26/17/17

Core Java

- Java is a system comprises of three things

1. Java Language

    - Here we discuss about the language and their syntax level etc.

2. Java Envirnoment

    - Java architecture , compilation and interpretation process

3. Java API (Application Programming Interface)

    - Packages by which we create different application

Edition

- There are three major editions which are in java they are

1. Java SE - Java Standard Edition

    - This edition is basically takes care about the core program and their concept

    - Some of the packages which we come across in this edition

java.lang , java.util , java.net , java. util.regex, java.io

    - This edition is the prerequisite for all the advance concepts of java.

    - This we say as core java or vanilla java.

2. Java EE - Java Enterprise Edition

    - This edition using which we can create web based or distributed level application.

    - Majorly three components we learn here

Servlet , JSP(Java Server Pages) , EJB (Enterprise Java Bean)

    - Servlet & JSP - Web Component

    - EJB - Business Component

3. Java ME - Java Micro Edition

    - This edition basically used in handheld devices like mobile , SetupBox.

Version

- Java has started its version from jdk1.1 to jdk 1.9

- Each version has added certain new concept

- jdk 1.5 - Tiger edition - Generics

- jdk 1.7 - Try with Resources , String with Switch case , NIO2

- jdk 1.8 – Lambda

Java Language

Characterset

- 0-9 , a-z , A-Z , Some Special characters(!@#$%^&*()_+)

Identifiers

- Identifiers are going to identify the programming element like variable , class , interface array , method etc.

- When we create an identifier we need to follow some rules.

1. It should start with alphabet and should not start with digit.

2. It can be followed by digit.

3. It should not contain any space and special character except (underscore).

4. It should not be same as that of java keywords.

Valid Identifier : num1, student_name,studentId

Invalid Identifier : 1num , student address , student.id , int

Keywords

- These are the reserved keywords which are available for certain purpose.

1. byte, short,int ,long, float , double ,char , boolean , void - Data Types

2. if , else , switch , case , break , continue , for , while , do - Condition and Control Structure

3. private , protected , public - Access Specifiers

4. static , final , abstract ,synchronized , transient , native , volatile - Access Modifier

5. class , interface - User defined data type

6. strictfp , instanceof - Special keywords

Sample program in Java

```
class HelloWorld
{
        public static void main(String arg[])
        {
        System.out.println("Hello World");
        }
}
```

To Compile the Program

javac FileName.java

- Here javac is the compiler tool which is used for compilation of the code and generating the byte code.

- So the File HelloWorld.java will have a file called HelloWorld.class which is the byte code.

- This byte code is the intermediatary code which can run in any JVM (Java Virtual Machine).

- So because of byte code java is platform independent as byte code can run in any jvm.

- In compilation the code will be compiled chunk.

Interpreting of the Program

java FileName

- Here java is the interpretor tool which is used to interpret the byte code.

- And the interpretation will be line by line.

- java interpretor will use the JVM to run the application.

- JVM is platform dependent as each and every os will have their own JVM.

Rules when we write the Class name

- Class name should be true case or upper camel case

Ex : StudentDetail , EmployeeDetail, CustomerInfo

System.out.println()

- This statement basically used to print anything to the command prompt.

- System - This is a predefined class in java.

- out - This is a printstream object with in System class.

- println() - This is a method to print in console|command prompt.

Variable

- It is a named location in the memory whose value can change at the runtime.

- Whenever we solve any problem temporarily we require some variable.

- System requires two instruction to create variable.

    DataType variableName;

    DataType variableName1,variableName2,...;

Data Type

- The type of the data the variable will hold will be defined by the data type.

- Java has provided four different categories of the primitive data type.

1. Integer - Whole Value	**Memory(Bytes)**	**Range**
A. byte	1	- 2^7 to 2^7-1
B. short	2	- 2^15 to 2^15-1
C. int	4	
D. long	8	
2. Floating - Decimal Values		
A. float	4	
B. double	8	
3. Character - Character		
A. char	2	
4. Boolean - Boolean values		
A. boolean	1	True , False

Operator

- Operator is a symbol which operates on one or more operands.

- Some of the operator which java supports.

1. Assignment Operator : =

- To assign any value to a variable or a variable value to another variable.

```
int num1;
num1=10;
```

2. Arithmetic Operator : + , - , * , / , %

This operator are used for doing arithmetic operation with the variables

```
int num1,num2,result;
num1=10;
num2=20;
result=num1+num2; // Here num1+num2 is called arithmetic expression
```

3. Relational Operator : > , < , >= , <= , != , ==

- This operator are used for finding or evaluating the relational expression.

int num1=10,num2=20;

num1>num2;// This is the relational expression which returns true or false value.

- Relational expression are called as condition.

- To evaluate the condition we need to use the conditional construct.

4. Logical Operator : and (&&) or (||) not(!)

- These operators are used to combine two or more relational expression and returns true or false value.

5. Unary Operator : ++ , --

- Unary operator which is going to operate on one operand.

- ++ operator is equals increment by 1

- -- operator is equal to decrement by 1

int num1=10;

num1++; => num1=num1+1

//here num1 is 11

ex-1

int num1=10;

System.out.println(++num1); // 11

ex-2

int num1=10;

System.out.println(num1++); // 10

6. Arithmetic Assignment Operator : += , -= , *=,/= , %=

- This operator is used for two different operations they are arithmetic and assignment operation.

int num1=10;

num1+=5; => num1=num1+5;

num1 value will be 15

int num2=16;

num2%=3;

now num2 will be 1

27/12/17

Demo : Write a simple program using Operators.

```
class Addition
{
	public static void main(String a[])
	{

	int num1,num2,result;

	num1=100;
	num2=200;
	 result=num1+num2;
	System.out.println("Addition Result:"+result);

	}
}
```

- Here num1 and num2 are the variables which has been initialized directly.

Demo : Write the Program to accept from Runtime

```
import java.util.*;
class Addition
{
	public static void main(String a[])
	{

	int num1,num2,result;

	Scanner scanner=new Scanner(System.in);

	System.out.println("Enter the First Value:");
	num1=scanner.nextInt();
```

```
        System.out.println("Enter the Second Value:");
        num2=scanner.nextInt();

        result=num1+num2;

        System.out.println("Addition Result:"+result);
        }
}
```

- Scanner is a built in class which is available in java.util package hence we need to import java.util package.

- Here System.in is a Stream which is pointing to the keyboard hence we are able to access the value from keyboard.

- nextInt() is a method of Scanner class which will return the integer by reading from keyboard.

Conditional Construct

- Basically conditional construct is used to evaluate the condition and based on the result it will display message.

- Java uses two different type of conditional construct they are

1. If **2.switch**

-If Conditional Construct

- The if conditional construct is again can be divided into four parts they are

1. Simple If **2.If....else**

if..else

Syntax

```
if(condition1)
{
        //code goes here
}
else
{
        //code goes here
}
```

3. If Ladder **4.Nested If**

If Ladder

```
if(condition1)
{
        //code goes here
}
else if(condition2)
{
        //code goes here
}
else if(condition3)
{
        //code goes here
}
else
{
        //code goes here
}
```

Demo : Write a Program to check the biggest number between three given integers using if Ladder.

```
import java.util.*;
class IfLadder
{
        public static void main(String arg[])
        {
        int num1,num2,num3;
        Scanner scanner=new Scanner(System.in);
        System.out.println("Enter the First Value:");
        num1=scanner.nextInt();
        System.out.println("Enter the Second Value:");
        num2=scanner.nextInt();
```

```
        System.out.println("Enter the Third Value:");
        num3=scanner.nextInt();
        if(num1>num2 && num1>num3)
                System.out.println("Number 1 is Biggest");
        else if(num2>num3)
                System.out.println("Number 2 is Biggest");
        else
                System.out.println("Number 3 is Biggest");
        }
}
```

Nested If

```
                if(condition1)
        {
                if(condition2)
                {
                //code goes here
                }
                else
                {
                //code goes here
                }
        }
        else
        {
                if(condition3)
                {
                //code goes here
                }
                else
                {
                //code goes here
```

```
                }
        }
```

Demo : Write a Program to check whether the year is a leap year or not using nested if.

```
import java.util.*;
class LeapYear
{
        public static void main(String arg[])
        {
        int year;
        Scanner scanner=new Scanner(System.in);
        System.out.println("Enter the Year");
        year=scanner.nextInt();
        if(year%100==0)
        {
                if(year%400==0)
                        System.out.println("Leap Year");
                else
                        System.out.println("Not a Leap year");
        }
        else
        {
                if(year%4==0)
                        System.out.println("Leap Year");
                else
                        System.out.println("Not a Leap year");
        }

        }
}
```

Switch Conditional Construct

- Switch is conditional construct is used whenever we want to go for value based checking.

Syntax

```
switch(variable)
{
        case value1:
                //code goes here
        break;
        case value2:
                //code goes here
        break;
        case value3:
                //code goes here
        break;
        default:
                //code goes here
}
```

Demo : Write a Program to check whether a given character is vowel or consonant.

```
import java.util.*;

class VowelCheck
{
        public static void main(String arg[])
        {
        char ch;
        Scanner scanner=new Scanner(System.in);
        System.out.println("Enter a Character");
        ch=scanner.next().charAt(0);
        switch(ch)
        {
                case 'A':
```

```
                case 'a':
                        System.out.println("Vowel");
                break;
                case 'E':
                case 'e':
                        System.out.println("Vowel");
                break;
                case 'I':
                case 'i':
                        System.out.println("Vowel");
                break;
                case 'O':
                case 'o':
                        System.out.println("Vowel");
                break;
                case 'U':
                case 'u':
                        System.out.println("Vowel");
                break;
                default:
                        System.out.println("Consonant");
        }

        }
}
```

Iterative Construct or Looping or Control Structure

- To execute a certain statement for number of times then we can go for loop.

- Java provides different looping construct they are

1. For Loop **2.while Loop** **3.do..while Loop**

For Loop

Syntax

```
for(Initialization;Condition ; Increment/Decrement)
{
//code goes here
}
```

While Loop

```
while(condition)
{
        //code goes here
}
```

do...while Loop

```
do
{
//code goes here
}while(condition);
```

Write a Program to Print 1 to 10 using For Loop

```
class ForLoop
{
        public static void main(String arg[])
        {
        int num1;
        for(num1=1;num1<=10;num1++)
        {
                System.out.println(num1);
        }
        }
}
```

Find the Output of the below Program

```
class ForLoop
{
        public static void main(String arg[])
        {
        int num1;
        for(num1=1;num1<=10;num1++);
        System.out.println(num1);
        }
}
```

Output

11

Example -1

```
class ForLoop
{
        public static void main(String arg[])
        {
        int num1=1;
        for(;num1<=10;)
        {
        System.out.println(num1);
        num1++;
        }
        }
}
```

Demo : Write a while loop for printing 1 to 10.

```
class WhileLoop
{
        public static void main(String arg[])
        {
```

```
        int num1=1;


        while(num1<=10)
        {
        System.out.println(num1);
        num1++;
        }
        }
}
```

28/12/17

Object Oriented Programming

- The language from evolution can be categorized into two different types they are

1. Procedure Oriented Approach

2. Object Oriented Approach

1. Procedure Oriented Approach- In this kind of approach the programs are written with procedure or functions and these functions usually called in the main function.

```
function1()
{
//code goes here
}
function2()
{
//code goes here
}
main()
{
function1();
function2();
}
```

- A function is a block of code which solves a sub problem and many different functions we create to solve the total problem.

- In this kind of approach data has not been given priority or importance hence data can be accessible anywhere hence it does not have any security.

- This type of programs are suitable for the scientific and mathematical problem solving.

- In this type of application we can reuse a portion of the code in another application.

- Some of the Language which has supported this concept of programming

  Ex : Pascal , Cobol , Fortran , Basic , C

- To overcome the limitation of this Object Oriented programming developed.

- This concept programmings are used to develop the real world programming.

- Some of the languages which support this concept.

  Ex : Java , C# , Vb.net , Small Talk , C++ , Python

- Object Oriented Programming has got the popularity because of the four features they are

1. Abstraction

- It says that the unnecesary components need to be hidden from the user.

- This concept is basically to avoid the complexity from the end user.

2. Encapsulation

- Implementing the abstraction is encapsulation.

- Wrapping the data member and code members which act upon the data into a single unit. And in any object oriented programming class is the basis of encapsulation.

3. Inheritance

- Reusing the existing code to create new application.

- Basically we use inheritance for reducing the redundancy(duplicate) of the code.

4. Polymorphism

- Poly means many and morphism means forms.

- Here one object which exibits different behaviours or forms.

- In object oriented programming there are two major concept they are class and object.

1. Class

- It is an user defined data type.

- It is a logical entity which does not take memory.

- It is a template to create object.

Why user Defined Data Type?

- Existing data types are not suitable to store the real data. To store the real data we need to create our own defined data type.

- Real data is nothing but the real orgranization data :

Ex : Employee , Student , Customer , Triangle , Box , Account , Loan

Syntax

```
class ClassName
{
        //Member variables


        //Member functions
}
```

- Here member variables are called as properties and functions are also called as behaviours.

Example

```
class Employee
{
        String empId,empName;
        double salary;

        public double calcSalary()
        {
        //code goes here
        }
}
```

```
class Box
{
        double width,height,depth;
```

```
        public double volume()
        {
        //code goes here
        }
}
```

- String is class in java which is built in class in java.lang package. Its object can store a string value.

Object

- It is a variable of type class.

- It is also called as runtime instance of a class.

- It is a physical entity as it takes memory.

Syntax

```
ClassName object_name=new Constructor([Parameters]);
```

Example

```
Employee employee1=new Employee();

Box box_obj=new Box();
```

- For primitive data type variable we need not to assign memory as java already knows how much memory need to be assigned.

- For object java does not know how much memory should be assigned hence we use new operator to allocate memory.

```
public class Box
{
        double width,height,depth;
}

public class BoxMain
{
        public static void main(String arg[])
        {
```

```
        Box boxobj=new Box();

        boxobj.width=10;
        boxobj.height=15;
        boxobj.depth=20;

        double volume;
        volume=boxobj.width*boxobj.height*boxobj.depth;

        System.out.println("Volume:"+volume);
    }
}
```

Method

- Method or function is a sub program solves a sub problem.
- Method we implement basically to modularize the program.

Syntax

```
Access_Specifier returnType methodName([Parameters])
{
//code goes here
}
```

Example

- Write a method to add two numbers and return the result.

```
public int add(int num1,int num2)
{
    int result;

    result=num1+num2;
    return result;
}
```

30/12/17

Inheritance

- Inheritance says the reusing the existing code into the new class.

- Creating a new class by using the existing class properties and methods.

- To understand inheritance we need to know the relationship between one class to another class.

- In Object orinted Programming one class to another class there may be 4 different type of relationship which may exist

1. Has -a **2. a-part-of**

Ex :

1. Student	Course
2. Car	Engine
3. Employee	Salary

- If there is has-a or a-part-of relationship existing between the two different classes then we can implement composition.

- In a composition relationship we will have the object of a particular class will be a member in another class.

Course.java

```
public class Course
{
    //Class Scope Variables
    String courseId,courseName;
    double price;
    int duration;
        public Course(String cId,String courseName,double price,int duration)
    {
        courseId=cId;
        this.courseName=courseName;
        this.price=price;
        this.duration=duration;
    }
    public void displayCourse()
    {
```

```
                System.out.println("Course ID:"+courseId);
                System.out.println("Course Name:"+courseName);
                System.out.println("Price:"+price);
                System.out.println("Duration:"+duration);
        }
}
```

Student.java

```
public class Student
{
        String studentId,studentName;
        Course course;
        public Student(String studentId,String studentName,String courseId,String
courseName,double price,int duration)
        {
                this.studentId=studentId;
                this.studentName=studentName;
                course=new Course(courseId,courseName,price,duration);
        }
        public void displayDetails()
        {
                System.out.println("Student ID:"+studentId);
                System.out.println("Student Name:"+studentName);
                course.displayCourse(); //calling the displayCourse() method of course class
        }
}
```

StudentMain.java

```
public class StudentMain
{
        public static void main(String arg[])
        {
                Student student=new Student("S1001","Vinod","DTEJA","DT Java",80000,5);
```

```
            student.displayDetails();
        }
}
```

Output

```
Student ID:S1001
Student Name:Vinod
Course ID:DTEJA
Course Name:DT Java
Price:80000.0
Duration:5
```

3. is-a	**4. a-kind-of**
1. Employee	AdminEmployee
2. Student	CarrierStudent
3. Customer	PremierCustomer

- If a class to another class have the is-a or a-kind of relationship existing then we can have inheritance relationship.

- In java to implement inheritance relationship we need to use the extends keyword.

Syntax

```
class SubClass_Name extends SuperClassName
{
        //code goes here
}
```

Getter Method

- To retrieve or display the member variable we need to use the getter method.

Setter Method

- To assign the value to the member variable we need to use the setter method.

```
public class Employee
{
        String employeeId,employeeName;
        public String getEmployeeId()
        {
```

```
		return employeeId;
	}
	public void setEmployeeId(String employeeId)
	{
		this.employeeId = employeeId;
	}
	public String getEmployeeName()
	{
		return employeeName;
	}
	public void setEmployeeName(String employeeName)
	{
		this.employeeName = employeeName;
	}
}
```

- Employee is the super class which is the generalized class which will be extended to other specific class.

- The properties of this class will be sharable to the specific class which has inherited this class.

SalariedEmployee.java

```
public class SalariedEmployee extends Employee
{
	int salary; //Specific member of the SalariedEmployee Class
	public int getSalary() {
		return salary;
	}
	public void setSalary(int salary) {
		this.salary = salary;
	}
}
```

- Here SalariedEmployee is a sub class for the Employee class. This class will inherit the properties and methods of the super class ie. employeeId and employeeName.

```
public class EmployeeMain
{
	public static void main(String arg[])
	{
		SalariedEmployee employeeObj=new SalariedEmployee();

		//Using setter method we have initialized the object
		employeeObj.setEmployeeId("E1001");
		employeeObj.setEmployeeName("Suresh Kumar");

		employeeObj.setSalary(60000);

		//Using getter method we can display the object values.
		System.out.println("Employee ID:"+employeeObj.getEmployeeId());
		System.out.println("Employee Name:"+employeeObj.getEmployeeName());

		System.out.println("Salary:"+employeeObj.getSalary());
	}
}
```

Overloading

- It is possible to write two or more methods with the same name provided their signatures are different.
- These methods are called as overloaded methods and the process is called method overloading.
- Signature include
  - Number of Parameters
    - int add(int v1,int v2);
    - int add(int v1,int v2,int v3);
  - Order of Parameters
    - int add(int v1,float v2);
    - int add(float v1,int v2);
  - Type of Parameters
    - int add(int v1,int v2);

- int add(double v1,double v2);

Note : Return type will not come under the signature.

```
public class AdditionOverload
{
	public int add(int v1,int v2)
	{
		return v1+v2;
	}
	public double add(double v1,double v2)
	{
		return v1+v2;
	}
	public int add(int v1,int v2,int v3)
	{
		return v1+v2+v3;
	}

	public static void main(String arg[])
	{
		AdditionOverload obj=new AdditionOverload();

		System.out.println("Two Integer Addition:"+obj.add(10, 20));
		System.out.println("Two Double Addition:"+obj.add(10.3,8));
		System.out.println("Three Integer Addition:"+obj.add(10,20,30));
	}
}
```

Constructor Overloading

- To create different kind of object we need to implement constructor overloading.

BoxOverload.java

```
public class BoxOverload
{
	double width,height,depth;

	public BoxOverload() //Default Box
	{
		width=10;
		height=15;
		depth=20;
	}

	public BoxOverload(double val) //Square Box
	{
		width=height=depth=(val>0)?val:1;
	}

	public BoxOverload(double w,double h,double d) //User Defined
	{
		width=(w>0)?w:1;
		height=(h>0)?h:1;
		depth=(d>0)?d:1;
	}
}
```

1/1/18

Method Overriding

- When a super class method is not suitable to the sub class then in the sub class we need to rewrite that method.

- The process of rewriting the method from the super class to the sub class is called method overriding.

Shape.java

```
public class Shape
{
	double dim1,dim2;

	public Shape(double d1,double d2)
	{
		dim1=d1;
		dim2=d2;
	}

	public double area()
	{
		return 0.0;
	}
}
```

Rectangle.java

```
public class Rectangle extends Shape
{
	public Rectangle(double d1,double d2)
	{
		super(d1,d2);
	}
}
```

MainShape.java

```
public class MainShape
{
	public static void main(String arg[])
	{
		Rectangle rectangle=new Rectangle(10,20);

		System.out.println("Rectangle Area:"+rectangle.area());
	}
}
```

- Now when we execute the above program then we will get the output as 0.0.
- So here the area() method which is in Shape class is not suitable to the sub class i.e Rectangle.
- Hence we need to override that method.
- Certain rules we need to follow when we do method overriding they are

1. Method name should be same.

2. Parameter level should be same.

3. Return type should be same

4. Access Specifier level should be same or higher.

- In java there are three access specifier and four access levels are there
- The three access specifiers are

1. private **2. protected** **3. public**

- Four access levels from lower to higher they are

1. private **2. default** **3.protected** **4.public**

- If the method is private in the super class we can't override
- If the method is package scope or default then it can overriden by default , protected or public.
- if the method is protected in super class then we can override using protected or public.
- The public method of the super class will be overriden by public only.

```
public class Rectangle extends Shape
{
	public Rectangle(double d1,double d2)
```

```
        {
                super(d1,d2);
        }

        public double area()
        {
                return dim1*dim2;
        }
}
```

Super Keyword

- Super is a keyword basically used to call the members of the super class.

- Bydefault in an inheritance the the super class default constructor will be called bydefault by the sub class any constructor.

SuperClass.java

```
public class SuperClass
{
        public SuperClass() //Default Constructor
        {
                System.out.println("Super Class Default Constructor");
        }
}
```

SubClass.java

```
public class SubClass extends SuperClass
{
        public SubClass()
        {
                System.out.println("Default Constructor of SubClass");
        }
```

```
	public SubClass(int i)
	{
		System.out.println("Overloaded Constructor of SubClass");
	}

	public static void main(String arg[])
	{
		SubClass obj1=new SubClass();

		SubClass obj2=new SubClass(11);
	}
}
```

Output

Super Class Default Constructor

Default Constructor of SubClass

Super Class Default Constructor

Overloaded Constructor of SubClass

- Here Super class default constructor will be called by the Subclass any constructor.
- To call the overloaded version of the constructor in super class we need to use the super keyword.
- super(parameters);
- super keyword should be the first statement in our derived class constructor.

```
public class SuperClass
{
	public SuperClass() //Default Constructor
	{
		System.out.println("Super Class Default Constructor");
	}
```

```
	public SuperClass(int i) //Overloaded Constructor
	{
		System.out.println("Super Class Overloaded Constructor");
	}
}

public class SubClass extends SuperClass
{
	public SubClass()
	{
		System.out.println("Default Constructor of SubClass");
	}

	public SubClass(int i)
	{
		super(i);
		System.out.println("Overloaded Constructor of SubClass");
	}

	public static void main(String arg[])
	{
		//SubClass obj1=new SubClass();

		SubClass obj2=new SubClass(11);
	}
}
```

Casting

- Casting means conversion from one datatype to another data type.
- We have two different type of conversion which are possible they are

1. Implicit Conversion or Implicit Casting

2. Explicit Conversion or Explicit Casting

1. Implicit Conversion

- In case of implicit conversion we dont require explicit code for conversion.

- This is generally done when a lower data type value assgning to the higher data type.

Ex : 1

```
double val;
int v=10;
val=v;
```

Here double is higher data type and int is the lower data type hence we can assign the integer value to the double val.

2. Explicit Conversion

- In case of the explicit conversion we require code for conversion.

Ex :1

```
double val=20.3;
int v;
v=val;//Error rather we need to write v=(int)val;
```

Similar to the primitive data types we can also implement to the extended or to the class data types.

- Implicit Data Type Casting

- When an sub class object instance will be assigned to the super class object reference there is not requirement of explicit casting.

Shape shape=new Rectangle(10,20);

- Using this object reference of super class we can call the methods of the sub class which are inherited from the super class to the sub class.

```
public class MainShape
{
    public static void main(String arg[])
    {
        Shape shape=new Rectangle(10,20);
```

```
        System.out.println("Rectangle Area:"+shape.area());
    }
}
```

Explicit Casting

- When an super class object reference again convert back to the sub class object reference then we need to explicitely write the code to convert.

Shape shape=new Rectangle(10,20); //Implicit Conversion

Rectangle rectangle=(Rectangle)shape; //Explicit Conversion

```
public class MainShape
{
    public static void main(String arg[])
    {
        Shape shape=new Rectangle(10,20); //Implicit Casting

        System.out.println("Rectangle Area:"+shape.area());

        //shape.display();//Error as display() method is specific to Rectangle

        Rectangle rectangle=(Rectangle)shape; //Explicit Casting

        rectangle.display(); //Calling specific meth od of the sub class.
    }
}
```

2/1/18

Abstract Keyword

- Abstract is a modifier which can be applied to the method or to the class.

- If a method is declared as abstract then it will not have any method body.

- To declare a method as abstract we need to have the following syntax

Access_Specifier abstract returnType methodName([Parameters]);

Example

public abstract double area();

- If a class contains atleast one abstract method then that class need to be declared as abstract.

- Abstract method need to be overriden by the derived class else the derived class will be declared as abstract.

```
public abstract class Shape
{
        double dim1,dim2;

        public Shape(double d1,double d2)
        {
                dim1=d1;
                dim2=d2;
        }

        public abstract double area();
}
```

Abstract Class

- A class which has atleast one abstract method is called abstract class.

- This class should be inherited by the derived class as we can't create an instance of this particular class.

- An abstract class can contain other methods , variables and constructor also.

- In java certain predefined class will be abstract class

Ex : InputStream , OutputStream , Reader , Writer

Note : Though we can't create an abstract class instance but we can create the object reference of the abstract class and we can store the instance of the class which has extended this abstract class.

```
Shape shape = new Rectangle();
```

Here shape is a object reference and new Rectangle() is an instance which is assigned to the shape class object reference.

Static Keyword

- Static is also a modifier which can applied to a variable , method or to the class.

Static Variable

- In a class there are two different kind of members which are possible they are

1. Static Members 2. Instance Member

1. Static Members : The member which are specific to the class is called static member or class member.

2. Instance Member : The member which are specific to the instance is called instance member

- If variable is declared as static then it will be create once and there will be only one copy of that variable will be provided and all the instance of the class will share that variable.

Demo : Implementing static variable

```
public class StaticDemo
{
    static int var1; //static variable
    int var2; //instance variable

    public StaticDemo() //Default Constructor
```

```
	{
		var1++;
		var2++;
	}

	public static void main(String arg[])
	{
		StaticDemo ob1=new StaticDemo();
		StaticDemo ob2=new StaticDemo();
		StaticDemo ob3=new StaticDemo();

		System.out.println("Static Variable var1:"+ob3.var1);
		System.out.println("Instance Variable var2:"+ob3.var2);
	}
}
```

Output

Static Variable var1:3

Instance Variable var2:1

- Static variable can be accessed by the class name instead of the object.

3/1/18

Static Method

- Static Method does not require any object to execute. It will be executed by the class name.

- It means it does not require instance so it will not use also the instance member.

- Non static variables can't be accessed in the static context or in the static method.

Syntax

```
Access_Specifier static returnType methodName([Parameters])
{
	//code goes here
}
```

Demo : Implementing Static Method

```
public class StaticDemo
{
	static int var1; //static variable
	int var2; //instance variable

	public StaticDemo() //Default Constructor
	{
		var1++;
		var2++;
	}

	public static void display()
	{
		System.out.println(var1);
		//System.out.println(var2);//Error as var2 is instance member or nonstatic
	}

	public static void main(String arg[])
	{
		StaticDemo ob1=new StaticDemo();
		StaticDemo ob2=new StaticDemo();
		StaticDemo ob3=new StaticDemo();

		StaticDemo.display();

	}
}
```

Static Block

- We can create a static block with in the class.

- The static block will be executed before the main method.

- Static block will access the static resources.

Syntax

```
static
{
        //code goes here
}
```

Demo : Static Block

```
public class StaticBlockDemo
{
        static
        {
                System.out.println("I am in static block");
        }

        public static void main(String arg[])
        {
                System.out.println("I am in Main Method");
        }

}
```

Output

```
I am in static block
I am in Main Method
```

Demo : Static Block

```
public class StaticBlockDemo
{
	static
	{
		System.out.println("I am in static block");
	}

	public static void main(String arg[])
	{
		System.out.println("I am in Main Method");
	}

	static
	{
		System.out.println("I am in a separate static block");
	}

}
```

Final Keyword

- Final is also modifier which can be applied to the variable , method or to a class also.

Final Variable

- If a variable is declared as final then it will be a constant.

- final variable need to be initialized when declared else compiler error will get.

- Conventionally final variable need to be written in upper case as constants in java written in upper case.

```
final string DRIVER="com.h2.Driver";

final int DIRECTION=1;

public class FinalDemo1
{
```

```
        int var1;

        final double PI;

        public FinalDemo1()
        {
                PI=3.141;
        }
}
```

- Here final variable PI has been initialized in constructor it can also be possible.

Final Method

- A final method can't be overriden by the derived class.
- It is reverse of abstract modifier property.
- we can't combine final and abstract modifier.

Syntax

```
Access_Specifier final returnType methodName([Parameters])
{
        //code goes here
}
```

```
public class Circle
{
        double PI=3.141;

        public final double area(double radius)
        {
                return PI*radius*radius;
        }
```

```
}


public class BigCircle extends Circle
{
	public static void main(String arg[])
	{
		BigCircle obj=new BigCircle();

		System.out.println("Area of Circle:"+obj.area(8.3));
	}
}
```

Final Class

- A final class can't be inherited to another class.

Syntax

```
Access_Specifier final class ClassName
{
//code goes here
}

public final class Circle
{

}
```

```
public final class Circle
{
	double PI=3.141;
```

```
	public final double area(double radius)
	{
		return PI*radius*radius;
	}
}

public class BigCircle
{
	public static void main(String arg[])
	{
		Circle circle=new Circle();

		System.out.println("Area of Circle:"+circle.area(8.3));
	}
}
```

- Some of the classes of java are final they are

	1. String		2.System

Closer look at System.out.println

- System is a class which is final class in java.lang package.
- out is the PrintStream class object which is static.
- println() is a method of PrintStream class to print in Console.

```
import java.io.*;
public final class System
{
	public static PrintStream out;
}
```

Interface

1. Interface is a 100% abstract class.
2. All the methods of the interface are bydefault public and abstract.
3. Interface can have variables which bydefault static and final.
4. Interface will be implemented to the class.
5. Class to another class there should be a kind of or is-a relationship has to be there but interface to the class there will no relationship still it can implement.
6. Many interfaces can be implemented to a class.
7. Interface will contain the methods which are common to some of the irrelevant classes.

Syntax for Creating Interface

```
interface InterfaceName
{
        dataType variable_name1=value1;
        dataType variable_name2=value2;
        .....
        .....
        .....
        returnType methodName1([Parameters]);
        returnType methodName2([Parameters]);
        .....
        .....
}
```

Example

```
interface CallBack
{
        int callback(int var1);
}
```

Implementing the Interface

- We can implement the interface to the class by using following syntax

Syntax

```
class SubClassname [extends SuperClassname] implements Interface1,Interface2.....
{
//code goes here
}
```

UseCallBack.java

```
public class UseCallBack implements CallBack
{

	public void callBack(int var1)
	{
		System.out.println("Variable var1:"+var1);
	}

	public static void main(String arg[])
	{
		UseCallBack useCallBack=new UseCallBack();

		useCallBack.callBack(15);
	}

}
```

Interface Object Reference

- We can create object reference of Interface and we can assign the class instance which has already implemented to the interface.

- We can't create an instance of interface as it is abstract.

```
public class UseCallBack implements CallBack
{

	public void callBack(int var1)
	{
		System.out.println("Variable var1:"+var1);
```

```
	}

	public static void main(String arg[])
	{
		CallBack callBack=new UseCallBack();

		callBack.callBack(15);
	}

}
```

4/1/18

Interface Inheritance

- An interface can extend multiple interfaces.

- In Java a class can extend only one class at a given point of time. But an interface can extend multiple interfaces.

- Using interface we can apply multiple inheritance.

Calculator.java

```
interface Addition
{
		int add(int var1,int var2);
}
interface Multiplication
{
		int mul(int var1,int var2);
}
interface Division
{
		int div(int var1,int var2);
```

```
}
interface Substraction
{
                    int substract(int var1,int var2);
}
public interface Calculator extends Addition,Multiplication,Division,Substraction
{
                    public void display();
}
```

UseCalculator.java

```
public class UseCalculator implements Calculator
{

                    @Override
                    public int add(int var1, int var2) {

                      return var1+var2;
                    }

                    @Override
                    public int mul(int var1, int var2) {
                      return var1*var2;
                    }

                    @Override
                    public int div(int var1, int var2) {
```

```
    return var1/var2;
}

@Override
public int substract(int var1, int var2) {

    return var1-var2;
}

@Override
public void display()
{
    System.out.println("Addition : "+add(10,20));
    System.out.println("Substraction : "+substract(20,10));
    System.out.println("Multiplication: "+mul(10,5));
    System.out.println("Division : "+div(80,4));
}

public static void main(String arg[])
{
    UseCalculator calculator=new UseCalculator();
    calculator.display();

    Calculator calc=new UseCalculator();
    calc.display();

    Addition addition=new UseCalculator();
    System.out.println("Using Addition Obj ref:"+addition.add(90,89));

}
```

```
}
```

- Here Addition interface object reference can also hold the UseCalculator object as indirectly it is implemented to the class.

- Using this object reference we can call the method which is inherited from Addition interface to the UseCalculator class.

Interface Variables

- Interface variables are bydefault static and final.

```
interface SharedConstant
{
                    int LEFT=1;
}
public class UseSharedConstant implements SharedConstant
{
                    public static void main(String arg[])
                    {
                       System.out.println("LEFT variable Value:"+LEFT);

                    }
}
```

Packages

- Package is a logical container which contains the relevant classes and interfaces.

- To create a package we need to type the following.

```
package package_name;

//class
//Interface
```

Demo : Creating a Simple Package and Stored a Class

```
package com;
```

```
public class Demo
{
                    private int v1;
                    int v2;
                    protected int v3;
                    public int v4;

                    public Demo()
                    {

                    }

                    public void display()
                    {
                       System.out.println("Display Method : Demo Class");
                    }
}
```

- To use the package there are two different ways

1. Using the fully qualified class name.
2. importing the package.

1. Using Fully Qualified class Name

- When you are using a classes from package and which is implemented only once or twice then we can implement using fully qualified class name.

- Fully qualified class name is nothing but packagename.classname

```
public class UseDemo
{
```

```
                    public static void main(String arg[])
                    {
                       com.Demo demo=new com.Demo();

                       //demo.v1=10;//Error as v1 is private
                       //demo.v2=10;//Error as v2 is package scope
                       //demo.v3=10;//Error as v3 is protected
                       demo.v4=10;

                       demo.display();
                    }
}
```

2. Using the import statement

- We can use also the import statement for using the class from the package.

```
import packagename.*;
```

- Here * denotes all the class with in the package.

```
import com.*;
public class UseDemo
{
                    public static void main(String arg[])
                    {
                       Demo demo=new Demo();

                       //demo.v1=10;//Error as v1 is private
                       //demo.v2=10;//Error as v2 is package scope
                       //demo.v3=10;//Error as v3 is protected
                       demo.v4=10;

                       demo.display();
```

```
                    }
}
```

Java Predefined Packages

- Java has provided number of predefined packages using which we can create different kind of application.

- Some of the Packages which we can see in the following.

1. java.lang **2.java.util** **3.java.io** **4.java.sq**

5.java.net

1. java.lang :

- This package is the default package for java which will be accessible for every class in java.

- All the classes which are available with in this package are basic classes which are required for the programs.

- Some of the classes and interfaces which are available in java.lang package they are

1. System **2. Object** **3. Thread**

4. Runtime **5. Class** **6.String**

1. Runnable **2.Comparable**

Object

- Object is super class hierarchy of all the java classes.

- It means even if our class is not extending to any other class then it is extending to the object class.

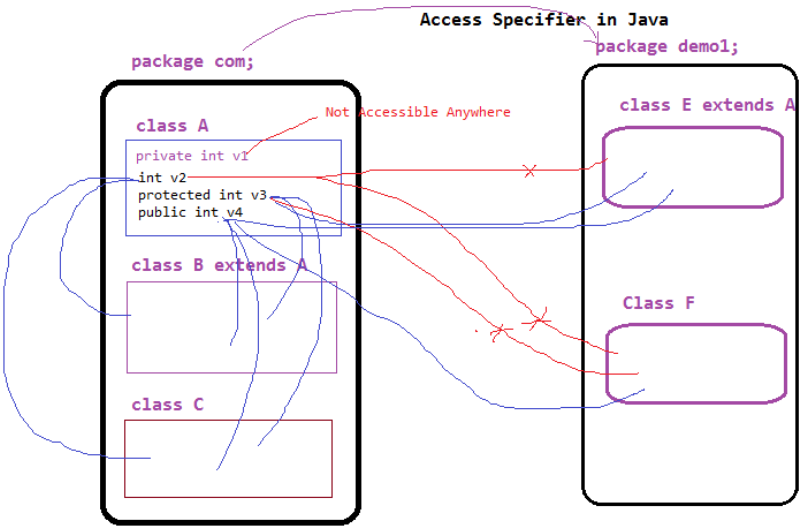
- All the methods of Object class can be accessible by the object of any java class.

toString()

- toString() is a method with in the Object class.

- Bydefault when we print the object of any directly the toString() method will be called and it will print ClassName@Hashcode bydefault.

- If your class has to customize the output then we need to override the toString() method of the Object class.



5/1/18

toString()

- toString() is a method with in the Object class.

- Bydefault when we print the object of any directly the toString() method will be called and it will print ClassName@Hashcode bydefault.

- If your class has to customize the output then we need to override the toString() method of the Object class.

```
public class ToStringImpl

{
```

```
    int var1,var2;

    public ToStringImpl()
    {
      var1=10;
      var2=20;
    }

    public String toString()
    {
      return "Var1:"+var1+"Var2:"+var2;
    }
    public static void main(String arg[])
    {
      ToStringImpl obj=new ToStringImpl();

      System.out.println(obj);
    }
}
```

String Class

- String is a class in java.lang package.
- String is a final class hence we can't inherit this particular class.
- A String literal in java is an object of String class.

```
String str="Gagan"
```

- Here "Gagan" is an object and using this we can call methods.

```
public class StringDemo
{
    public static void main(String arg[])
```

```
{
   String str="Gagan";

   System.out.println(str.length());  //5
   System.out.println("VinodKumar".length()); //10

}
}
```

Constructors of String

```
public class StringDemo
{
                public static void main(String arg[])
                {

   char myString[]= {'S','U','D','H','I','R'};

   String stringObj1=new String(myString);

   byte bytearray[]= {65,66,67,68};

   String stringObj2=new String(bytearray);

   System.out.println("Character Array :"+stringObj1);

   System.out.println("Byte Array:"+stringObj2);

                }
}
```

String Functions

```
public class StringDemo
{
                    public static void main(String arg[])
                    {

                      String str1="Srinivas";

                      System.out.println("UpperCase:"+str1.toUpperCase());

                      System.out.println("FAHAD".toLowerCase());

                      String studentid="S180076300090";

                      System.out.println("Year:"+studentid.substring(0,3));

                      System.out.println("CenterCode:"+studentid.substring(3,8));

                      System.out.println("Roll No :"+studentid.substring(8));

                      String record="E1001,Suresh,Hyderabad,9888112239";

                      String empDetail[]=record.split(",");

                      System.out.println("Employee ID:"+empDetail[0]);
                      System.out.println("Employee Name:"+empDetail[1]);
                      System.out.println("Employee Address:"+empDetail[2]);
                      System.out.println("Employee Mobile:"+empDetail[3]);

                    }
}
```

String as Immutable Object

- String is a immutable object as we can't change the value of the string instance.

```
public class StringDemo2
{
                    public static void main(String arg[])
                    {

                      String str="Sudhir";
                      str.concat("Kumar");
                      System.out.println(str);

                    }

}
```

- The above program when we execute it will print as "Sudhir" as concat method returns a String which will have "SudhirKumar".

- Here str which the contain the string instance will concat with new Sting "Kumar" but it is not refered by anyother string.

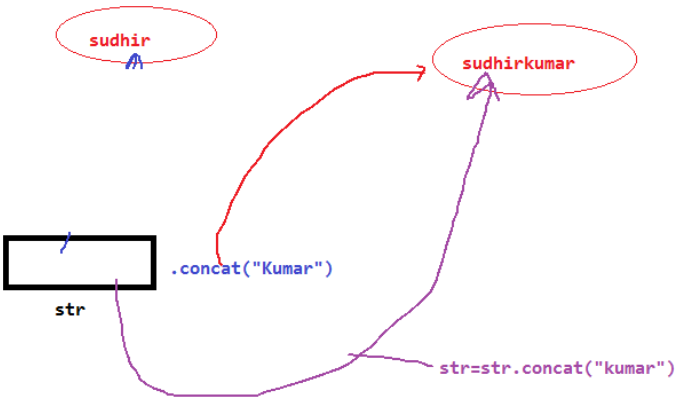
```
public class StringDemo2
{
                    public static void main(String arg[])
                    {

                      String str="Sudhir";
                      str=str.concat("Kumar");
                      System.out.println(str);
```

```
                    }

}
```

- This time the output will print "SudhirKumar"



6/1/18

Mutable String

- To implement mutable String in java we have two different classes they are

1. StringBuilder 2.StringBuffer

- StringBuilder method are not synchronized it means in concurrent envirnoment multiple threads can access the same object.

- StringBuffer methods are synchronized it means in concurrent envirnoment a single thread only can access the object at a given of time.

```
public class MutableString
{
                    public static void main(String arg[])
                    {
                       StringBuffer stringBuffer=new StringBuffer();

                       stringBuffer.append("Mr.");
                       stringBuffer.append("Sudhir");
                       stringBuffer.append(" Kumar");

                       System.out.println("String Buffer:"+stringBuffer);

                       StringBuilder stringBuilder=new StringBuilder();

                       stringBuilder.append("Mr.");
                       stringBuilder.append("Sharath");
                       stringBuilder.append(" Kumar");

                       System.out.println("String Builder:"+stringBuilder);

                    }
}
```

Regular Expression

- Regular Expression is a string which is used for searching a particular pattern of string with in a big string.

- Regular expression is used for validating a string or searching a particular string.

- To implement the regular expression pattern matching we can use the java.util. regex package.

- Certain characters which we are using for searching

- * - Which represents all characters.

- \d - digit

- \w - character

- To implement the regular expression we need to implement two main classes they are

1. Pattern 2. Matcher

1. Pattern

- This class defines the pattern for which we need to search in target string.

- There are different pattern which we can be applied to the target string using the meta characters of regular express (Ex : * , + , \d , \w , \W)

- We can create complex searching string

1. Email Validation

2. Mobile Number Validation

3. House Number Validation

Syntax

- To create a Pattern object we need to use the compile() method of the Pattern class which returns the Pattern object only.

- This compile(String) method which is basically a static method and which accept an string argument.

Syntax

Pattern pattern=Pattern.compile("string");

Example

Pattern pattern=Pattern.compile("b.st");

Here "." is representing a single character

b.st => best , bbst , bkst

2. Matcher

- This class is used for matching the pattern with target string.

- To create the object of this class we need to call the matcher() method which takes an argument of String and it is in Pattern class.

- Certain method which are available in Matcher class are

1. group() 2.find()

Demo : Simple Pattern implemention to Find whether Pattern found in Target String

```
import java.util.regex.*;
public class PatternDemo1
{
                    public static void main(String arg[])
                    {
                      String target_string="It was the best of times.bbst bkst";

                      Pattern pattern= Pattern.compile("b.st");

                      Matcher matcher=pattern.matcher(target_string);

                      if(matcher.find())
                              System.out.println("Pattern Found");
                      else
                              System.out.println("Pattern Not Found");
                    }
}
```

Demo : Finding all the Pattern matched String and printing them

```
import java.util.regex.*;
public class PatternDemo1
{
                    public static void main(String arg[])
                    {
                      String target_string="It was the best of times.bbst bkst";

                      Pattern pattern= Pattern.compile("b.st");

                      Matcher matcher=pattern.matcher(target_string);
```

```
                while(matcher.find())
                {
                        System.out.println(matcher.group());
                }

        }
}
```

Demo : Finding all two digit based numbers with in a given string.

```
import java.util.regex.*;
public class PatternDemo2
{
        public static void main(String arg[])
        {
                String target_String="Steve told me to get 20 Apple for 80 ruppes";

                Pattern pattern=Pattern.compile("(\\d\\d)");

                Matcher matcher=pattern.matcher(target_String);

                while(matcher.find())
                {
                        System.out.println(matcher.group());
                }
        }
}
```

Demo : Implementing the Boundary Match using ^ character in a Pattern

```
import java.util.regex.*;
public class PatternDemo3
```

```
{
                    public static void main(String arg[])
                    {
                       String target_String="Steve told me to get 20 Apple for 80 ruppes";

                       Pattern pattern=Pattern.compile("^Steve.*?ruppes");

                       Matcher matcher=pattern.matcher(target_String);

                       if(matcher.find())
                              System.out.println("Pattern was Found");
                       else
                              System.out.println("Pattern was not Found");
                    }
}
```

8/1/18

Exception Handling

- When a program we execute generally we find three different kind of errors they are

1. Compile Time Error - This error will be happen when we have any syntax error like , missing or any spelling mistake.

- We need to reedit the program and then we need to execute.

2. Logical Error - Will happen basically in conditions and looping structures.

- This error can be resolved by debugging or step by step execution.

- Bug - Error - Debug - Error Free

3. Runtime Error - This error will happen at the runtime and suspend the execution of the program.

- Runtime error will happen for the various cases.

1. Calling a method with null Object
2. Querying a database which is in shut mode.
3. Any number by zero operation

- To resolve this we have exception handling.

Exception

- Exception is an abnormality which occurs at the runtime and suspends the execution of the program.

What happens when Exception Occur?

- When an exception occur it will create an instance of that particular exception and throws to the program.

- If the program has handled that exception then as per the exception handler the code goes else it will suspend the execution and prints the error.

- Every exception is a class in java hence we can create an instance or jvm creates and instance of that class.

- All exception classes are derived from Exception class of java.lang package.

- The Throwable is a class in java.lang package which is the super class of Exception.

- Some of the Exception Classes as follows

1. ArithmeticException : This exception class object will be create when an number by zero will happen.

2. ArrayIndexOutOfBoundsException: This exception will occur when we trying to access the value of the array which is beyond the index.

3. NullPointerException : This exception will occur when we trying to access a method with null object.

```
Player p;
p.play();
```

4. FileNotFoundException : This exception will occur when we are trying to write a File program but the file is not exist.

UnHandled Exception Code

```
public class UnHandled
{
    public static void main(String arg[])
    {
```

```
                int num1=10,num2=0,result;

                result=num1/num2; //Exception occured

                System.out.println("Result:"+result);

                System.out.println("After the Exception:"); //Valid Code
            }
}
```

- When the above program has compiled and executed the following error will occur.

Exception in thread "main" java.lang.ArithmeticException: / by zero

at UnHandled.main(UnHandled.java:8)

- So the moment the exception has occured it has suspended the execution.

- Even after that if any valid code is there still it will no execute.

- To handle this situation java provided the five keywords for handling exception.

1. Try 2.catch 3. Finally 4.throw 5.throws

1. Try

- This is block of code where the exception may arise.

```
try
{
            //code which may arise exception
}
```

2. Catch

- It will follow the try block and this will take an exception object reference the exception which may have arised in the try block.

- when a try block have exception then the catch block will execute else it will not.

Handled Code

```
public class Handled
{
            public static void main(String arg[])
            {
```

```
                int num1=10,num2=0,result;

                try
                {
                result=num1/num2;
                System.out.println("Result:"+result);
                }
                catch(ArithmeticException obj)
                {
                        System.out.println("Exception Arised:"+obj);
                }

                System.out.println("After the Exception:");
        }
}
```

- Now the code which has handled the exception will not suspend the execution.

MultipleCatch

- Whenever our program have multiple exceptions we can handle them using multiple catch handler.

```
import java.util.*;
public class MultipleCatch
{
                public static void main(String arg[])
                {
                int arr[]= {2,3,0},num1=42,index,result;

                Scanner scanner=new Scanner(System.in);
                try
                {
                        System.out.println("Enter the Index Value:");
```

```
                index=scanner.nextInt();

                result=num1/arr[index];

                        //index=0 => 42/arr[0]=>42/2=>21

                        //index=1 => 42/arr[1]=>42/3=>14

                        //index=2 => 42/arr[2]=>42/0=>ArithmeticException

                        //index=3 => 42/arr[3]=>
```
ArrayIndexOutOfBoundsException

```
                        System.out.println("Result:"+result);
                }
                catch(ArithmeticException ae)
                {
                        System.out.println("Exception Arised:"+ae);
                }
                catch(ArrayIndexOutOfBoundsException ae1)
                {
                        System.out.println("Exception Arised:"+ae1);
                }
                catch(Exception e) //Global catch Exception
                {
                        System.out.println("Global catch handler:"+e);
                }

                System.out.println("After the Exception Code");
        }
}
```

- Here we have written the three catch block

1. Which handles the ArithmeticExceptions

2. Which handles the ArrayIndexOutOfBoundsException

3. The last one which is global catch handler which can handle any exception.

9/1/18

Finally Block

- Finally block basically used when we want to implement any resource cleaning.
- Resource Cleaning - File Pointer closing , Database Connection Closing
- Finally block will be executed irrespective of exception occured or may not occured.
- Finally block can followed by the try or catch block.

```
try
{
//code which may arise exception
}
catch(Exception obj_ref)
{
//code if any exception occurs
}
finally
{
//finally block.
}
```

Demo : Finally Block Demo

```
public class FinallyDemo
{
                    public static void main(String arg[])
                    {
                      try
                      {
```

```
                System.out.println("Try block");
                int result;
                result=10/0;
            }
            catch(Exception obj_ref)
            {
                System.out.println("Exception Arised:"+obj_ref);
            }
            finally
            {
                System.out.println("Finally Block");
            }
        }

}
```

User Defined Exception

- All the built in exception object will be created by jvm and thrown to the program when there is an abnormality occurs for jvm in the program.

- If there any validational condition which occurs with in our application specific to a business logic then java will not throw the exception for use.

- For this we need to create our own exceptions which are user defined exception.

- To create user defined exception we need to create a class by extending to the Exception class of java.lang package.

Userdefined Exception

```
public class BankException extends Exception
{
        public BankException()
        {
            super("Balance is not Sufficient");
        }

        public String toString()
```

```
                    {
                        return "Balance is not Sufficient";
                    }
}
```

- The super is the keyword which we have used here to initialize the parent constructor or call the parent constructor. Here it will initialize the message attribute of parent class.

- toString() method is implemented if anyone print this object then the message should be printed.

Now this class is userdefined class will be thrown by us if the user is not maintaining the specified balance what bank has told.

throw

- throw keyword is used for throwing any exception mostly userdefined exception.

Syntax

throw throwableInstance;

throws

- When we are not handling the exception in a method then it is better to tell that this method is throwing an exception.

- To tell this way we need to use throws keyword.

Access_Specifier returnType methodName([Parameters])throws Exception1,Exception2...

{

//code may throws exception

}

Demo

```
import java.util.*;
public class Bank
{
                    public void withDraw(int amount)throws BankException
                    {
                        int balance=1000;

                        if((balance-amount)<500)
```

```
                throw new BankException();
        else
                System.out.println("Transaction Successful");
    }

    public static void main(String arg[])
    {
        int amount;

        Bank bank=new Bank();

        try
        {
                Scanner scanner=new Scanner(System.in);
                System.out.println("Enter the Amount:");
                amount=scanner.nextInt();

                bank.withDraw(amount);
        }
        catch(BankException objRef)
        {
                System.out.println("Exception Arised:"+objRef);
        }
    }
}
```

Collection

- A Collection is a group of objects arranging in a logical manner.
- A collection is a growable array
- With Collection we can do the following activities.

1. Adding the item to the collection
2. Deleting the item from the collection

3. Updating the item from the collection

4. Displaying the item from the collection

Collection Vs Array

- A collection is a dynamic data structure whose memory can be allocated at the runtime.

  - We can add an element at the runtime

  - We can delete the element at the runtime

- An array is a static data structure whose memory can be allocated at the compile time.

  - We can add an element until the size of the array.

  - We can delete the element but the not the memory.

- To implement the collection java has provided the collection framework

- This framework has already provided interfaces and classes to implement the collection.

- All this interfaces and classes related to the collection is available in java.util package.

- In the Collection Framework Collection is the super interface for all the collection interfaces and their related classes.

- This collection can be brodly categorized into two different types

1. Ordred Collection

  - This type of collection will have the ordered info using index.

  - All these collection have implemented from List.

  - They can contain duplicate values.

2. UnOrdered Collection

  - This type of collection we can using Set.

  - They will not accept duplicate values.

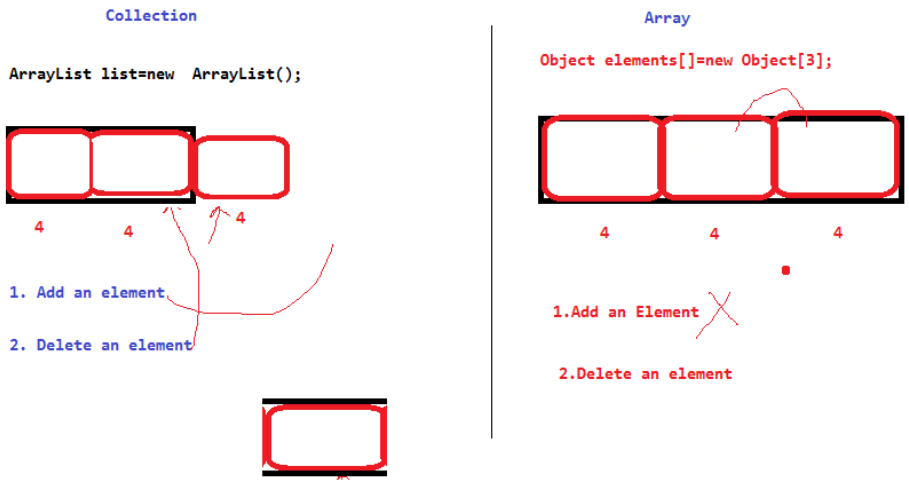
Demo : Implementing List Collection

```
import java.util.*;
public class ListDemo
{
        public static void main(String arg[])
        {
           ArrayList list=new ArrayList();
```

```
        list.add("Sridhar");
        list.add("Vinod");
        list.add("Suresh");
        list.add(100);
        list.add(200);

        System.out.println("List :"+list);

        System.out.println("Collection Size:"+list.size());
    }
}
```



10/1/18

Traversal of Collection

- We can traverse the collection in three different ways they are

1. Using enhanced for loop.

2. Using Iterator

3. Using ListIterator

1. Enhanced For Loop

- In this case we will try to display the collection elements using enhanced for loop.

Syntax

```
for(DataType elementName:Collection/Array)
{
//code goes here
}
```

instanceof

- instanceof operator is used for basically to know what kind of instance the super class object reference is carrying.

- instanceof operator will return boolean value.

Demo : Implementing Enhanced For Loop.

```
import java.util.*;
public class ListDemo
{
    public static void main(String arg[])
    {
      ArrayList list=new ArrayList();

      list.add("Sridhar");
      list.add("Vinod");
      list.add("Suresh");
      list.add(100);
      list.add(200);

      System.out.println("List :"+list);
      System.out.println("Collection Size:"+list.size());
```

```
                for(Object element:list)
                {
                        if(element instanceof String)
                        {
                                String str=(String)element;
                                System.out.println(str+"-String");
                        }
                        else
                        {
                                Integer intr=(Integer)element;
                                System.out.println(intr+"-Integer");
                        }
                }
        }
}
```

Iterator

- Iterator is an interface is java.util package which help us to traverse the collection.
- To get Iterator object we need to call the iterator() method.
- Iterator interface contains mainly two methods they are

1. hasNext() : This method returns a boolean value which says whether the collection contains any element.

2. next() : This method returns an object which is in the collection

- Iterator can remove a particular value from a collection during the iteration.
- remove() method is there in Iterator interface which will do that.

Demo : Implementing Iterator

```
import java.util.*;
public class ListDemo
{
        public static void main(String arg[])
        {
```

```
ArrayList list=new ArrayList();

list.add("Sridhar");
list.add("Vinod");
list.add("Suresh");
list.add(100);
list.add(200);

System.out.println("List :"+list);
System.out.println("Collection Size:"+list.size());

//iterator() method returns Iterator object
Iterator itr=list.iterator();

while(itr.hasNext())
{
        Object element=itr.next();

        if(element instanceof String)
        {
                String str=(String)element;
                System.out.println(str+"-String");

                if(str.equals("Vinod"))
                        itr.remove();

        }
        else
        {
                Integer intr=(Integer)element;
                System.out.println(intr+"-Integer");
```

```
                    }
                }

                System.out.println(list);
            }
}
```

ListIterator

- The ListIterator interface which is basically to iterate over the collection of List type.
- To get an object of ListIterator then we can implement listIterator() method.

ListIterator listIterator = list.listIterator();

- ListIterator has already extended the Iterator interface.

```
import java.util.*;
public class ListDemo
{
        public static void main(String arg[])
        {
            ArrayList list=new ArrayList();

            list.add("Sridhar");
            list.add("Vinod");
            list.add("Suresh");
            list.add(100);
            list.add(200);

            System.out.println("List :"+list);
            System.out.println("Collection Size:"+list.size());

            //iterator() method returns Iterator object
            ListIterator itr=list.listIterator();
```

```
                while(itr.hasNext())
                {
                        Object element=itr.next();

                        if(element instanceof String)
                        {
                                String str=(String)element;
                                System.out.println(str+"-String");

                                if(str.equals("Vinod"))
                                        itr.set("Vikash");

                        }
                        else
                        {
                                Integer intr=(Integer)element;
                                System.out.println(intr+"-Integer");
                        }
                }

                System.out.println(list);

        }
}
```

Iterator - We have only forward Traversal and remove option

ListIterator - We have forward and backward Traversal and update option.

Set Collection

- Set is a unordered collection basically will not take duplicate values.

- There are majorly three classes we use to create set type collection they are

1. HashSet **2.LinkedHashSet** **3.TreeSet**

Demo : Implementing the Set Collection.

```
import java.util.*;
public class SetDemo
{
                    public static void main(String arg[])
                    {
                      HashSet hs=new HashSet();  //Display no order

                      hs.add("Vinod");
                      hs.add("Harish");
                      hs.add("Avinash");
                      hs.add("Saket");
                      hs.add("Fareed");

                      System.out.println(hs);

                      LinkedHashSet lhs=new LinkedHashSet(); //Display in Order how it
has added

                      lhs.add("Vinod");
                      lhs.add("Harish");
                      lhs.add("Avinash");
                      lhs.add("Fareed");

                      System.out.println(lhs);

                      TreeSet ts=new TreeSet(); //Display in order in sorting

                      ts.add("Vinod");
                      ts.add("Harish");
                      ts.add("Avinash");
                      ts.add("Saket");
```

```
                ts.add("Fareed");


                System.out.println(ts);
            }
}
```

Output

[Avinash, Fareed, Saket, Harish, Vinod]

[Vinod, Harish, Avinash, Fareed]

[Avinash, Fareed, Harish, Saket, Vinod]

Generics

- It has implemented from jdk1.5 onwards.

- Generic is used to avoid the runtime error of casting.

- Generic can be implemented in Collection also.

```
import java.util.*;


public class NonGenericImpl
{
            public static void main(String arg[])
            {
               ArrayList list=new ArrayList();


               list.add("Suresh");
               list.add("Vinod");


               Integer intr=(Integer)list.get(0);
               System.out.println(intr);
            }
}
```

- The above code when we compile and execute we get the following exception.

Exception in thread "main" java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Integer at NonGenericImpl.main(NonGenericImpl.java:12)

- The error has occured as ArrayList object takes any kind of object and hence we are able to cast to any type which leads to the runtime error.

- To avoid this type of runtime error we need to use Generic.

```
import java.util.*;

public class GenericImpl
{
                    public static void main(String arg[])
                    {
                       ArrayList<String> list=new ArrayList<String>();

                       list.add("Suresh");
                       list.add("Vinod");
                       //list.add(100);//Error

                       //Integer intr=(Integer)list.get(0);//Error

                    }
}
```

Map Collection

- Map is another type collection which will take key and value pair.

- This is not from the Collection interface hiearchy.

- Map is the super interface hiearchy for all the map based collections.

- There are three major classes which we can use they are

1. HashMap **2.LinkedHashMap** **3.TreeMap**

Demo : Implementing the Map Collection

```
import java.util.*;
public class MapDemo
{
                    public static void main(String arg[])
```

```
{
    HashMap<Integer,String> mapObj=new HashMap<Integer,String>();

    mapObj.put(1001, "Vinod");
    mapObj.put(1002, "Sunil");
    mapObj.put(1003, "Suresh");
    mapObj.put(1004, "Fareed");

    //keySet() is a method which returns all the key values of map
    Set<Integer> allKeys=mapObj.keySet();

    for(Integer element:allKeys)
    {
        System.out.println(element+":::"+mapObj.get(element));
    }

}
}
```

11/01/18

File IO

Stream - Stream is a via media by which we can write the data to the sink and read the data from the source.

- Here sink is the one which can consume the data

Ex : Monitor , File , Socket ,

- Here Source is the one which can produce the data

Ex : Keyboard , File , Socket

- Everything in java it is considered as file and we need write to this file or read from the file.

- The stream in java can be divided into two different types they are

1. Binary Stream **2. Character Stream**

1. Binary Stream

- Binary stream is the one where we can read or write the data by byte.

- Again this stream can be divided into two different types they are

1. InputStream

2. OutputStream

1. InputStream

- This stream is used for reading the data from the Source.

- This is the super class hierarchy of all the binary reading streams.

- This class is available in java.io package.

- This is an abstract class and there are other classes which have already inherited this particular class.

- To read the data from the File Source we have FileInputStream.

Reading a File Demo

```
import java.io.*;
public class FileReadDemo
{
                    public static void main(String arg[])
                    {

                      try
                      {
                              FileInputStream fileInputStream=new
FileInputStream("c:\\File.txt");

                              int avl=0;

                              avl=fileInputStream.available();

                              byte buff[]=new byte[avl];
```

```
                    fileInputStream.read(buff,0,avl);

                    String content=new String(buff);

                    System.out.println(content);

                    fileInputStream.close();
                }
                catch (Exception e)
                {
                    System.out.println("Exception Arised:"+e);
                }

            }
}
```

OutputStream

- OutputStream is the super class hierarchy of all the writing streams.
- FileOutputStream is the class which will write to a specific file.

Writing a File

```
import java.io.*;
public class FileWriteDemo
{
            public static void main(String arg[])
            {
                try
                {
                    FileOutputStream foutputStream=new
FileOutputStream("c:\\Demo1.txt",true);
```

```
                    String myContent="This is the Content will be Written to
File";

                    for(int position=0;position<myContent.length();position++)
                    {
                        foutputStream.write(myContent.charAt(position));
                    }

                    System.out.println("The String has written");

                    foutputStream.close();
                }
                catch(Exception e)
                {
                    System.out.println("Exception Arised:"+e);
                }
            }
}
```

Serialization

- Serialization is a process by which we can write an object to the permanent location.

- It is a process to make an object persist.

- Bydefault an object can't be written to a file or to a persistent location.

- To store into persistent location the object need to be serialized and to make that we need to make the class to be serialized.

- To make the class to be serialized we need to implement Serializable interface of java.io package.

Employee.java

```
import java.io.Serializable;

public class Employee implements Serializable

{
```

```
private String empId;
private String empName;
private String addr;

public Employee()
{
}

public String getEmpId() {
   return empId;
}

public void setEmpId(String empId) {
   this.empId = empId;
}

public String getEmpName() {
   return empName;
}

public void setEmpName(String empName) {
   this.empName = empName;
}

public String getAddr() {
   return addr;
}

public void setAddr(String addr) {
   this.addr = addr;
}
```

```
                    public String toString()
                    {
                       return "Employee ID:"+empId+" Employee Name:"+empName+"
Address:"+addr;
                    }

}
```

Writing an Object to the File

```
import java.io.*;
public class ObjectWriteDemo
{
                    public static void main(String arg[])
                    {

                       try
                       {
                              FileOutputStream foutputStream=new
FileOutputStream("c:\\MyEmp.txt");
                              ObjectOutputStream oos=new
ObjectOutputStream(foutputStream);

                              Employee employee=new Employee();
                              employee.setEmpId("E1001");
                              employee.setEmpName("Suresh");
                              employee.setAddr("Hyderabad");

                              oos.writeObject(employee);

                              System.out.println("Object Written to a File:");

                              oos.close();
```

```
                foutputStream.close();
            }
            catch(Exception e)
            {
                System.out.println("Exception Arised:"+e);
            }
            }
}
```

DeSerailization

- It is reverse of the serialization process where we will read the content of the object from permanent location and store it into to the temporary location.

```
import java.io.*;
public class ObjectReadDemo
{
        public static void main(String arg[])
        {

          try
          {
                FileInputStream fis=new FileInputStream("c:\\MyEmp.txt");
                ObjectInputStream ois=new ObjectInputStream(fis);

                Employee empObj=(Employee)ois.readObject();

                System.out.println(empObj);

          }
          catch(Exception e)
          {
                System.out.println("Exception Arised:"+e);
          }
```

```
                    }
}
```

12/01/18

Transient

- Transient is a modifier basically used for not serailizing certain members of the class during serialization.

- To declare some variables as transient

transient datatype variable_name;

Demo : Implementing Transient Modifier

```
import java.io.Serializable;


public class Employee implements Serializable
{
                    private String empId;
                    private String empName;
                    private String addr;

                    private transient String password;

                    public String getPassword() {
                       return password;
                    }

                    public void setPassword(String password) {
                       this.password = password;
                    }

                    public Employee()
                    {
                    }
```

```
public String getEmpId() {
   return empId;
}

public void setEmpId(String empId) {
   this.empId = empId;
}

public String getEmpName() {
   return empName;
}

public void setEmpName(String empName) {
   this.empName = empName;
}

public String getAddr() {
   return addr;
}

public void setAddr(String addr) {
   this.addr = addr;
}

public String toString()
{
   return "Employee ID:"+empId+" Employee Name:"+empName+"
Address:"+addr+" Password:"+password;
}
```

}

Character Stream

- Character stream is used to implement the reading and writing by character.

- The Reader and Writer are two abstract classes which are the super class for all character base stream classes.

- Reader is the super class hierarchy for all Reading stream classes

  - FileReader , BufferedReader etc

- Writer is the super class hierarchy for all Writing based stream classes.

  - FileWriter , BufferWriter

Demo : Reading the File using FileReader

```
import java.io.*;

public class FileReaderDemo

{

        public static void main(String arg[])

        {

          try

          {

                FileReader fileReader=new FileReader("c:\\Student.txt");


                char studentDetail[]=new char[500];


                fileReader.read(studentDetail);


                String records=new String(studentDetail);


                System.out.println(records);


                fileReader.close();

          }

          catch(Exception e)

          {
```

```
                    System.out.println("Exception Arised:"+e);
                }
            }
}
```

Thread

- In a multitask envirnoment of operating system where multiple process are running at a single time.

- These are task are managed by using CPU scheduling or process scheduling.

- There are various schedules algorithms they are FCFS , SJF , RoundRobin etc.

- These are managed by the Operating system.

- Similarly if we want to implement multiple tasks with in a single application then we can use the thread based implementation.

- For an example if we want to run a Greeting application there are two concurrent task has to run 1. Animation 2. Sound. So to run these two task concurrently we can use Thread.

- Each Thread will be managed by JVM.

- Thread is a sequential execution with in a process.

- Thread is a light weight process as it is going to use the JVM instead the native hardware.

MainThread

- Every application when it is running it run with the main thread.

- To get the main thread we need to call the currentThread() method of the Thread Class.

```
Thread mainThread=Thread.currentThread();
```

- Any thread will have a name and priority.

- Bydefault the main thread name is main and priority is 5 we can change afterwords.

- We can retrieve the name of thread using getName() method and to change the name then we can implement setName() method.

```
public class Demo1
{
        public static void main(String arg[])
        {
            Thread mainThread=Thread.currentThread();

            System.out.println("Main Thread Name:"+mainThread.getName());
```

```
                    System.out.println("Main Thread
Priority:"+mainThread.getPriority());

                    mainThread.setName("ImpThread");
                    mainThread.setPriority(8);

                    System.out.println("==========After Changing
Value=============");

                    System.out.println("Main Thread Name:"+mainThread.getName());

                    System.out.println("Main Thread
Priority:"+mainThread.getPriority());

                }
}
```

Child Thread

- Like main thread we can create our own thread by using two process.

1. Creating a class by extending to the Thread class of java.lang package

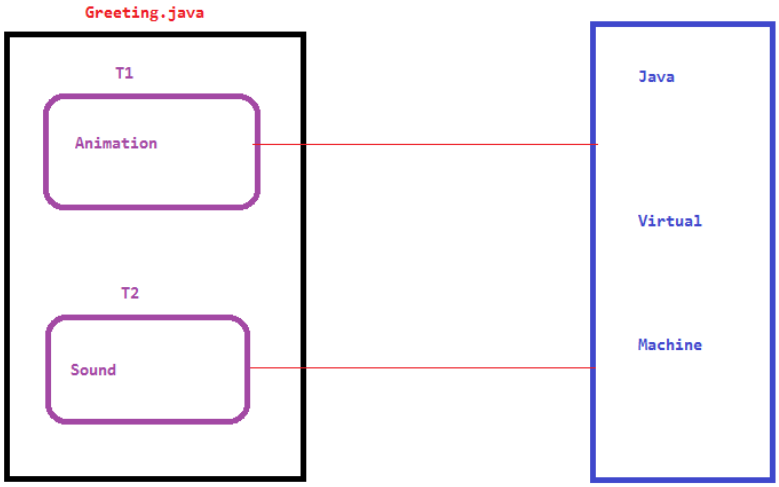
2. Creating a class by implementing Runnable interface

1. Extending to Thread Class

```
public class ChildThread extends Thread
{
                public void run()
                {
                  for(int i=1;i<=10;i++)
                  {

                System.out.println(Thread.currentThread().getName()+":"+i);
                  }
                }
                public static void main(String arg[])
```

```
        {
            ChildThread T1=new ChildThread();
            ChildThread T2=new ChildThread();
            T1.setName("FirstThread");
            T2.setName("SecondThread");
            T1.start(); //It will call run() method
            T2.start(); //It will call run() method
        }
}
```



13/01/18

Using Runnable Interface

- Runnable interface contains run() method which need to overriden by the implemented class.

- The run() method should contain the code what a thread must do.

- If a class is implementing Runnable interface then the class object can't be thread it is the classs which is called as RunnableTarget class where it contains the code what the thread has to do.

Demo : Implementing Runnable interface

```
public class RunnableTargetDemo implements Runnable
{
                    @Override
                    public void run()
                    {
                      for(int i=1;i<=10;i++)
                      {

                    System.out.println(Thread.currentThread().getName()+":"+i);
                      }
                    }

                    public static void main(String arg[])
                    {
                      RunnableTargetDemo obj1=new RunnableTargetDemo();
                            Thread T1=new Thread(obj1);
                            Thread T2=new Thread(obj1);
                            T1.start();//It will call the run() method of Thread class
                            T2.start();
                    }

}
```

Synchronized Keyword

- synchronized is a modifier which is used to implement synchronous calling of the thread.

- synchronized can be implemented to a block or to a method.

```
class Sync
{
                    public synchronized static void withDraw(String[] operations)
                    {
                      for(String operation:operations)
                      {
                              try
                              {
                                      Thread.sleep(500);
                              }
                              catch(Exception e)
                              {


                              }

                    System.out.println(Thread.currentThread().getName()+"::"+operation)
;;
                      }
                    }
}
public class ThreadSync extends Thread
{
                    String operation[]= {"Enter Pin","Enter Amount","Validate
Balance","Process Tran","Update Tran","Complete"};


                    public void run()
                    {
                      Sync.withDraw(operation);
```

```
    }
    public static void main(String arg[])
    {
      ThreadSync T1=new ThreadSync();
      T1.setName("Rohit");

      ThreadSync T2=new ThreadSync();
      T2.setName("Harish");

      T1.start();

      T2.start();
    }
}
```

Output

```
Harish::Enter Pin
Harish::Enter Amount
Harish::Validate Balance
Harish::Process Tran
Harish::Update Tran
Harish::Complete
Rohit::Enter Pin
Rohit::Enter Amount
Rohit::Validate Balance
Rohit::Process Tran
Rohit::Update Tran
Rohit::Complete
```

- In synchornous envirnoment at a given point of time one thread will execute.
- This we are implemented when multiple threads want to access the same resource.
- There is a monitor which is going to monitor in a synchronous envirnoment.

JDBC - Java Database Connectivity

- JDBC is an API (Application Programming Interface) which is used to connect to the database.

- This API will contain various classes and interfaces using which we can manipulate the database table data.

- All these classes are available in java.sql package.

- To implement JDBC we need to follow seven steps they are

1. import the java.sql. package

import java.sql.*;

2. Load and register the driver.

Driver - Driver is an interface between our application to the database.

- To load and register we can have two different approaches

1. call the forName() method of the class Class in java.lang package.

```
Class.forName("org.h2.Driver");
```

Note :

- Here forName() method is a static method and it will take the driver class name as its parameter.

- Here org.h2 is the package and Driver is the class name and it will be available in h2.jar file which we need to incorporate in our application.

- This will throw an exception called ClassNotFoundException.

2. Call the registerDriver() method of the DriverManager class.

```
org.h2.Driver myDriver=new org.h2.Driver();
DriverManager.registerDriver(myDriver);
```

- To implement the oracle database

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

- As discussed for each every database we will have separate jar file or also called dependency required to the application need to be added.

Adding the Jar File to the Application

- Right Click on the Project

- Click on the Properties

- Go to the Java Build Path
- Go to the Libraries
- Click on Add External Jars
- Select the Jar file
- Click on Apply
- Click on Ok

3. Creating a Connection Object

- To create a connection object we need to call the getConnection() method of DriverManager class.

Connection conn=DriverManager.getConnection(url,username,password);

Example

Connection conn=DriverManager.getConnection("jdbc:h2:tcp://localhost/~/DT248","cdt","cdt");

4. Create the Statement Object

- Statement object is used for built the query and send it to the database.
- There are three different type of statements which are been available they are.

1. Statement

- It is used for basically execute the static level queries

Statement statement=conn.createStatement();

2. PreparedStatement

- It is used for basically execute runtime based queries

PreparedStatement psmt=conn.prepareStatement("select * from Student where studname=?");

PreparedStatement psmt=conn.prepareStatment("insert into Employee values(?,?,?,?)");

Here ?,? called as placeholder which runtime we can pass the value for it.

3. CallableStatement

To execute the procedure based implementation

Callablestatement cs=conn.prepareCall("?procedurename");

5. Create the ResultSet

- To create the ResultSet we need to call the executeQuery() method of Statement object.

ResultSet resultset=statement.executeQuery("select * from Employee");

6. Process the ResultSet

- To process the resultset we need to use the getXXX() method and here XXX denotes the data type.

```
while(resultset.next())
{
   resultset.getInt(1);
   resultset.getString(2);
   ..............
   ..............
}
```

7. Close the Connection and Statement object

```
statement.close();
conn.close();
```

16/01/18

Creating Table in H2 Database

```
create table Emp
(
empid varchar(10),
empname varchar(40),
salary int,
address varchar(80)
)
```

Insert the Data to H2 table

```
insert into Emp values('E1001','Vinod',67000,'Delhi')
```

Displaying the Records

select * from EMP

Demo : Implementing of simple JDBC program.

```
import java.sql.*;
public class DisplayRecord
{
                    public static void main(String arg[])
                    {
                      try
                      {
                              Class.forName("org.h2.Driver");

                              Connection
conn=DriverManager.getConnection("jdbc:h2:tcp://localhost/~/DT255","dteja","dteja");

                              Statement stmt=conn.createStatement();

                              ResultSet resultSet=stmt.executeQuery("select * from
Emp");

                              while(resultSet.next())
                              {
                                      System.out.print(resultSet.getString(1)+"    ");
                                      System.out.print(resultSet.getString(2)+"    ");
                                      System.out.print(resultSet.getInt(3)+"    ");
                                      System.out.println(resultSet.getString(4));
                              }

                              stmt.close();
                              conn.close();
```

```
                    }
                    catch(Exception e)
                    {
                            System.out.println("Exception Arised:"+e);
                    }
                }
}
```

ResultSetMetaData

- ResultSetMetaData will contain the meta information of the resultset.
- Here the meta information are 1. Column name , 2. Column Datatype , 3. Column # ,
- To get the resultset metadata object we need to call the getMetaData() method.

```
ResultSetMetaData rsmd=rs.getMetaData();
```

Demo : Implementing ResultSet MetaData

```
import java.sql.*;
public class DisplayRecord
{
                    public static void main(String arg[])
                    {
                      try
                      {
                            Class.forName("org.h2.Driver");

                            Connection
conn=DriverManager.getConnection("jdbc:h2:tcp://localhost/~/DT255","dteja","dteja");

                            Statement stmt=conn.createStatement();

                            ResultSet resultSet=stmt.executeQuery("select * from
Emp");
```

```
                    ResultSetMetaData rsmd=resultSet.getMetaData();

                    int colcount=rsmd.getColumnCount();

                    for(int count=1;count<=colcount;count++)
                    {
                        System.out.print(rsmd.getColumnName(count)+"
");
                    }

                    System.out.println();

                    while(resultSet.next())
                    {
                        System.out.print(resultSet.getString(1)+"    ");
                        System.out.print(resultSet.getString(2)+"    ");
                        System.out.print(resultSet.getInt(3)+"    ");
                        System.out.println(resultSet.getString(4));
                    }

                    stmt.close();
                    conn.close();
                }
                catch(Exception e)
                {
                    System.out.println("Exception Arised:"+e);
                }
            }
}
```

PreparedStatement

- This statement is used for executing the runtime based queries.

- To create prepared statement we need to call the prepareStatement() method of connection object.

```
PreparedStatement psmt=conn.prepareStatement(String runtimeQuery);
```

Demo : Implementing Insertion to Table

```
import java.sql.*;
import java.util.*;
public class PreparedStatementDemo
{
                    public static void main(String arg[])
                    {
                      try
                      {
                              Class.forName("org.h2.Driver");

                              Connection
conn=DriverManager.getConnection("jdbc:h2:tcp://localhost/~/DT255","dteja","dteja");

                              PreparedStatement psmt=conn.prepareStatement("insert
into Emp values(?,?,?,?)");

                              String empid,empname,addr;
                              int salary;

                              Scanner scanner=new Scanner(System.in);

                              System.out.println("Enter Your Employee ID:");
                              empid=scanner.next();
                              System.out.println("Enter Your Employee Name:");
                              empname=scanner.next();
```

```
                System.out.println("Enter Your Employee Salary:");
                salary=scanner.nextInt();
                System.out.println("Enter Your Employee Address:");
                addr=scanner.next();

                psmt.setString(1, empid);
                psmt.setString(2, empname);
                psmt.setInt(3, salary);
                psmt.setString(4, addr);

                int row_eff=psmt.executeUpdate();

                if(row_eff>0)
                        System.out.println("Row Inserted");
                else
                        System.out.println("Error in Insertion");


                psmt.close();
                conn.close();
            }
            catch(Exception e)
            {
                System.out.println("Exception Arised:"+e);
            }
        }
}
```

Html

- Hypertext Markup Language

Web Page

- A web page is an electronical document which is consisting of various components they are

1. Text 2.Images 3. Links 4. List 5.Frames 6.Forms 7.Tablesand certain other programs.

Who Construct Web Page?

- Browser application construct web page for us.

- It can't create on its own it requires some instruction which we are giving using HTML.

Ex : Internet Explore , Mozilla Firefix , Opera , Chrome

Html

- This is markup language which make the text to be more informative text.

Markup Language

- Markup mean we mark the text with tags.

Tag

- A tag is word which enclosed with angular brackets.

ex : <age> , <font> , <hr>

10

<age>10</age>