Java Notes

Prepared by Uday Pawar

Java Course Content (50-55 Sessions)

1. **Variables**
2. **Datatypes**
3. **Operators**
4. **Structure**
5. **Conditional Statements**
6. **Looping Statements**

----

1. **Object, Class, Keywords and Identifiers**
2. **Methods, Constructor and Blocks**
3. **Inheritance**
4. **Overloading and Overriding**
5. **Access Modifiers and Encapsulation**
6. **Casting, Abstract keyword, Interface and Arrays**
7. **Polymorphism**
8. **Abstraction**

----

1. **Java Libraries -> String -> Lambda Functions**
2. **Exception Handling**
3. **File Handling**
4. **Multi-Threading**
5. **Collection Framework**

Note: After the completion of java course, mini project would be done.

What is Java?

- Java is a high level, platform independent, object-oriented programming language.

- High Level Language is a language which is in normal English i.e. Human Understandable Form.

- Programming Language is a medium to interact or communicate with the system.

Why do we use Java? or Features of Java.

1. Simple
2. Platform Independent
3. Extensible
4. Object Oriented
5. Automatic Garbage Collector
6. Secured
7. Robust

Working of a Java Program (or) How is Java platform independent (or) WORA Architecture

1. We develop a java program and save the file with the extension (.java).
2. Next, we compile a java program to check if there are any error in the program or not.
3. If the compilation is unsuccessful (program has errors) then we need to Debug.
4. If the compilation is successful, the byte code (intermediate code) gets generated with the extension (.class).

5. Once the Byte code is generated, we can execute (interpret) the program on all operating systems.
6. WORA stands for Write Once Run Anywhere.

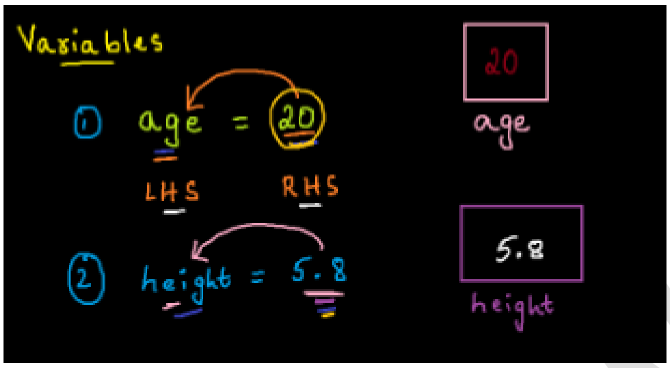
Note: Refer Screenshot day3_2

Note:

1. **Java was introduced by a company called as Sun Micro Systems.**
2. **Java is owned by a company called as Oracle presently.**
3. **James Gosling was the Person who developed JAVA.**
4. **Previous Names of Java are Green Talk and Oak.**

Variables

1. Variable is Container in order to store some data or information.
   Example:



Datatypes

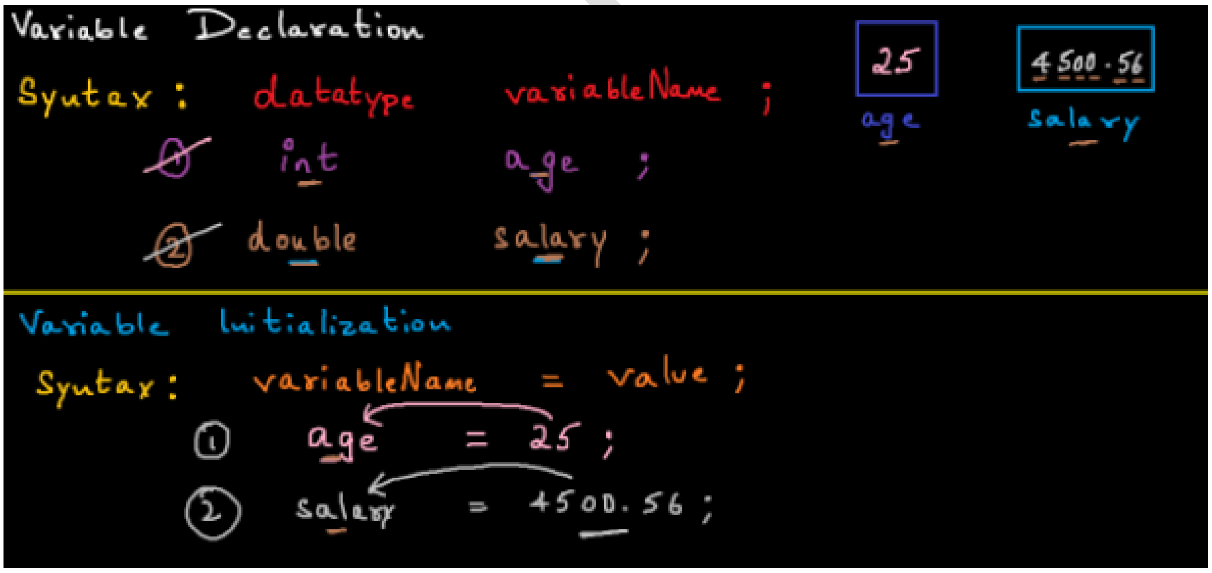
1. Datatype is an indication of the type of data stored into a variable.
2. In order to store Non-Decimal Numeric Values, we make use byte, short, int, long.
3. In order to store Decimal Numeric Values, we make use float, double.
4. In order to store true/false, we make use boolean.
5. In order to store Single Character in single quotes, we make use char.

Note: All the above 8 Datatypes are referred Primitive Datatypes

6. In order to store a sequence of characters we make use of String.

Datatype	Memory Size	
	bytes	bits
byte	1	8
short	2	16
int	4	32
long	8	64
float	4	32
double	8	64
boolean	1	8
char	2	16
String	×	×

1. Variable Declaration and Initialization



Variable Declaration

Syntax : datatype variableName ;

1 int age ;

2 double salary ;

$\boxed{25}$ age

$\boxed{4500.56}$ salary

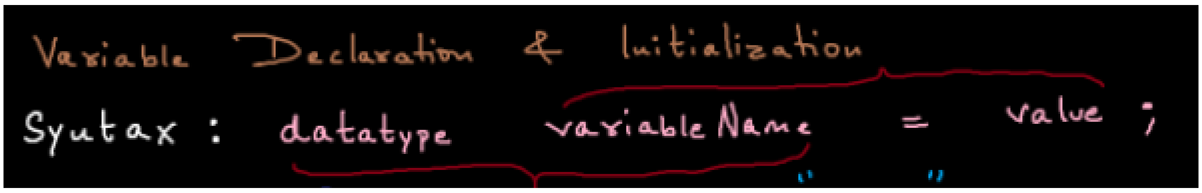
Variable Initialization

Syntax : variableName = value ;

1 age = 25 ;

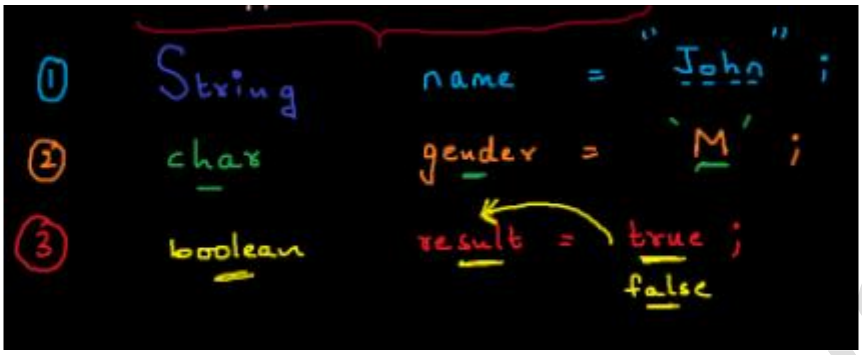
2 salary = 4500.56 ;

2. Variable Declaration and Initialization



Variable Declaration & Initialization

Syntax : datatype variableName = value ;



```
1   String    name   = "John" ;
2   char      gender = 'M' ;
3   boolean   result = true ;
                       false
```

Structure of a Java Program

```
class ClassName
{
	public static void main(String[] args)
	{
		System.out.println();
	}
}
```

Program

1.

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

o/p:

```
Hello World!!!
```

2.

```
class Greet
{
      public static void main(String[] args)
      {
            System.out.println("Welcome to Java Online Session");
      }
}
```

o/p:

```
Welcome to Java Online Session
```

3.

```
class Student
{
      public static void main(String[] args)
      {
            int age;
            age = 22;
            String name = "jerry";
            double height = 5.8;

            System.out.println(age);
            System.out.println(name);
            System.out.println(height);
      }
}
```

o/p:

```
22
jerry
5.8
```

4.

```
class Employee
{
	public static void main(String[] args)
	{
		int id = 101;
		String name = "tom";
		double salary = 1500.67;

		System.out.println(id);
		System.out.println(name);
		System.out.println(salary);

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

		System.out.println("Employee Id: "+id);
		System.out.println("Employee Name is " + name);
		System.out.println("Employee Salary = "+salary);

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

		System.out.println(id+name+salary);

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

		System.out.println(id+" "+name+" "+salary);

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

		String address = "#40, Rajajinagar, Bengaluru-51";
```

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

o/p:
101
tom
1500.67
------------
Employee Id: 101
Employee Name is tom
Employee Salary = 1500.67
------------
101tom1500.67
------------
101 tom 1500.67
------------
#40, Rajajinagar, Bengaluru-51
```

1. **Arithmetic Operators**
    a. +
    b. –
    c. *
    d. /
    e. %

2. **Assignment Operators**
    a. =
    b. +=
    c. -=
    d. *=
    e. /=
    f. %=

3. **Conditional or Comparison or Relational Operators**
    a. <
    b. <=
    c. >
    d. >=
    e. ==
    f. !=

Note: Return type is Boolean value (true or false)

4. Logical Operators

a. && -> AND
b. || -> OR
c. ! -> Not

Note: Return type is Boolean value

Truth Tables: **True-T** **False-F**

AND (&&)

A	B	O/P
T	T	T
T	F	F
F	T	F
F	F	F

OR (||)

A	B	O/P
T	T	T
T	F	T
F	T	T
F	F	F

NOT

T → F

F → T

5. Unary Operators

++ → Increment by 1

-- → Decrement by 1

int x = 5;

int y = x++;

Post-Increment -> First Assign, Then Increment

----

int a = 10;

int b = ++a;

Pre-Increment -> First Increment, Then Assign

---

int x = 5;

int y = x--;

Post-Decrement -> First Assign, Then Decrement

----

int a = 10;

int b = --a;

Pre-Decrement -> First Decrement, Then Assign

Comments

1. Additional Information which will not affect the execution of a program.

```
// Single Line Comment
/* Multi
     Line
         Comment */
```

1.

```
class ArithmeticOperators
{
     public static void main(String[] args)
     {
          int a = 10;
          int b = 20;
          int sum = a+b;

          System.out.println("Sum of "+a+" & "+b+" is "+sum);
          System.out.println("Sum = "+sum);

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

          System.out.println(a-7);
          System.out.println(b*4);
          System.out.println(10/5);
          System.out.println(10%2);

     }
}
```

o/p:

```
Sum of 10 & 20 is 30
Sum = 30
-------------
3
80
2
0
```

2.

```
class AssignmentOperators
{
	public static void main(String[] args)
	{
		int x = 5;
		System.out.println("Value of x: "+x);

		x += 20;
		System.out.println("Value of x: "+x);

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

		int a = 6;
		System.out.println("Value of a: "+a);

		// multiplying 7 with a(6) and re-initializing back to a
itself
		a *= 7;
		System.out.println("Value of a: "+a);
	}
}
```

o/p:

```
Value of x: 5
Value of x: 25
---------------
Value of a: 6
Value of a: 42
```

3.

```
class ComparisonOperators
{
	public static void main(String[] args)
	{
		int x = 10;
		int y = 20;

		boolean result1 = x<y;
		boolean result2 = x<=10;

		System.out.println(result1); // true
		System.out.println(result2); // true

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

		System.out.println(x>y);    // false
		System.out.println(y>=21);  // false
		System.out.println(y>=20);  // true

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

		System.out.println(x == 10);    // true
		System.out.println(x == y);     // false

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

		System.out.println(x != 10);    // false
		System.out.println(x != y);  // true

	}
}

o/p:
true
true
-------------
false
false
true
-------------
true
false
-------------
false
true
```

4.

```
class LogicalOperators
{
      public static void main(String[] args)
      {
            int a = 1;
            int b = 2;

            boolean result1 = a<b && b>a; // 1<2 && 2>1 // true &&
true
            boolean result2 = a>5 && b<=2; // 1>5 && 2<=2 // false &&
true

            System.out.println(result1); // true
            System.out.println(result2); // false
            System.out.println("---------");

            System.out.println(1<=2 || 4>2); // true
            System.out.println(1>2 || 5==6); // false

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

            System.out.println(!true); // false
            System.out.println(!false); // true

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

            System.out.println( !(1<2) ); // false
            System.out.println(!(20<10)); // true
      }
}
```

```
o/p:
true
false
---------
true
false
---------
false
true
---------
false
true
```

5.

```
class UnaryOperators
{
      public static void main(String[] args)
      {
            int i=5;
            System.out.println("i: "+i); // 5

            i++;
            System.out.println("i: "+i); // 6

            i++;
            System.out.println("i: "+i); // 7

            i--;
            System.out.println("i: "+i); // 6

            i--;
            System.out.println("i: "+i); // 5

            i++;
            System.out.println("i: "+i); // 6
      }
}
```

```
o/p:
i: 5
i: 6
i: 7
i: 6
i: 5
i: 6
```

6.

```
class Demo
{
      public static void main(String[] args)
      {
            int a = 20;
            int b = a++;
            System.out.println(a+" "+b); // 21 20

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

            int x = 88;
            int y = x--;
            System.out.println(x+" "+y); // 87 88

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

            int i = 45;
            int j = ++i;
            System.out.println(i+" "+j); // 46 46
            System.out.println("-------------");

            int p = 10;
            int q = --p;
            System.out.println(p+" "+q); // 9 9

      }
}
```

```
o/p:
21 20
-------------
87 88
-------------
46 46
-------------
9 9
```

Conditional Statements or Decisional Statements

- Conditional Statements are used to take some decision based on the condition specified.
- They are used for decision making.
- Different Decisional Statements are as follows:
  1. Simple If
  2. If Else
  3. If Else If
  4. Nested If
  5. Switch Statement

1. Simple If:

- It is a Decision-Making statement wherein we execute a set of the instructions if the condition is true.

```
1.
class IfDemo
{
        public static void main(String[] args)
        {
           System.out.println("Start");

           int n = 10;

           if(n<=10) //if(10<=10) --> if(true)
           {
                System.out.println(n+" is lesser than or equal
to 10");
           }

           System.out.println("End");
        }
}

o/p:
Start
10 is lesser than or equal to 10
En
```

2. If Else:

It is a Decision-Making statement wherein we execute a set of the instructions if the condition is true and another set of instructions if the condition is false.

```
2.
class IfElseDemo
{
        public static void main(String[] args)
        {
           // Check if a No is Positive or Negative

           int n = 10;

           if(n>0) // if(10>0) -> if(true)
           {
                System.out.println(n+" is a Positive Number");
           }
           else
           {
                System.out.println(n+" is a Negative Number");
           }

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

           if(true)
           {
                System.out.println("I am in Bengaluru");
           }
           else
           {
                System.out.println("I am not in Bengaluru");
           }

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

           if(false)
           {
                System.out.println("java");
           }
           else
           {
                System.out.println("sql");
           }
```

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

		int x = 20;
		int y = 100;

		System.out.println("x:"+x+" y:"+y); // x:20 y:100

		if(x>y) // if(20>100) // if(false)
		{
			System.out.println("x is largest");
		}
		else
		{
			System.out.println("y is largest");
		}
	}
}

o/p:
10 is a Positive Number
----------------------
I am in Bengaluru
----------------------
sql
----------------------
x:20 y:100
y is largest


3.
class EvenOrOdd
{
	public static void main(String[] args)
	{
		int number = 21;

		if(number%2 == 0) // if(21%2 == 0) -> if(1 == 0) ->
if(false)
		{
			System.out.println(number+" is a Even
Number");
		}
		else
		{
			System.out.println(number+" is a Odd Number");
```

```
                }
            }
    }

o/p:
21 is a Odd Number
```

3. If Else If

- If Else If Condition is used when we need to check or compare multiple conditions.

1.

```
class IfElseIfDemo
{
	public static void main(String[] args)
	{
		int a = 20;
		int b = 20;

		System.out.println("a:"+a+" b:"+b);

		if(a<b)
		{
			System.out.println("a is lesser than b");
		}
		else if(a>b)
		{
			System.out.println("a is greater than b");
		}
		else		// else if(a==b)
		{
			System.out.println("a is equal to b");
		}
	}
}
```

o/p:

```
a:20 b:20
a is equal to b
```

2.

```
class MarksValidation
{
	public static void main(String[] args)
	{
		int marks = -26;

		if(marks>=81 && marks<=100)
		{
			System.out.println("DISTINCTION");
		}
		else if(marks>=35 && marks<=80)
		{
			System.out.println("FIRST CLASS");
		}
		else if(marks>=0 && marks<=34)
		{
			System.out.println("FAIL");
		}
		else
		{
			System.out.println("INVALID MARKS");
		}
	}
}
```

```
o/p:INVALID MARKS
```

4. Nested If

- Nested If is a decision-making statement wherein one if condition is present inside another if condition.

3.

```
class NestedIf
{
      public static void main(String[] args)
      {
            int n = 70;

            if(n<=10)       // outer if
            {
                  System.out.println(n+" is lesser than or equal to 
10");

                  if(n==7) // inner if
                  {
                        System.out.println(n+" is equal to 7");
                  }
                  else    // inner else
                  {
                        System.out.println(n+" is not equal to 7");
                  }
            }
            else        // outer else
            {
                  System.out.println(n+" is greater than 10");
            }
      }
}
```

```
o/p:
70 is greater than 10
```

4.

```
class LoginValidation
{
	public static void main(String[] args)
	{
		String id = "pawar";
		int password = 1234;

		if(id == "uday") // false
		{
			System.out.println("User Id is Valid");

			if(password == 1234)   // false
			{
				System.out.println("Password is Valid");
				System.out.println("Login Successful");
			}
			else
			{
				System.out.println("Password is Invalid");
				System.out.println("Login Unsuccessful");
			}
		}
```

```
          else
          {
                System.out.println("User Id is Invalid");
                System.out.println("Login Unsuccessful");
          }
     }
}
```

```
o/p:
User Id is Invalid
Login Unsuccessful
```

5.

```
class LargestOfThreeNumbers {

     public static void main(String[] args) {

          int a = 10;
          int b = 50;
          int c = 300;

          System.out.println("a="+a+" b="+b+" c="+c);

          if(a>b && a>c) {
                System.out.println("a is largest");
          }
          else if (b>a && b>c) {
                System.out.println("b is largest");
          }
          else {
                System.out.println("c is largest");
            }


        }
    }
```

```
o/p:
a=10 b=50 c=300
c is largest
```

Switch Statement:

- **Switch Statement is generally used for Character Comparison.**

```
1.
class SwitchExample
{
        public static void main(String[] args)
        {
                int choice = 30;

                switch(choice)
                {
                        case 1 : System.out.println("In Case 1");
1                                                 break;

                        case 2 : System.out.println("In Case 2");
                                                  break;

                        case 3 : System.out.println("In Case 3");
                                                  break;

                        default : System.out.println("Invalid Choice");
                }
        }
}
```

```
o/p:
Invalid Choice
```

2.

```
class GradeValidation
{
	public static void main(String[] args)
	{
		char grade = 'C';

		switch(grade)
		{
			case 'A' : System.out.println("Excellent");
						break;

			case 'B' : System.out.println("Good");
						break;

			case 'C' : System.out.println("Average");
						break;

			case 'D' : System.out.println("BAD!!");
						break;

			default : System.out.println("Invalid Grade");
		}
	}
}
```

o/p:

```
Average
```

Looping Statements:

- **Looping Statements are generally used to perform repetitive task.**
- **Loops are used to repeat the execution of a set of instructions and traverse a group of elements.**
- **Different Looping Statements are as follows:**
  - **For Loop**
  - **While Loop**
  - **Do-While Loop**
  - **Nested For Loop**

For Loop:

- **For loop is used to execute a set of instructions for a fixed no of times.**
- **It has a logical start point and end point.**

```
1.
class ForLoopDemo
{
	public static void main(String[] args)
	{
		System.out.println("Hello");
		System.out.println("Hello");
		System.out.println("Hello");
		System.out.println("Hello");
		System.out.println("Hello");

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

		for(int i=1; i<=5; i++)
		{
			System.out.println("Hello World!");
		}
```

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

          for(int i=1; i<=5; i++)
          {
                System.out.println(i);
          }

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

          for(int i=156; i<=160; i++)
          {
                System.out.println("i: "+i);
          }

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

          for(int i=2; i<=10; i=i+2)  // i=i+2 -> i+=2  -> i++ ->
i+=1 -> i=i+1
          {
                System.out.println(i);
          }

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

          for(int i=1; i<=9; i+=2)  // 1 3 5 7 9
          {
                System.out.println(i);
          }

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

```
            for(int i=3; i<=15; i+=3)  // 3 6 9 12 15
            {
                  System.out.println(i);
            }
      }
}
```

```
o/p:
Hello
Hello
Hello
Hello
Hello
------------
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
------------
1
2
3
4
5
------------
i: 156
i: 157
```

```
i: 158

i: 159

i: 160

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

2

4

6

8

10

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

1

3

5

7

9

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

3

6

9

12

15
```

While Loop:

- While loop is a looping statement which keeps on executing until the condition is false.

Do-While Loop:

- Do-While loop is similar to while loop but do while loop executes a set of instructions and then checks the condition.

1.

```
class WhileLoopDemo
{
	public static void main(String[] args)
	{
		int i = 1;

		while(i<=5)
		{
			System.out.println(i);
			i++;
		}

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


		int n = 1;

		do
		{
```

```
                System.out.println(n);
                n++;
            }
            while(n<=5);

        }
}
```

```
o/p:
1
2
3
4
5
---------
1
2
3
4
5
```

****

Difference between while loop and do-while loop

While Loop	Do While Loop
1. Checks Condition first and then executes a set of instructions.	1. Executes a set of instructions first and then checks the condition.
2. Does not execute even once if the initial condition is false.	2. executes at least once even if the initial condition is false.

Break:

- **Break is a keyword which is used to transfer the control outside the currently executing loop.**

1.

```
class BreakDemo
{
	public static void main(String[] args)
	{
		System.out.println("start");

		for(int i=1; i<=10; i++)
		{
			if(i==3)
			{
				break;
			}

			System.out.println(i); // 1 2
		}

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

		for(int i=1; i<=10; i++)
		{
			System.out.println(i); // 1 2 3

			if(i==3)
			{
				break;
			}
```

```
        }

        System.out.println("end");
    }
}
```

```
o/p:
start
1
2
-------
1
2
3
end
```

Continue:

- Continue is a keyword which is used to transfer the control back to the update part.

1.

```
class ContinueDemo
{
    public static void main(String[] args)
    {

        for(int i=1; i<=5; i++)
        {
            if(i==4)
            {
                continue;
            }

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

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

        for(int i=1; i<=5; i++)
        {
            System.out.println(i); // 1 2 3 4 5

            if(i==4)
            {
                continue;
            }
        }

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

        for(int i=1; i<=5; i++)
        {
            if(i==2 && i==3)
            {
                continue;
            }
            System.out.println(i); // 1 2 3 4 5
        }

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

        for(int i=1; i<=5; i++)
        {
            if(i==2 || i==3)
            {
                continue;
            }

            System.out.println(i); // 1 4 5
        }
    }
}
```

```
o/p:
1
2
3
5
------
1
2
3
4
```

```
5
------
1
2
3
4
5
------
1
4
5
```

Object Oriented Programming

Object

- Anything which is present in the real world and physically existing can be termed as an Object.
- The Properties of every object is categorized into 2 types:
    - States
    - Behaviours
- States are the properties used to store some data.
- Behaviour are the properties to perform some task.

Class

- class is a blue print of an object.
- It's a platform to store states and behaviours of an object.
- class has to be declared using class keyword.
- class can also act as datatype.

Object Creation or Instantiation

- Object is a copy or instance of a class.

- In order to store or load non static members/properties within the memory, we need to create an object.

- Objects are created inside a memory location called as HEAP Area.

- new Operator is responsible for creating an object internally.

syntax: ClassName referenceName = new ClassName();

- Any number of objects can be created for a class.

- We can access non static properties in same class or another class with the help of object creation and dot operator.

syntax: objectReferenceName.variableName

1.

```
class Student
{
	// NON STATIC VARIABLES
	int age = 20;
	String name = "Dinga";
	double height = 5.8;

	public static void main(String[] args)
	{
		Student s = new Student();  // OBJECT CREATION

		System.out.println(s.age);
		System.out.println(s.name);
		System.out.println(s.height);
	}
}
```

o/p:

```
20
Dinga
5.8
```

2.

```
class Car
{
	String brand = "BMW";
	int cost = 4500;

	public static void main(String[] args)
	{
		Car c = new Car();

		System.out.println(c.brand);
		System.out.println(c.cost);

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

		System.out.println(c.brand+" "+c.cost);

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

		System.out.println("Cost of "+c.brand+" is Rs."+c.cost);
	}
}
```

o/p:

```
BMW
4500
-------
BMW 4500
-------
Cost of BMW is Rs.4500
```

3a.

```
class Employee
{
	int id = 101;
	String name = "Tim Cook";
	double salary = 1500.56;
}
```

```
3b.
/* Accessing non static members in different class */

class Test
{
	public static void main(String[] args)
	{
		Employee emp = new Employee();

		System.out.println(emp.id);
		System.out.println(emp.name);
		System.out.println(emp.salary);
	}
}
```

o/p:

```
101
Tim Cook
1500.56
```

Default Value

- If a variable is declared and not initialized to any value, then the compiler will automatically initialize to its default value.

- Default values are applicable only for Member Variables (Static and Non-Static Variables).

Default values are as follows

byte, short, int, long ----> 0

float, double ----> 0.0

char ----> '/n10' (Unicode value) Java does not understand empty white space( )

boolean ----> false

String ----> null

Assignment: Write a Java Program to store 3 Employee Details wherein every employee will have id, name, email and salary.

1.

```
class Person {

      int age = 10;
      String name = "Tom";

      public static void main(String[] args) {

            Person p1 = new Person();
            Person p2 = new Person();
            System.out.println(p1.age+" "+p1.name); // 10 Tom
            System.out.println(p2.age+" "+p2.name); // 10 Tom

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

            p1.age = 33;
            p2.name = "Jerry";

            System.out.println(p1.age+" "+p1.name); // 33 Tom
            System.out.println(p2.age+" "+p2.name); // 10 Jerry
      }

}
```

```
o/p:
10 Tom
10 Tom
**************
33 Tom
10 Jerry
```

2.

```
class DefaultValuesDemo {

      int a;
      double b;
      char c;
      boolean d;
      String e;
```

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

		DefaultValuesDemo dvd = new DefaultValuesDemo();

		System.out.println(dvd.a);
		System.out.println(dvd.b);
		System.out.println(dvd.c);
		System.out.println(dvd.d);
		System.out.println(dvd.e);

	}

}
```

```
o/p:
0
0.0

false
null
```

3.

```
class Student {

	int age;
	String name;

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

```
            Student s1 = new Student();
            Student s2 = new Student();

            s1.age = 21;
            s1.name = "Tom";

            s2.age = 22;
            s2.name = "Jerry";

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

            System.out.println(s1.name+" is "+s1.age+" years old");

            System.out.println(s2.name+" is "+s2.age+" years old");

      }

}
```

o/p:

```
Student Details
---------------
Tom is 21 years old
Jerry is 22 years old
```

4.

```
class Demo
{
	public static void main(String[] args) {

		System.out.println(10);
		System.out.println(20);
		System.out.println(30);

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

		System.out.print(1);
		System.out.print(2);
		System.out.print(3);
	}

}
```

```
o/p:
10
20
30
----
123
```

5.

```
class Test
{
	public static void main(String[] args) {

		System.out.println(10);
		System.out.print(20+"hello");
		System.out.println(30+" ");
```

```
                System.out.print("bye");
                System.out.print(" java");
                System.out.println("program "+" hi");
                System.out.print("done");

        }

}
```

```
o/p:
10
20hello30
bye javaprogram  hi
done
```

Methods or Functions

1. A Method is a set of instructions or a block of code in order to perform a specific task.

```
syntax: AccessSpecifier returnType methodName( arguments )
       {
            /* statement-1
                    .
                    .
               statement-n
                 return     */
       }
```

2. If we want to execute a method, we need to call it or invoke it.

syntax: objectName.methodName();

3. A method can be called multiple times.

4. Reusuablity of code is increased with the help of methods.

5. If a method returns something, store it or print it.

void: void is an indication to the caller, that the method does not return anything.

return: return is used to transfer the control back to the caller.

Different ways of writing a method:

1. Method without arguments and without return statement.
2. Method with arguments and without return statement.
3. Method without arguments and with return statement.
4. Method with arguments and with return statement.

1.

```
class Demo
{
      /* Method without arguments and without return statement */
      void display() // non static method
      {
            System.out.println("Hello World");
      }

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

            Demo d = new Demo();  // Object Creation

            d.display();    // calling or invoking method

            System.out.println("end");
      }
}
```

o/p:

```
start
Hello World
end
```

2.

```
/* Write a Java Program in order to create a method to add 2 nos */

class Addition
```

```
{
	/* Method with arguments and without return statement */
	void add(int a, int b)
	{
		int sum = a+b;
		System.out.println("Sum of "+a+" & "+b+" is "+sum);
	}

	public static void main(String[] args)
	{
		Addition obj = new Addition();

		obj.add(4,8);
		obj.add(10, 20);
		obj.add(786, 234);
	}
}
```

```
output:
Sum of 4 & 8 is 12
Sum of 10 & 20 is 30
Sum of 786 & 234 is 1020
```

3.

```
/* Write a Java Program in order to create a method
            to find the product of 3 Numbers */

class Multiplication
{
	void multiply(int x, int y, int z)
```

```
        {
                System.out.println(x*y*z);
        }

        public static void main(String[] args)
        {
                Multiplication m = new Multiplication();

                m.multiply(2, 3, 4);
        }
}
```

```
o/p:
24
```

4.

```
class Test
{
        /* Method without arguments and with return statement */
        String display()
        {
                return "dinga";
        }

        public static void main(String[] args)
        {
                Test t = new Test();

                String name = t.display();
```

```
            System.out.println("My Name is not "+name);

            System.out.println(t.display());
      }
}
```

```
o/p:
My Name is not dinga
dinga
```

5.

```
class Example
{
      /* Method with argument and with return statement */
      int findPower(int n)
      {
            return n*n;
      }

      public static void main(String[] args)
      {
            Example e = new Example();

            int result = e.findPower(5);
            System.out.println(result);

            System.out.println(e.findPower(4));
      }
}
```

```
o/p:
25
16
```

6a.

```
class Solution {

	/* Method without arguments and without return statement */
	void m1() {
		System.out.println("Welcome to Java Online Session");
	}

	/* Method with arguments and without return statement */
	void m2(String name, int id) {
		System.out.println("Name: "+name);
		System.out.println("Id: "+id);
	}

	/* Method without arguments and with return statement */
	double m3() {
		return 45.6;
	}

	/* Method with arguments and with return statement */
	double m4(double x, double y) {
		return x+y;
	}

}
```

6b.

```
class MainClass {

	public static void main(String[] args) {

		Solution s = new Solution();

		s.m1();
		System.out.println("------");

		s.m2("Guldu", 123);
		System.out.println("------");

		double x = s.m3();
		System.out.println(x);

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

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

		double sum = s.m4(1.2, 3.2);
		System.out.println(sum);

		System.out.println(s.m4(1.1, 7.2));
	}
}
```

output:

```
Welcome to Java Online Session
------
```

```
Name: Guldu
Id: 123
------
45.6
45.6
------
4.4
8.3
```

Method Overloading

1. In a class having multiple methods with the same name, but difference in arguments is called as Method Overloading.

In order to achieve method overloading we need to satisfy either 1 of the following 3 rules.

1. There should be a change in the No of Arguments.

2. There should be a change in the Datatype of the Arguments.

3. There should be a change in the order/sequence of the Datatypes.

Note:

1. Both Static and Non-Static methods can be Overloaded.

2. Yes, we can overload Main() as well, But the execution starts from the main() which accepts String[] as the argument.

3. returntype might be same or different.

4. Method Overloading is also referred as Compile time Polymorphism.

1.

```
package jspiders;


class Demo

{

	public static void main(String[] args)

	{
		System.out.println("Hello World");
	}
}
```

```
o/p:
Hello World
```

2a.

```
package com;

class Example
{
	void display(int a)
	{
		System.out.println("a: "+a);
	}

	void display(double a)
	{
		System.out.println("a: "+a);
	}

	void display()
	{
		System.out.println("Hello DabbaFellow");
	}

	void display(int x, String y)
	{
		System.out.println("x:"+x+" y:"+y);
	}

	void display(String x, int y)
	{
		System.out.println("x:"+x+" y:"+y);
	}

}
```

2b.

```
package com;

class Solution
{
	public static void main(String[] args)
	{
		Example e = new Example();

		e.display();

		e.display(45);

		e.display(10, "java");

		e.display(45.6);

		e.display("tom", 34);
	}
}
```

```
o/p:
Hello DabbaFellow
a: 45
x:10 y:java
a: 45.6
x:tom y:34
```

3.

```
package com;

class Netflix
{
	void login(String emailId, int password)
	{
		System.out.println("Email Id: "+emailId);
		System.out.println("Password: "+password);
	}

	void login(int contactNo, int password)
	{
		System.out.println("Contact No: "+contactNo);
		System.out.println("Password: "+password);
	}

	public static void main(String[] args)
	{
		Netflix n = new Netflix();

		n.login(987745, 123);
```

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

                n.login("dinga@gmail.com", 456);
        }
}
```

o/p:

```
Contact No: 987745
Password: 123
-------
Email Id: dinga@gmail.com
Password: 456
```

4.

```
package com;

class Test
{
        public static void main(int a)
        {
                System.out.println("java");
        }

        public static void main(double a)
        {
                System.out.println("bye");
        }

        public static void main(String[] args)
```

```
	{
		System.out.println("hello");
		main(10);
		main(10.5);
	}
}
```

```
o/p:
hello
java
bye
```

Scanner

1. Scanner is a pre-defined class in java.util package.

2. Scanner class is used to accept dynamic input from the User.

Rules to accept dynamic input from the user (or) Rules to work around with Scanner class

1. Create an object of Scanner class.

2. Pass System.in to the Constructor call.

syntax: `Scanner scan = new Scanner(System.in);`

3. import Scanner class from java.util package.

syntax: `import java.util.Scanner;`

4. Make use of pre-defined methods to accept dynamic input.

Important method used with respect to Scanner class

```
1. byte   - nextByte()
2. short  - nextShort()
3. int    - nextInt()
4. long   - nextLong()
5. float  - nextFloat()
6. double - nextDouble()
7. boolean - nextBoolean()
```

8. String - next() or nextLine()

9. char   - next().charAt(0)

1.

```
package com;

import java.util.Scanner;

public class Demo {

	public static void main(String[] args) {

		Scanner scan = new Scanner(System.in);

		System.out.println("Enter the value of a:");
		int a = scan.nextInt();

		System.out.println("Enter the value of b:");
		int b = scan.nextInt();

		System.out.println(a+b);

	}

}
```

o/p:

```
Enter the value of a:
```

```
10

Enter the value of b:

20

30
```

2.

```
package com;

import java.util.Scanner;

public class Person {

	public static void main(String[] args) {

		Scanner s = new Scanner(System.in);

		System.out.println("Enter Age:");
		int age = s.nextInt();

		System.out.println("Enter Name:");
		String name = s.next();

		System.out.println("Enter Height:");
		double height = s.nextDouble();

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

		System.out.println(name+" is "+age+" years old and his 
height is "+height);
```

```
	}
}
```

```
o/p:
Enter Age:
25
Enter Name:
tom
Enter Height:
5.6
---------------
tom is 25 years old and his height is 5.6
```

3.

```
package com;

import java.util.Scanner;

public class Calculator
{
	void add(int x, int y)
	{
		int sum = x+y;
		System.out.println("Sum of "+x+" & "+y+" is "+sum);
		System.out.println("-----------------");
	}

	public static void main(String[] args)
	{
		Calculator c = new Calculator();

		Scanner s = new Scanner(System.in);

		for(int i=1; i<=3; i++)
		{
			System.out.println("Enter 2 Numbers:");
			int x = s.nextInt();
			int y = s.nextInt();
			c.add(x,y);
		}
	}
}
```

```
o/p:
Enter 2 Numbers:
1
2
Sum of 1 & 2 is 3
-----------------
Enter 2 Numbers:
5
36
Sum of 5 & 36 is 41
-----------------
Enter 2 Numbers:
896
5
Sum of 896 & 5 is 901
-----------------
```

4.

```
package com;

import java.util.Scanner;

public class Test {

      void findPosOrNeg(int n) {

            if(n>0) {
                  System.out.println(n+" is a Positive Number");
            }
            else {
                  System.out.println(n+" is a Negative Number");
            }

      }

      public static void main(String[] args) {

            Test t = new Test();
            Scanner s = new Scanner(System.in);

            for(int i=1; i<=3; i++)
            {
                  System.out.println("Enter Number:");
                  t.findPosOrNeg(s.nextInt());
                  System.out.println("--------------");
```

```
          }
     }
}

/*int n = s.nextInt();
t.findPosOrNeg(n);*/

o/p:
Enter Number:
7
7 is a Positive Number
--------------
Enter Number:
-9
-9 is a Negative Number
--------------
Enter Number:
6
6 is a Positive Number
--------------
```

static

1. static is a keyword which can be used with class, variable, method and blocks.

2. The Class Loader loads all the static properties inside a memory location called as Class Area or Static Pool.

3. All the static properties has to accessed with help of ClassName.

4. static properties can be accessed directly or with the help of ClassName in the same class.

5. static properties can be accessed only with the help of ClassName in the different/another class.

syntax: ClassName.variableName or ClassName.methodName()

Note:

1. STATIC PROPERTIES ARE LOADED ONLY ONCE, THEREFORE THEY WILL HAVE A SINGLE COPY.

2. All Objects will implicitly be pointing to the static pool, therefore we can access static properties with object reference but not a good practice.

1.

```
package com;


/* Accessing static members inside same class */


class Student {


	static int age = 20;
	static void study() {
		System.out.println("Studying");
	}

	public static void main(String[] args) {

		System.out.println(Student.age);
		Student.study();

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

		System.out.println(age); // ClassName.age -> Student.age
		study(); // ClassName.study();  ->  Student.study();

	}
}
```

o/p:

```
20
Studying
------
20
Studying
```

2a.

```
package com;

class Employee {

	static int id = 1203;
	static void work() {
		System.out.println("Working");
	}
}
```

2b.

```
package com;

/* Accessing static members in different class */

public class Test {

	public static void main(String[] args) {

		System.out.println(Employee.id);
		Employee.work();

		// Error System.out.println(id);  ClassName.id -> Test.id

		// Error work(); ClassName.work(); -> Test.work();

	}

}
```

o/p:

```
1203
Working
```

3.

```
package com;

class Company {

	static String companyName;

	String branchName;

	public static void main(String[] args) {

		Company.companyName = "Jspiders";

		Company c1 = new Company();
		c1.branchName = "RajajiNagar";

		Company c2 = new Company();
		c1.branchName = "BTM";

	}
}
```

Blocks

1. Blocks are a set of Instructions/Block of code used for initialization.

2. Blocks are generally categorized into

   a. static block

   b. non-static block

static block

1. Static blocks are a set of instructions used to initializing static variables.

syntax:
```
static
{


}
```

2. Static blocks always gets executed even before main() or during class loading time.

3. We can have Multiple Static Blocks and the execution will happen in a sequential Manner.

1.

```
package com;

class Demo {

	static {
		System.out.println("In Static Block-1");
	}
	static {
		System.out.println("In Static Block-2");
	}

	public static void main(String[] args) {
		System.out.println("Hello");
	}

	static {
		System.out.println("In Static Block-3");
	}
}
```

```
o/p:
In Static Block-1
In Static Block-2
In Static Block-3
Hello
```

2.

```
package com;

class Student {

	static int age;

	static
	{
		System.out.println("Initializing age to 20");
```

```
          age = 20;    //Student.age = 20;
      }

      public static void main(String[] args)
      {
          System.out.println("Age: "+Student.age);
      }

      static
      {
          System.out.println("Initializing age to 30");
          age = 30;
      }

}

// age = 0 -> 20 -> 30

o/p:
Initializing age to 20
Initializing age to 30
Age: 30
```

non-static block (Instance Block)

1. Non-Static blocks are a set of instructions used to initializing static variables and non-static variables.

syntax:

```
          {
         }
```

2. Non-Static blocks always gets executed during Object creation (Instantiation).

3. We can have Multiple Non-Static Blocks and the execution will happen in a sequential Manner.

1.

```
package com;

class Test
{

    {
         System.out.println("in non-static block-1");
    }

    {
         System.out.println("in non-static block-2");
    }

    public static void main(String[] args)
    {
         Test t = new Test();
    }

    {
         System.out.println("in non-static block-3");
    }
}
```

```
o/p:
in non-static block-1
in non-static block-2
in non-static block-3
```

2.

```
package com;

class Employee {

      int id;

      {
            id = 100;
      }

      public static void main(String[] args)
      {
            Employee e = new Employee();
            System.out.println(e.id);
      }

      {
            id = 200;
      }

}
```

```
// id = 0 -> 100 -> 200
```

o/p:

```
200
```

3.

```
package com;

class Solution
{
	static
	{
		System.out.println(20);
	}

	public static void main(String[] args)
	{
		System.out.println(10);
		Solution s = new Solution();
	}

	{
		System.out.println(30);
	}

}
```

o/p:

```
20

10

30
```

4.

```
package com;

class Car
{
	static int a; // 0 -> 20
	int b; // 30

	static
	{
		a = 20;
	}

	{
		b = 30;
	}

	public static void main(String[] args)
	{
		System.out.println(Car.a); // 20

		Car c = new Car();
		System.out.println(c.b); // 30

	}
}
```

```
o/p:
20
30
```

JDK

- Java Development Kit

- JDK is a software which contains all the resources in order to develop and execute java programs.

JRE

- Java Runtime Environment

- JRE is a software which provides a platform for executing java programs.

JIT Compiler

- Just In Time Compiler

- JIT Compiler complies/coverts java program(High Level) into machine understandable language.

Class Loader

- Loads the Class from secondary storage to executable area.

Interpreter

- Interprets the code line by line.

JVM

- Java Virtual Machine
- JVM is the Manager of the JRE.

JVM Architecture

1. Heap Area - Objects get created here.
2. Class Area - or Static Pool - All the static members gets stored here.
3. Stack - Execution happens inside stack.
4. Method Area - Implementation of methods is stored here.
5. Native Area

1.

```
package com;

class Example
{
	static void m1(int a) // a=10
	{
		System.out.println("Hai");
		System.out.println(m2(50));
		System.out.println("Bye");
	}

	static int m2(int n) // n=50
	{
		return n+100;
	}

	public static void main(String[] args)
```

```
        {
                System.out.println("Start");
                m1(10);
                System.out.println("End");
        }
}
```

```
o/p:

    Start
    Hai
    150
    Bye
    End
```

Constructor

1. Constructor is a set of instructions used for initialization(Assigning) and Instantiation (Object Creation).

2. Constructor Name and Class Name should always be same.

3. Constructors will not have return type.

4. Constructors will get executed at the time of object creation.

5. Constructors are categorized into 2 types:

   a. Default Constructor

   b. Custom/User-Defined Constructor

syntax:
```
AccessSpecifier ClassName(optional arguments)
      {
          // Set of Instructions
      }
```

Default Constructor

1. If a constructor is not explicitly present in a class, then the compiler will automatically generate a constructor and those constructors are called as Default Constructor.

2. Default constructor neither accepts any arguments nor has any implementation.

Custom/User-Defined Constructor

1. If a constructor is explicitly defined inside a class by the user or the programmer, then we refer it as custom/user-defined constructor.

2. They are further categorized into 2 types:

   i. Non-Parameterized Custom Constructor

   ii. Parameterized Custom Constructor

NOTE: WHEN THERE IS DEFAULT CONSTRUCTOR, THEN CUSTOM CONSTRUCTOR CANNOT BE PRESENT AND VICE VERSA.

Global/Member Variable and Local variable

1. Global/Member variables are those variables which are declared in the class limit/Scope.

2. They can be accessed globally ie. throughout the class.

3. Global/Member variables are categorized into

   a. static

   b. Non-static

4. Local Variables are those variables which are declared within a specific scope or limit such as method, constructor, etc.

5. Local variables are accessible within that specific scope.

6. Default Values are applicable only for Member Variables.

this keyword

1. In java, we can have both member/global and local variable names same, then always the local variables will dominate the member variables.

2. In order to avoid the dominating part we make use of "this" keyword.

3. this is keyword which is used to point to the current object/instance.

*******************PROGRAMS*******************

1.

```
package com;

class Car
{
	/* Non Parameterized Custom/User-Defined Constructor */
	Car()
	{
		System.out.println("In Car Class Constructor");
	}

	public static void main(String[] args)
	{
		System.out.println("start");
		Car c = new Car();
		System.out.println("end");
	}
}
```

o/p:

```
start
In Car Class Constructor
end
```

2.

```
package com;

class Bike
{
	/* Parameterized Custom/User-Defined Constructor */
```

```
    Bike(String brand)
    {
        System.out.println("Brand: "+brand);
    }

    public static void main(String[] args)
    {
        Bike b = new Bike("suzuki");
    }
}
```

```
o/p:
Brand: suzuki
```

3.

```
package com;

class Kangaroo
{
    double height = 5.5; // Member/Global Variable

    void display()
    {
        double height = 4.4; // Local Variable

        System.out.println(height);
        System.out.println(this.height);
    }

    public static void main(String[] args)
```

```
	{
		Kangaroo k = new Kangaroo();
		k.display();
	}
}
```

```
o/p:
4.4
5.5
```

4.

```
package com;

class Student
{
	int age;

	Student(int age)
	{
		this.age = age;
	}

	public static void main(String[] args)
	{
		Student s1 = new Student(25);
		Student s2 = new Student(24);

		System.out.println("Age:"+s1.age);
		System.out.println("Age:"+s2.age);
	}
}
```

o/p:

```
Age:25
Age:24
```

5.

```
package com;

class Person {

	String name;

	Person(String name) {
		this.name = name;
	}

	public static void main(String[] args) {

		Person p1 = new Person("Dinga");
		Person p2 = new Person("Tom");
		Person p3 = new Person("Dingi");

		System.out.println(p1.name+" "+p2.name+" "+p3.name);

	}
}
```

o/p:

```
Dinga Tom Dingi
```

6.

```
package com;

class Employee {

	int id;
	String name;

	Employee(int id, String name) {
		this.id = id;
		this.name = name;
	}

	public static void main(String[] args) {

		Employee e1 = new Employee(101, "Tom");
		Employee e2 = new Employee(102, "Ali");
		Employee e3 = new Employee(103, "Ram");

		System.out.println("Name: "+e1.name+" Id: "+e1.id);
		System.out.println("Name: "+e2.name+" Id: "+e2.id);
		System.out.println("Name: "+e3.name+" Id: "+e3.id);

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

		System.out.println("Employee Id of "+e1.name+" is 
"+e1.id);
		System.out.println("Employee Id of "+e2.name+" is 
"+e2.id);
```

```
            System.out.println("Employee Id of "+e3.name+" is
"+e3.id);
      }
}
```

```
o/p:
Name: Tom Id: 101
Name: Ali Id: 102
Name: Ram Id: 103
-------------------
Employee Id of Tom is 101
Employee Id of Ali is 102
Employee Id of Ram is 103
```

INHERITANCE

1. Inheritance is a process of one class acquiring the properties of another class.

2. A class which gives or shares the properties are called as Super Class, Base Class or Parent Class.

3. A class which acquires or accepts the properties are called as Sub Class, Derived Class or Child Class.

4. In java, we achieve inheritance with the help of 'extends' keyword.

5. Inheritance is also referred as "IS-A" Relationship.

6. In java, Only Variables and methods are inherited whereas blocks and constructors are not INHERITED.

Types of Inheritance

1. Single Level Inheritance

2. Multi-Level Inheritance

3. Hierarchical Inheritance

4. Multiple Inheritance

5. Hybrid Inheritance

1a.

```
package singlelevel;


public class Father

{

	String name = "Dinga";

}
```

1b.

```
package singlelevel;

public class Son extends Father
{
	int age = 20;
}
```

1c.

```
package singlelevel;

public class Test {

	public static void main(String[] args) {

		Son s = new Son();

		System.out.println(s.name+" "+s.age);

	}

}
```

output:

```
Dinga 20
```

2a.

```
package multilevel;

class University {
	String universityName = "VTU";

	void conductExams() {
		System.out.println("VTU is Conducting Examination");
	}

}
```

2b.

```
package multilevel;

class College extends University {

	String collegeName = "Jspiders";

	void providePlacements() {
		System.out.println("Jspiders will provide Interview
Opportunities");
	}

}
```

2c.

```
package multilevel;

class Department extends College {

	String departmentName = "Computer Science";

	void conductFest() {
```

```
            System.out.println("CS Department planned an intra
college fest");
      }


}
```

2d.

```
package multilevel;

public class Student {

      public static void main(String[] args) {

            Department d = new Department();

            System.out.println("University Name: "+d.universityName);
            System.out.println("College Name: "+d.collegeName);
            System.out.println("Department Name: "+d.departmentName);

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

            d.conductExams();
            d.providePlacements();
            d.conductFest();

      }
}
```

o/p:

```
University Name: VTU
College Name: Jspiders
Department Name: Computer Science
------------
VTU is Conducting Examination
Jspiders will provide Interview Opportunities
CS Department planned an intra college fest
```

3a.

```java
package hierarchical;

class Vehicle {
	String brand = "Audi";
}
```

3b.

```java
package hierarchical;

class Car extends Vehicle {

	String fuel = "Petrol";

	void start() {
		System.out.println("Car Started");
	}
}
```

3c.

```java
package hierarchical;
```

```
class Bike extends Vehicle {

      int cost = 1000;

      void stop() {
            System.out.println("Bike Stopped");
      }
}
```

3d.

```
package hierarchical;

public class Solution {
      public static void main(String[] args) {

            Car c = new Car();
            System.out.println(c.brand+" "+c.fuel);
            c.start();

            Bike b = new Bike();
            System.out.println(b.brand+" "+b.cost);
            b.stop();
      }
}
```

o/p:

```
Audi Petrol
Car Started
Audi 1000 Bike Stopped
```

Constructor Overloading

1. The Process of having multiple constructors in the same class but difference in arguments.

In order to achieve Constructor Overloading we have to either follow 1 of the following rules.

a. There should be a change in the (length)No of Arguments.

b. There should be a change in the Datatype of the Arguments.

c. There should be a change in the Sequence/Order of Datatype.

1a.

```
package hierarchical;

class Employee {

	Employee() {
		System.out.println("Hai");
	}

	Employee(int age) {
		System.out.println("Age: "+age);
	}

	Employee(String name) {
		System.out.println("Name: "+name);
	}

	Employee(String name, int id) {
		System.out.println("Name: "+name+" Id:"+id);
	}
	Employee(int id, String name) {
		System.out.println("Id:"+id + " Name: "+name);
	}
}
```

1b.

```
package hierarchical;

public class Test {

	public static void main(String[] args) {

		Employee e1 = new Employee(25);
		Employee e2 = new Employee("tom", 25);
		Employee e3 = new Employee();
		Employee e4 = new Employee("jerry");
		Employee e5 = new Employee(78, "smith");

	}
}
```

o/p:

```
Age: 25
Name: tom Id:25
Hai
Name: jerryId:78 Name: smith
```

CONSTRUCTOR CHAINING

1. The Process of one constructor calling another constructor is called as constructor chaining.

2. Constructor Chaining can be achieved only in case of constructor overloading.

3. Constructor Chaining can happen in two ways:

a. Constructor Chaining in same class can be achieved using this calling statement ie. this().

b. Constructor Chaining in another class can be achieved using super calling statement ie. super() & Is-a relationship.

Note:

1. this() or super() should always be the first executable line within the constructor.

2. Recursive Chaining is not possible, Therefore if there are 'n' constructors we can have a maximum of 'n-1' caling statements.

1.

```
package com;

class Demo
{
	Demo()
	{
		this(999);
		System.out.println(1);
	}

	Demo(int a)
	{
		System.out.println(2);
	}

	public static void main(String[] args)
	{
		Demo d = new Demo();
	}
}
```

```
o/p:
2
1
```

2.

```
package com;

class Student {

	Student(int age) // age=20
	{
		this(5.5);
		System.out.println("Age: "+age);
	}

	Student(double height) // height=5.5
	{
        System.out.println("Height: "+height);
    }

    Student(String name) //name="tom"
    {
        this(20);
        System.out.println("Name: "+name);
    }

    public static void main(String[] args)
    {
        Student s = new Student("tom");
    }
}
```

```
o/p:
Height: 5.5
Age: 20
Name: tom
```

3.

```
package com;

class Employee
{
    Employee(int id) // id=101
    {
        System.out.println(id);
    }
```

```
        Employee(int id, String name) // id=101  name="jerry"
        {
                this(id); // this(101);
                System.out.println(name);
        }

        public static void main(String[] args)
        {
                Employee e = new Employee(101, "Jerry");
        }
}
```

```
o/p:
101
Jerry
```

4.

```
package com;

class Car
{
        Car()
        {
                this(20);
                System.out.println("hai");
        }

        Car(int x) // x=20
        {
                System.out.println("bye");
```

```
    }

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

        Car c = new Car();

        System.out.println("end");
    }
}
```

```
o/p:
start
bye
hai
end
```

1. super() is used to invoke or call constructor of another class.
    a. IS-A (Inheritance) --> extends
    b. super() ie. super calling statement.

super() can be used in 2 ways:

i. implicitly

When we create an object of a class, and if that class has a super class, and if that super class has a non-

parameterized constructor, then the sub class constructor will invoke the super class constructor implicitly.

1a.

```
package com;

class Father
{
	Father()
	{
		System.out.println(1);
	}
}
```

1b.

```
package com;

class Son extends Father
{
	Son()
	{
		// implicitly super();
		System.out.println(2);
	}
}
```

1c.

```
package com;
```

```
public class Test {

	public static void main(String[] args) {

		Son s = new Son();

	}
}
```

```
o/p:
1
2
```

ii. explicitly

When we create an object of a class, and if that class has a super class, and if that super class has a parameterized constructor, then the sub class constructor should invoke the super class constructor explicitly, otherwise we get compile time error.

1a.

```
package com;

class Father
{
	Father(int a)
	{
		System.out.println(1);
	}
}
```

1b.

```
package com;

class Son extends Father
{
	Son()
	{
		super(10);
		System.out.println(2);
	}
}
```

1c.

```
package com;

public class Test {

	public static void main(String[] args) {

		Son s = new Son();

	}
}
```

o/p:

```
1
2
```

2a.

```
package com;

class Vehicle
{
	Vehicle(String brand)
	{
		System.out.println("brand: "+brand);
	}
}
```

2b.

```
package com;

class Bike extends Vehicle
{
	Bike(int cost)
	{
		super("BMW");
		System.out.println("cost: "+cost);
	}
}
```

2c.

```
package com;

public class Solution {

	public static void main(String[] args) {

		Bike b = new Bike(200);
```

```
    }
}
```

```
o/p:
brand: BMW
cost: 200
```

super keyword

1. super is a keyword which is used to access the super class properties.

syntax: super.variableName or super.methodName()

Note:

1. this -> points to current object

2. super -> points to super class object

1a.

```
package com;

class Employee
{
	int id = 30;
}
```

1b.

```
package com;

class Developer extends Employee
{
	int id = 20;

	void display()
	{
```

```
            int id = 10;


            System.out.println(id); // points to local variable -> 10
            System.out.println(this.id); //  points to current
object's variable -> 20
            System.out.println(super.id); //  points to super class
variable -> 30
      }


      public static void main(String[] args)
      {
            Developer d = new Developer();


            d.display();
      }
}
```

```
o/p:
10
20
30
```

Method Overriding

1. The process of Inheriting the method and changing the implementation/Definition of the inherited method is called as method overriding.

2. In order to achieve method overriding, we have to follow the below rules:

i. Method Name must be same.

ii. Arguments should be same

iii. returntype should also be same.

Note:

1. Access Specifier should be same or of Higher Visibility.

2. While Overriding a method we can optionally use annotation ie. @Override

3. annotation was introduced from JDK 1.5

1a.

```
package com;

class Father {

	void bike() {
		System.out.println("Old Fashioned Father's Bike");
	}
}
```

1b.

```
package com;

class Son extends Father {

	@Override
	void bike() {
		System.out.println("New Modified Son's Bike");
	}

	public static void main(String[] args) {
		Son s = new Son();
		s.bike();
	}

}
```

o/p:

```
New Modified Son's Bike
```

2a.

```
package com;

public class Vehicle {

	void start() {
        System.out.println("Vehicle Started");
    }

}
```

2b.

```
package com;

class Bike extends Vehicle {

    @Override
    void start() {
        super.start();
        System.out.println("Bike Started");
        super.start();
    }

}
```

2c.

```
package com;

public class Test {
    public static void main(String[] args) {

        Bike b = new Bike();
        b.start();

    }
}
```

```
o/p:
Vehicle Started
Bike Started
Vehicle Started
```

3a.

```
package com;

public class WhatsApp1 {

	void display() {
		System.out.println("Single Ticks Supported");
	}

}
```

3b.

```
package com;

public class WhatsApp2 extends WhatsApp1 {

	@Override
	void display() {
		super.display();
		System.out.println("Double Ticks Supported");
	}
	void call() {
		System.out.println("Voice Call Supported");
	}

}
```

3c.

```
package com;

public class WhatsApp3 extends WhatsApp2 {

	@Override
	void display() {
		super.display();
		System.out.println("Blue Ticks Supported");
	}

	@Override
	void call() {
		super.call();
		System.out.println("Video Call Supported");
	}

	void story() {
		System.out.println("Can upload images as Story");
	}

}
```

3d.

```
package com;

public class User {

	public static void main(String[] args) {

		WhatsApp3 w3 = new WhatsApp3();

		w3.display();

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

		w3.call();

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

		w3.story();

	}

}
```

o/p:

```
Single Ticks Supported
Double Ticks Supported
Blue Ticks Supported
-------------
Voice Call Supported
Video Call Supported
-------------
Can upload images as Story
```

Packages

1. Packages are nothing but Folder or Directory.

2. Packages are used to store classes and interfaces.

3. Searching becomes easy.

4. Better Maintenance of the Programs.

example:

```
package com.google.gmail;


class Inbox {


}
```

Access Modifiers or Access Specifiers

1. Access Modifiers or Access Specifiers is used to specify the accessibility (boundary or range) of a specific member (class, variable, method and Constructor).

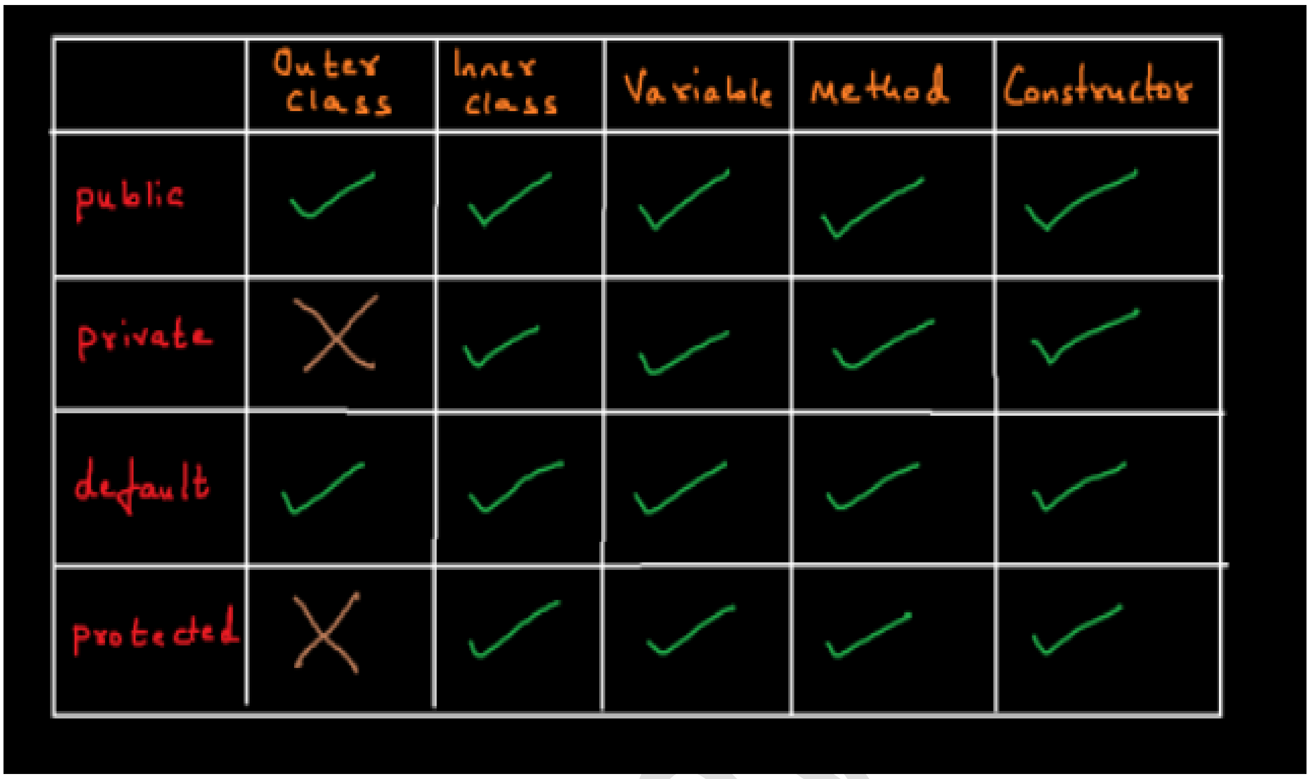
2. The different Access Modifiers are as follows:

   i. public

   ii. private

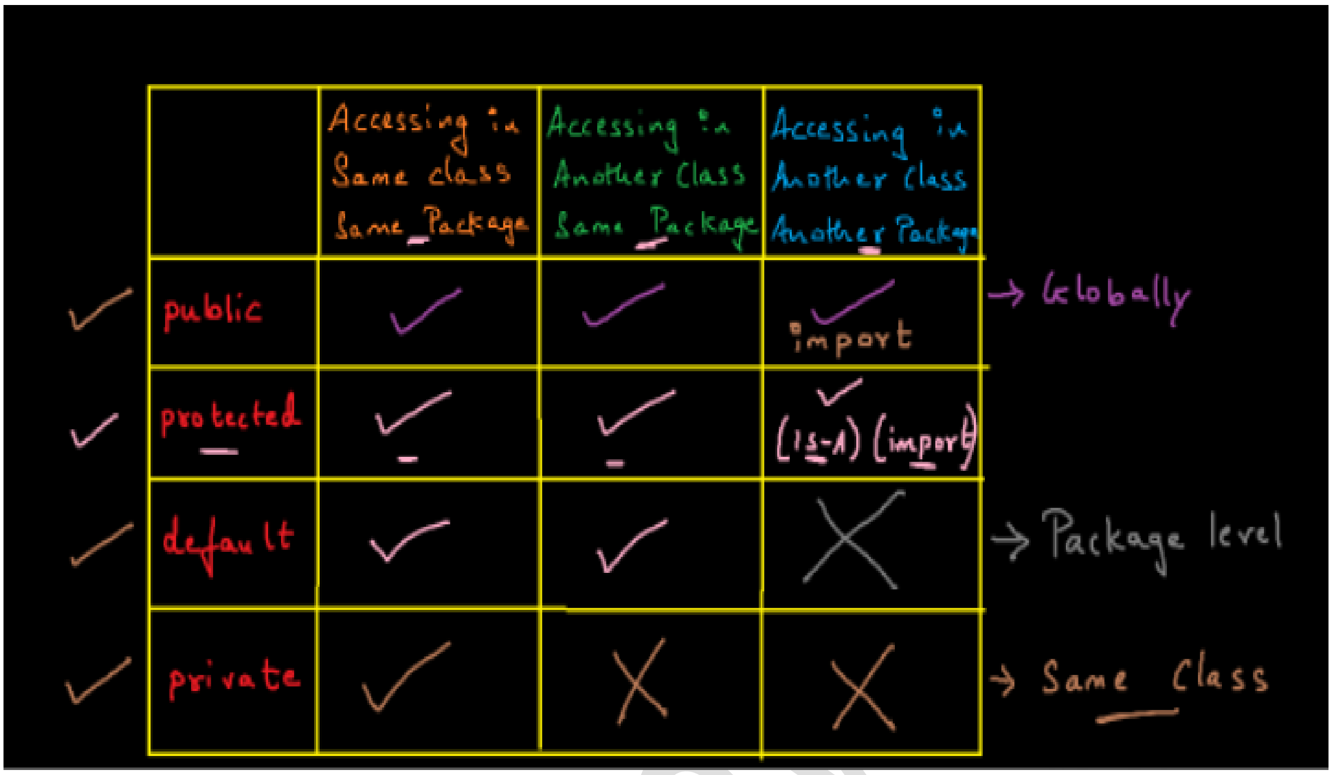
   iii. protected

   iv. default

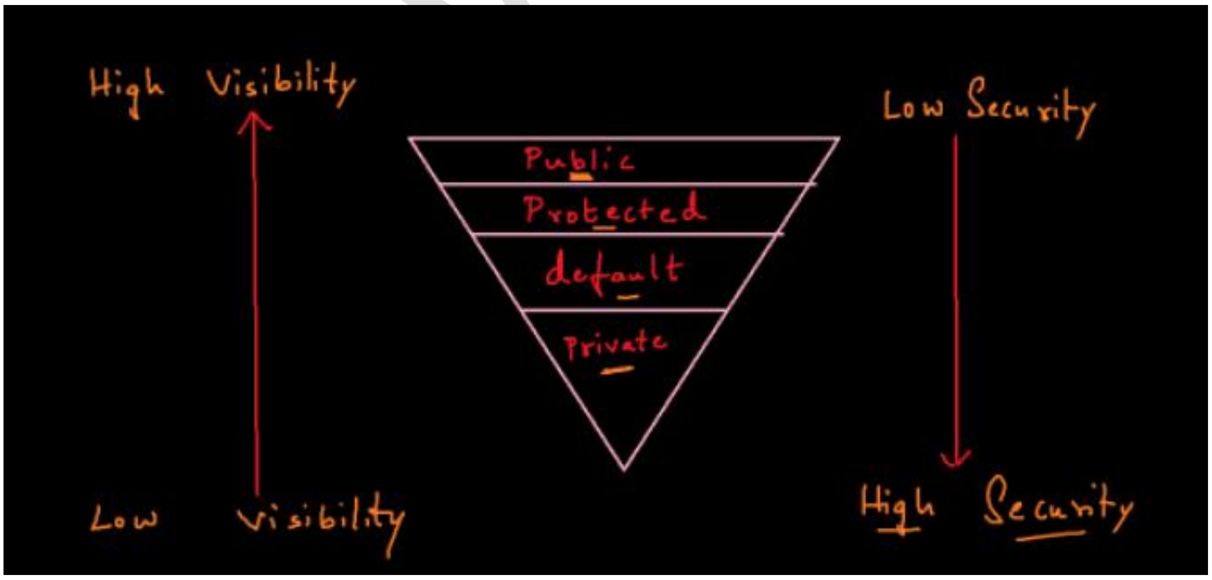


	Outer Class	Inner Class	Variable	Method	Constructor
public	✓	✓	✓	✓	✓
private	✗	✓	✓	✓	✓
default	✓	✓	✓	✓	✓
protected	✗	✓	✓	✓	✓

1.



		Accessing in Same class Same Package	Accessing in Another Class Same Package	Accessing in Another Class Another Package	
✓	public	✓	✓	✓ import	→ Globally
✓	protected	✓	✓	✓ (IS-A) (import)	
✓	default	✓	✓	✗	→ Package level
✓	private	✓	✗	✗	→ Same Class



1.

```
package com;

/* Accessing Public Members In Same Class */

public class Person {

	public int age = 20;

	public Person() {
		System.out.println("In Person Class Constructor");
	}

	public void eat() {
		System.out.println("Person is Eating");
	}

	public static void main(String[] args) {

		Person p = new Person();
		System.out.println(p.age);
		p.eat();

	}
}
```

o/p:

```
In Person Class Constructor
20
Person is Eating
```

2a.

```
package com;

/* Accessing Public Members In Another Class Same Package */

public class Person {

	public int age = 20;

	public Person() {
		System.out.println("In Person Class Constructor");
	}

	public void eat() {
		System.out.println("Person is Eating");
	}

}
```

2b.

```
package com;

public class Test {

	public static void main(String[] args) {

		Person p = new Person();
		System.out.println(p.age);
```

```
          p.eat();

     }

}
```

```
o/p:
In Person Class Constructor
20
Person is Eating
```

3a.

```
package com;

/* Accessing Public Members In Different Class Different Package */

public class Person {

     public int age = 20;

     public Person() {
          System.out.println("In Person Class Constructor");
     }

     public void eat() {
          System.out.println("Person is Eating");
     }
}
```

3b.

```
package org;

import com.Person;

public class Solution {

	public static void main(String[] args) {

		Person p = new Person();
		System.out.println(p.age);
		p.eat();

	}

}
```

o/p:

```
In Person Class Constructor
20
Person is Eating
```

4.

```
package com;
```

```
/* Accessing Private Members In Same Class Same Package */

public class Mobile {

	private int cost = 2000;

	private void chat() {
		System.out.println("Chatting");
	}

	public static void main(String[] args) {

		Mobile m = new  Mobile();
		System.out.println(m.cost);
		m.chat();

	}

}
```

```
o/p:
2000
Chatting
```

5a.

```
package com;

/* Accessing Private Members In Another Class Same Package */
```

```
public class Mobile {

	private int cost = 2000;

	private void chat() {
		System.out.println("Chatting");
	}
}
```

5b. ERROR

```
package com;

public class Test {

	public static void main(String[] args) {

		Mobile m = new  Mobile();
		// System.out.println(m.cost);
		// m.chat();

	}
}
```

6.

```
package com;

/* Accessing default Members In Same Class */
```

```
class Television {

      String brand = "Sony";

      Television() {
            System.out.println("In Constructor");
      }

      void watchMovie() {
            System.out.println("Watching Movie");
      }

      public static void main(String[] args) {

            Television t = new Television();
            System.out.println(t.brand);
            t.watchMovie();

      }

}
```

```
o/p:
In Constructor
Sony
Watching Movie
```

```
7a.
package com;

/* Accessing default Members In Different Class Same Package */

class Television {

      String brand = "Sony";

      Television() {
            System.out.println("In Constructor");
      }

      void watchMovie() {
            System.out.println("Watching Movie");
      }

}

7b.
package com;

class User {

      public static void main(String[] args) {

            Television t = new Television();
            System.out.println(t.brand);
            t.watchMovie();
      }

}
```

```
o/p:
In Constructor
Sony
Watching Movie
```

8a.

```
package com;

/* Accessing default Members In Different Class Different Package */

class Television {

      String brand = "Sony";

      Television() {
            System.out.println("In Constructor");
      }

      void watchMovie() {
            System.out.println("Watching Movie");
      }

}
```

8b. ERROR

```
package org;

// import com.Television;

public class Solution {

	public static void main(String[] args) {

		// Television	t = new Television();

	}

}
```

9.

```
package com;

/* Accessing Protected Members In Same Class */

public class Television {

	protected String brand = "LG";

	protected Television() {
		System.out.println("In Television Constructor");
	}

	protected void watchMovie() {
		System.out.println("Watching Movie");
	}

	public static void main(String[] args) {

		Television t = new Television();
		System.out.println(t.brand);
		t.watchMovie();

	}
}
```

```
o/p:
In Television Constructor
LG
Watching Movie
```

9a.

```
package com;

/* Accessing Protected Members In Different Class Same Package*/

public class Television {

	protected String brand = "LG";

	protected Television() {
		System.out.println("In Television Constructor");
	}

	protected void watchMovie() {
```

```
            System.out.println("Watching Movie");
      }
}
```

9b.

```
package com;

class User {

      public static void main(String[] args) {

            Television t = new Television();
            System.out.println(t.brand);
            t.watchMovie();

      }

}
```

o/p:

```
In Television Constructor
LG
Watching Movie
```

10a.

```
package com;
```

```
/* Accessing Protected Members in Different class Different Package
*/

public class Father {

	protected int age = 50;

}

10b.
package org;

import com.Father;

public class Son extends Father {

	public static void main(String[] args) {

		Son s = new Son();
		System.out.println(s.age);

	}

}

o/p:
50
```

Encapsulation

1. Encapsulation is a Process of Wrapping/Binding/Grouping Data Members with its related Member Functions in a Single Entity called as Class.

2. The process of grouping and protecting data members and member functions in a single entity called as class.

3. The Best Example for Encapsulation is JAVA BEAN CLASS.

Specifications of JAVA BEAN CLASS (1, 3, 4)

1. Class should be public non abstract class.

2. Class should have public non parameterized constructor.

3. Variables or Data Members should be declared as private.

4. Those Data Members should have respective public getter method and public setter method.

5. Class should implement a marker interface called as Serializable.

Advantages of Encapsulation

1. Validation can be done. (Protecting -> data security)

2. Flexible ie. Data can be made either write only or read only.

1a.

```
package javabean;

public class Person
{
	private int age;
	public void setAge(int age)
	{
		this.age = age;
	}

	public int getAge()
	{
		return age;
	}
}
```

1b.

```
package javabean;

public class TestPerson {

	public static void main(String[] args) {

		Person p = new Person();

		p.setAge(25);

		int age = p.getAge();
		System.out.println("Age: "+age);

		System.out.println(p.getAge());

	}
}
/*System.out.println(p.age);
p.age = 25; */
```

o/p:

```
Age: 25
25
```

2a.

```
package javabean;

public class Employee {
	private int id;
	private String name;

	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

}
```

2b.

```
package javabean;

public class TestEmployee {

	public static void main(String[] args) {

		Employee e = new Employee();

		e.setId(101);
		e.setName("Tom");

		System.out.println("Id: "+e.getId());
		System.out.println("Name: "+e.getName());

	}

}
```

o/p:

```
Id: 101
Name: Tom
```

3a.

```
package javabean;

public class Student {

	private int age;

	public void setAge(int age) {

		if(age>0) {
			System.out.println("Initializing age");
			this.age = age;
		}
		else {
			System.out.println("Cannot initialize");
		}

	}

	public int getAge() {
		return age;
	}

}
```

3b.

```
package javabean;

public class TestStudent {

	public static void main(String[] args) {

		Student s = new Student();

		s.setAge(25);

		System.out.println(s.getAge());
	}
}
```

o/p:

```
25
```

final

- **final is a keyword which we can use it with a variable, method and class.**
- **final variables cannot be re-initialized, it acts as a constant.**
- **final methods cannot be overridden, but can be inherited.**
- **final class cannot be inherited.**

1.

```
package javabean;

public class Demo {
	public static void main(String[] args) {

		final int a = 10;
		System.out.println(a);

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

		//a=30;
		System.out.println(a);

		final double PI = 3.14;
	}
}
```

2a.

```
package javabean;
```

```
public class A {

	final void m1() {

	}

}
```

2b.

```
package javabean;

public class B extends A {

	@Override
	void m1() {

	}
}
```

3a.

```
package javabean;

public final class A {



}
```

3b.

```
package javabean;

public class B extends A {

	}
```

Array

1. Array is a container to store a group of Data/Elements.

2. Array is of Fixed Size.

3. Array can store only Homogeneous Data.

4. Array is of Indexed Based and index position starts from 0.

1a.

```
package org;

public class Demo {

	public static void main(String[] args) {

		/* Array Declaration */
		int[] a;

		/* Array Creation */
		a = new int[3];

		/* Printing Array Elements */
		System.out.println(a[0]);
		System.out.println(a[1]);
		System.out.println(a[2]);

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

		/* Array Initialization */
		a[1] = 35;
```

```
        a[2] = 12;

        System.out.println(a[0]);
        System.out.println(a[1]);
        System.out.println(a[2]);

        System.out.println("Length: "+a.length);

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

        /* Array Declaration & Creation */
        String[] names = new String[3];

        System.out.println(names[0]);
        System.out.println(names[1]);
        System.out.println(names[2]);

        names[0] = "dinga";
        names[1] = "guldu";
        names[2] = "dingi";

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

        System.out.println(names[0]);
        System.out.println(names[1]);
        System.out.println(names[2]);

        System.out.println("Length: "+names.length);
    }
```

```
}

o/p:
0
0
0
=======
0
35
12
Length: 3
*****************
null
null
null
=======
dinga
guldu
dingi
Length: 3
```

2.

```
package org;

public class Test {
	public static void main(String[] args) {

		/* Array Initialization Directly */
		int[] a = {10, 20, 30};
```

```
          for(int i=0; i<a.length; i++)
          {
               System.out.println(a[i]);
          }

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

          for(int i=a.length-1; i>=0; i--)
          {
               System.out.println(a[i]);
          }

     }
}
```

```
o/p:
10
20
30
-----------
30
20
10
```

3.

```
package org;

public class SumOfArrayElements {
	public static void main(String[] args) {

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

		int sum = 0;

		for(int i=0; i<a.length; i++)
		{
			sum += a[i];  // sum = sum + a[i];
		}

		System.out.println("Sum: "+sum);
		System.out.println("Average: "+(sum/a.length));

	}

}
```

```
o/p:
Sum: 60
Average: 20
```

4a.

```
package org;

public class Person
{
	int id;
	String name;
```

```
     Person(int id, String name)
     {
          this.id = id;
          this.name = name;
     }
}
```

4b.

```
package org;

public class Employee extends Person
{
     double salary;

     Employee(int id, String name, double salary)
     {
          super(id, name);
          this.salary = salary;
     }
}
```

4c.

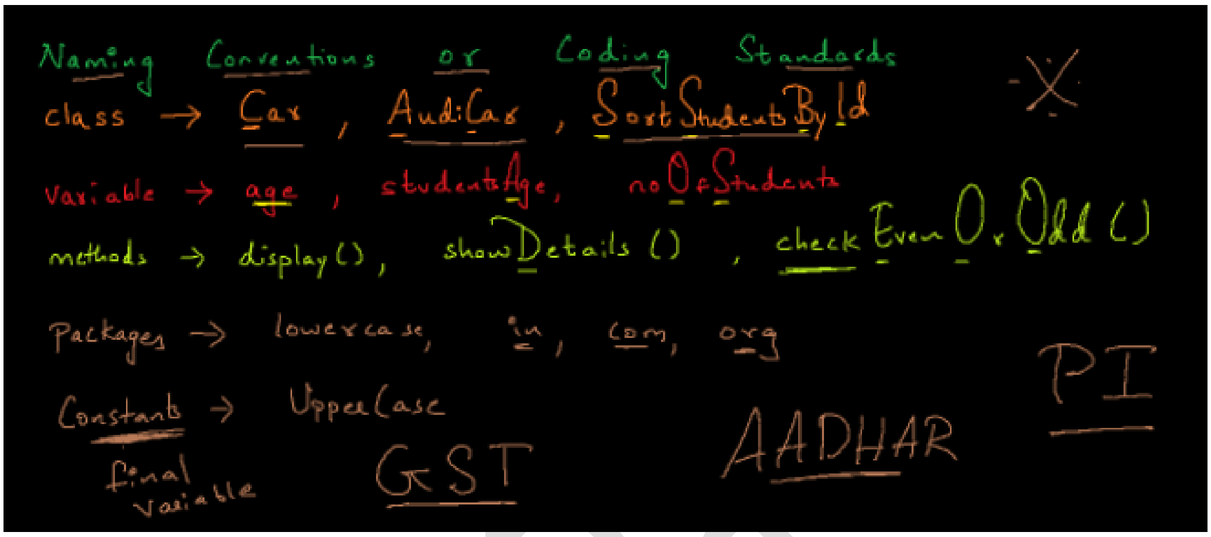
```
package org;

public class Student extends Person
{
     int marks;

     Student(int id, String name, int marks) // 101, Tom, 34
     {
```

```
		super(id, name); // super(101, "Tom");
		this.marks = marks;
	}
}

4d.
package org;

public class MainClass {

	public static void main(String[] args) {

		Student s = new Student(101, "Tom", 34);
		Employee e = new Employee(222, "Jerry", 345.67);

		System.out.println(s.id+" "+s.name+" "+s.marks);
		System.out.println(e.id+" "+e.name+" "+e.salary);

	}
}
o/p
101 Tom 34
222 Jerry 345.67

// Assignment
// 1. Similar
// 2. array -> No of occurrences
// a-> 10, 20, 30, 20, 10, 40
```

Advantages of Inheritance

1. Repetition/Redundancy is avoided.
2. Reusability of code is increased.

Naming Conventions



Naming Conventions or Coding Standards ×

class → Car , AudiCar , SortStudentsById

variable → age , studentAge, noOfStudents

methods → display(), showDetails() , checkEvenOrOdd()

Packages → lowercase, in, com, org

Constants → UpperCase
final variable

PI

GST

AADHAR

abstract

1. abstract is a keyword which can be used with class and method.

2. A class which is not declared using abstract keyword is called as Concrete class.

3. Concrete class can allow only concrete methods.

4. A class which is declared using abstract keyword is called as Abstract class.

5. Abstract class can allow both abstract and concrete methods.

6. Concrete method has both declaration and implementation/definition.

7. Abstract method has only declaration but no implementation.

8. All Abstract methods should be declared using abstract keyword.

Contract of Abstract or What should we do when a class extends abstract class:

1. When a class Inherits an abstract class, override all the abstract methods.

2. When a class Inherits an abstract class and if we do not want to override the inherited abstract method, then make the sub class as abstract class.

----

1. Can abstract class have constructors?

Yes. But we cannot invoke indirectly, it has to be invoked by the sub class constructor either implicitly or explicitly using super().

NOTE:

1. Can a class inherit an abstract class? -> YES
2. We cannot create an object of abstract class.
3. Abstract methods cannot be private.
4. Abstract methods cannot be static.
5. Abstract methods cannot be final.

1a.

```
package org;

public abstract class Father
{
    Father()
    {
        System.out.println(1);
    }
}
```

1b.

```
package org;
```

```
public class Son extends Father
{
     Son()
     {
          // implicitly super();
          System.out.println(2);
     }
}
```

1c.

```
package org;

public class Test {

     public static void main(String[] args) {

          Son s = new Son();

     }

}
```

o/p:

```
1
2
```

2a.

```
package org;

public abstract class Father
{
    Father(int a)
    {
        System.out.println(1);
    }
}
```

2b.

```
package org;

public class Son extends Father
{
    Son()
    {
        super(10);
        System.out.println(2);
    }
}
```

2c.

```
package org;

public class Test {
    public static void main(String[] args) {

        Son s = new Son();

    }

}
```

```
o/p:
1
2
```

TYPE CASTING

1. Type Casting is a process of converting/storing one type of value/data into another type of value/data.

2. They are classified into 2 types:

    i. Primitive Type Casting

    ii. Non-Primitive Type Casting (Derived Casting)

3. Primitive Type Casting is further divided into 2 types:

i. Widening:

- Converting Smaller type of data into Bigger type of data.

- Widening happens Implicitly/Automatically.

ii. Narrowing:

- Converting Bigger type of data into Smaller type of data.

- Narrowing happens Explicitly.

3.

```
package primitivecasting;

public class Demo {

	public static void main(String[] args) {

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

		int a = 10;
		double b = a; // double b = 10;
		System.out.println(a+" "+b); // 10 10.0

		char c = 'A';
		int d = c;
		System.out.println(c+" "+d); // A 65

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

		double x = 3.56;
		int y = (int) x;
		System.out.println(x+" "+y); // 3.56 3
```

```
        int i = 66;

        char j = (char)i;

        System.out.println(i+" "+j); // 65 B


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


        char z = 97;

        System.out.println(z);


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


        System.out.println((char)98); // b


        System.out.println( (int) 78.54); // 78


        System.out.println("A"+"B"); // AB


        System.out.println("A"+20); //A20


        System.out.println('A'+'B'); // 65+66 -> 131


        System.out.println('a'+20); // 97+20 -> 117


    }


}
```

```
// ASCII Value -> American Standard Code for Information
Interchange

// A -> 65 -> 0101010

// a -> 97 -> 1010101


o/p:
---Widening---
10 10.0
A 65
---Narrowing---
3.56 3
66 B
-----------------------
a
-----------------------
b
78
AB
A20
131
117
```

4.

```
package primitivecasting;

public class Test {
    public static void main(String[] args) {

        System.out.println(10); // int
        System.out.println(10.45); // double

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

        float x = (float) 5.6;

        double a = 5.6;
        float b = (float) a;

        float i = 1.5f;
        float j = 1.5F;

        System.out.println(a+" "+b+" "+i+" "+j+" "+x);


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

        long contactNo = 9999988771;
        long contactNum = 9999988877L;
        System.out.println(contactNo+" "+contactNum);

    }
}
```

```
o/p:
10
10.45
-------------
5.6 5.6 1.5 1.5 5.6
---------------
9999988877 9999988877
```

```
5.
package primitivecasting;

import java.util.Scanner;

public class Solution {

     public static void main(String[] args) {

          Scanner scan = new Scanner(System.in);

          System.out.println("Enter the No of Elements to
be inserted:");
          int size = scan.nextInt(); // 3

          int[] a = new int[size];

          System.out.println("Enter "+a.length+"
Elements:");
```

```
        for(int i=0; i<a.length; i++)
        {
            a[i] = scan.nextInt();
        }

        System.out.println("Array Elements Are:");

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

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

        int sum = 0;

        for(int i=0; i<a.length; i++)
        {
            sum+=a[i];
        }

        System.out.println("Sum: "+sum);
        System.out.println("Average: "+ (sum/a.length));

    }
```

```
}
```

o/p:

```
Enter the No of Elements to be inserted:
3
Enter 3 Elements:
10
50
30
Array Elements Are:
10
50
30
-----------
Sum: 90
Average: 30
```

Non-Primitive Casting or Derived Casting

1. Non-Primitive Casting or Derived Casting or Class Type Casting can be divided into 2 types:

   i. Up-Casting

   ii. Down-Casting

Upcasting

1. Creating an object of sub class, and storing it's address into a reference of type Superclass.

2. With Upcasted Reference we can access only superclass Members/Properties.

3. In order to achieve upcasting, IS-A Relationship mandatory.

4. Upcasting will have implicitly/Automatically.

5. Superclass reference, subclass object.

Down-Casting

1. The process of converting the upcasted reference back to Subclass type reference is called as Down-casting.

2. With the Subclass/Down-casted reference we can access both superclass and subclass members properties.

3. In order to achieve down-casting, upcasting is mandatory.

4. Down-casting has to be done explicitly.

syntax: (SubClassName) SuperClassReference;

1a.

```
package nonprimitivecasting;
```

```
public class Father
{
     int age = 40;
}
```

1b.

```
package nonprimitivecasting;

public class Son extends Father
{
     String name = "Dinga";
}
```

1c.

```
package nonprimitivecasting;

public class Test {

     public static void main(String[] args) {

          /* UPCASTING
          Son s1 = new Son();
          Father f = s1; */

          Father f = new Son();
          System.out.println(f.age); // f.name -> Error
```

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

                /* DOWNCASTING */
                Son s = (Son) f;
                System.out.println(s.age+" "+s.name);
        }

}
```

```
o/p:
40
--------
40 Dinga
```

2a.

```
package nonprimitivecasting;

public class Vehicle {

        String brand ="BMW";

        void start() {
                System.out.println("Vehicle Started");
        }

}
```

2b.

```
package nonprimitivecasting;

public class Car extends Vehicle {

    int cost = 70000;

    void shiftGears() {
        System.out.println("Shifting Gears");
    }

}
```

2c.

```
package nonprimitivecasting;

public class Person {

    public static void main(String[] args) {

        Vehicle v = new Car();

        System.out.println(v.brand); // v.cost will give
error
        v.start(); // v.shiftGears(); will give error
```

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

          Car c = (Car) v;

          System.out.println(c.brand+" "+c.cost);
          c.start();
          c.shiftGears();

     }

}
```

```
o/p:
BMW
Vehicle Started
-------
BMW 70000
Vehicle Started
Shifting Gears
```

ClassCastException

1. If an object has been upcasted we have to downcast to same type else we get ClassCastException.

2. In other words, if one type of reference is upcasted and downcasted to some other type of reference we get ClassCastException.

3. If we Downcast, without upcasting even then we get ClassCastException.

4. In order to avoid ClassCastException we make use of instanceof operator.

instanceof

1. instanceof is an operator in order to check if an object is an instance of a specific class type or not.

2. In other words, instanceof is an operator in order to check if an object is having the properties of a specific class type or not.

3. instanceof will return boolean value.

syntax: object instanceof ClassName

1a.

```
package org;

public class Father
{
	int x = 30;
}
```

1b.

```
package org;

public class Son extends Father
{
	int y = 20;
}
```

1c.

```
package org;

public class Daughter extends Father
{
	int z = 10;
}
```

1d.

```
package org;

public class Test {

	public static void main(String[] args) {

		// Father obj = new Daughter();

		Father obj;
		obj = new Son();

		if(obj instanceof Son)
		{
			System.out.println("Downcasting to Son");
			Son s = (Son) obj;
			System.out.println(s.x+" "+s.y);
		}
		else if(obj instanceof Daughter)
```

```
         {
              System.out.println("Downcasting to
Daughter");
              Daughter d = (Daughter) obj;
              System.out.println(d.x+" "+d.z);
         }

     }
}

/*
 Father f = new Son();
 Daughter s = (Daughter) f;

 Exception in thread "main" java.lang.ClassCastException:
org.Son cannot be cast to org.Daughter
     at org.Test.main(Test.java:9)
*/

o/p:
Downcasting to Son
30 20

1e.
package org;

public class Solution {
     public static void main(String[] args) {

         Father f = new Father();
         Son s = new Son();
         Daughter d = new Daughter();

         System.out.println(s instanceof Son); // true
         System.out.println(s instanceof Father); // true

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

         System.out.println(d instanceof Daughter); //
true
         System.out.println(d instanceof Father); // true

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

         System.out.println(f instanceof Father); // true
         System.out.println(f instanceof Son); // false
         System.out.println(f instanceof Daughter); //
false

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

         System.out.println(new Son() instanceof Father);
// true
         System.out.println(new Son() instanceof Son); //
true
```

```
    }
}
```

```
o/p:
true
true
--------
true
true
--------
true
false
false
--------
true
true
```

2a

```
package org;

public class Vehicle
{
    String brand = "Suzuki";
}
```

2b.

```
package org;

public class Car extends Vehicle
{
	String fuel = "Diesel";
}
```

2c.

```
package org;

public class Bike extends Vehicle
{
	String color = "Black";
}
```

2d.

```
package org;

public class Example {

	public static void main(String[] args) {

		Vehicle v = new Car();

		if(v instanceof Bike)
		{
```

```
              Bike b = (Bike) v;
              System.out.println("Brand: "+b.brand+"
Color: "+b.color);
         }
         else if(v instanceof Car)
         {
              Car c = (Car) v;
              System.out.println("Brand: "+c.brand+" Fuel:
"+c.fuel);
         }

    }
}
```

```
o/p:
Brand: Suzuki Fuel: Diesel
```

3a.

```
package org;

public class Food {

}
```

3b.

```
package org;
```

```
public class Pizza extends Food {

}

3c.
package org;

public class Burger extends Food {

}

3d.
package org;

public class Sandwich extends Food {

}

3e.
package org;

public class KFC {

    Food order(int choice)
    {
        if(choice == 1)
        {
            Pizza p = new Pizza();
            return p;
        }
        else if(choice == 2)
        {
            Burger b = new Burger();
            return b;
        }
        else
        {
            Sandwich s = new Sandwich();
            return s;
        }
    }

}
```

```
3f.
package org;

public class Customer {

    public static void main(String[] args) {

        KFC k = new KFC();
```

```
          Food obj = k.order(3); // Food obj = new
Sandwich();

          if(obj instanceof Pizza) {
               System.out.println("Eating Pizza");
          }
          else if(obj instanceof Burger) {
               System.out.println("Eating Burger");
          }
          else if(obj instanceof Sandwich) {
               System.out.println("Eating Sandwich");
          }

     }
}
```

o/p:

```
Eating Sandwich
```

4a.

```
package org;

public class Beverage {

}
```

4b.

```
package org;

public class Coffee extends Beverage {

}
```

4c.

```
package org;

public class Tea extends Beverage {

}
```

4d.

```
package org;

public class Company {

    Beverage vendingMachine(int choice)
    {
        if(choice == 1)
        {
            return new Coffee();
```

```
			}
			else if(choice == 2)
			{
				return new Tea();
			}
			else
			{
				return null;
			}
		}
	}

	/*
	Beverage vendingMachine(int choice)
	{
			if(choice == 1)
			{
				Coffee c = new Coffee();
				return c;
			}
			else if(choice == 2)
			{
				Tea t = new Tea();
				return t;
			}
			else
			{
				return null;
			}
}
*/

4e.
package org;

import java.util.Scanner;

public class Employee {

	public static void main(String[] args) {

		Scanner scan = new Scanner(System.in);

		Company c = new Company();

		System.out.println("Enter Choice:");
		System.out.println("1:Coffee\n2:Tea");
		int choice = scan.nextInt();

		Beverage obj = c.vendingMachine(choice);

		if(obj instanceof Coffee) {
			System.out.println("Drinking Coffee");
		}
```

```
        else if(obj instanceof Tea) {
            System.out.println("Drinking Tea");
        }
        else {
            System.out.println("Invalid Choice");
        }
    }

}
```

```
o/p:
Enter Choice:
1:Coffee
2:Tea
1
Drinking Coffee
```

Method Binding

Associating or Mapping the Method Caller to its Method Implementation or Definition is called Method Binding.

Polymorphism

1. Polymorphism means many forms.

2. The ability of a method to behave differently, when different objects are acting upon it.

3. The ability of a method to exhibit different forms, when different objects are acting upon it.

4. Different types of polymorphism are as follows:

- **i. Compile time polymorphism.**
- **ii. Run time polymorphism.**

****

Compile time Polymorphism

1. Compile time Polymorphism is achieved with the help of Method Overloading.

2. Compile time Polymorphism is also referred as Early Binding or Static Binding.

3. Method Binding is happening as compile time, Hence we call Method Overloading as Early Binding or Static Binding or Compile time Polymorphism.

4. Out of so many overloaded methods, which method implementation should get executed is decided by the compiler during compile time.

1a.

```
package compiletime;
```

```
public class Myntra {

     void purchase(int cost)
     {
          System.out.println("Cost: Rs."+cost);
     }

     void purchase(String brand, String product)
     {
          System.out.println("Brand: "+brand+" Product 
"+product);
     }

     void purchase(String paymentGateway)
     {
          System.out.println("Payment Gateway: 
"+paymentGateway);
     }

     void purchase(String product, int cost)
     {
          System.out.println("Product: "+product+" Cost: 
"+cost);
     }

     void purchase(int cost, String product)
     {
```

```
		System.out.println("Cost: "+cost+" Product: 
"+product);
	}


}
```

1b.

```
package compiletime;

public class Customer {

	public static void main(String[] args) {

		Myntra m = new Myntra();

		m.purchase("GooglePay");
		m.purchase(2500);
		m.purchase("Shoe", 3000);
		m.purchase(15000, "Mobile");
		m.purchase("Adidas", "T-Shirt");

	}
}
```

o/p:

```
Payment Gateway: GooglePay
```

```
Cost: Rs.2500
Product: Shoe Cost: 3000
Cost: 15000 Product: Mobile
Brand: Adidas Product T-Shirt
```

Run time Polymorphism

1. Run time Polymorphism is achieved with the help of
   i. Inheritance (IS-A Relationship)
   ii. Method Overriding
   iii. Upcasting
2. When we call an Overridden method on the superclass reference, the method implementation which gets executed is dependent on the subclass acting upon it.
3. Out of so many Overridden method, which method implementation should get executed is decided by the JVM at runtime.
4. Runtime Polymorphism is also called as Late Binding, Dynamic Binding or Dynamic Method Dispatch.

Note:

If we call an overridden method on the superclass reference, always the overridden method implementation only gets executed.

1a.

```
package runtime;


public class Employee {
	void work() {
		System.out.println("Working");
	}

}
```

1b.

```
package runtime;

public class Developer extends Employee {

	@Override
	void work() {
		System.out.println("Developer is "
				+ "developing an application");
	}

}
```

1c.

```
package runtime;

public class Tester extends Employee {

	@Override
	void work() {
        System.out.println("Tester is "
                + "testing an application");
    }

}
```

1d.

```
package runtime;

public class Test {

    public static void main(String[] args) {

        Employee e = new Developer();
        e.work();

        Employee emp = new Tester();
        emp.work();

    }
}
```

o/p:

```
Developer is developing an application
Tester is testing an application
```

2a.

```
package compiletime;

public class Vehicle
{
     void start()
     {
          System.out.println("Vehicle Started");
     }
}
```

2b.

```
package compiletime;

public class Car extends Vehicle // 1
{
     @Override
     void start() // 2
     {
          System.out.println("Car Started");
     }
}
```

2c.

```
package compiletime;

public class Bike extends Vehicle { // 1
```

```
    @Override
    void start() { // 2
        System.out.println("Bike Started");
    }

}
```

2d.

```
package compiletime;

public class Test {
    public static void main(String[] args) {

        Vehicle v1 = new Car();
        v1.start();

        Vehicle v2 = new Bike();
        v2.start();
    }
}
```

o/p:

```
Car Started
Bike Started
```

2e.

```
package compiletime;
```

```
public class Demo {

     void invokeStart(Vehicle v) // Vehicle obj = new
Car();  ->  Vehicle obj = new Bike();
     {
          v.start();
     }

     public static void main(String[] args)
     {
          Demo d = new Demo();

          d.invokeStart(new Car());
          d.invokeStart(new Bike());
     }

}
```

```
o/p:
Car Started
Bike Started
```

Interface

1. Interface is a Java Type Definition which has to be declared using interface keyword.

2. Interface is a media between 2 systems, wherein 1 system is the client/user and another system is object with resources/services.

syntax:
```
interface InterfaceName
      {


      }
```

3. Interface can have variables, those variables are automatically public, static and final.

4. Interface can allow only abstract methods, and those methods are automatically public and abstract.

5. class can achieve IS-A Relationship with an interface using implements keyword.

6. When a class implements an interface, mandatorily override the abstract method.

7. While Overriding a method, Access Specifier/Modifier should be same or of Higher Visibility.

8. A class can implement any number of Interfaces (Multiple Interfaces).

9. A class can extend 1 class and implement any number of interfaces.

10. Interfaces does not contain Constructors.

11. We cannot create an object of interface.

Programs

1a.

```
package org;

public interface Person {

	int id = 101; // public static final int id = 101;

	void eat(); // public abstract void eat();

}
```

1b.

```
package org;

public class Dinga implements Person {

	@Override
	public void eat() {
		System.out.println("Eating");
	}

	public static void main(String[] args) {

		System.out.println(Person.id);
```

```
          Dinga d = new Dinga();
          d.eat();

     }

}
```

```
o/p:
101
Eating
```

2a.

```
package org;

public interface ReserveBank
{
     void deposit();
}
```

2b.

```
package org;

public interface ICICIBank extends ReserveBank
{
     void withdraw();
}
```

2c.

```
package org;

public class Customer implements ICICIBank {

	@Override
	public void deposit() {
		System.out.println("Depositing Amount");
	}

	@Override
	public void withdraw() {
		System.out.println("Withdrawing Amount");
	}

	public static void main(String[] args) {

		Customer c = new Customer();
		c.deposit();
		c.withdraw();

	}
}
```

o/p:

```
Depositing Amount
Withdrawing Amount
```

3a.

```
package org;

public interface Jspiders {

	void devlop();

}
```

3b.

```
package org;

public interface Qspiders {

	void test();

}
```

3c.

```
package org;

public class TestYantra {

	void work() {
		System.out.println("Working");
	}
}
```

3d.

```
package org;

public class Uday extends TestYantra implements Qspiders,
Jspiders  {

	@Override
	public void devlop() {
		System.out.println("Developing");
	}

	@Override
	public void test() {
		System.out.println("Testing");
	}

}
```

3e.

```
package org;

public class Solution {

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

```
                Uday u = new Uday();

                u.devlop();
                u.test();
                u.work();
        }
}
```

```
o/p:
Developing
Testing
Working
```

Abstraction

1. The process of Hiding the Implementation details (unnecessary details) and showing only the functionalities (Behaviour) to the user with the help of an abstract class or interface is called as Abstraction.

2. The process of Hiding the Implementation and showing only the functionality is called as Abstraction.

3. Abstraction can be achieved by following the below rules:

    i. Abstract class or Interface.

    ii. Is-A (Inheritance).

    iii. Method Overriding.

    iv. Upcasting.

1a.

```
package com;

public abstract class Person {

     abstract void work();

}

/*public interface Person {

     void work();

}*/
```

1b.

```
package com;

public class Employee extends Person { // implements
Person

	@Override
	public void work() {
		System.out.println("Employee is Working");
	}

}
```

1c.

```
package com;

public class Test {

	public static void main(String[] args) {

		/*Employee e = new Employee();
		e.work();*/

		Person p = new Employee();
		p.work();
```

```
     }

}
```

o/p:

```
Employee is Working
```

2a.

```
package org.bankapp;

public interface Bank {

     void deposit(int amount);
     void withdraw(int amount);
     void checkBalance();

}
```

2b.

```
package org.bankapp;

public class ATM implements Bank {

     int balance = 10000;

     @Override
```

```
	public void deposit(int amount) {
		System.out.println("Depositing Rs."+amount);
		balance = balance + amount;
		System.out.println("Amount Deposited 
Successfully");
	}

	@Override
	public void withdraw(int amount) {
		System.out.println("Withdrawing Rs."+amount);
		balance -= amount; // balance = balance - amount;
		System.out.println("Amount Withdrawn 
Successfully");
	}

	@Override
	public void checkBalance() {
		System.out.println("Available Balance: 
Rs."+balance);
	}

}
```

2c.

```
package org.bankapp;

public class AccountHolder {
    public static void main(String[] args) {

        Bank obj = new ATM();

        obj.checkBalance();

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

        obj.deposit(5000);
        obj.checkBalance();

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

        obj.withdraw(4500);
        obj.checkBalance();

    }

}
```

o/p:

```
Available Balance: Rs.10000
------------------
Depositing Rs.5000
Amount Deposited Successfully
Available Balance: Rs.15000
------------------
```

```
Withdrawing Rs.4500
Amount Withdrawn Successfully
Available Balance: Rs.10500
```

2d.

```
package org.bankapp;

import java.util.Scanner;

public class Solution {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);
        Bank b = new ATM();

        while(true)
        {
            System.out.println("Enter Choice");

    System.out.println("1:Deposit\n2:Withdraw\n3:CheckBal
ance\n4:Exit");
            int choice = scan.nextInt();

            switch(choice)
            {
            case 1:
                    System.out.println("Enter Amount to be
Deposited:");
                    int amount = scan.nextInt();
                    b.deposit(amount);
                    break;

               case 2:
                    System.out.println("Enter Amount to be
Withdrawn:");
                    int amt = scan.nextInt();
                    b.withdraw(amt);
                    break;

               case 3:
                    b.checkBalance();
                    break;

               case 4:
                    System.out.println("Thank You!!");
                    System.exit(0);

               default:
                    System.out.println("Invalid Choice");
               }
               System.out.println("---------------------");
          }

     }
```

```
}
```

```
o/p:
Enter Choice
1:Deposit
2:Withdraw
3:CheckBalance
4:Exit
1
Enter Amount to be Deposited:
2000
Depositing Rs.2000
Amount Deposited Successfully
---------------------
Enter Choice
1:Deposit
2:Withdraw
3:CheckBalance
4:Exit
3
Available Balance: Rs.12000
---------------------
Enter Choice
1:Deposit
2:Withdraw
```

```
3:CheckBalance
4:Exit
2
Enter Amount to be Withdrawn:
5000
Withdrawing Rs.5000
Amount Withdrawn Successfully
----------------------
Enter Choice
1:Deposit
2:Withdraw
3:CheckBalance
4:Exit
3
Available Balance: Rs.7000
----------------------
Enter Choice
1:Deposit
2:Withdraw
3:CheckBalance
4:Exit
4
Thank You!!
```

2.

```
package org.bankapp;
```

```
import java.util.Scanner;

public class Demo {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        while(true)
        {
            System.out.println("Enter Choice:");
            int choice = scan.nextInt();

            switch(choice)
            {
            case 1:
                System.out.println("Hai");
                break;

            case 2:
                System.out.println("Bye");
                break;

            case 3:
                System.exit(0);

            default:
                System.out.println("Invalid Choice");
            }

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

        }

    }

}
```

```
o/p:
Enter Choice:
1
Hai
----------
Enter Choice:
2
Bye
----------
Enter Choice:
5
Invalid Choice
----------
Enter Choice:
3
```

3a.

```
package org.bankapp;

public interface A
{
	void m1();
}
```

3b.

```
package org.bankapp;

public interface B
{
	void m1(int a);
}
```

3c.

```
package org.bankapp;

public class Test implements A, B
{
	@Override
	public void m1()
	{

	}
```

```
    @Override
    public void m1(int a)
    {

    }
}
```