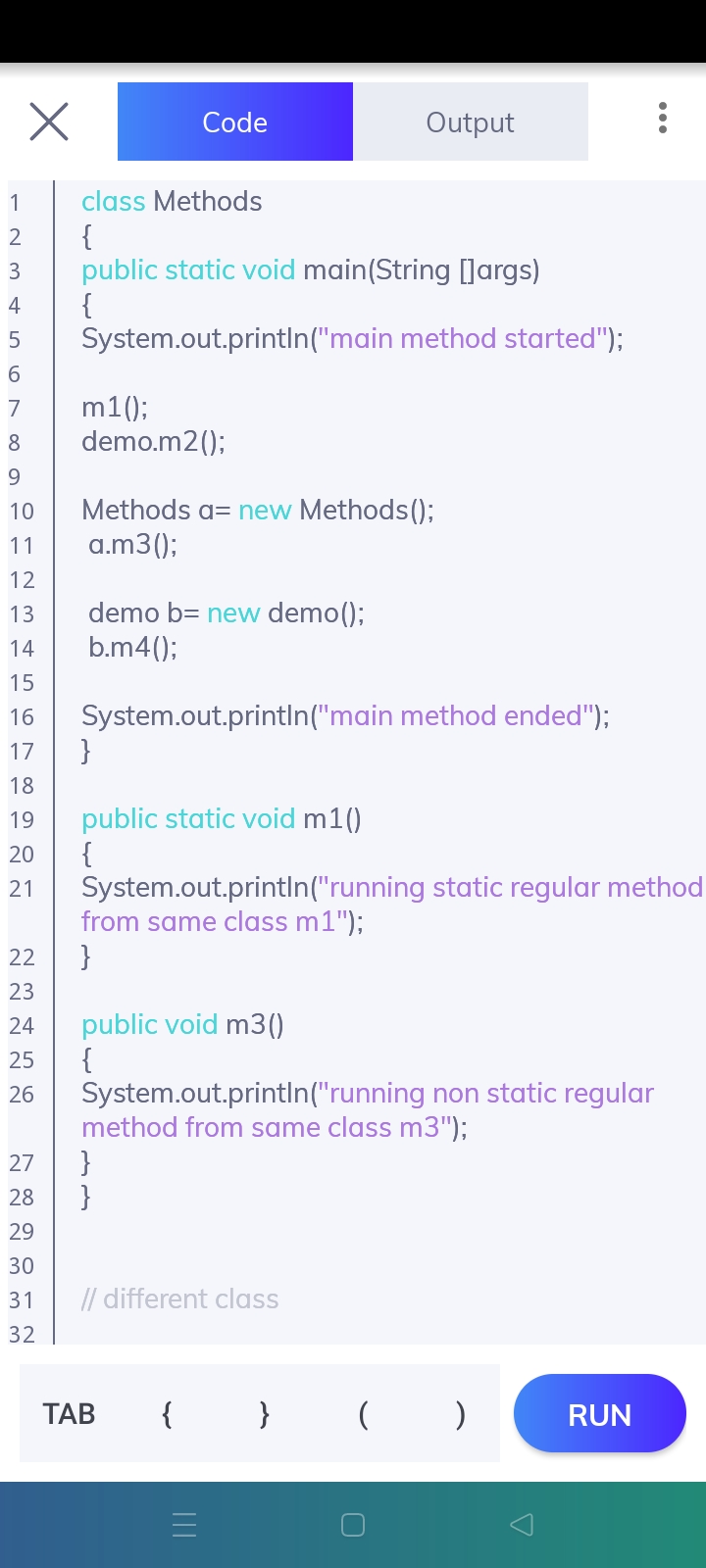
public void m3()

{

System.out.println("running non static regular method from same class m3");

}

}



[06/08, 8:55 am] Sanjay Sir Velocity: package Methods;

public class demo8 {

// 6. method with parameter

public static void main(String[] args) {

System.out.println("main method started");

addition();

addition(10, 20);

demo8 d8=new demo8();

d8.addition(10, 20, 30);

System.out.println("main method ended");

}

//static regular method

//method without/zero parameter

public static void addition()

{

int a=10;

int b=20;

int c=a+b;

System.out.println(c);

}

//static regular method

//method with parameter -- int ,int

public static void addition(int a, int b)

{

int c = a + b;

System.out.println(c);

}

//non-static regular method

//method with parameter -- int ,int,int

public void addition(int a, int b, int d)

{

int c = a + b +d;

System.out.println(c);

}

}

[06/08, 8:55 am] Sanjay Sir Velocity: package Methods;

public class demo7

{

//5. method without/zero parameter

public static void main(String[] args)

{

System.out.println("main method started");

addition();

System.out.println("main method ended");

}

//static regular method

//method without/zero parameter

public static void addition()

{

int a=10;

int b=20;

int c=a+b;

System.out.println(c);

}

}

[09/08, 8:51 am] Sanjay Sir Velocity: package Methods;

public class demo9 {

// 6. method with parameter

public static void main(String[] args) {

System.out.println("main method started");

studentName();

studentName("sanjay");

studentName("rahul");

System.out.println("main method ended");

}

//static regular method

//method without parameter --

public static void studentName()

{

String sname="suraj";

System.out.println(sname);

}

//static regular method

//method with parameter --String parameter

public static void studentName(String sname)

{

System.out.println(sname);

}

}

[09/08, 8:51 am] Sanjay Sir Velocity: package Methods;

public class demo10

{

public static void main(String[] args) {

System.out.println("main method started");

studentInfo("abc",200, 'C', 85.2f);

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

studentInfo("xyz",300, 'A', 60.2f);

System.out.println("main method ended");

}

public static void studentInfo(String sname, int sRollNum, char sGrade, float sper)

{

System.out.println(sname);

System.out.println(sRollNum);

System.out.println(sGrade);

System.out.println(sper);

}

//public static void studentInfo()

//{

// String StudentName;

// int StudentRollNum;

// char StudentGrade;

// float StudentPer;

//

// StudentName ="abc";

// StudentRollNum = 100;

// StudentGrade ='B';

// StudentPer = 76.6f;

//

// System.out.println(StudentName);

// System.out.println(StudentRollNum);

// System.out.println(StudentGrade);

// System.out.println(StudentPer);

//}

}

[09/08, 8:52 am] Sanjay Sir Velocity: package Loops;

public class example1\_ForLoop1 {

public static void main(String[] args) {

//6<=5 6

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

{

System.out.println(i); //1 2 3 4 5

}

}

}

[09/08, 8:52 am] Sanjay Sir Velocity: package Loops;

public class example1\_ForLoop2 {

public static void main(String[] args) {

for(int i=1; i<=20; i++)

{

System.out.println(i);

}

}

}

[09/08, 8:52 am] Sanjay Sir Velocity: package Loops;

public class example1\_ForLoop3 {

public static void main(String[] args) {

for(int i=51; i<=55; i++)

{

System.out.println(i);

}

// for(int i=11; i<=25; i++)

// {

// System.out.println(i);

// }

}

}

[09/08, 8:52 am] Sanjay Sir Velocity: package Loops;

public class example1\_ForLoop4\_print\_odd\_Numbers\_From\_1\_To\_99

{

public static void main(String[] args)

{

for(int i=1; i<=99; i=i+2)

{

System.out.println(i);

}

}

}

[09/08, 8:52 am] Sanjay Sir Velocity: package Loops;

public class example1\_ForLoop5\_print\_even\_Numbers\_From\_2\_To\_100

{

public static void main(String[] args)

{

for(int i=2; i<=100; i=i+2)

{

System.out.println(i);

}

}

}

[10/08, 8:50 am] Sanjay Sir Velocity: package Loops;

public class example1\_ForLoop9\_print\_odd\_Numbers\_From\_99\_To\_1

{

public static void main(String[] args)

{

for(int i=99; i>=1; i=i-2)

{

System.out.println(i);

}

}

}

[10/08, 8:50 am] Sanjay Sir Velocity: package Loops;

public class example1\_ForLoop10\_print\_even\_Numbers\_From\_100\_To\_2

{

public static void main(String[] args)

{

for(int i=100; i>=2; i=i-2)

{

System.out.println(i);

}

}

}

[10/08, 8:50 am] Sanjay Sir Velocity: package Loops;

public class example2\_whileLoop1 {

public static void main(String[] args) {

int i=51; //start condition

// 6<=5

while(i<=60) //end condition

{

System.out.println(i); //1 2 3 4 5

i++; //6 //increment or decrement

}

}

}

[10/08, 8:50 am] Sanjay Sir Velocity: package Loops;

public class example2\_whileLoop2\_print\_odd\_num\_from\_1\_to\_99 {

public static void main(String[] args) {

int i=1; //start condition

while(i<=99) //end condition

{

System.out.println(i);

i=i+2; //increment or decrement

}

}

}

[10/08, 8:50 am] Sanjay Sir Velocity: package Loops;

public class example2\_whileLoop3\_print\_num\_from\_15\_to\_5 {

public static void main(String[] args) {

int i=15; //start condition

while(i>=5) //end condition

{

System.out.println(i);

i--; //increment or decrement

}

}

}

[10/08, 8:51 am] Sanjay Sir Velocity: package Loops;

public class example2\_whileLoop4\_print\_odd\_num\_from\_99\_to\_1 {

public static void main(String[] args) {

int i=99; //start condition

while(i>=1) //end condition

{

System.out.println(i);

i=i-2; //increment or decrement

}

}

}

[10/08, 8:51 am] Sanjay Sir Velocity: package Loops;

public class example3\_DoWhile1 {

public static void main(String[] args) {

int i=1; //start condition

do

{

System.out.println(i); //1 2 3 4 5 //statements

i++; //6 //increment or decrement

}

while (i<=5); //end condition

//1>=5

}

}

[10/08, 8:54 am] Sanjay Sir Velocity: Loops:

In programming languages, loops are used to execute a set of instructions/functions repeatedly when some conditions become true.

There are 4 types of loops in Java.

1. for loop

2. while loop

3. do while

4. for each--array/collection--> selenium

for (statement 1; statement 2; statement 3)

{

// code block to be executed

}

Statement 1 is executed (one time) before the execution of the code block.

Statement 2 defines the condition for executing the code block.

Statement 3 is executed (every time) after the code block has been executed.

[12/08, 8:46 am] Sanjay Sir Velocity: package Control\_Statements;

public class example1\_IF\_Statement {

public static void main(String[] args) {

int marks= 20;

//20>=35

if(marks>=35)

{

System.out.println("Pass");

}

// if(condition)

// {

//

// }

}

}

[12/08, 8:46 am] Sanjay Sir Velocity: package Control\_Statements;

public class example2\_IF\_Else\_Statement {

public static void main(String[] args) {

int marks = 35;

// 35>=35

if(marks>=35)

{

System.out.println("Pass");

}

else

{

System.out.println("Fail");

}

}

}

[12/08, 8:46 am] Sanjay Sir Velocity: package Control\_Statements;

public class example3\_else\_If\_Stement {

public static void main(String[] args) {

int marks = 40;

if(marks>=65)

{

System.out.println("distinction");

}

else if (marks>=60)

{

System.out.println("1st class");

}

else if (marks>=55)

{

System.out.println("higher 2nd class");

}

else if (marks>=50)

{

System.out.println("2nd class");

}

else if (marks>=35)

{

System.out.println("Pass");

}

else

{

System.out.println("Fail");

}

}

}

[12/08, 8:46 am] Sanjay Sir Velocity: Java Keywords:

Keywords are predefined, reserved words used in Java programming that have special meanings to the compiler. For example:

abstract assert boolean break byte

case catch char class const

continue default do double else

enum extends final finally float

for goto if implements import

instanceof int interface long native

new package private protected public

return short static strictfp super

switch synchronized this throw throws

transient try void volatile while

Java identifiers

Identifiers are the name given to variables, classes, methods, package etc

Rules for Naming an Identifier

1.Identifiers cannot be a keyword.

2.Identifiers are case-sensitive.

3. It can have a sequence of letters and digits. However,

it must begin with a letter, $ or \_. The first letter of an identifier cannot be a digit.

4. It's a convention to start an identifier with a letter rather and $ or \_.

5. Whitespaces are not allowed.Similarly, you cannot use symbols such as @, #, and so on.

eg.

String str123 , abc1, $ab, \_xyz , s42424798

//1abc, 5xy, 10ab

//s name , sample 1, demo 10

//abc@123 , xyz#1

[13/08, 8:49 am] Sanjay Sir Velocity: package Control\_Statements;

public class example4\_nested\_IF

{

public static void main(String[] args)

{

String UN="abc";

String PWD="xyz";

if("ab"==UN)

{

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

System.out.println("Enter your password");

if("xyz"==PWD)

{

System.out.println("Correct PWD--> Login Scuccessful");

}

else

{

System.out.println("Wrong PWD--> Login Failed");

}

}

else

{

System.out.println("Wrong UN --> Login Failed");

}

}

}

[13/08, 8:49 am] Sanjay Sir Velocity: package Control\_Statements;

public class example5\_Switch {

public static void main(String[] args) {

int day=15;

switch (day)

{

case 1: System.out.println("Today is mon");

break;

case 2: System.out.println("Today is Tue");

break;

case 3: System.out.println("Today is Wed");

break;

case 4: System.out.println("Today is Thr");

break;

case 5: System.out.println("Today is Fri");

break;

case 6: System.out.println("Today is Sat");

break;

case 7: System.out.println("Today is Sun");

break;

default: System.out.println("invalid input");

break;

}

}

}

[13/08, 8:49 am] Sanjay Sir Velocity: package Control\_Statements;

public class example6\_Switch {

public static void main(String[] args) {

String inp="MT";

switch (inp)

{

case "BI": System.out.println("Running BI Code");

break;

case "MT": System.out.println("Running MT Code");

break;

case "MS": System.out.println("Running MS Code");

break;

case "CW": System.out.println("Running CW Code");

break;

case "CD": System.out.println("Running CD Code");

break;

case "PC": System.out.println("Running PC Code");

break;

default: System.out.println("wrong input");

break;

}

}

}

[13/08, 8:49 am] Sanjay Sir Velocity: package Types\_Of\_Variables;

public class sample1 {

public static void main(String[] args) {

sample2 s2=new sample2();

s2.m1(50);

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

s2.m2();

}

}

[13/08, 8:49 am] Sanjay Sir Velocity: package Types\_Of\_Variables;

public class sample2

{

int c=30; //global variable

public void m1(int d) // int d--> local variable

{

int a=10; //local variable

System.out.println(a); //10

System.out.println(d); // 50

System.out.println(c); //30

}

public void m2()

{

//System.out.println(a);

int b=20; //local variable

System.out.println(b); //20

System.out.println(c); //30

}

}

[13/08, 8:50 am] Sanjay Sir Velocity: Types of variable:

1. local variable

Creating variable inside method/block is known as local variable.

Scope of local varaible remains only within the method/block & they are temporary.

2. global variable

Creating variable outside method/block is known as global variable.

Scope of global varaible remains throught the class & they are permanent.

[14/08, 10:27 am] Sanjay Sir Velocity: package Contructor;

public class sample1

{

//example of default constructor

//constructor --> default constructor privided by compiler

// use1: use to copy all the members of class into object

// sample1()

// {

//

// }

public static void main(String[] args)

{

sample1 s1=new sample1();

s1.addition();

//sample1 --> classname-->dataType

//s1 --> objectName

// new --> keyword --> blank object

//sample1() --> classname(); --> constructor

sample2 s2=new sample2();

s2.mul();

}

//non-static regular from same class

public void addition()

{

int a=10;

int b=20;

int sum=a+b;

System.out.println(sum);

}

}

[14/08, 10:27 am] Sanjay Sir Velocity: package Contructor;

public class sample2

{

//example of default constructor

//default condtictor--> provided by compiler

// use1: use to copy all the members of class into object

// sample2()

// {

//

// }

//non-static regular from diff class

public void mul()

{

int a=10;

int b=20;

int mul=a\*b;

System.out.println(mul);

}

}

[14/08, 10:27 am] Sanjay Sir Velocity: package Contructor;

public class sample3

{

//example of user defined contructor

//Step1: declaration

int a; //50

int b; //60

//contructor--> user defined

// use1: use to copy all the members of class into object

// use2. To initialize data member/variable

//step2: initialization

sample3()

{

a=50;

b=60;

}

public static void main(String[] args)

{

sample3 s3=new sample3();

s3.addition();

s3.mul();

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

sample4 s4 =new sample4();

s4.div();

}

//step3: usage

public void addition()

{

int sum=a+b;

System.out.println(sum);

}

//step3: usage

public void mul()

{

int mul=a\*b;

System.out.println(mul);

}

}

[14/08, 10:27 am] Sanjay Sir Velocity: package Contructor;

public class sample4

{

//example of user defined contructor

//step1: declaration

int a;

int b;

//constructor--> user defined

// use1: use to copy all the members of class into object

// use2. To initialize data member/variable/objects

//step2: initialization

sample4()

{

a=10;

b=5;

}

//step3: usage

public void div()

{

int divValue= a/b;

System.out.println(divValue);

}

}

Constructor:

A constructor in Java is a special method that is used to initialize objects/variables.

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

At the time of constructor declaration below points need to folow:

1. Constructor name should be same as class name

2. you should not declare any return type for the constrictor(like void).

3. Any no of constructor can be declared in a java class but constructor name should be same as class name,

but arguments/parameter should be diffrent.

Use of Constructor

1. To copy/load all members of class into object --> when we create object of class

2. To initialize data member/variable

Types of Constructor

1. Default Constructor

2. User defined Constructor

1. Default Constructor

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

If programer has declared the constructor in the class then compiler will not provided default Constructor.

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

2. User defined Constructor

If programer is declaring constructor in java class then it is considered to be as User defined constructor.

User defined Constructor are classified into 2 types

1. Without/zero parameter constructor

// example-without parameter constructor --eg. addition

2. With parameter constructor

// example-with parameter constructor- only 1 constructor -- eg. addidtion

// example-with parameter constructor- multiple constructor -- eg. addition

[16/08, 8:55 am] Sanjay Sir Velocity: package Contructor;

public class sample5

{

//example with parameter -->single constructor

//step1: declaration

int num1; //10, 15

int num2;

//step2: initialization

//constructor --> user defined --> only 1 constructor

//use1: copy all members of class into object

//use2: to perform initialization

sample5(int a, int b) // int , int parameter constructor a=10, b=15

{

num1 = a; //assign local variable info to global variable

num2 = b;

}

public void addition()

{

int c= num1+num2;

System.out.println(c);

}

public static void main(String[] args)

{

sample5 s5=new sample5(30,40);

s5.addition();

sample5 s55=new sample5(5,6);

s55.addition();

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

sample6 s6=new sample6(25, 50);

s6.addition();

s6.mul();

}

}

[16/08, 8:55 am] Sanjay Sir Velocity: package Contructor;

public class sample6 {

//example with parameter -->single constructor

int num1; //25

int num2; //50

//user defined constructor-->with parameter --> 1 constructor

sample6(int a, int b)

{

num1 =a; //25

num2 =b; //50

}

public void addition()

{

int c=num1 +num2;

System.out.println(c);

}

public void mul()

{

int mul=num1\*num2;

System.out.println(mul);

}

}

[16/08, 8:55 am] Sanjay Sir Velocity: package Contructor;

public class sample7 {

public static void main(String[] args) {

//example with parameter -->multiple constructor

sample8 s8 =new sample8(); //create object of without parameter constructor

s8.addition(); //50

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

sample8 s9=new sample8(10, 15);

s9.addition();

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

sample8 s10=new sample8("abc");

s10.studentName();

}

}

[16/08, 8:55 am] Sanjay Sir Velocity: package Contructor;

public class sample8

{

//example with parametr -->multiple constructor

int num1;

int num2;

String sname;

//user defined -->without parameter

sample8()

{

num1=20;

num2=30;

}

//user defined -->with parameter(2 int parameter)

sample8(int a,int b)

{

num1=a; //10

num2=b; //15

}

//user defined -->with parameter(String parameter)

sample8(String str)

{

sname=str;

}

public void addition()

{

int sum = num1 + num2;

System.out.println(sum);

}

public void studentName()

{

System.out.println(sname);

}

}

Types of variables:

1. global variable:

the variable which is declared outside the method/block/constructor is called global variable.

scope of global variable remains throught the class.

global variable is also called permanant variable.

2. local variable:

The variable which is declared inside the method/block/constructor is called local variable.

scope of local variable remains only within the method/block/constructor.

local variable is also called temporary variable.

3. static/class variable:

1. static variable call from same class -->variableName

2. static variable call from diff class--> className.variableName

Note: we can access static global variable in both static & non-static method

4. non-static/instance variable: (instance-object)

3. non-static variable call from same class

4. non-static variable call from diff class

[17/08, 8:41 am] Sanjay Sir Velocity: package Types\_Of\_Variables;

public class sample3

{

//example of static/class variable

static int a=10; //global variable --> static variable

static int b=20; //global variable --> static variable

public static void main(String[] args)

{

//static variable call from same class

int sum=a+b; //variablename;

System.out.println(sum);

//static variable call from diff class

System.out.println(sample4.c); //classname.variablename

}

//we can access static variable in both static & non-static method

public void m1()

{

System.out.println(a);

}

}

[17/08, 8:41 am] Sanjay Sir Velocity: package Types\_Of\_Variables;

public class sample4

{

static int c=30; //static global variable ---> diff class

}

[17/08, 8:42 am] Sanjay Sir Velocity: package Types\_Of\_Variables;

public class sample5

{

int a=50; //non-static global variable

int b=60;

public static void main(String[] args) {

//non-static variable call from same class

sample5 s5=new sample5(); //create object of same class

System.out.println(s5.a); //objectname.variable

System.out.println(s5.b);

System.out.println(s5.a+s5.b);

//non-static variable call from diff class

sample6 s6=new sample6(); //create object of diff class

System.out.println(s6.a); //objectname.variable

}

//non-static regular method

public void addition()

{

System.out.println(a+b);

}

}

[17/08, 8:42 am] Sanjay Sir Velocity: package Types\_Of\_Variables;

public class sample6

{

int a=70; //non-static global variable

}

[18/08, 8:49 am] Sanjay Sir Velocity: package Static\_non\_Static\_Use;

public class static\_use {

public static void main(String[] args) {

emp rahul = new emp();

rahul.eid=100;

rahul.esal=10000;

//rahul.eceoname="abc";

emp.eceoname="abc";

emp Ajay=new emp();

Ajay.eid=200;

Ajay.esal=20000;

//Ajay.eceoname="abc";

emp.eceoname="abc";

emp balaji=new emp();

balaji.eid=300;

balaji.esal=30000;

//balaji.eceoname="xyz";

emp.eceoname="xyz";

System.out.println("rahul info---");

rahul.showInfo();

System.out.println("Ajay info---");

Ajay.showInfo();

System.out.println("balaji info---");

balaji.showInfo();

}

}

[18/08, 8:49 am] Sanjay Sir Velocity: package Static\_non\_Static\_Use;

public class emp

{

int eid; //non-static global variable

int esal;

static String eceoname;

public void showInfo()

{

System.out.println(eid+" :"+esal+ " :"+eceoname);

}

}

[18/08, 8:49 am] Sanjay Sir Velocity: package Start\_Pattern;

public class pattern1

{

// \*

// \*

// \*

// \*

public static void main(String[] args)

{

//1 //5<=4 5

for(int i=1; i<=4; i++)

{

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

}

}

}

[18/08, 8:49 am] Sanjay Sir Velocity: package Start\_Pattern;

public class pattern2

{

// \*\*\*\*

public static void main(String[] args)

{

//1 5<=4 5

for(int i=1; i<=4; i++)

{

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

}

}

}

what is Object?

In the filed of java each and everything is consider to be as object.

Object is copy of class or instance class which is having state & behaviour.

where state stands for variable(data member) & behaviour stands for methods(member functions).

Characteristics of objects are:

1. State

2. Behaviour

eg. Marker

state- colour, size,price, weight

behaviour- write, throw

eg. Car

state: What the objects have: Speed,Gear, Direction, Fuel level,Engine temperature

Behaviors: Change Gear, Go faster/slower,Go in reverse, Stop, Shut-off

Student:

State: what the objects have, Student have a first name, last name, age, etc

Behavior: what the objects do, Student attend a course "Software testing"

------------------------------------------------------

Object-Oriented Programming System(OOPS)

Java programming language is very popular in software industry because of OOPS concept.

OOps concept provides 5 important pillers/principles for the language they are

1. Inheritance

2. Polymorphism

3. Encapsulation

4. Interface

5. Abstraction

--------------------------------------------------------------------------------------------------------

Section-2

--------------------------------------------------------------------------------------------------------

Inheritance:

It is one of the Oops principle where one class acquires properties of

another class with the help of 'extends' keywords is called Inheritance.

The class from where properties are acquiring/inheriting is called super/base/parent class.

The class too where properties are inherited/delivered is called sub/child class.

Inheritance takes place between 2 or more than 2 classes.

Inheritance is classified into 4 types:

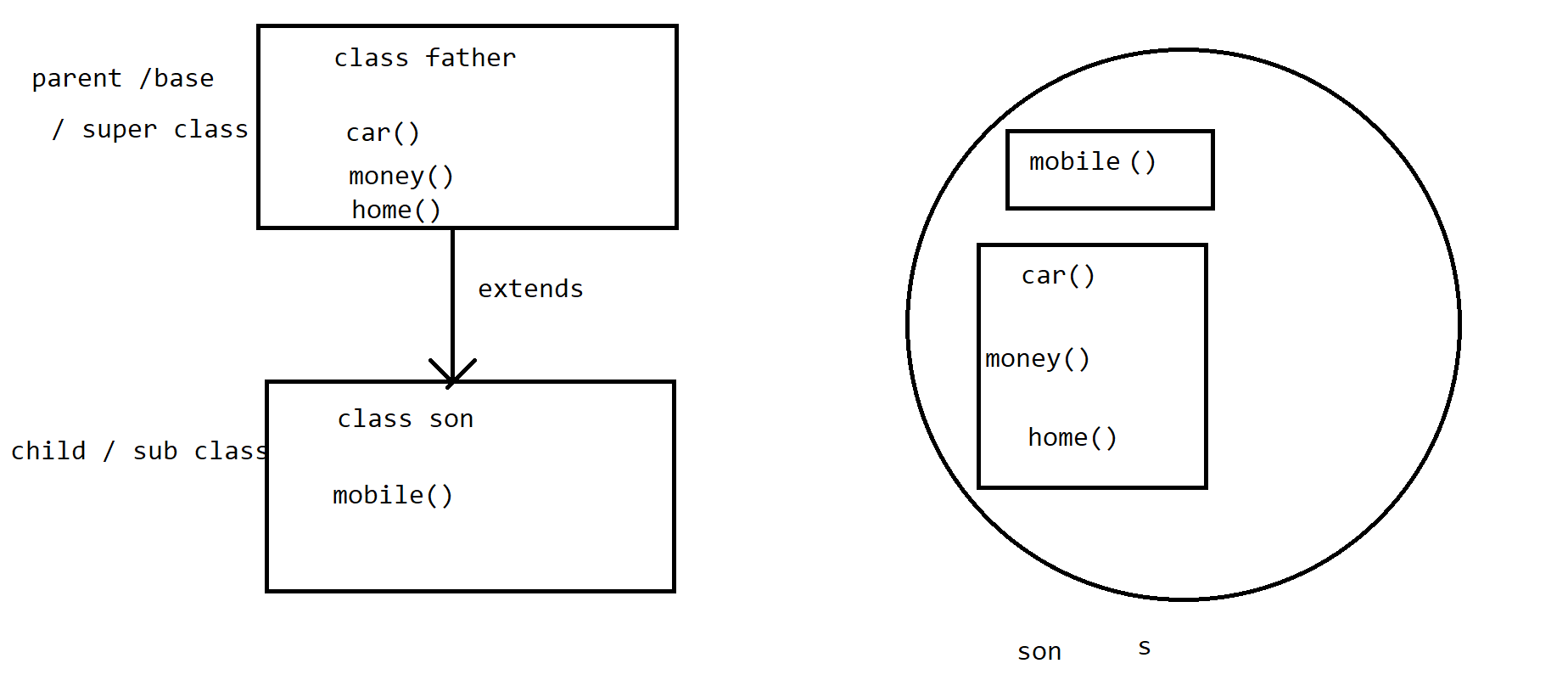
1. Singlelevel Inheritance

2. Multilevel Inheritance

3. Multiple Inheritance

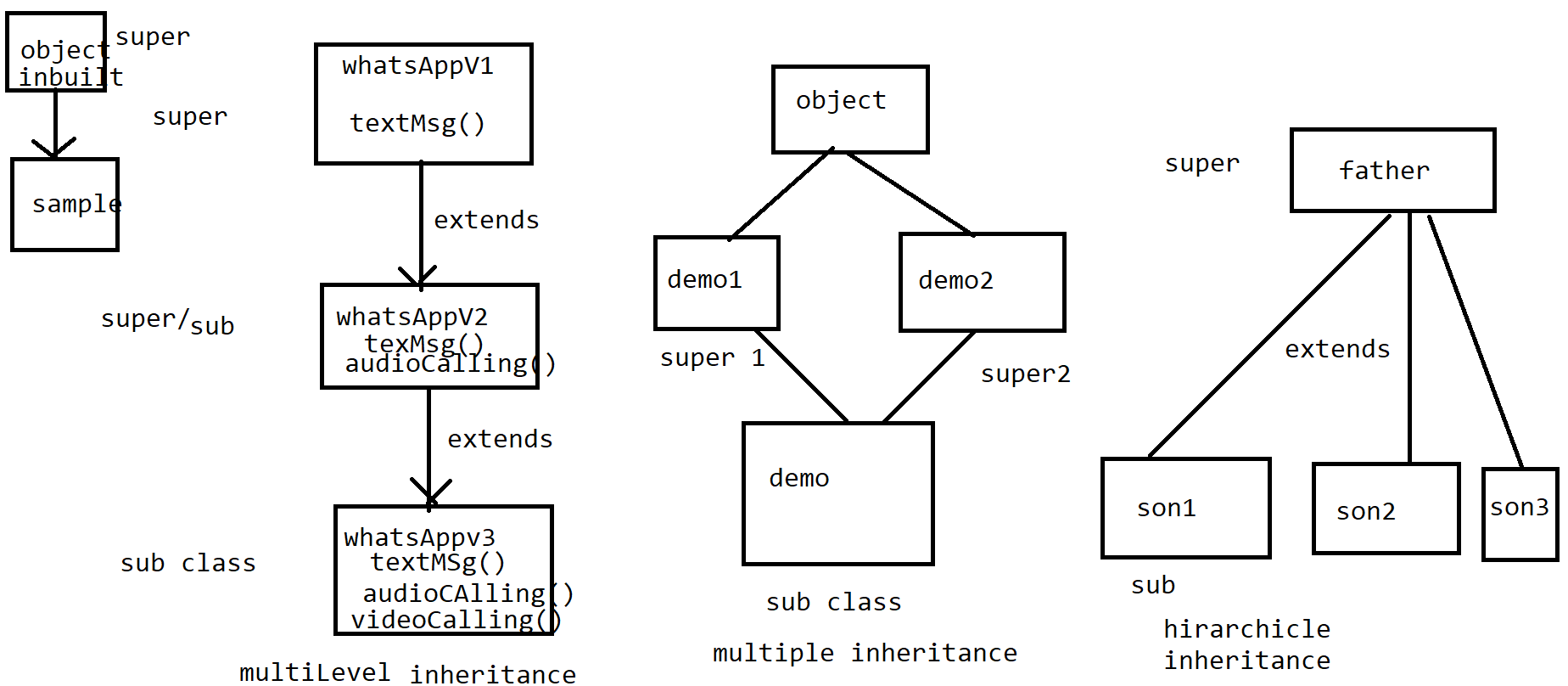
4. hirarchicle

1. Singlelevel Inheritance:



It is a operation where inheritance takes place between 2 classes.

To perform singlelevel inheritance only 2 classes are mandatory.



2. Multilevel Inheritance:

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

In Multilevel Inheritance 1 sub class acquires properties of another super class &

that class acquires properties of its another super class & phenomenon continuous.

3. Multiple Inheritance:

1 subclass acquiring properties of 2 super classes at the same time is known as Multiple Inheritance.

Java doesn't support Multiple Inheritance using class because of "dimond ambiguity" problem.

Note: object class is the super most class in java

By using interface we can achieve Multiple Inheritance.

4. Hirarchicle Inheritance:

multiple sub classes can acquire properties of 1 super class is known as hirarchicle Inheritance.

[19/08, 9:03 am] Sanjay Sir Velocity: package Inheritance;

// super/base/parent class

public class father

{

public void car()

{

System.out.println("Car: kia seltos");

}

public void money()

{

System.out.println("money: 2L");

}

public void home()

{

System.out.println("home: 2BHK");

}

}

[19/08, 9:03 am] Sanjay Sir Velocity: package Inheritance;

//sub / child class

public class son extends father

{

public void mobile()

{

System.out.println("mobile: samsung A50");

}

// public void car()

// {

// System.out.println("Car: kia seltos");

// }

//

// public void money()

// {

// System.out.println("money: 2L");

// }

//

// public void home()

// {

// System.out.println("home: 2BHK");

// }

}

[19/08, 9:03 am] Sanjay Sir Velocity: package Inheritance;

public class SingleLevel\_Inheritance {

public static void main(String[] args) {

son s=new son();

s.mobile();

s.car();

s.money();

s.home();

}

}

[19/08, 9:04 am] Sanjay Sir Velocity: package Inheritance;

public class whatsAppV1

{

public void textMsg()

{

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

}

}

[19/08, 9:04 am] Sanjay Sir Velocity: package Inheritance;

public class whatsAppV2 extends whatsAppV1

{

public void audioCalling()

{

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

}

// public void textMsg()

// {

// System.out.println("text Msg");

// }

}

[19/08, 9:04 am] Sanjay Sir Velocity: package Inheritance;

public class whatsAppV3 extends whatsAppV2

{

public void videoCalling()

{

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

}

// public void audioCalling()

// {

// System.out.println("audio Calling");

// }

// public void textMsg()

// {

// System.out.println("text Msg");

// }

}

[19/08, 9:04 am] Sanjay Sir Velocity: package Inheritance;

public class MultiLevel\_Inheritance

{

public static void main(String[] args) {

whatsAppV3 v3=new whatsAppV3();

v3.textMsg();

v3.audioCalling();

v3.videoCalling();

}

}

[20/08, 8:52 am] Sanjay Sir Velocity: package Inheritance;

// super/base/parent class

public class father

{

public void car()

{

System.out.println("Car: kia seltos");

}

public void money()

{

System.out.println("money: 2L");

}

public void home()

{

System.out.println("home: 2BHK");

}

}

[20/08, 8:52 am] Sanjay Sir Velocity: package Inheritance;

public class son1 extends father

{

public void mobile()

{

System.out.println("mobile");

}

}

[20/08, 8:52 am] Sanjay Sir Velocity: package Inheritance;

public class son2 extends father

{

public void laptop()

{

System.out.println("laptop");

}

}

[20/08, 8:52 am] Sanjay Sir Velocity: package Inheritance;

public class son3 extends father

{

public void PC()

{

System.out.println("PC");

}

}

[20/08, 8:52 am] Sanjay Sir Velocity: package Inheritance;

public class Hirarchicle\_Inheritance {

public static void main(String[] args) {

System.out.println("---properties of son1---");

son1 s1 =new son1();

s1.mobile();

s1.car();

s1.home();

s1.money();

System.out.println("---properties of son2---");

son2 s2=new son2();

s2.laptop();

s2.car();

s2.home();

s2.money();

System.out.println("---properties of son3---");

son3 s3=new son3();

s3.PC();

s3.car();

s3.home();

s3.money();

}

}

[20/08, 8:53 am] Sanjay Sir Velocity: package This\_and\_Super\_Keywords;

public class sample1

{

//example of this keyword

int a=10; //global variable

public void m1()

{

int a=20; //local variable

System.out.println(a); //20

System.out.println(this.a); //10

}

}

[20/08, 8:53 am] Sanjay Sir Velocity: package This\_and\_Super\_Keywords;

public class TestSample1

{

//example of this keyword

public static void main(String[] args) {

sample1 s1=new sample1();

s1.m1();

}

}

[20/08, 8:56 am] Sanjay Sir Velocity: package This\_and\_Super\_Keywords;

//demo1 --> super class

public class demo1

{

//example of super keyword

int b =50; //global variable from super class

}

[20/08, 8:56 am] Sanjay Sir Velocity: package This\_and\_Super\_Keywords;

//demo2--> sub class

public class demo2 extends demo1

{

//example of super keyword

//int b =50; //global variable from super class

int b=60; // global variable from sub class

public void m2()

{

int b = 70; //local variable

System.out.println(b); // 70 --> local variable call

System.out.println(this.b); // 60 --> global variable call from same/sub class

System.out.println(super.b); // 50 --> global variable call from super/diff class

}

}

[20/08, 8:56 am] Sanjay Sir Velocity: package This\_and\_Super\_Keywords;

public class TestDemo {

//example of super keyword

public static void main(String[] args) {

demo2 d2=new demo2();

d2.m2();

}

}

[20/08, 8:56 am] Sanjay Sir Velocity: This & super Keyword

1.this keyword

this keyword is use to access global variable from same class.

2. super keyword

super keyword is use to access global variable from super/diffrent class.

[23/08, 8:59 am] Sanjay Sir Velocity: package Polymorphism;

public class MethodOverloading

{

public void addition(int a, int b)

{

int sum=a+b;

System.out.println(sum);

}

public void addition(int a, int b, int c)

{

int sum = a+b+c;

System.out.println(sum);

}

}

[23/08, 8:59 am] Sanjay Sir Velocity: package Polymorphism;

public class TestMethodOverloading {

public static void main(String[] args) {

MethodOverloading mo=new MethodOverloading();

mo.addition(10,20,30);

mo.addition(20, 30);

}

}

[23/08, 9:00 am] Sanjay Sir Velocity: package Polymorphism;

public class father

{

public void money()

{

System.out.println("money: 100");

}

public void car()

{

System.out.println("car");

}

}

[23/08, 9:00 am] Sanjay Sir Velocity: package Polymorphism;

public class son extends father

{

//method overriding

public void money()

{

System.out.println("money: 50");

}

// public void car()

// {

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

// }

}

[23/08, 9:00 am] Sanjay Sir Velocity: package Polymorphism;

public class TestMethodOverriding {

public static void main(String[] args) {

son s=new son();

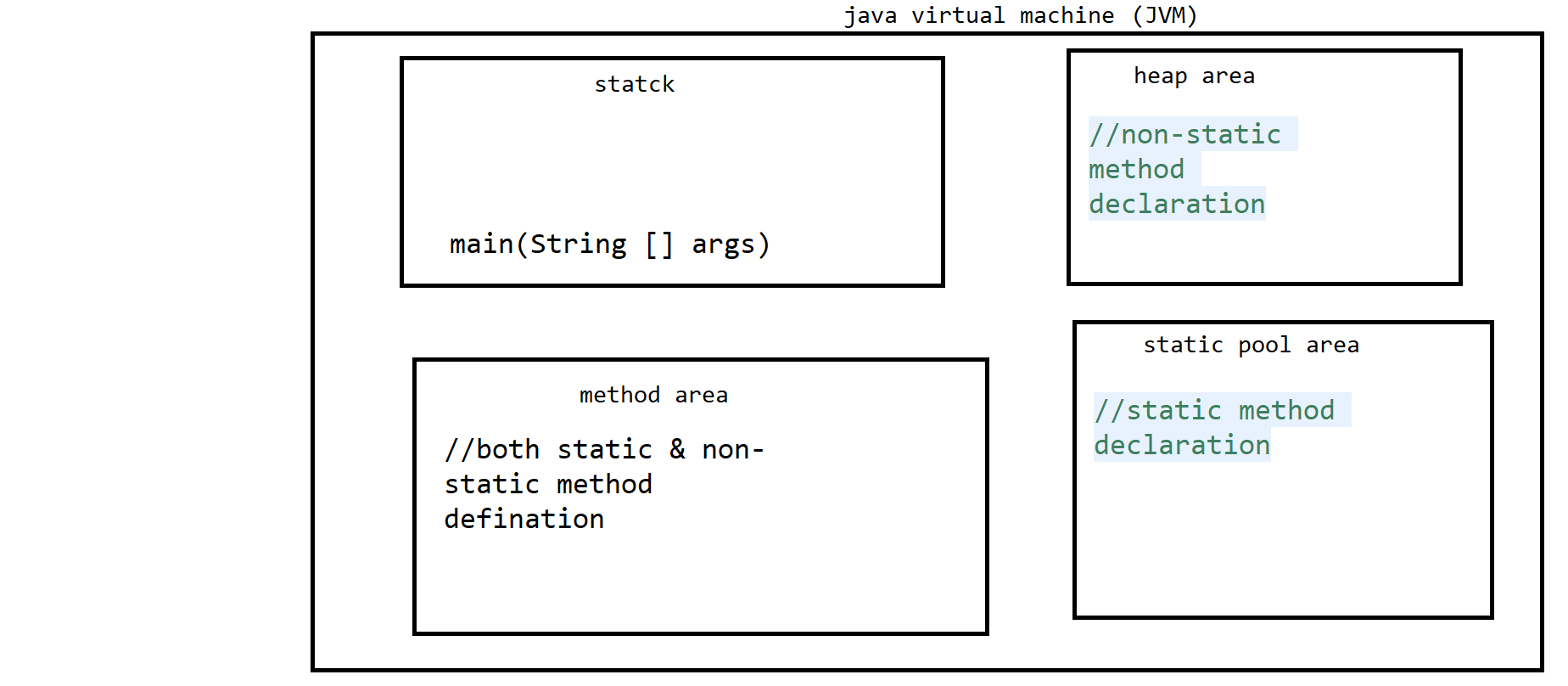
s.money();

s.car();

}

}

[23/08, 9:01 am] Sanjay Sir Velocity: diff types of JVM memories:



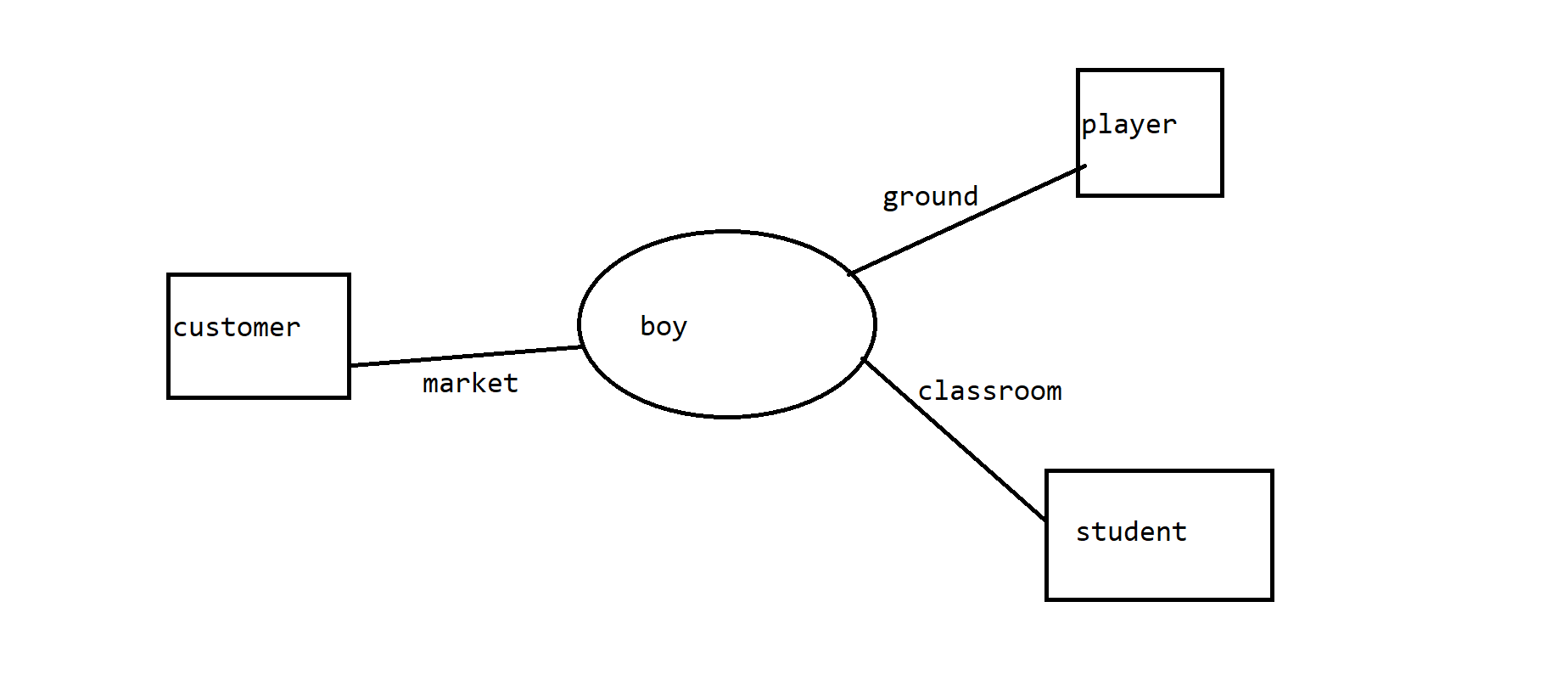
1. Heap area--> non-static method declaration

2. Static pool area--> static method declaration

3. method area --> static & non-static method defination

4. stack --> main()--> method execution flow

Polymorphism:



It is one of the OOPs principle where one object showing diffrent behaviour at diffrent stages of life cycle.

Polymorphism is an latin word where poly stand for many & morphism stands for forms.

In java Polymorphism is classified into 2 types:

1. Compiletime Polymorphism

2. Runtime Polymorphism

1. Compiletime Polymorphism:

In Compiletime Polymorphism method declaration is going to get binded to its defination at compilation time,

based on argument/input/parameter is known as compiletime Polymorphism.

As binding takes during compilation time only, so it is also known as early binding.

//once binding is done, again rebinding can't be done, so it is called static binding.

Method overloading is an example of compiletime Polymorphism.

2. Runtime Polymorphism:

In Runtime Polymorphism method declaration is going to get binded to its defination at Runtime/execution time,

based on object creation is known as runtime Polymorphism.

As binding takes during Runtime/execution time, so it is also known as late binding.

//once binding is done, again rebinding can be done, so it is called dynamic binding.

Method overriding is an example of Runtime Polymorphism.

Method overloading:

Declaring multiple method with same method name but with diffrent argument/inputs/parameter in a same class is called

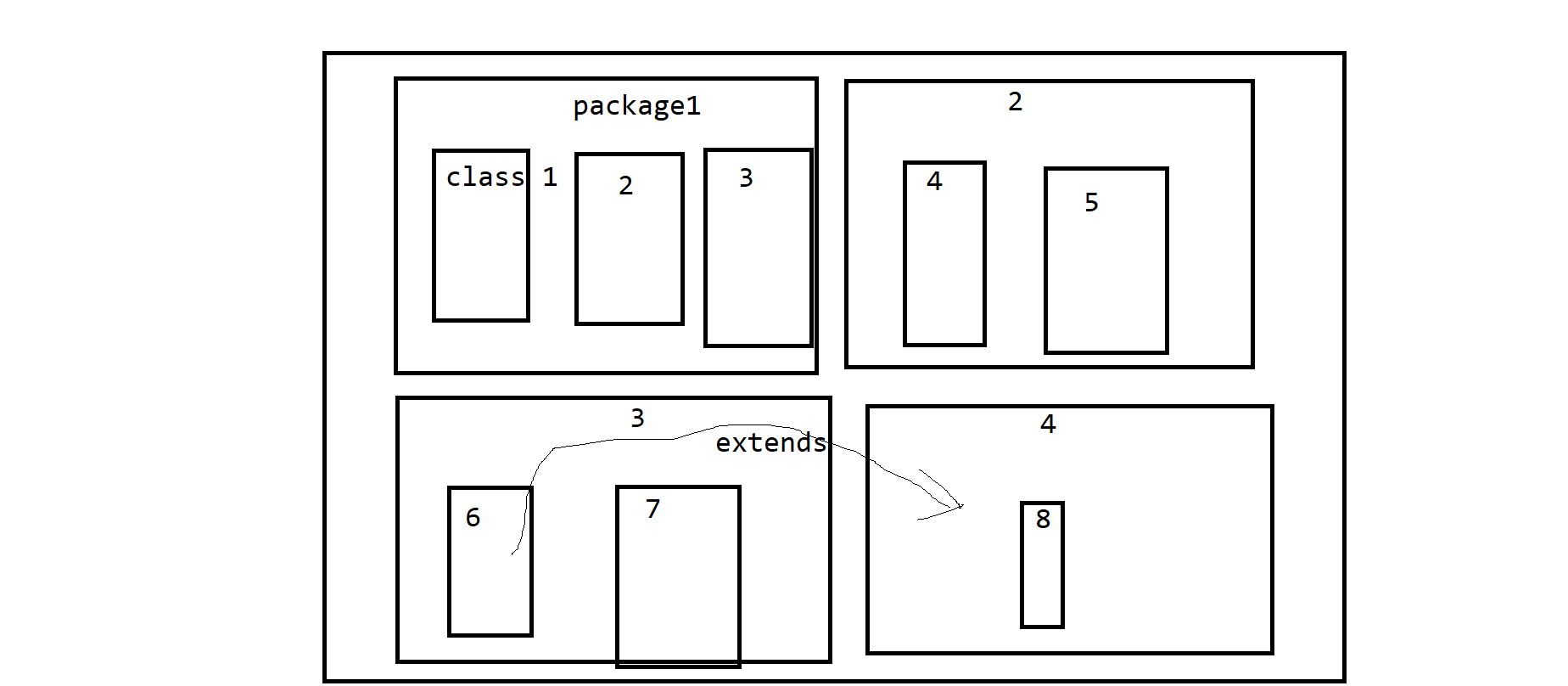
method overloading

Method overriding:

Acquiring super class method into sub class with the help of extends keyword & changing implementaion/defination

according to subclass specification is called method overriding

Access specifiers:



Access specifiers are use to represent scope of members of class.

In java Access specifiers are classified into 4 types

1. private

2. default

3. protected

4. public

1. private: If you declare any member of class as private then scope of that member remains only within the class

It can't be access from other classes.

2. default: If you declare any member of class as default then scope of that member remains only within the package

It can't be access from other packages.

There is no keyword to represent default access specifier.

3. protected: If you declare any member of class as protected then scope of that member remains only within the package

that class which is present outside the package can access it by one condition ie. inheritance operation

4. public: If you declare any member of class as public then scope of that member remains throught the project.

Abstract Class:

A class declared with "abstract" keyword is called abstract class.

an Abstract class is nothing but an incomplete class where programer

can declare complete as well as incomplete methods in it.

programer can declare incomplete methods as abstract method, by declaring keyword

called "abstract" infront of method.

we can't create object of abstract class, to create object of abstract class we need

to make use of concrete class.

Concrete class:

A class which provides definations for all the incomplete methods which are

present in abstarct class with the help of extends keywords is called concrete class.

[25/08, 8:44 am] Sanjay Sir Velocity: package Abstarct\_Concrete\_Class;

//incomplete class---> abstarct class

abstract public class sample1

{

//complete method

public void m1() //method declaration

{

System.out.println("method m1: completed in abstarct class"); //method defination

}

//incomplete method

abstract public void m2(); //method declaration

//incomplete method

abstract public void m3(); //method declaration

}

[25/08, 8:44 am] Sanjay Sir Velocity: package Abstarct\_Concrete\_Class;

//concrete class

public class sample2 extends sample1

{

// public void m1()

// {

// System.out.println("running m1 method"); //method defination

// }

public void m2()

{

System.out.println("method m2: completed in concrete class");

}

public void m3()

{

System.out.println("method m3: completed in concrete class");

}

}

[25/08, 8:44 am] Sanjay Sir Velocity: package Abstarct\_Concrete\_Class;

public class TestSample {

public static void main(String[] args) {

sample2 s2=new sample2();

s2.m1();

s2.m2();

s2.m3();

}

}

[25/08, 8:44 am] Sanjay Sir Velocity: package Start\_Pattern;

public class pattern3 {

public static void main(String[] args) {

// \*\*\*\*

// \*\*\*\*

// \*\*\*\*

//outer for loop for rows

//4<=3 4

for(int i=1; i<=3; i++)

{

//inner for loop cols

//1<=4

for(int j=1; j<=4; j++)

{

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

}

System.out.println();

}

}

}

[25/08, 8:44 am] Sanjay Sir Velocity: package Start\_Pattern;

public class pattern4 {

//\*\*\*\*\*\*

//\*\*\*\*\*\*

//\*\*\*\*\*\*

//\*\*\*\*\*\*

public static void main(String[] args) {

//outer for loop for rows

for(int i=1; i<=4; i++)

{

//inner for loop for cols

for(int j=1; j<=6; j++)

{

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

}

System.out.println();

}

}

}

Interface:

It is one of the oops principle.

It is pure 100% abstract in nature.

Interface is use to declare only incomplete methods in it.

Features of Interface:

1. Data Members/variable declared inside Interface are by default static and final.

2. methods declared inside Interface are by default public & abstract.

3. constructor concept in not present inside Interface.

4. object of Interface can't be created.

5. Interface support multiple inheritance.

6. To create object of Interface programmer need to make use of Implementation class using implements keyword.

Implementation class:

A class which provides definations for all the incomplete methods which are

present in interface with the help of "implements" keyword is called Implementation class.

[26/08, 8:40 am] Sanjay Sir Velocity: package Interface\_ImplementationClass;

//demo1 -->interface name

public interface demo1

{

int num=10; //variables are by default static & final

void m1(); //methods are by default public & abstract

void m2();

}

[26/08, 8:40 am] Sanjay Sir Velocity: package Interface\_ImplementationClass;

//implementation class

public class demo2 implements demo1

{

public void m1()

{

System.out.println("method m1 completed in implementation class");

}

public void m2()

{

System.out.println("method m2 completed in implementation class");

}

}

[26/08, 8:41 am] Sanjay Sir Velocity: package Interface\_ImplementationClass;

public class TestDemo2 {

public static void main(String[] args) {

//example of interface & implementation class

demo2 d2 =new demo2();

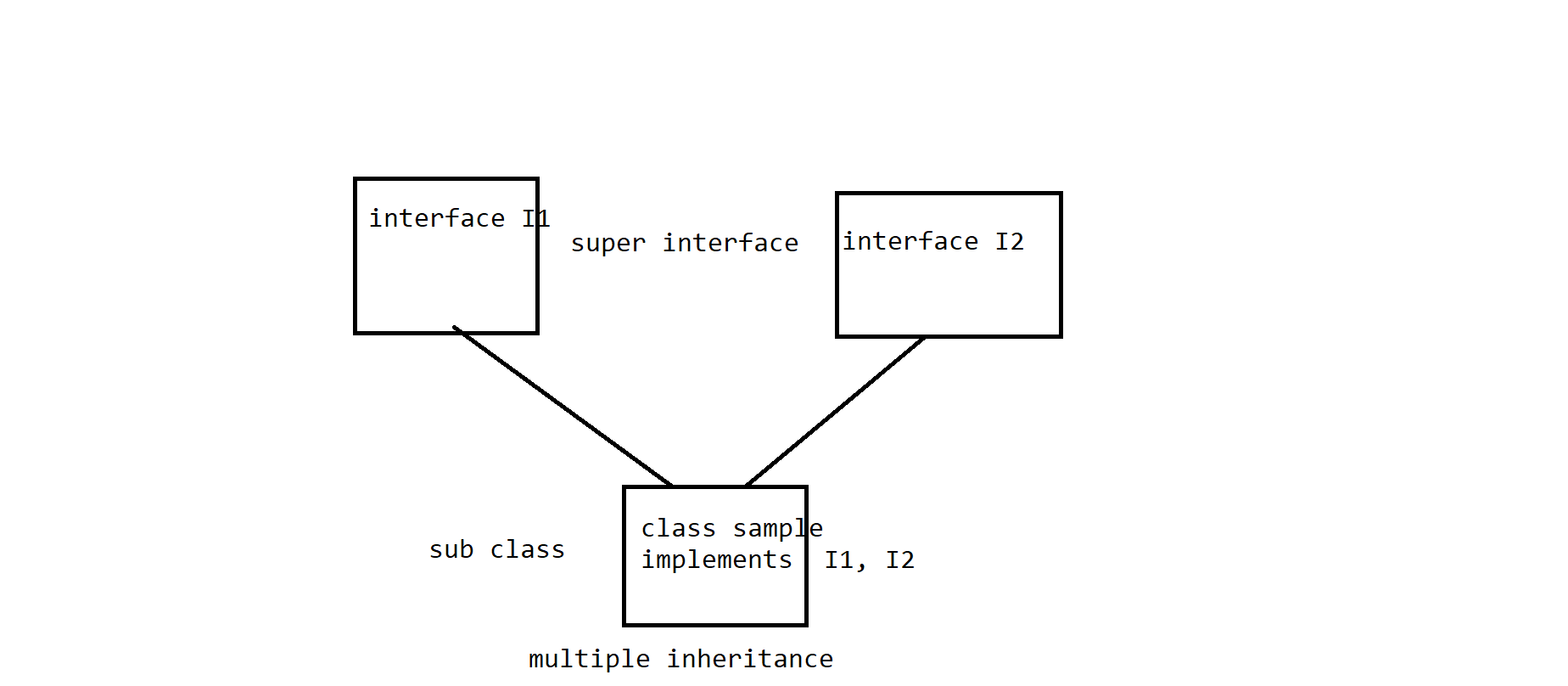
d2.m1();

d2.m2();

}

}

Multiple inheritance:-



[26/08, 8:41 am] Sanjay Sir Velocity: package Interface\_ImplementationClass;

//super interface 1

public interface I1

{

void m3();

void m4();

}

[26/08, 8:41 am] Sanjay Sir Velocity: package Interface\_ImplementationClass;

//super interface 2

public interface I2

{

void m5();

}

[26/08, 8:41 am] Sanjay Sir Velocity: package Interface\_ImplementationClass;

//sub class or implementation class

public class Sample2 implements I1, I2

{

//example of multiple inheritance using interface

public void m3() {

System.out.println("running method m3 from Interface I1");

}

public void m4() {

System.out.println("running method m4 from Interface I1");

}

public void m5() {

System.out.println("running method m5 from Interface I2");

}

}

[26/08, 8:41 am] Sanjay Sir Velocity: package Interface\_ImplementationClass;

public class MultipleInheritance

{

//example of multiple inheritance using interface

public static void main(String[] args) {

Sample2 s2=new Sample2();

s2.m3();

s2.m4();

s2.m5();

}

}

Casting:

Converting one type of information into another type is called casting

In java casting is classified into 2 types:

1. primitive casting

2. non-primitive casting

1. Primitive-casting:

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

primitive-casting is classified into 3 types:

1. implicit casting

2. explicit casting

3. boolean casting

1. implicit casting:

converting lower data type info into higher data type info is called implicit casting.

implicit casting is also called widning casting, where memory size goes on incresing,

eg.

int a=5 // (memory size of int is 4 byte)

sop(a) // 5

double b = a //(memory size of double is 8 byte)

sop(b) //5.0

2. explicit casting:

converting higher data type info into lower data type info is called explicit casting.

explicit casting is also called narrowing casting, where memory size goes on decresing.

In explicit casting data loss takes place

eg.

double b = 2.5 //(memory size of double is 8 byte)

sop(b) //2.5

int a= (int)b // (memory size of int 4 byte)

sop(a) //2

3. boolean casting:

boolean casting is considered to be incompatible casting type, because boolean data type is unique type of data type

where information is already predeclared inside it.

boolean str = true

2. non-primitive casting

converting one type of class into another type of class is called non-primitive casting.

non-primitive is classified into 2 types:

\*\* 1. up casting

2. down casting

1. up casting:



Assigning subclass property into superclass is called upcasting.

before performing upcasting 1st we need to perform inheritance operation.

after performing inheritance, the property which are present inside superclass comes into subclass

In the subclass programer can declare new properties.

At the time of upcasting operation the properties which are inherited from superclass are only eligible for the upcasting operation.

The new property which were declared inside subclass are not eligible for upcasting operation.

2. down casting:

Assigning superclass property into subclass is called downcasting.

before performing downcasting 1st we need to perform upcasting.

[27/08, 9:00 am] Sanjay Sir Velocity: package Upcasting;

// super/base/parent class

public class father

{

public void car()

{

System.out.println("Car: Honda city");

}

public void money()

{

System.out.println("money: 1L");

}

public void home()

{

System.out.println("home: 2BHK");

}

}

[27/08, 9:00 am] Sanjay Sir Velocity: package Upcasting;

//sub / child class

public class son extends father

{

public void mobile() //original method of son clas

{

System.out.println("mobile: samsung A50");

}

public void car() //overriding

{

System.out.println("Car: kia seltos");

}

public void money() //overriding

{

System.out.println("money: 0.75L");

}

// public void home()

// {

// System.out.println("home: 2BHK");

// }

}

[27/08, 9:00 am] Sanjay Sir Velocity: package Upcasting;

public class UpCasting

{

public static void main(String[] args) {

father s=new son(); //upcasting

s.car();

s.money();

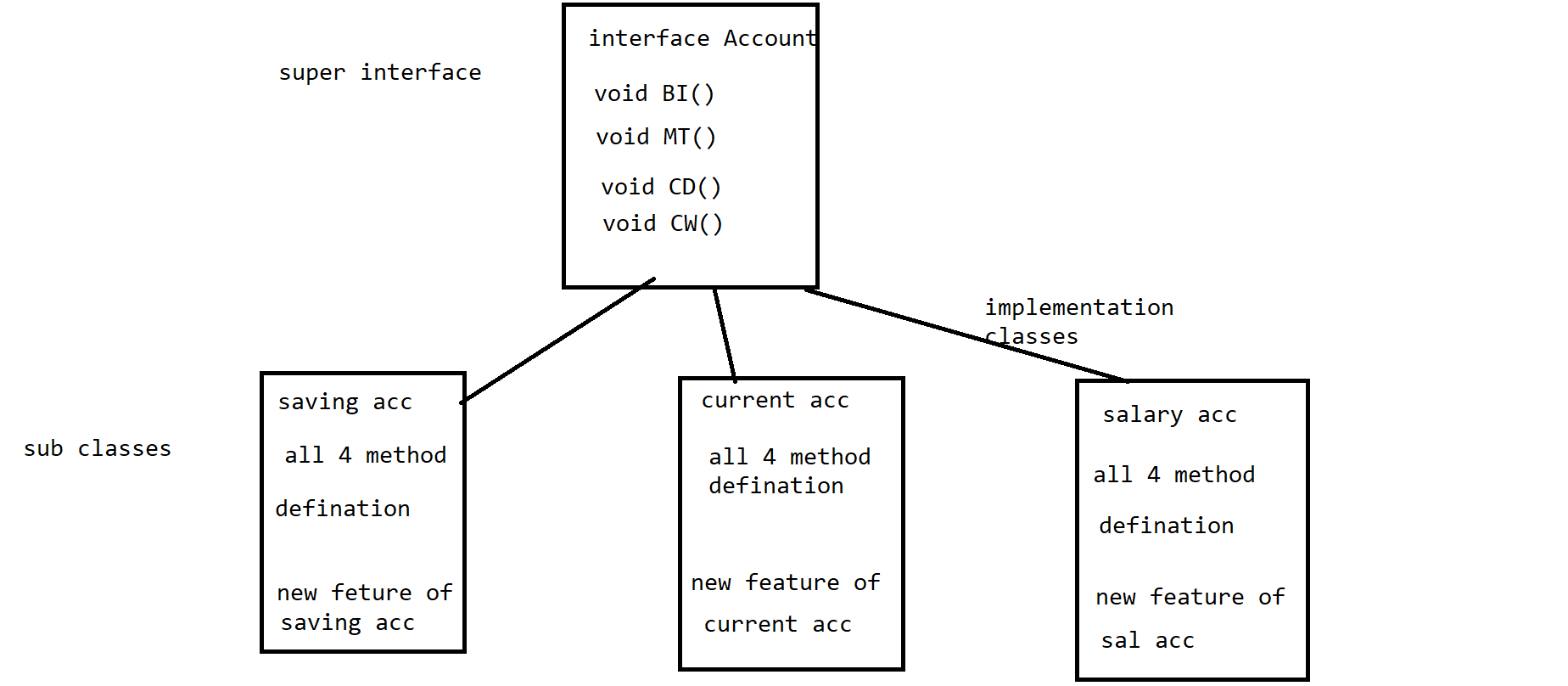
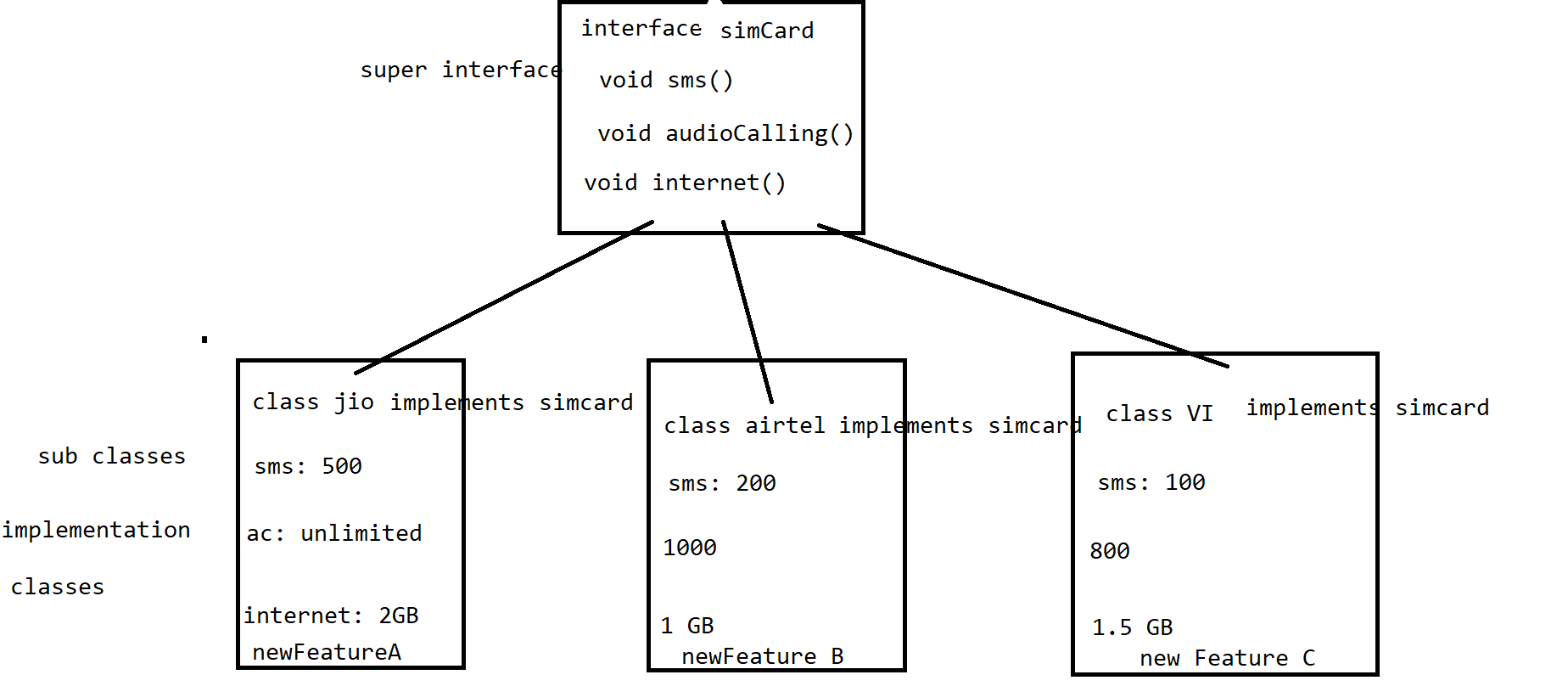
s.home();

//create object of sub class provide reference/type of super class

}

}

Generalization:



Extracting all the important common properties & declaring it in super class(ie. super interface) & providing

implementation/defination according to subclass specification is called Generalization.

generalization file can be normal java class or abstract class or Interface, but only Interface is recommonded.

Abstraction:

abstraction is one of the oops principle in java.

Hiding the implementation code and providing only functionality to the end user is called abstraction.

The scenario of Abstraction is "if customer is visiting or making use of any application,

then he should utilize functionality only & he should not feel any backend code processing"

[30/08, 8:46 am] Sanjay Sir Velocity: package Generalization;

public class Jio implements SimCard

{

public void sms()

{

System.out.println("sms: 500");

}

public void audioCalling() {

System.out.println("Audio calling: unlimited");

}

public void internet() {

System.out.println("Internet: 2GB");

}

public void newFeatureA() {

System.out.println("newFeature: A");

}

}

[30/08, 8:47 am] Sanjay Sir Velocity: package Generalization;

public class Airtel implements SimCard

{

public void sms()

{

System.out.println("sms: 200");

}

public void audioCalling() {

System.out.println("Audio calling: 1000");

}

public void internet() {

System.out.println("Internet: 1GB");

}

public void newFeatureB() {

System.out.println("newFeature: B");

}

}

[30/08, 8:47 am] Sanjay Sir Velocity: package Generalization;

public class VI implements SimCard

{

public void sms()

{

System.out.println("sms: 100");

}

public void audioCalling() {

System.out.println("Audio calling: 800");

}

public void internet() {

System.out.println("Internet: 1.5 GB");

}

public void newFeatureC() {

System.out.println("newFeature: C");

}

}

[30/08, 8:47 am] Sanjay Sir Velocity: package Generalization;

public interface SimCard

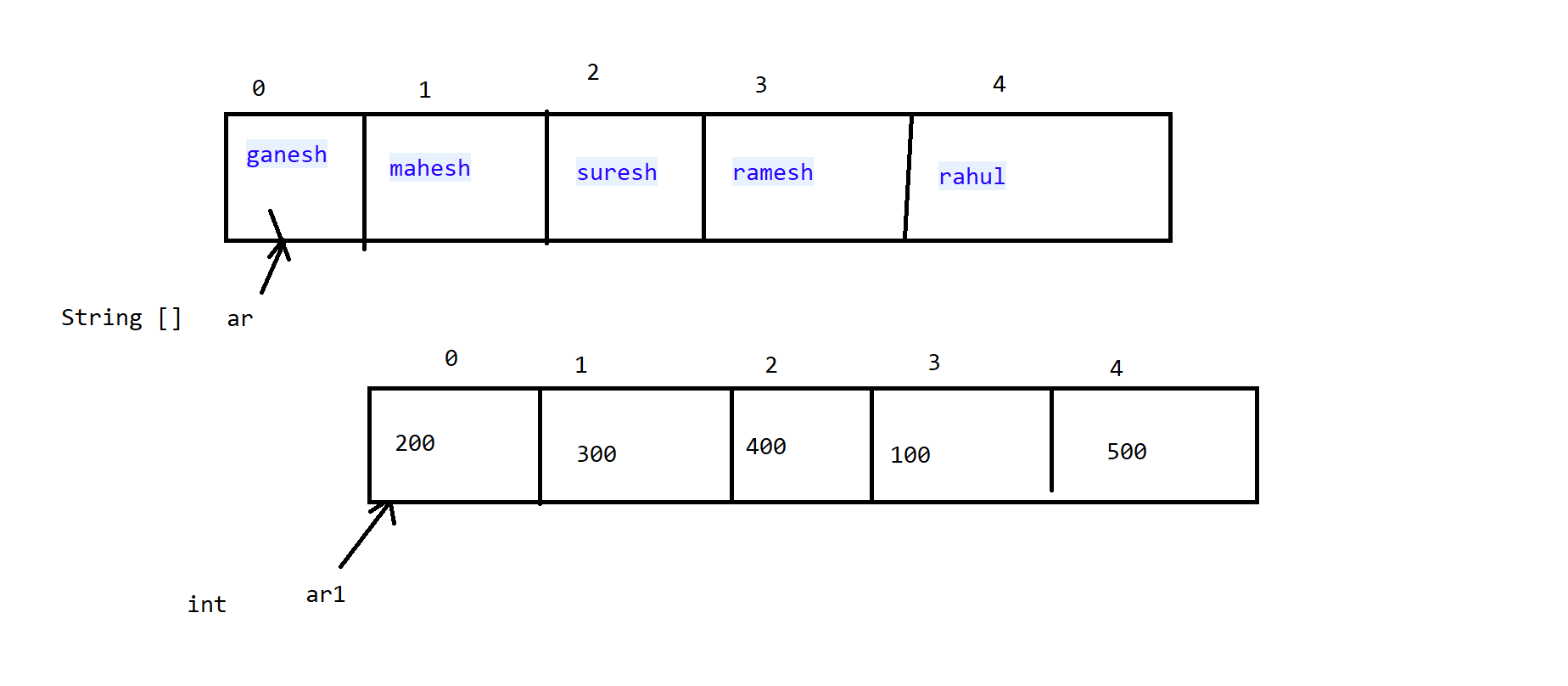
{

void sms();

void audioCalling();

void internet();

}



[31/08, 9:01 am] Sanjay Sir Velocity: package Array;

public class example1\_StringArray

{

public static void main(String[] args) {

//step1: Array declaration

String [] ar =new String[5];

//step2: Array initialization

ar[0] = "ganesh";

ar[1] = "mahesh";

ar[2] = "suresh";

ar[3] = "ramesh";

ar[4] = "rahul";

//ar[5] = "rohit";

System.out.println(ar.length); //5

//step3: usage

System.out.println(ar[2]);

System.out.println("-------Print all info in String array---------");

//5<=4 5

// for(int i=0; i<=4; i++)

// { //ar[4]

// System.out.println(ar[i]);

// }

for(int i=0; i<=ar.length-1; i++)

{ //ar[4]

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

}

}

}

[31/08, 9:01 am] Sanjay Sir Velocity: package Array;

public class example2\_intArray {

public static void main(String[] args) {

int [] ar1=new int[5];

ar1[0]=200;

ar1[1]=300;

ar1[2]=400;

ar1[3]=100;

ar1[4]=500;

//ar1[5]=600; exception

System.out.println(ar1.length);

System.out.println(ar1[3]);

System.out.println("---print all info in int array----");

// for(int i=0; i<=4; i++)

// {

// System.out.println(ar1[i]);

// }

for(int i=0; i<=ar1.length-1; i++)

{

System.out.println(ar1[i]);

}

}

}

[31/08, 9:01 am] Sanjay Sir Velocity: package Array;

public class example3\_printArrayIn\_ReverseOrder

{

public static void main(String[] args) {

int [] ar1=new int[5];

ar1[0]=200;

ar1[1]=300;

ar1[2]=400;

ar1[3]=100;

ar1[4]=500;

// for(int i=4; i>=0; i--)

// {

// System.out.println(ar1[i]);

// }

for(int i=ar1.length-1; i>=0; i--)

{

System.out.println(ar1[i]);

}

}

}

[31/08, 9:01 am] Sanjay Sir Velocity: package Array;

import java.util.Arrays;

public class example4\_Sort\_intArray

{

public static void main(String[] args) {

int [] ar1=new int[5];

ar1[0]=200; //100

ar1[1]=300; //200

ar1[2]=400; //300

ar1[3]=100; //400

ar1[4]=500; //500

System.out.println("----print original info-----");

for(int i=0; i<=ar1.length-1; i++)

{

System.out.println(ar1[i]);

}

System.out.println("----print info in asscending order-----");

Arrays.sort(ar1);

for(int i=0; i<=ar1.length-1; i++) {

System.out.println(ar1[i]);

}

System.out.println("----print info in desscending order-----");

for(int i=ar1.length-1; i>=0; i--) {

System.out.println(ar1[i]);

}

}

}

[31/08, 9:01 am] Sanjay Sir Velocity: package Array;

import java.util.Arrays;

public class example5\_Sort\_StringArray

{

public static void main(String[] args) {

String [] ar =new String[5];

ar[0] = "ganesh";

ar[1] = "mahesh";

ar[2] = "suresh";

ar[3] = "ramesh";

ar[4] = "rahul";

Arrays.sort(ar);

for(int i=0; i<=ar.length-1; i++)

{

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

}

}

}

[01/09, 8:44 am] Sanjay Sir Velocity: package Array;

public class example6\_intArray\_declaration\_initialization\_in\_singlestep {

public static void main(String[] args) {

//array declaration & initialization in single step

int [] ar= {50,30,20,40,10,60};

System.out.println(ar.length);

System.out.println(ar[5]);

System.out.println("----print all info from array------");

for (int i = 0; i <= ar.length-1; i++) {

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

}

}

}

[01/09, 8:44 am] Sanjay Sir Velocity: package Array;

public class example7\_StringArray\_declaration\_initialization\_in\_singlestep {

public static void main(String[] args) {

String [] ar= {"rahul","mahesh", "ganesh","ramesh"};

System.out.println(ar.length);

System.out.println(ar[1]);

System.out.println("---------print all info-------------");

for (int i = 0; i <= ar.length-1; i++) {

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

}

}

}

[01/09, 8:44 am] Sanjay Sir Velocity: package Array;

public class example8

{// 0 1 2

//0 10 20 30

//1 40 50 60

public static void main(String[] args) {

//step1: array declaration

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

//step2: array initialization

ar[0][0]=10;

ar[0][1]=20;

ar[0][2]=30;

ar[1][0]=40;

ar[1][1]=50;

//ar[1][2]=60;

//step3: usage

System.out.println(ar.length); //print only row size

System.out.println(ar[1][2]);

System.out.println("---print all info from array---");

//outer for loop for rows

//2<=1 2

for(int i=0; i<=1; i++)

{

//inner for loop for col

//3<=2 3

for(int j=0; j<=2; j++)

{ // 1 2

System.out.print(ar[i][j]+" "); //10 20 30

} // 40 50 60

System.out.println(); //

}

}

}

[01/09, 8:45 am] Sanjay Sir Velocity: package Array;

public class example9 {

public static void main(String[] args) {

//0 1

int [][] ar= {{10,20,30},{40,50,60}};

//outer for loop for rows

for(int i=0; i<=1; i++)

{

//inner for loop for col

for (int j = 0; j <=2; j++)

{

System.out.print(ar[i][j]+ " ");

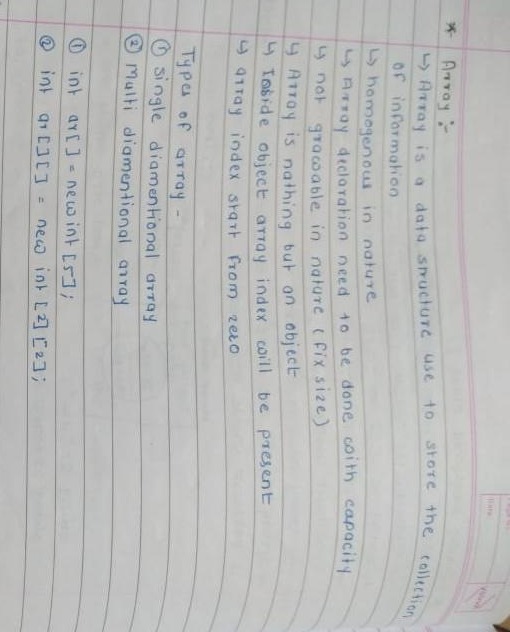
}

System.out.println();

}

}

}



[02/09, 8:56 am] Sanjay Sir Velocity: package StringClass\_Methods;

public class sample1 {

public static void main(String[] args) {

String str="abc";

str=str+"d"; //abcd

System.out.println(str);

//1. create object of string without using new keyword

String s1="xyz";

//2. create object of string using new keyword

String s2=new String("xyz1");

}

}

[02/09, 8:56 am] Sanjay Sir Velocity: package StringClass\_Methods;

public class sample2

{

public static void main(String[] args) {

//1. create object of string without using new keyword

String s1="xyz";

String s2="xyz";

String s3="xyz1";

//2. create object of string using new keyword

String s4=new String("xyz");

String s5=new String("xyz");

System.out.println(s1==s2); //true

System.out.println(s1==s3); //false

System.out.println(s1==s4); //false

System.out.println(s4==s5); //false

}

}

[02/09, 8:56 am] Sanjay Sir Velocity: package StringClass\_Methods;

public class StringClass\_Methods

{

public static void main(String[] args) {

String s1="velocity";

String s2="VELOCITY";

String s3="";

System.out.println(s1.length()); //8

System.out.println(s1.toUpperCase()); //VELOCITY

//s1=s1.toUpperCase(); //re-initialization

System.out.println(s2.toLowerCase()); //velocity

//s2=s2.toLowerCase();

System.out.println(s1.isEmpty()); //false

System.out.println(s3.isEmpty()); //true

}

}

[02/09, 8:57 am] Sanjay Sir Velocity: String class:

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

2. String is use to store collection of characters.

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

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

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

6. String objects are immutable in nature/cant be change.

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

1. without using new keyword

2. using new keyword

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

String pool area:

It is use to store String objects.

It is classified into 2 areas:

1. constant pool area

2. non-constant pool area.

1. constant pool area:

1. During object creation time if you don't make use of new keyword then

object creation takes place inside constant pool area.

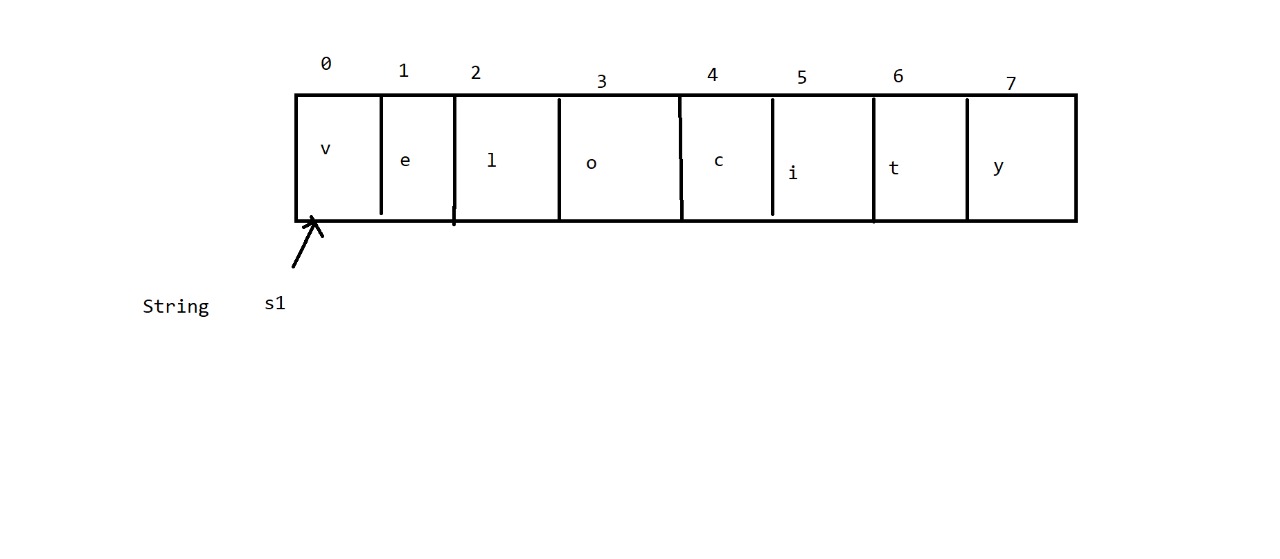
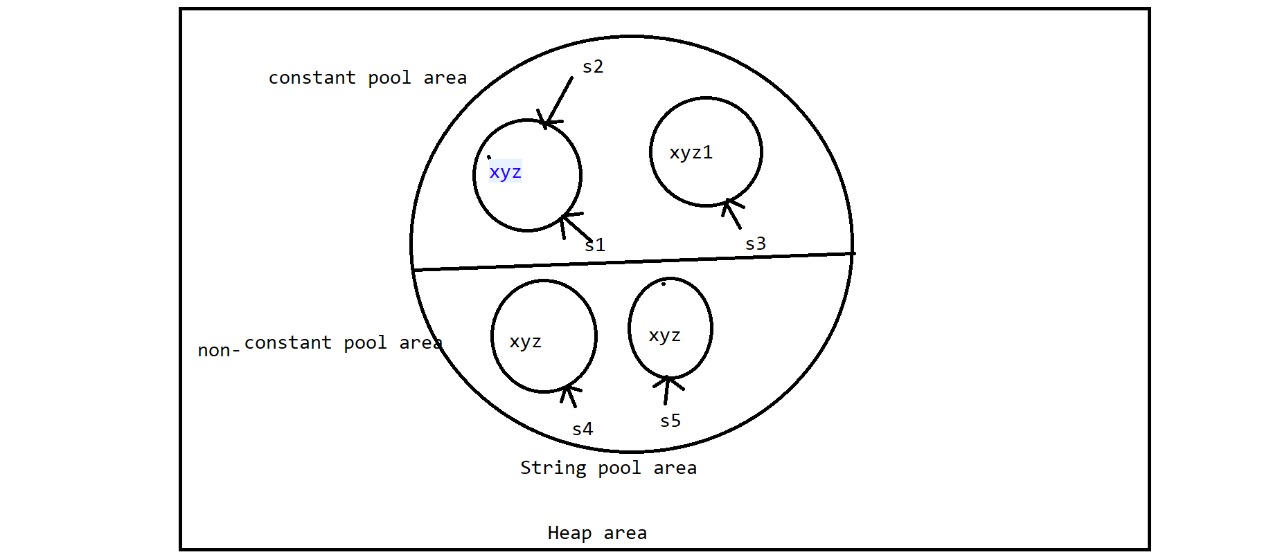
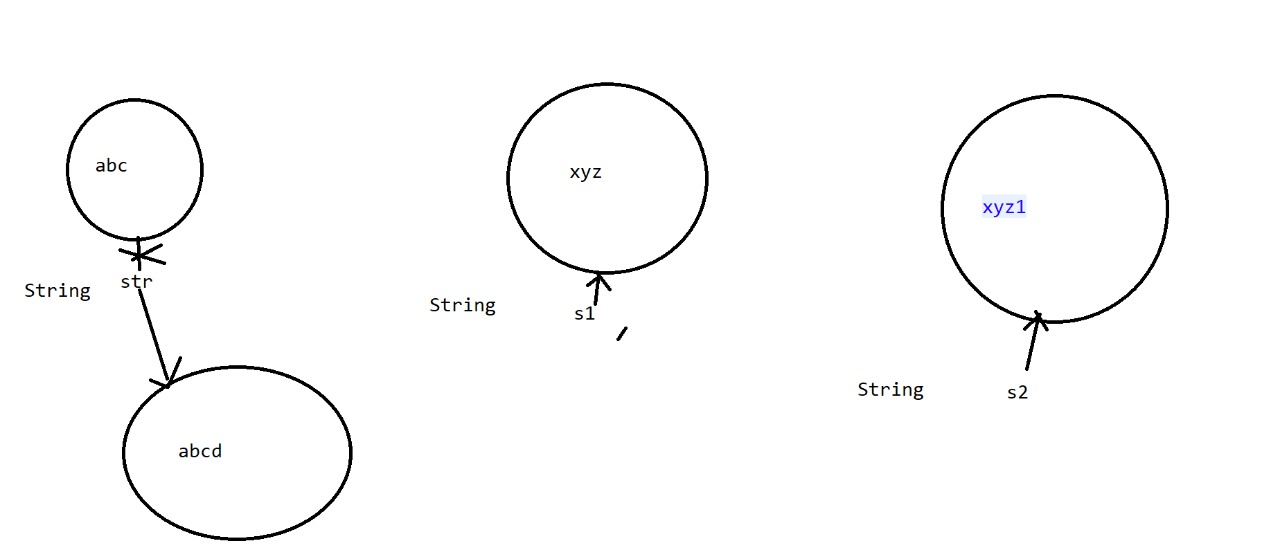
Duplicate objects are not allowded inside constant pool area.

2. non-constant pool area:

2. During object creation time if you make use of new keyword then

object creation takes place inside non-constant pool area.

Duplicate objects are allowded inside non-constant pool area.



package StringClass\_Methods;

public class StringClass\_Methods

{

public static void main(String[] args) {

String s1="velocity";

String s2="ABCD";

String s3="";

String s4="VELOCITY";

String s5 ="city";

String s6 ="abcabcab";

String s7 = "java classes";

System.out.println(s1.substring(4,6)); //ci

System.out.println(s1.substring(6)); //city

System.out.println(s7.replace("java", "selenium"));

System.out.println(s2.concat(s5)); //ABCDcity

System.out.println(s2+s5); //ABCDcity

System.out.println(s1.endsWith("city")); //true

System.out.println(s1.startsWith("ve")); //true

System.out.println(s6.lastIndexOf('a')); //6

System.out.println(s6.indexOf('b')); //1

System.out.println(s1.charAt(5)); //i

System.out.println(s1.contains(s5)); //true

System.out.println(s1.equalsIgnoreCase(s4)); //true

System.out.println(s1.equals(s4)); //false

System.out.println(s1.length()); //8

System.out.println(s1.toUpperCase()); //VELOCITY

// s1=s1.toUpperCase(); //re-initialization

// System.out.println(s1);

System.out.println(s2.toLowerCase()); //abcd

// s2=s2.toLowerCase(); //re-initialization

// System.out.println(s2);

System.out.println(s1.isEmpty()); //false

System.out.println(s3.isEmpty()); //true

}

}

[04/09, 8:54 am] Sanjay Sir Velocity: package ExceptionHandling;

public class example1 {

public static void main(String[] args) {

int num1=10;

int num2=0;

int div = 0;

try

{

div= num1/num2; //10/2; 5

}

catch(ArithmeticException e)

{

System.out.println("arithmatic exception handled");

div=num1/1; //10/1 =10

}

System.out.println(div);

System.out.println("Hi");

}

}

[04/09, 8:54 am] Sanjay Sir Velocity: package ExceptionHandling;

public class example2 {

public static void main(String[] args) {

System.out.println("GM");

int [] ar= new int[5];

try

{

ar[2]=10;

}

catch(ArrayIndexOutOfBoundsException a)

{

System.out.println("ArrayIndexOutOfBounds Exception handled");

System.out.println("wrong array index");

}

System.out.println("Hi");

}

}

[04/09, 8:54 am] Sanjay Sir Velocity: package ExceptionHandling;

public class example3 {

public static void main(String[] args) {

int num1=10;

int num2=0;

int div = 0;

try

{

div= num1/num2;

}

catch (NullPointerException e)

{

System.out.println("NullPointer Exception handled");

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("ArrayIndexOutOfBounds Exception handled");

}

catch(ArithmeticException e)

{

System.out.println("arithmatic exception handled");

}

System.out.println(div);

System.out.println("Hi");

}

}

[04/09, 8:54 am] Sanjay Sir Velocity: package ExceptionHandling;

public class example4 {

public static void main(String[] args) {

int num1=10;

int num2=0;

int div = 0;

try

{

div= num1/num2; //risky code

}

catch(Exception a) ///generic exception

{

System.out.println("generic exception handled");

a.printStackTrace();

}

System.out.println(div);

System.out.println("Hi");

}

}

[04/09, 8:54 am] Sanjay Sir Velocity: package ExceptionHandling;

public class example5 {

public static void main(String[] args) {

int num1=10;

int num2=0;

int div = 0;

try

{

div= num1/num2; //risky code

}

catch (ArrayIndexOutOfBoundsException e)

{

System.out.println("ArrayIndexOutOfBounds Exception handled");

}

catch(NullPointerException e)

{

System.out.println("NullPointer Exception handled");

}

catch(Exception a) ///generic exception

{

System.out.println("generic exception handled");

a.printStackTrace();

}

System.out.println(div);

System.out.println("Hi");

}

}

[04/09, 8:54 am] Sanjay Sir Velocity: package ExceptionHandling;

public class example6

{

public static void main(String[] args) {

try

{

}

catch (Exception e) {

}

try

{

}

catch (Exception e) {

}

}

}

[04/09, 8:54 am] Sanjay Sir Velocity: package ExceptionHandling;

public class example7

{

public static void main(String[] args) {

try

{

try

{

}

catch (Exception e) {

}

}

catch (Exception e)

{

}

}

}