

# Java Notes



Java<sup>TM</sup>

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 for diagram**

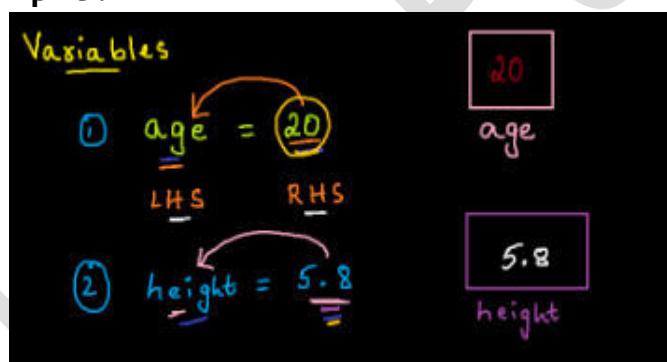
## History:

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**.

## 1. Variable Declaration and Initialization

Variable Declaration

Syntax: datatype variableName ;

① int age ;      25      age

② double salary ;      4500.56      salary

---

Variable Initialization

Syntax: variableName = value ;

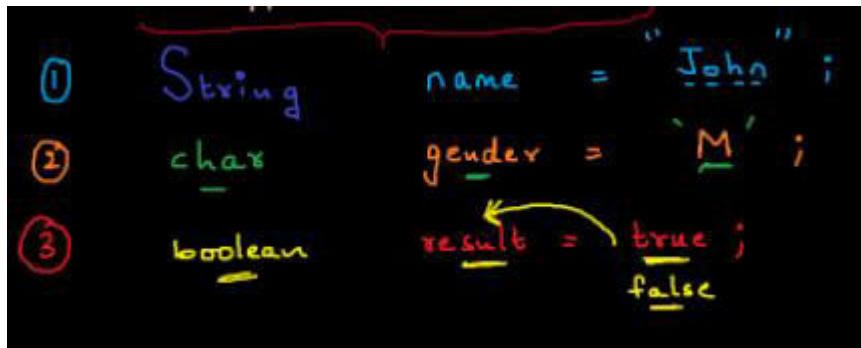
① age = 25 ;      25      age

② salary = 4500.56 ;      4500.56      salary

## 2. Variable Declaration and Initialization

Variable Declaration & Initialization

Syntax: datatype variableName = value ;



## Programs

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

## **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 Student  
{  
    public static void main(String[] args)  
    {  
        // Variable Declaration  
        int age;  
  
        // Variable Initialization  
        age = 20;  
  
        // Variable Declaration & Initialization  
        String name = "Jerry";  
  
        System.out.println(age);  
        System.out.println(name);  
  
        System.out.println("-----");  
  
        System.out.println("Student Age = "+age);  
        System.out.println("Student Name is " + name);  
  
        System.out.println("-----");  
  
        System.out.println(age+name);  
  
        System.out.println("-----");  
  
        System.out.println(age+" "+name);  
  
    }  
}
```

```
/* this  
is  
a  
java program */
```

**o/p:**

**20**

**Jerry**

-----

**Student Age = 20**

**Student Name is Jerry**

-----

**20Jerry**

-----

**20 Jerry**

**2.**

**class Operators**

**{**

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

**{**

```
/* Arithmetic Operators */
```

```
int a = 5;  
int b = 10;  
int sum = a+b;
```

```
System.out.println("Sum: "+sum);
```

```
System.out.println(a-2);
```

```
System.out.println(4*b);
```

```
System.out.println(10/5);
```

```
System.out.println(10%2);
```

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

/* Assignment Operators */

int x = 10;
System.out.println("value of x = "+x);

x+=20; // x = x+20;
System.out.println("value of x = "+x);

int i = 45;
System.out.println("value of i = "+i);

i -= 13;
System.out.println("value of i = "+i);

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

}

o/p:
Sum: 15
3
40
2
0
-----
value of x = 10
value of x = 30
value of i = 45
value of i = 32
-----
```

**3.**

**class Demo**

**{**

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

**{**

**/\* Comparison Operators \*/**

**int x = 10;**

**int y = 20;**

**System.out.println(x < y); // true**

**System.out.println(x <= 5); // false**

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

**System.out.println(y > 50); // false**

**System.out.println(y >= 20); // true**

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

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

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

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

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

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

**/\* Logical Operators \*/**

**int a = 1;**

```
int b = 2;
boolean result = a<b && a==1;

System.out.println(result); // true
System.out.println(a<=4 && a==b); // false

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

System.out.println(a<5 || b==5); // true
System.out.println(a==b || b!=2); // false

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

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

System.out.println(!(1<2)); // false
System.out.println(!(20<5)); // true

}

}

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

**false**

**true**

**false**

**false**

**true**

**false**

**true**

**4.**

**class Unary**

**{**

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

**{**

**int x = 5;**

**System.out.println("x: "+x); // 5**

**x++;**

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

**x++;**

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

**x--;**

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

**x--;**

**System.out.println("x: "+x); // 5**

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

```
int i = 40;  
int j = i++; // Post Increment -> First Assign, Then  
Increment
```

```
System.out.println(i+" "+j); // 41 40
```

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

```
int a = 5;  
int b = --a; // pre decrement -> first decrement,  
then assign
```

```
System.out.println(a+" "+b); // 4 4
```

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

```
/* Ternary Operators */
```

```
int p = 10;  
int q = 50;  
  
int max = p>q ? p : q;  
  
System.out.println("Maximum of "+p+" & "+q+" is  
"+max);
```

```
}  
}
```

**o/p**  
x: 5

```
x: 6  
x: 7  
x: 6  
x: 5  
=====  
41 40  
-----  
4 4  
=====
```

**Maximum of 10 & 50 is 50**

### 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

#### Simple If:

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

#### 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.

#### If Else If

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

1.

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

        int n = 5;

        if(n<=10) // if(5<=10) -> if(true)
        {
            System.out.println(n+" is lesser than or equal
to 10");
        }

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

o/p:

```
start
5 is lesser than or equal to 10
end
```

2.

```
class IfElseDemo
{
    public static void main(String[] args)
    {
        int a = 50;
        int b = 20;

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

```
        System.out.println(a+" is greater than "+b);
    }

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

    if(true)
    {
        System.out.println("Hai Dinga");
    }
    else
    {
        System.out.println("Bye Dinga");
    }

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

    if(false)
    {
        System.out.println("Hai Dinga");
    }
    else
    {
        System.out.println("Bye Dinga");
    }

}

}
```

**o/p:**  
50 is greater than 20

-----  
Hai Dinga  
-----  
Bye Dinga

3.

```
class IfElseIfDemo
{
    public static void main(String[] args)
    {
        int a = 10;
```

```
int b = 10;

if(a<b) // if(10<10) -> if(false)
{
    System.out.println(a+" is lesser than "+b);
}
else if(a>b) // if(10>10) -> if(false)
{
    System.out.println(a+" is greater than "+b);
}
else
{
    System.out.println(a+" is equal to "+b);
}
}

o/p:
```

10 is equal to 10

4.

```
class MarksValidation
{
    public static void main(String[] args)
    {
        int marks = -7;

        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, Study Well!! :(");
        }
        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.

**1.**

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

        if(a<=10) // false outer if
        {
            System.out.println("Inside Outer If");

            if(a==5) // inner if
            {
                System.out.println("a is equal to 5");
            }
            else // inner else
            {
                System.out.println("a is not equal to
5");
            }
        }
    }
}
```

```
        }

    }

else // outer else

{

    System.out.println("a greater than 10");

}

}

}

o/p:
```

a greater than 10

2.

```
class LoginValidation

{

    public static void main(String[] args)

    {

        String id = "adminxyz";

        int password = 123;

        if(id == "admin")

        {

            System.out.println("User Id is Valid");

            if(password == 123)

            {
```

```
        System.out.println("Password is  
Valid");  
  
        System.out.println("Login Successful  
:));  
    }  
  
    else  
    {  
  
        System.out.println("Password is  
Invalid");  
  
        System.out.println("Login Unsuccessful  
:(");  
    }  
  
}  
}  
}
```

**o/p:**  
**User Id is Invalid**  
**Login Unsuccessful :(**

**3.**

**class AssignmentPrograms**

```
{  
    public static void main(String[] args)  
    {  
        int num = -10;  
  
        if(num>0)  
        {  
            System.out.println(num+" is a Positive  
Number");  
        }  
        else  
        {  
            System.out.println(num+" is a Negative  
Number");  
        }  
  
        System.out.println("-----");  
  
        int n = 215;  
  
        if(n%2 == 0) // if(215%2 == 0) -> 1 == 0  
        {  
            System.out.println(n+" is a Even Number");  
        }  
        else  
        {  
            System.out.println(n+" is a Odd Number");  
        }  
    }  
}
```

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

int a = 20;
int b = 20;

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

if(a>b)
{
    System.out.println("a is Largest");
}
else if(a<b) // b>a
{
    System.out.println("b is Largest");
}
else
{
    System.out.println("a and b are both
equal");
}
}
```

**o/p:**

-10 is a Negative Number

215 is a Odd Number

-----

a:20 b:20

a and b are both equal

### Switch Statement:

- Switch Statement is generally used for Character Comparison.

1.

```
class SwitchDemo
{
    public static void main(String[] args)
    {
        int choice = 3;

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

```
    }  
}  
}
```

**o/p:**

**In Case-3**

**2.**

```
class GradeValidation  
{  
    public static void main(String[] args)  
    {  
        char grade = 'C';  
  
        switch(grade)  
        {  
            case 'A': System.out.println("Excellent ->  
Distinction");  
                    break;  
  
            case 'B': System.out.println("Good -> First  
Class");  
                    break;  
  
            case 'C': System.out.println("Bad -> Fail  
:(");  
                    break;  
        }  
    }  
}
```

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

}
```

o/p:

Bad -> Fail :(

3.

```
class LargestOfThreeNumbers
{
    public static void main(String[] args)
    {
        int a = 20;
        int b = 150;
        int c = 10;

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

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

```
        }

    }

    else if(b>c)

    {

        System.out.println("b is Largest");

    }

    else

    {

        System.out.println("c is Largest");

    }

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

    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:20 b:150 c:10**

**b is Largest**

-----

**b is Largest**

**4.**

**class Demo**

**{**

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

**{**

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

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

**System.out.print(" bye"+10);**

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

**System.out.print("hai"+" hello");**

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

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

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

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

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

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

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

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

```
        System.out.print("bye");

    }

}
```

**o/p:**

```
hai
hai bye10java
hai hellohaibye
hai
Good Morning
Bye
=====
haiMorningbye
```

**5.**

```
class MatrimonyPortal
{
    public static void main(String[] args)
    {
        char gender = 'M';
        int age = 19;

        if(gender == 'M')
        {
            System.out.println("Gender is Male");
        }
    }
}
```

```
if(age>=21)
{
    System.out.println("Yes, You can get
Married");
}
else
{
    System.out.println("Have Patience :)");
}
else if(gender == 'F')
{
    System.out.println("Gender is Female");

    if(age>=18)
    {
        System.out.println("Yes, You can get
Married");
    }
    else
    {
        System.out.println("Have Patience :)");
    }
}
else
{
    System.out.println("Invalid");
```

```
    }  
}  
}  
  
o/p  
Gender is Male  
Have Patience :)
```

### 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.

### Assignment

1. Write a Java Program to print even numbers from 1-10
2. Write a Java Program to print odd numbers from 1-10
3. Write a Java Program to print multiples of 3 from 1-15

1.

```
class ForLoopDemo  
{
```

```
public static void main(String[] args)
{
    // Print Hello 5 times

    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");
    }

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

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

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

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

**o/p:**

Hello

Hello

Hello

Hello

Hello

-----

Hello

Hello

Hello

Hello

Hello

-----

1

2

3

4

5

-----

1

2  
3  
4  
5

2.

```
class ForLoopExample
{
    public static void main(String[] args)
    {
        int n = 5;
        int sum = 0;

        for(int i=1; i<=n; i++)
        {
            sum = sum + i;          // sum += i;
        }

        System.out.println("Sum: "+sum);

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

        // 146 147 148 149 150
        for(int x=146; x<=150; x++)
        {
            System.out.println(x);
        }
    }
}
```

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

// 5 4 3 2 1
for(int i=5; i>=1; i--)
{
    System.out.println(i);
}

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

// 2 4 6 8 10
for(int i=2; i<=10; i=i+2)
{
    System.out.println(i);
}

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

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

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

```
for(int i=2; i<=10; i+=2)
{
    System.out.print(i+" ");
}
}
```

**o/p:**

**Sum: 15**

-----  
**146**

**147**

**148**

**149**

**150**

-----  
5

4

3

2

1

-----  
2

4

6

8

10  
-----

1  
3  
5  
7  
9  
-----

2 4 6 8 10

### 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.

\*\*\*\*\*

### 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.

### Nested For Loop:

- One For Loop present inside another For Loop.

1.

```
class WhileLoopDemo
{
    public static void main(String[] args)
    {
        int n = 1;
        while(n<=5)
        {
            System.out.println(n);
            n++;
        }

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

        int x = 1;
        do
        {
            System.out.println(x);
            x++;
        }
        while(x<=5);

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

        int i = 5;;
        while(i>=1)
        {
    }
```

```
        System.out.println(i);
        i--;
    }

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

    int j = 5;
    do
    {
        System.out.println(j);
        j--;
    }
    while(j>=1);

}
```

**o/p:**

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

4  
5  
-----  
5  
4  
3  
2  
1  
-----  
5  
4  
3  
2  
1  
2.  
**class NestedForLoopDemo**  
{  
    **public static void main(String[] args)**  
    {  
        **for(int i=1; i<=3; i++)**  
        {  
            **for(int j=5; j<=6; j++)**  
            {  
                **System.out.println("i:"+i + " j:"+j);**  
            }  
            **System.out.println("-----");**  
    }

```
}

for(int i=1; i<=5; i++)
{
    for(int j=1; j<=5; j++)
    {
        System.out.print("* ");
    }
    System.out.println();
}
}
```

**o/p:**

```
i:1 j:5
i:1 j:6
-----
i:2 j:5
i:2 j:6
-----
i:3 j:5
i:3 j:6
-----
* * * * *
* * * * *
* * * * *
* * * * *
```

\* \* \* \* \*

Uday Pawar

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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:
  - o States
  - o 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.
/*
Accessing Non-Static Variables inside same class
*/

class Student
{
    //  NON-STATIC VARIABLES
    int age = 20;
    String name = "Dinga";

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

        Student s = new Student();           // OBJECT
CREATION

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

        System.out.println("-----");
        System.out.println("Age: "+s.age);
        System.out.println("Name: "+s.name);

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

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

```

**o/p**

```
start
*****
20
Dinga
-----
Age: 20
Name: Dinga
-----
Dinga is 20 years old
*****
end
```

2a.

```
class Employee
{
    int id = 101;
    String name = "Tom";
    double salary = 123.45;
}
```

2b.

```
/*
Accessing Non-Static Variables in another 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

**Tom**  
**123.45**

Uday Pawar

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

### 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:**

1. Create a class called as Employee.
2. Declare 3 attributes as id, name and salary.
3. Create another class called as Solution.
4. Under main(), create 3 objects of Employee, then re-initialize to their respective values and display the contents.

1.

```
package com;
```

```
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

2.

```
package com;

class Car {

    int cost = 10;

    public static void main(String[] args) {

        Car c1 = new Car();
        Car c2 = new Car();

        System.out.println(c1.cost+
"+c2.cost);
```

```
    c1.cost = 15;
    c2.cost = 23;

    System.out.println(c1.cost+
"+c2.cost);

}
```

o/p:  
10 10  
15 23

3.

```
package com;
```

```
class Student {
```

```
    String name;
    int marks;
```

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

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

```
        System.out.println(s1.name+
"+s1.marks);
```

```
        System.out.println(s2.name+
"+s1.marks);
```

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

```
s1.name = "Tom";
s1.marks = 36;

s2.name = "Jerry";
s2.marks = 40;

System.out.println(s1.name+
"+s1.marks);
System.out.println(s2.name+
"+s2.marks);

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

System.out.println(s1.name+ " has scored
"+s1.marks+ " marks in Java");
System.out.println(s2.name+ " has scored
"+s2.marks+ " marks in Java");

}

o/p:
null 0
null 0
-----
Tom 36
Jerry 40
-----
Tom has scored 36 marks in Java
Jerry has scored 40 marks in Java
```

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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. Reusability 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.

```
package com;

class Demo {

    /* Method without Arguments and without return statement */
    void display()
    {
        System.out.println("Hello World!");
    }

    public static void main(String[] args) {

        System.out.println("start");

        Demo d = new Demo();

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

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

o/p:

```
start
Hello World!
end
```

2.

```
package com;

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

    public static void main(String[] args) {

        Addition obj = new Addition();
    }
}
```

```
        obj.add(10, 20);
        obj.add(6, 3);
        obj.add(123, 456);

    }
}
```

**o/p:**

```
Sum of 10 & 20 is 30
Sum of 6 & 3 is 9
Sum of 123 & 456 is 579
```

**3.**

```
package com;

class Test
{
    /* Method without Arguments and with return statement */
    int display()
    {
        return 10;
    }

    public static void main(String[] args) {

        System.out.println("start");

        Test t = new Test();

        int result = t.display();
        System.out.println(result);

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

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

**o/p:**

```
start
10
10
end
```

4.

```
package com;

class Example
{
    /* Method with Arguments and with return statement */
    int findSquare(int n)
    {
        return n*n;
    }

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

        int res = e.findSquare(5);
        System.out.println(res);

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

**o/p:**

```
25
16
```

5a.

```
package com;

class Solution {
    /* Method without Arguments and without return statement */
    void m1() {
        System.out.println("Learning Methods");
    }

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

    /* Method without Arguments and without return statement */
    String m3() {
        return "Jspiders";
    }

    /* Method with Arguments and with return statement */
    int m4(int a, int b) {
```

```

        return a+b;
    }

}

5b.

package com;

public class MainClass {

    public static void main(String[] args) {

        Solution s = new Solution();

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

        s.m2("john", 25);
        System.out.println("-----");

        String company = s.m3();
        System.out.println(company);

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

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

        int sum = s.m4(4, 5);
        System.out.println(sum);

        System.out.println(s.m4(12, 78));

    }
}

```

o/p:

Learning Methods

-----

Name: john Age:25

-----

Jspiders

Jspiders

-----

9

90

6.

```
package com;

public class Runner {

    int m1()
    {
        return 10+10;
    }

    double m2()
    {
        return 10+10.7;
    }

    String m3()
    {
        return "hai"+10;
    }

    String a = "java";
}
```

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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.

## 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)

**1a.**

**package com;**

```
public class MethodOverloading {  
  
    void display() {  
        System.out.println("Hello DabbaFellow");  
    }  
  
    void display(int x) {  
        System.out.println(x);  
    }  
  
    void display(double x) {  
        System.out.println(x);  
    }  
  
    void display(int x, String y) {  
        System.out.println(x+" "+y);  
    }  
  
    void display(String x, int y) {  
        System.out.println(x+" "+y);  
    }  
}
```

1b.

```
package com;  
  
public class Runner {
```

```
public static void main(String[] args) {  
  
    MethodOverloading mol = new MethodOverloading();  
  
    mol.display(45);  
    mol.display(12, "java");  
    mol.display(45.65);  
    mol.display();  
    mol.display("eclipse", 789);  
  
}  
}
```

**o/p:**

```
45  
12 java  
45.65  
Hello DabbaFellow  
eclipse 789
```

**2.**

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

```
{  
    System.out.println("Hello");  
    main(10);  
    main(12.45);  
    System.out.println("Bye");  
  
}  
  
public static void main(int a)  
{  
    System.out.println("a:"+a);  
}  
  
public static void main(double b)  
{  
    System.out.println("b:"+b);  
}  
}  
  
o/p:  
Hello  
a:10  
b:12.45  
Bye  
  
3.  
package com;
```

```
import java.util.Scanner;

public class Test {

    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);

        scan.close();
    }
}
```

**o/p:**

Enter the value of a:

5

Enter the value of b:

20

25

4.

```
package com;

import java.util.Scanner;

public class Employee {

    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 Salary:");
        double salary = s.nextDouble();

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

        System.out.println("Age:"+age+"\nName:"+name+"\nSalary:"+salary);
    }
}
```

```
}
```

**o/p:**

**Enter Age:**

**25**

**Enter Name:**

**John**

**Enter Salary:**

**1200.35**

**-----**

**Age:25**

**Name:John**

**Salary:1200.35**

**5.**

```
package com;
```

```
import java.util.Scanner;
```

```
public class Solution {
```

```
    void add(int a, int b) {
```

```
        System.out.println("Sum of "+a+" and "+b+" is "+  
(a+b) );
```

```
}
```

```
public static void main(String[] args) {  
  
    Solution s = new Solution();  
    Scanner scan = new Scanner(System.in);  
  
    for(int i=1; i<=3; i++)  
    {  
        System.out.println("Enter First Number:");  
        int a = scan.nextInt();  
        System.out.println("Enter Second Number:");  
        int b = scan.nextInt();  
  
        s.add(a, b);  
  
        System.out.println("-----");  
    }  
}  
}
```

**o/p:**

**Enter First Number:**

**2**

**Enter Second Number:**

**3**

**Sum of 2 and 3 is 5**

-----  
**Enter First Number:**

**45**

**Enter Second Number:**

**65**

**Sum of 45 and 65 is 110**

-----

**Enter First Number:**

**526**

**Enter Second Number:**

**9562**

**Sum of 526 and 9562 is 10088**

-----

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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.

1a.

```
package com;
```

```
/* Accessing static properties within the same class */
```

```
public class Student
```

```
{
```

```
    static int age = 20;
```

```
static void study()
{
    System.out.println("Student is Studying");
}

public static void main(String[] args)
{
    System.out.println(Student.age);
    Student.study();

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

    System.out.println(age); //  

ClassName.variableName -> ClassName.age -> Student.age
    study(); // ClassName.methodName() ->  

ClassName.study() -> Student.study()
}

}
```

**o/p:**  
**20**  
**Student is Studying**  
-----  
**20**  
**Student is Studying**

2a.

```
package com;

class Employee {

    static int id = 101;

    static void work() {
        System.out.println("Employee is Working");
    }

}
```

2b.

```
package com;

/* Accessing static properties in another class */

public class Test {

    public static void main(String[] args) {

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

        // System.out.println(id); Test.id
        // work(); Test.work();

    }
}
```

```
}
```

**o/p:**

**101**

**Employee is Working**

**3.**

```
package com;
```

```
public class Car {
```

```
    static int cost = 10;
```

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

```
        System.out.println(cost); // Car.cost
```

```
        cost = 20; // Car.cost = 20;
```

```
        System.out.println(cost); // Car.cost
```

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

```
    Car c1 = new Car();
```

```
        System.out.println(c1.cost); // not a good
practice

    }

}
```

**o/p:**

```
10
20
-----
20
```

**4a.**

```
package com;

public class Demo {

    void checkEvenOrOdd(int n) {
        if(n%2 == 0)
        {
            System.out.println(n+" is a Even Number");
        }
        else
        {
            System.out.println(n+" is a Odd Number");
        }
    }
}
```

```
    }

}

4b.

package com;

import java.util.Scanner;

public class Solution {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);
        Demo d = new Demo();

        for(int i=1; i<=4; i++)
        {
            System.out.println("Enter Number:");
            int num = scan.nextInt();

            d.checkEvenOrOdd(num);

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

        scan.close();
    }
}
```

}

**o/p:**

**Enter Number:**

**1**

**1 is a Odd Number**

-----

**Enter Number:**

**2**

**2 is a Even Number**

-----

**Enter Number:**

**3**

**3 is a Odd Number**

-----

**Enter Number:**

**4**

**4 is a Even Number**

-----

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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 jspiders;
```

```
public 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 jspiders;
```

```
public class Student
{
    static int age;

    static
    {
        age = 10; // Student.age = 10;
    }

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

    static
    {
        Student.age = 20; // age = 20;
    }
}

// age = 0 -> 10 -> 20
```

**o/p:**

Age: 20

### 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.

3.

```
package jspiders;
```

```
public class Test
```

```
{
```

```
{
```

```
    System.out.println("In Non-Static Block-1");
```

```
}
```

```
{
```

```
    System.out.println("In Non-Static Block-2");
```

```
}
```

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

```
{
```

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

```
Test t = new Test();
```

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

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

}
```

**o/p:**

**Start**

**In Non-Static Block-1**

**In Non-Static Block-2**

**In Non-Static Block-3**

**End**

**4.**

```
package jspiders;

public class Employee
{
```

```
    int id;
```

```
{
```

```
    id = 10;
```

```
}
```

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

    System.out.println("ID: "+e.id);
}

{
    id = 20;
}

}

// id = 0 -> 10 -> 20
```

**o/p:**

ID: 20

5.

```
package jspiders;
```

```
public class Car
```

```
{
```

```
    static
```

```
{
```

```
    System.out.println(1);
```

```
}
```

```
public static void main(String[] args)
{
    Car c = new Car();
    System.out.println(3);
}

{
    System.out.println(2);
}
}
```

o/p:

1  
2  
3

6.

```
package jspiders;

public class Pen
{
    static int x = 10;

    static
    {
        x = 20;
    }
}
```

```
public static void main(String[] args)
{
    Pen p = new Pen();
    System.out.println(x);
}

{
    x = 30;
}

// x -> 10 -> 20 -> 30
```

**o/p:**

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/converts 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**

7.

```
package jspiders;
```

```
public class Runner {
```

```
    static void m1()
```

```
{
```

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

```
    int result = m2();
```

```
    System.out.println("Bye "+result);
```

```
}

static int m2()
{
    return 20;
}

public static void main(String[] args)
{
    System.out.println("start");
    m1();
    System.out.println("end");
}
}

o/p:
start
Hai
Bye 20
end
```

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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\*\*\*\*\*

1a.

```
package com;

class Car
{
    // Non-Parameterized Custom Constructor
    Car()
    {
        System.out.println("Hello");
    }

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

        Car c = new Car();

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

**o/p:**

**start**  
**Hello**  
**end**

2.

```
package com;

class Bike
{
    // Parameterized Custom Constructor
    Bike(int cost) // cost=1000
    {
        System.out.println("Cost: "+cost);
    }

    public static void main(String[] args)
    {
        Bike b = new Bike(1000);
    }
}
```

**o/p:**

Cost: 1000

3.

```
package com;

public class Pen
{
    /* implicitly present -> Default Constructor
    Pen()
```

```
{  
}  
*/  
public static void main(String[] args)  
{  
    Pen p = new Pen();  
}  
}
```

4.

```
package com;
```

```
class Student
```

```
{  
    int age;  
  
    Student(int a)  
    {  
        age = a;  
    }
```

```
public static void main(String[] args) {  
  
    Student s1 = new Student(21);  
    Student s2 = new Student(22);
```

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

**o/p:**

Age: 21

Age: 22

5.

```
package com;

class Kangaroo
{
    double height = 5.5; // Member/Global
Variable (Non-Static)

    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

6.

```
package com;
```

```
class Person {
```

```
    int age;  
    String name;
```

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

```
    public static void main(String[] args) {  
  
        Person p1 = new Person(20, "Tom");  
        Person p2 = new Person(22, "Tim");  
    }
```

```
        System.out.println("Name: "+p1.name+"    Age:  
" +p1.age);  
  
        System.out.println("Name: "+p2.name+"    Age:  
" +p2.age);  
  
    }  
  
}
```

**o/p:**

```
Name: Tom    Age: 20  
Name: Tim    Age: 22
```

**7.**

```
package com;  
  
class Vehicle  
{  
    int x = 10;  
}  
  
class Test  
{  
    public static void main(String[] args)  
    {  
        Vehicle v = new Vehicle();  
        System.out.println(v.x);  
    }  
}
```

```
    }  
}
```

**o/p:**

**10**

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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
{
    int age = 40;
}
```

1b.

```
package singlelevel;

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

1c.

```
package singlelevel;

public class Test {

    public static void main(String[] args) {

        Son s = new Son();

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

    }
}
```

**o/p:**

**40**

**Tom**

2a.

```
package multilevel;

public class University {

    String universityName = "VTU";

    void conductExams() {
        System.out.println("VTU is conducting Exams");
    }

}
```

2b.

```
package multilevel;

public class College extends University {

    String collegeName = "Jspiders";

    void providePlacements() {
        System.out.println("Jspiders Provides
Placement");
    }

}
```

2c.

```
package multilevel;

public class Department extends College {

    String departmentName = "Computer Science";

    void fest() {
        System.out.println("CS Department had a Fest
called as Equinox");
    }

}
```

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.fest();  
        d.providePlacements();  
  
    }  
  
}
```

**o/p:**

```
University Name: VTU  
College Name: Jspiders  
Department Name: Computer Science  
-----  
VTU is conducting Exams  
CS Department had a Fest called as Equinox  
Jspiders Provides Placement
```

**3a.**

```
package hierarchical;  
  
public class Vehicle {  
  
    String brand = "BMW";
```

```
}
```

3b.

```
package hierarchical;

public class Car extends Vehicle {

    int cost = 45000;

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

}
```

3c.

```
package hierarchical;

public class Bike extends Vehicle {

    String fuel = "Petrol";

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

}
```

```
}
```

3d.

```
package hierarchical;
```

```
public class Test {
```

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

```
        Car c = new Car();
```

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

```
        c.start();
```

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

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

```
        System.out.println(b.brand+" "+b.fuel);
```

```
        b.stop();
```

```
}
```

```
}
```

o/p:

```
BMW 45000
```

```
Car Started
```

```
-----
```

```
BMW Petrol
```

```
Bike Stopped
```

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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.

## Programs

1.

```
package org;
```

```
class Employee {
```

```
    int id;  
    String name;  
    double salary;
```

```
Employee(int id, String name, double salary) {
```

```
    this.id = id;  
    this.name = name;  
    this.salary = salary;  
}
```

```
public static void main(String[] args) {  
  
    Employee e1 = new Employee(101, "Dinga",  
1200.34);  
  
    Employee e2 = new Employee(201, "Guldu", 700.89);  
  
    System.out.println("Employee Details");  
    System.out.println("=====");  
  
    System.out.println("Employee Id: "+e1.id);  
    System.out.println("Employee Name: "+e1.name);  
    System.out.println("Employee Salary:  
"+e1.salary);  
  
    System.out.println("-----");  
  
    System.out.println("Employee Id: "+e2.id);  
    System.out.println("Employee Name: "+e2.name);  
    System.out.println("Employee Salary:  
"+e2.salary);  
  
}  
}
```

2.

```
package com;  
  
public class Employee {
```

```
int id;
String name;
double salary;

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

void display() {
    System.out.println("Employee Id: "+this.id);
    System.out.println("Employee Name: "+this.name);
    System.out.println("Employee Salary: "+salary);
// this.salary
}

public static void main(String[] args) {
    Employee e1 = new Employee(101, "Dinga",
1200.34);
    Employee e2 = new Employee(201, "Guldu", 700.89);

    System.out.println("Employee Details");
    System.out.println("=====");
    e1.display();
```

```
        System.out.println("-----");
        e2.display();
    }

}
```

**o/p of both the programs:**

**Employee Details**

=====

**Employee Id: 101**

**Employee Name: Dinga**

**Employee Salary: 1200.34**

-----

**Employee Id: 201**

**Employee Name: Guldu**

**Employee Salary: 700.89**

**3.**

**package com;**

**public class Vehicle {**

**Vehicle(String brand, int cost) {**

**System.out.println("Brand: "+brand+
Cost:"+cost);**

**}**

```
Vehicle(String brand) {  
    System.out.println("Brand: "+brand);  
}  
  
Vehicle() {  
    System.out.println("No Argument Constructor");  
}  
  
Vehicle(String brand, String fuel) {  
    System.out.println("Brand: "+brand+" Fuel:  
"+fuel);  
}  
  
Vehicle(int cost, String brand) {  
    System.out.println("Cost: "+cost+" Brand:  
"+brand);  
}  
  
public static void main(String[] args) {  
  
    Vehicle v1 = new Vehicle("BMW");  
    Vehicle v2 = new Vehicle("Audi", 45000);  
  
    new Vehicle(8999, "Suzuki");  
    new Vehicle("Honda", "Petrol");  
  
    new Vehicle();  
    Vehicle obj = new Vehicle();
```

```
    }  
}
```

**o/p:**

**Brand: BMW**

**Brand: Audi Cost:45000**

**Cost: 8999 Brand: Suzuki**

**Brand: Honda Fuel: Petrol**

**No Argument Constructor**

**No Argument Constructor**

### [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

**4a.**

```
package org;
```

```
public class Father
```

```
{
```

```
    int age = 50;
```

```
}
```

4b.

```
package org;
```

```
public class Son extends Father
```

```
{
```

```
    int age = 30;
```

```
    void display()
```

```
{
```

```
    int age = 10;
```

```
    System.out.println(age); // 10
```

```
    System.out.println(this.age); // 30
```

```
    System.out.println(super.age); // 50
```

```
}
```

```
}
```

4c.

```
package org;
```

```
public class MainClass {
```

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

```
        Son s = new Son();
```

```
s.display();  
}  
  
}  
  
o/p:  
10  
30  
50
```

Uday Pawar

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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' calling statements.

1.

```
package chaining;
```

```
public class Demo
```

```
{
```

```
    Demo(int a) // a=10
```

```
{
```

```
    this();
```

```
    System.out.println(1);
```

```
}
```

```
Demo()  
{  
    System.out.println(2);  
}  
  
public static void main(String[] args)  
{  
    System.out.println("START");  
  
    Demo d = new Demo(10);  
  
    System.out.println("END");  
}  
}
```

o/p:

START  
2  
1  
END

2.

package chaining;

public class Student {

```
Student(int age) // age = 22
{
    this("Tom");
    System.out.println("Age: "+age);
}

Student(double height) // height = 5.8
{
    this(22);
    System.out.println("Height: "+height);
}

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

public static void main(String[] args)
{
    Student s = new Student(5.8);
}
}
```

**o/p:**

Name: Tom

Age: 22

Height: 5.8

3.

```
// Recursive Chaining  
package chaining;  
  
public class Car {  
  
    Car(String brand)  
    {  
        // this(500);  
    }  
  
    Car(int cost)  
    {  
        // this("bmw");  
    }  
  
    public static void main(String[] args)  
    {  
        Car c = new Car("bmw");  
    }  
}
```

4.

```
package chaining;
```

```
public class Employee {  
  
    Employee(int id) // id = 101  
    {  
        System.out.println(id);  
    }  
  
    Employee(int id, String name) // id=101  
    name="Dinga"  
    {  
        this(id); // this(101);  
        System.out.println(name);  
    }  
  
    public static void main(String[] args)  
    {  
        new Employee(101, "Dinga");  
    }  
}
```

o/p:

101

Dinga

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

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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;

public class Car extends Vehicle {

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

}
```

**2c.**

```
package com;

public class Test {

    public static void main(String[] args) {

        Car c = new Car();

        c.start();

    }

}
```

**o/p:**

**Vehicle Started**  
**Car 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**

#### **final keyword**

- final keyword can be used with a variable, method, and class.
- final variable acts as a constant, whose value cannot be re-initialized.
- final methods can be inherited but cannot be Overridden.
- final class cannot be Inherited.

4.

```
package com;
```

```
public class Demo {
```

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

```
        final double PI = 3.14;
```

```
// PI = 3.4;

int a = 10;

a = 20;

a = 30;

}

}
```

5a.

```
package com;

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

5b.

```
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();
}
```

6a.

```
package com;

final class Father
{
```

```
}
```

**6b.**

```
package com;
```

```
class Son extends Father
```

```
{
```

```
}
```

```
// Error
```

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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.

1.

```
package com;

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[0] = 10;
        a[1] = 20;
        a[2] = 40;

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

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

```
// Array Declaration and Creation
double[] x = new double[4];

System.out.println(x[0]);
System.out.println(x[1]);
System.out.println(x[2]);
System.out.println(x[3]);

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

x[0] = 1.2;
x[1] = 3.4;
x[2] = 5.6;
x[3] = 7.8;

System.out.println(x[0]);
System.out.println(x[1]);
System.out.println(x[2]);
System.out.println(x[3]);

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

}

```

**o/p:**

```
0
0
0
-----
10
20
40
=====
0.0
0.0
0.0
0.0
-----

```

1.2  
3.4  
5.6  
7.8  
Length: 4

2.

```
package com;

public class Test {

    public static void main(String[] args) {

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

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

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

        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]);
        }

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

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

        int sum = 0;
```

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

    System.out.println("Sum: "+sum);

    System.out.println("Average: "+ (sum/data.length))
};

}

}
}
```

o/p:

```
10
20
30
-----
10
20
30
-----
30
20
10
=====
Sum: 150
Average: 30
```

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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:**

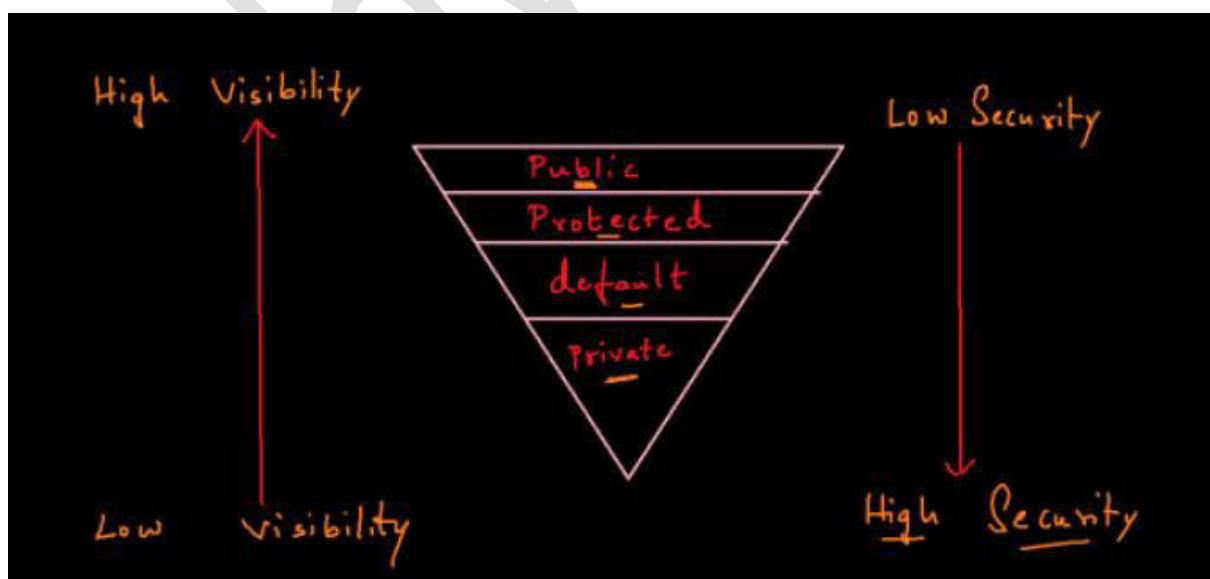
```
package com.google.gmail;  
  
class Inbox {  
  
}
```

## 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	✗	✓	✓	✓	✓

	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

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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.

## Programs

1a.

```
package com.javabean;

public class Person
{
    private int age;

    public void setAge(int age)
```

```
{  
    this.age = age;  
}  
  
public int getAge()  
{  
    return age; // return this.age;  
}  
}  
  
1b.  
  
package com.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 = 20;*/
```

o/p:

Age: 25  
25

2a.

```
package com.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 com.javabean;  
  
public class TestEmployee {  
  
    public static void main(String[] args) {  
  
        Employee emp = new Employee();  
  
        System.out.println(emp.getName()+"  
"+emp.getId());  
  
        System.out.println("-----");  
  
        // Accept using Scanner Class  
        emp.setName("Tom");  
    }  
}
```

```
        emp.setId(101);

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

    }

}
```

o/p:

```
null 0
-----
Name: Tom
Id: 101
```

3a.

```
package com.javabean;

public class Student {

    private int age;

    public void setAge(int age) {

        if(age>0) {
            System.out.println("Age Initialized");
            this.age = age;
        }
        else {
            System.out.println("Age Cannot be
Initialized");
        }
    }

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

}

3b.

```
package com.javabean;

import java.util.Scanner;

public class TestStudent {

    public static void main(String[] args) {

        Student s = new Student();

        Scanner scan = new Scanner(System.in);

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

        s.setAge(age);

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

o/p:

```
Enter Age:
25
Age Initialized
25
```

4a.

```
package com.javabean;

public class Car {

    private int cost;

    public Car(int cost) {
```

```

        if(cost>0) {
            this.cost = cost;
        }
    }

    public int getCost() {
        return cost;
    }

    public void setCost(int cost) {
        if(cost>0) {
            this.cost = cost;
        }
    }
}

```

4b.

```

package com.javabean;

public class TestCar {

    public static void main(String[] args) {

        Car c = new Car(100);

        System.out.println("Cost: "+c.getCost());

        c.setCost(200);
        System.out.println("Cost: "+c.getCost());

        c.setCost(300);
        System.out.println("Cost: "+c.getCost());
    }
}

```

o/p:

Cost: 100

Cost: 200

Cost: 300

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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.

1.

```
package primitivecasting;
```

```
public class Demo {
```

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

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

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

char c = 'a';
int d = c;
System.out.println(c+" "+d);

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

double x = 3.45;
int y = (int) x;
System.out.println(x+" "+y);

int p = 66;
char q = (char)p;
System.out.println(p+" "+q);

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

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

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

System.out.println('A'+'B');
```

```
        System.out.println('a'+20);

        System.out.println('a'+"a");

    }

}
```

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

```
// A -> 65 -> 101010101
```

```
// a -> 97
```

**o/p:**

**---Widening---**

**10 10.0**

**a 97**

**---Narrowing---**

**3.45 3**

**66 B**

**=====**

**AB**

**A20**

**131**

**117**

**aa**

**2.**

```
package primitivecasting;

public class Runner {

    public static void main(String[] args) {

        System.out.println(10);      // int
        System.out.println(10.5);    // double

        double x = 4.7;
        float y = (float) x; // float y = (float) 4.7;

        float a = (float) 3.5;

        float b = 3.5f;

        float c = 3.5F;

        float i = (float)1.2;

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

        long mobNo = 9999999999L;

        // long mobNo = 99999999991;

    }
}
```

```
}
```

**o/p:**

**10**

**10.5**

**3.**

```
package primitivecasting;
```

```
import java.util.Scanner;
```

```
public class Test {
```

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

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

```
        System.out.println("Enter No of Elements to be  
        Inserted:");
```

```
        int size = scan.nextInt();
```

```
        int[] a = new int[size];
```

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

```
        for(int i=0; i<a.length; i++) // i=3  
        {
```

```
            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("Enter the Element to be find
the No of Occurences:");
int element = scan.nextInt();

int count = 0;

for(int i=0; i<a.length; i++) // 10 20 30
20
{
    if(element == a[i]) {
        count++;
    }
}

System.out.println("No of Occurences of
"+element+" is "+count);

}
```

```
// 100 200 300  
// 0 1 2
```

**o/p:**

**Enter No of Elements to be Inserted:**

**3**

**Enter 3 Elements:**

**10**

**201**

**10**

**Array Elements are:**

**10**

**201**

**10**

**Enter the Element to be find the No of Occurences:**

**10**

**No of Occurences of 10 is 2**

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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 nonprimitive;  
  
public class Father  
{
```

```
        int age = 45;  
    }
```

1b.

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

1c.

```
package nonprimitive;  
  
public class Test {  
  
    public static void main(String[] args) {  
  
        /* UPCASTING */  
        Father f = new Son();  
        System.out.println(f.age); //f.name will give  
error  
  
        /* DOWNCASTING */  
        Son s = (Son) f;  
        System.out.println(s.age+" "+s.name);  
  
        /*Son s = new Son();  
        Father f = s;*/  
  
    }  
}
```

o/p:

45

45 Dinga

2a.

```
package nonprimitive;

public class Vehicle {

    String brand = "BMW";

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

}
```

2b.

```
package nonprimitive;

public class Car extends Vehicle {

    String fuel = "Petrol";

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

}
```

2c.

```
package nonprimitive;

public class Demo {

    public static void main(String[] args) {

        Vehicle v = new Car();
        System.out.println(v.brand);
        v.start();
```

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

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

}

}
```

o/p:

BMW  
Vehicle Started  
-----  
BMW Petrol  
Vehicle Started  
Car Stopped

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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**

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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 at 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 completime;  
  
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 completime;
```

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

2b.

```
package completime;

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

2c.

```
package completime;

public class Bike extends Vehicle { // 1

    @Override
    void start() { // 2
```

```
        System.out.println("Bike Started");  
    }  
  
}
```

2d.

```
package completime;
```

```
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 completime;
```

```
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

**eg:**

```
package runtime;

public class MainClass
{
    static void invokeWork(Employee emp)
    {
        emp.work();
    }
}
```

```
}

public static void main(String[] args)
{
    invokeWork(new Tester());
    invokeWork(new Developer());

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

    Tester t = new Tester();
    invokeWork(t);

    Developer d = new Developer();
    invokeWork(d);
}
}
```

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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 com;

public abstract class Person
{
    abstract void work();
}
```

1b.

```
package com;

public class Employee extends Person {

    @Override
    void work() {
        System.out.println("Working");
    }

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

```
        e.work();
    }

}
```

o/p:

Working

2a.

```
package com;

public abstract class Vehicle {

    abstract void start();

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

2b.

```
package com;

public abstract class Car extends Vehicle
{
    abstract void stop();

    // start() and shiftGears();
}
```

2c.

```
package com;

public class User extends Car {

    @Override
```

```

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

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

// optionally override shiftGears() as well

public static void main(String[] args) {

    User u = new User();
    u.start();
    u.shiftGears();
    u.stop();
}
}

```

o/p:

```

Car Started
Shifting Gears!
Car Stopped

```

### **Constructors in Abstract Class.**

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

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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 Emp 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) {

        Emp u = new Emp();

        u.devlop();
    }
}
```

```
    u.test();  
    u.work();  
}  
}
```

**o/p:**

**Developing**  
**Testing**  
**Working**

Uday Pawar

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## 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:CheckBalance\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);
        // b.withdraw(scan.nextInt());
        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**

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## Java Libraries

1. It is a collection of pre-defined packages.
2. Each package/folder is collection of pre-defined classes and pre-defined interfaces.
3. Each class or interface is a collection of variables and methods.
4. All the pre-defined classes and interfaces are present inside a jar file called as rt.jar (rt->runtime) or zip file -> src.zip

## Write Any 6 pre-defined package Names

- 1.java.lang
  - 2.java.util
  - 3.java.io
  - 4.java.sql
  - 5.java.applet
  - 6.java.awt .....
- 

### **1. java.lang package**

- This package is implicitly or automatically imported in all java classes and interfaces.

### **2. Object Class**

- Object is a pre-defined class present in java.lang package.
- Object class is referred as super-most class in java.
- Object class is implicitly inherited by all java classes.

### methods present in Object Class

1. protected Object clone()
  2. public boolean equals(Object o)
  3. public int hashCode()
  4. public String toString()
  5. public void wait()
  6. public void wait(long a)
  7. public void wait(long a, int b)
  8. public void notify()
  9. public void notifyAll()
  10. public Class getClass()
  11. protected void finalize()
- 

#### 1. **toString()**

syntax : public String toString()

- **toString()** returns the String representation of an Object in the below format.

**FullyQualifiedClassName @ HexadecimalValueOfTheHashCode**

#### **Programs:**

1.

```
package com;
```

```
import java.util.Scanner;
```

```
import java.util.ArrayList;

public class Demo
{
    Scanner s = new Scanner(System.in);

    ArrayList l = new ArrayList();

    Thread t = new Thread();

    Object o = new Object();
}
```

2.

```
package com;

public class Car {
    public static void main(String[] args) {
        // Up-casting
        Object obj = new Car();

        Car c = new Car();
    }
}
```

3. Without Overriding `toString()`

```
package com;
```

```
public class Person
{
    public static void main(String[] args)
    {
        Person p = new Person();

        System.out.println(p); // implicitly it calls
        toString() -> String Representation
        System.out.println(p.toString()); // explicitly
        calling toString()
    }
}
```

**o/p:**

```
com.Person@15db9742
com.Person@15db9742
```

#### 4. After overriding `toString()`

```
package org;

public class Person
{
    @Override
    public String toString()
    {
        return "Hi Guldu!";
    }

    public static void main(String[] args)
    {
        Person p = new Person();

        System.out.println(p); // implicitly calls
        toString()

        System.out.println(p.toString()); // explicitly
        calls toString()
    }
}
```

```
}
```

**o/p:**

```
Hi Guldu!  
Hi Guldu!
```

**5.**

```
package org;  
  
public class Student  
{  
    String name;  
  
    Student(String name) {  
        this.name = name;  
    }  
  
    @Override  
    public String toString() {  
        return "Name: "+name;  
    }  
  
    public static void main(String[] args)  
    {  
        Student s1 = new Student("Tom");  
        Student s2 = new Student("Jerry");  
  
        System.out.println(s1);  
        System.out.println(s2);  
  
        System.out.println(s1.toString());  
        System.out.println(s2.toString());  
    }  
}
```

**o/p:**

Name: Tom  
Name: Jerry  
Name: Tom  
Name: Jerry

6.

```
package org;

public class Employee {

    int id;
    String name;

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

    @Override
    public String toString() {
        return "Employee Id of "+name+" is "+id;
    }

    public static void main(String[] args) {

        Employee e1 = new Employee(101, "Ambani");
        Employee e2 = new Employee(102, "Tata");
        Employee e3 = new Employee(103, "Cook");

        System.out.println(e1);
        System.out.println(e2);
        System.out.println(e3);
    }
}
```

o/p:

Employee Id of Ambani is 101  
Employee Id of Tata is 102  
Employee Id of Cook is 103

Uday Pawar

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## Storing Objects inside Array

1.

```
package storingobjects;

public class Car
{
    String brand;

    Car(String brand)
    {
        this.brand = brand;
    }

    public static void main(String[] args)
    {
        Car c1 = new Car("BMW");
        Car c2 = new Car("Audi");

        Car[] c = new Car[2];
        c[0] = c1;
        c[1] = c2;

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

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

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

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

for(int i=0; i<c.length; i++)
{
    System.out.println(c[i]);
    System.out.println("Brand: "+c[i].brand);
}
}

/*
int[] a = new int[2];

a[0] = 10;
a[1] = 20;
*/

```

**o/p:**

**storingobjects.Car@15db9742**  
**storingobjects.Car@6d06d69c**

-----  
BMW

Audi  
-----

storingobjects.Car@15db9742

Brand: BMW

storingobjects.Car@6d06d69c

Brand: Audi

## 2.

```
package storingobjects;

public class Student {

    int age;
    String name;

    Student(int age, String name) {
        this.age = age;
        this.name = name;
    }

    public static void main(String[] args) {

        Student s1 = new Student(21, "Dinga");
        Student s2 = new Student(22, "Guldu");
        Student s3 = new Student(23, "Dingi");

        Student[] s = new Student[3];      // Array of type
        Student
        s[0] = s1;
        s[1] = s2;
        s[2] = s3;

        // Without Overriding toString
```

```

        for(int i=0; i<s.length; i++) {
            System.out.println(s[i]);
            System.out.println(s[i].age+" "+s[i].name);
        }
    }
}

```

o/p:

```

storingobjects.Student@15db9742
21 Dinga
storingobjects.Student@6d06d69c
22 Guldu
storingobjects.Student@7852e922
23 Dingi

```

3.

```

package storingobjects;

public class Person {

    int age;
    String name;

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

    @Override
    public String toString() {
        return "Age: "+age+" Name: "+name;
    }

    public static void main(String[] args) {

        Person p1 = new Person(21, "Dinga");
        Person p2 = new Person(22, "Guldu");
    }
}

```

```

Person p3 = new Person(23, "Dingi");

Person[] p = new Person[3];

p[0] = p1;
p[1] = p2;
p[2] = p3;

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

}

// p1  p2  p2
// 0    1    2

```

**o/p:**

```

Age: 21 Name: Dinga
Age: 22 Name: Guldu
Age: 23 Name: Dingi

```

**4a.**

```

package storingobjects;

class Laptop {

    String brand;
    int cost;
    String ramSize;

    Laptop(String brand, int cost, String ramSize) {
        this.brand = brand;
        this.cost = cost;
        this.ramSize = ramSize;
    }

    @Override

```

```

    public String toString() {
        return "Brand:" + brand +
            " Cost:" + cost +
            " RAM Size:" + ramSize;
    }
}

```

4b.

```

package storingobjects;

class Customer {

    public static void main(String[] args) {

        Laptop l1 = new Laptop("HP", 2000, "4GB");
        Laptop l2 = new Laptop("Dell", 3000, "8GB");
        Laptop l3 = new Laptop("Lenovo", 7000, "16GB");
        Laptop l4 = new Laptop("Acer", 2500, "2GB");

        Laptop[] l = {l1, l2, l3, l4};

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

    /*
    int[] a = new int[2];
    a[0] = 10;
    a[1] = 20;

    int[] a = {10, 20};

    Laptop[] l = new Laptop[4];
    l[0] = l1;
    l[1] = l2;
    */
}

```

```
l[2] = 13;  
l[3] = 14;  
*/
```

o/p:

```
Brand:HP Cost:2000 RAM Size:4GB  
Brand:Dell Cost:3000 RAM Size:8GB  
Brand:Lenovo Cost:7000 RAM Size:16GB  
Brand:Acer Cost:2500 RAM Size:2GB
```

## 2. **hashCode()**

```
*****
```

**syntax : public int hashCode()**

**hashCode()** is used to identify an object uniquely

**hashCode()** return a number or id to uniquely Identify an Object.

5.

```
package com;  
  
public class Employee {  
  
    public static void main(String[] args) {  
  
        Employee e = new Employee();  
  
        System.out.println(e.hashCode());  
  
    }  
  
}
```

**o/p:**

366712642

6.

```
package com;

public class Employee {

    @Override
    public int hashCode() {
        return 124;
    }

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

**o/p:**

124

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## **equals()**

- equals() is used to compare the content/members of two objects.

- equals() is used for content comparison.

**syntax:** public boolean equals(Object obj)

**1.**

**package comparingobjects;**

**public class Student {**

**int age;**

**Student(int age) {**

**this.age = age;**

**}**

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

**Student s1 = new Student(20);**

**Student s2 = new Student(20);**

**System.out.println(s1 == s2); // false**

**System.out.println(s1.equals(s2)); // false**

```
}
```

```
}
```

2.

```
package comparingobjects;
```

```
public class Student {
```

```
    int age;
```

```
    Student(int age) {
```

```
        this.age = age;
```

```
    }
```

```
    @Override
```

```
    public boolean equals(Object obj)
```

```
    {
```

```
        Student s = (Student) obj;
```

```
        return this.age == s.age;
```

```
    }
```

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

```
Student s1 = new Student(20);
Student s2 = new Student(20);

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

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

}

// s1 -> this
// s2 -> obj -> s
```

**o/p:**  
false  
true

**3.**

```
package comparingobjects;
```

```
public class Employee {
```

```
int id;  
double height;  
  
Employee(int id, double height) {  
    this.id = id;  
    this.height = height;  
}  
  
@Override  
public boolean equals(Object obj)  
{  
    Employee emp = (Employee) obj;  
    return this.id == emp.id && this.height == emp.height;  
}  
  
public static void main(String[] args) {  
    Employee e1 = new Employee(101, 5.8);  
    Employee e2 = new Employee(102, 5.8);  
  
    System.out.println(e1.equals(e2));  
  
    System.out.println("-----");
```

```
System.out.println(new Employee(1, 5.7).equals(new  
Employee(1, 5.4)));
```

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

```
if(e1.equals(e2)) {  
    System.out.println("Contents are Same");  
}  
else {  
    System.out.println("Contents are Different");  
}
```

```
}
```

**o/p:**

**false**

**false**

**Contents are Different**

4a.

```
package comparingobjects;
```

```
public interface A
```

```
{
```

```
    void m1(String a);
```

```
}
```

4b.

```
package comparingobjects;
```

```
public interface B
```

```
{
```

```
    void m1(int a);
```

```
}
```

4c.

```
package comparingobjects;
```

```
public class Demo implements A, B {
```

```
    @Override
```

```
    public void m1(int a) {
```

```
}

@Override
public void m1(String a) {

}

}
```

Uday Pawar

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## Singleton Design Pattern or Singleton Class

- It is a Design Pattern wherein we can create only a single instance or a single object for a class.

## Rules in order to Develop Singleton Class/Singleton Design Pattern

1. Declare a Private Constructor.
2. Have a public static helper/Factory method to create a single Object.
3. Declare a private static non primitive reference variable.

---

1a.

```
package com;

public class Account {

    private static Account a = null; // a = 0x11;

    private Account() {
        System.out.println("Object Created");
    }

    public static void createObject()
    {
        if(a==null)
```

```
        {  
            a = new Account();  
        }  
    else  
    {  
        System.out.println("Cannot Create");  
    }  
}  
}
```

1b.

```
package com;
```

```
public class Test {
```

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

```
        Account.createObject();
```

```
        Account.createObject();
```

```
        Account.createObject();
```

```
}
```

```
}
```

o/p:

**Object Created**

**Cannot Create**

**Cannot Create**

**2a.**

```
package singleton;
```

```
public class AadhaarCard {
```

```
    private static AadhaarCard ac;
```

```
    private AadhaarCard() {
```

```
        System.out.println("AadhaarCard Object Created");
```

```
    }
```

```
    public static void createAadhaarCardObject() {
```

```
        if(ac == null) {
```

```
            ac = new AadhaarCard();
```

```
        }
```

```
    }
```

```
}
```

**2b.**

```
package singleton;
```

```
public class Person {  
  
    public static void main(String[] args) {  
  
        AadhaarCard.createAadhaarCardObject();  
        AadhaarCard.createAadhaarCardObject();  
        AadhaarCard.createAadhaarCardObject();  
        AadhaarCard.createAadhaarCardObject();  
  
    }  
  
}
```

**o/p:**  
**AadhaarCard Object Created**

**3a.**  
**package singleton;**

```
public class PrimeMinister {  
  
    String name = "Modi";
```

```
private static PrimeMinister pm; // pm = null

private PrimeMinister() {
    System.out.println("PM got Elected");
}

public static PrimeMinister createAndReturnPMObject() {
    if(pm == null) {
        pm = new PrimeMinister();
    }
    return pm;
}

}
```

3b.

```
package singleton;
```

```
public class Test {
```

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

```
        PrimeMinister pm =
        PrimeMinister.createAndReturnPMObject();
```

```
        System.out.println("Name: "+pm.name);  
  
    }  
  
}
```

**o/p:**

**PM got Elected**

**Name: Modi**

**4a.**

```
package singleton;
```

```
public class Marriage {
```

```
    int age = 29;
```

```
    private static Marriage m;
```

```
    private Marriage() {
```

```
        System.out.println("Got Married");
```

```
}
```

```
public static Marriage getInstance() {  
    if(m == null) {  
        m = new Marriage();  
    }  
    return m;  
}  
}
```

4b.

```
package singleton;  
  
public class Solution {  
  
    public static void main(String[] args) {  
        Marriage obj = Marriage.getInstance();  
        System.out.println("at the age of "+obj.age);  
    }  
}
```

**o/p:**

**Got Married**

**at the age of 29**

Uday Pawar

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## String

- > String is pre-defined final class present in java.lang package.
  - > String Objects Immutable in Nature.
  - > String is a Collection/Set of Characters.
  - > String is also a Non-Primitive Datatype.
  - > String implements Serializable, Comparable, CharSequence
- 
- > String Objects can be created in 2 ways.
    1. literal (" ") double quotations
    2. using new Operator

---

    1. String s = "java";
    2. String s = new String("java");
- 
- > String Objects are stored inside a memory location called as String pool.
  - > String Pool is further divided into 2 types
    1. Constant Pool
    2. Non-Constant Pool
  - > Literal Objects are stored inside constant pool and constant pool does not allow duplicates.
  - > String Objects created using new operator are stored inside Non-Constant Pool and Non-Constant Pool allows duplicates.
  - > String class has automatically overridden 3 methods from Object class

1. `toString()`
2. `hashCode()`
3. `equals()`

-> `toString()` of the Object Class is Overridden in String Class to return the actual data passed to the Constructor during object creation.

-> `hashCode()` of the Object Class is overridden in String class to return a number based on the ASCII value.

-> `equals()` of the Object Class is Overridden in String Class to compare the contents of both the objects.

#### Constructors in String Class

```
package jspiders;

public class Test {

    public static void main(String[] args) {

        // Empty Representation of a String Object
        String s1 = new String();
        System.out.println(s1);

        // Passing a String Object
        String s2 = new String("dinga");
        System.out.println(s2);

        char[] ch = {'j', 'a', 'v', 'a'};
    }
}
```

```
// Converting an Array of Characters to String  
String s3 = new String(ch);  
System.out.println(s3);  
  
}  
  
}
```

**o/p:**

dinga  
java

### 1. How String Objects are Immutable? Explain String Immutability Concept.

- When we re-initialize a String object, rather than modifying the same object, a new object is created and the reference pointing to the old object gets de-referenced and starts pointing to the newly created object.

This is String Immutability Concept.

### Mutable Version of String

1. **StringBuffer**
2. **StringBuilder**

---

1.

```
package jspiders;

public class Student {

    public static void main(String[] args) {

        Student s = new Student();

        System.out.println(s); // implicitly calls
        toString()

        System.out.println(s.toString()); // explicitly
        calling toString()

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

        Student std = new Student();
        System.out.println(std.hashCode());

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

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

        System.out.println(s1.equals(s2));

    }
}
```

**o/p:**

```
jspiders.Student@15db9742
```

```
jspiders.Student@15db9742
```

```
-----
```

```
1829164700
```

```
-----
```

```
false
```

2.

```
package jspiders;
```

```
public class Demo {
```

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

```
        // String s = "java";
```

```
        String s = new String("java");
```

```
        System.out.println(s); // implicitly calls  
        toString()
```

```
        System.out.println(s.toString()); // explicitly  
        calling toString()
```

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

```
        String s1 = new String("a");
```

```
        System.out.println(s1.hashCode());
```

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

String a = new String("Dinga");
String b = new String("Dinga");

System.out.println(a.equals(b));
}

}
```

**o/p:**

**java**

**java**

-----

**97**

-----

**true**

### Methods Present in String class

```
package jspiders;

public class MethodsDemo {

    public static void main(String[] args) {

        String s = "Software Developer";

        System.out.println(s.length()); // 18
    }
}
```

```
System.out.println("-----");
System.out.println(s.toUpperCase());
System.out.println("-----");
System.out.println(s.toLowerCase());
System.out.println("-----");
System.out.println(s.startsWith("soft"));
System.out.println(s.startsWith("Soft"));
System.out.println("-----");
System.out.println(s.endsWith("er"));
System.out.println(s.endsWith("Eloper"));
System.out.println("-----");
System.out.println(s.contains("dev"));
System.out.println(s.contains("Dev"));
System.out.println("-----");
System.out.println(s.concat(" in TY"));
System.out.println("-----");
// Software Developer
System.out.println(s.charAt(2));
System.out.println(s.charAt(14));
System.out.println("-----");
System.out.println(s.indexOf('t'));
System.out.println(s.indexOf('D'));
System.out.println(s.indexOf('e'));
System.out.println("-----");
```

```

String a = "java";
String b = "JavA";
String c = "java";

System.out.println(a.equals(b)); // false
System.out.println(a.equals(c)); // true

System.out.println(a.equalsIgnoreCase(b));

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

String x = "hello dinga";

System.out.println(x.substring(3)); // lo dinga
System.out.println(x.substring(7)); // inga

System.out.println(x.substring(2, 7)); // llo d
System.out.println(x.substring(4, 10)); // o ding

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

}

```

o/p:

18

-----  
SOFTWARE DEVELOPER  
-----

software developer  
-----

false

true

-----  
true

false

-----  
false

true

-----

Software Developer in TY

-----

f

o

-----

3  
9  
7

-----

false

true

true

-----

lo dinga  
inga  
llo d  
o ding

-----

# Java Notes



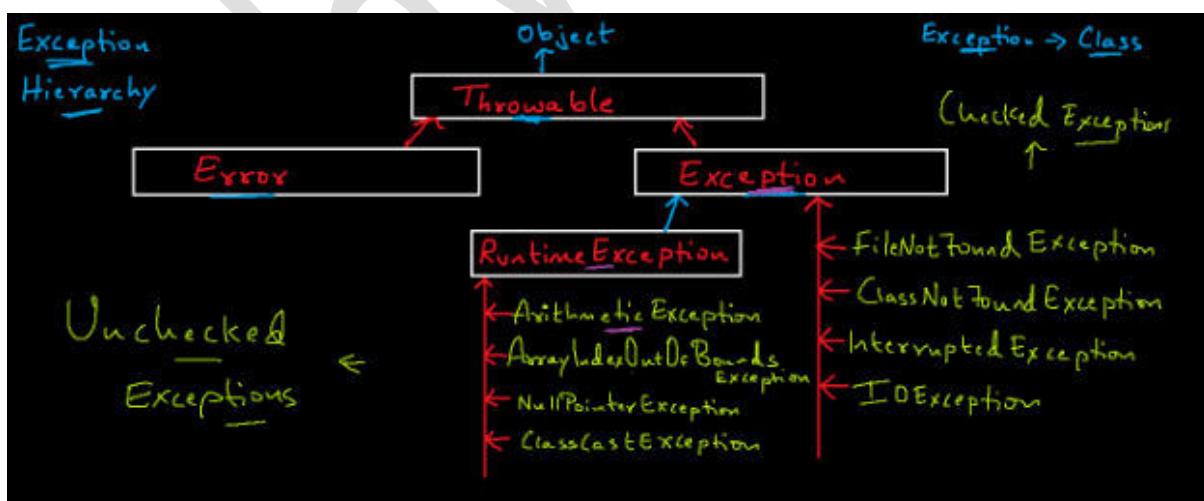
Java<sup>TM</sup>

Prepared by Uday Pawar

## Exception Handling

1. **Exception** is an event or an interruption which stops the execution of a program ie. abrupt termination wherein below lines of code will not get executed.
2. In other words, Exception is a Runtime Interruption which can be HANDLED.
3. **ERROR:** Error is also a Runtime Interruption/Mistake which cannot be HANDLED (We have to Debug)
  - Errors can occur during
    - i. Compile time -> **Compilation error** -> Syntax Mistakes
    - ii. Runtime -> **Runtime Error** -> Executing a class without main()

## Exception Hierarchy



1. The Process of handling an Exception is called as Exception Handling.
2. Typically an Exception is Handled using Try Block and Catch Block.

### try and catch block

1. The critical lines of code which gives an exception should be written inside the try block.
2. If there is try block, mandatorily catch block should be present and vice versa.
3. The Solution should always be written within the catch block.
4. catch block will be executed only if an exception occurs.

### syntax:

```
try
{
    // set of instructions
}
catch(ExceptionName referenceVariable)
{
    // set of instructions
}
```

4. One try block can have any number of catch blocks.

5. There should not be any executable lines of code between try and catch block.

6. It is always a good practice to handle the superclass exception as the last catch block.

## Programs

1.

```
package com;
```

```
import java.util.Scanner;
```

```
public class Demo {
```

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

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

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

```
        System.out.println("Enter the value of  
a:");
```

```
        int a = scan.nextInt(); // 10
```

```
        System.out.println("Enter the value of  
b:");
```

```
int b = scan.nextInt(); // 0

try
{
    System.out.println(a/b); // 10/0 ->
ArithmetricException
}
catch(ArithmetricException e) // Specific
Exception Handler
{
    System.out.println("Dabbafellow, do not
divide by 0");
}

scan.close();

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

**o/p:**

**start**

**Enter the value of a:**

**10**

**Enter the value of b:**

0

Dabbafellow, do not divide by 0

end

2.

```
package com;
```

```
public class Test {
```

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

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

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

```
        try {
```

```
            System.out.println(a[99]);
```

```
        }
```

```
        catch(ArrayIndexOutOfBoundsException e) {
```

```
// Specific Exception Handler
```

```
            System.out.println("Invalid Index");
```

```
        }
```

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

```
}
```

```
}
```

**o/p:**

**start**

**Invalid Index**

**end**

**3.**

```
package com;
```

```
public class Runner {
```

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

```
        try {
```

```
            System.out.println(10/0); // Object  
of ArithmeticException is thrown
```

```
        }
```

```
        catch(ArithmeticException e) {
```

```
            System.out.println("Invalid  
Denominator");
```

```
        }
```

```
    }  
}
```

```
// Exception e = new ArithmeticException();
```

**o/p:**

**Invalid Denominator**

```
/* internally  
 * 1. An object of ArithmeticException is created  
 * 2. the object is thrown  
 * 3. it is caught by the catch block  
 */
```

4.

```
package com;
```

```
public class Example {
```

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

```
        try {  
            System.out.println(10/0);  
        }
```

```
        catch(ArrayIndexOutOfBoundsException e) {
```

```
        System.out.println("Invalid Index  
Position");  
    }  
    catch(NullPointerException e) {  
        System.out.println("Invalid");  
    }  
    catch(ArithmeticException e) {  
        System.out.println("Invalid  
Denominator");  
    }  
    catch(Exception e) {  
        System.out.println("SuperClass  
Exception Handler");  
    }  
}  
}
```

**o/p:**  
**Invalid Denominator**

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

### Important methods present in Throwable Class:

1. **printStackTrace():** This method is used to get the complete information about the Exception.
2. **getMessage():** This method is used to return a small message about the Exception occurred.

### finally block

1. The Set of Instructions which has to be executed all the time has to written within the finally block.
2. Finally Block is a block of code which gets executed all the time. ie. irrespective of exception occurs or not.

### syntax:

```
finally  
{  
}
```

### Note:

1. In java we can have nested try and catch block.
2. We can have try and catch block within finally block as well.

1.

```
package org;

public class Demo {

    public static void main(String[] args) {

        System.out.println("start");

        try
        {
            System.out.println(10/0);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }

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

}

/*
Exception in thread "main"
java.lang.ArithmetricException: / by zero
```

```
        at org.Demo.main(Demo.java:7)
*/

```

**o/p:**

**start**

**java.lang.ArithmetricException: / by zero**

```
        at org.Demo.main(Demo.java:11)
```

**end**

**2.**

```
package org;
```

```
public class Test {
```

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

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

```
    try
```

```
    {
```

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

```
    }
```

```
    catch(Exception e)
```

```
    {
```

```
        System.out.println(e.getMessage());
```

```
        String message = e.getMessage();
        System.out.println(message);
    }

    System.out.println("end");

}

/*
Exception in thread "main"
java.lang.ArithmetricException: / by zero
at org.Test.main(Test.java:7)
*/

```

**o/p:**  
**start**  
**/ by zero**  
**/ by zero**  
**end**

**3.**

```
package org;
```

```
public class FinallyBlockExample {  
  
    public static void main(String[] args) {  
  
        System.out.println("start");  
  
        try  
        {  
            System.out.println(10/0);  
        }  
        catch(ArithmetricException e)  
        {  
            System.out.println("Invalid");  
        }  
        finally  
        {  
            System.out.println("Inside Finally  
Block");  
        }  
  
        System.out.println("end");  
  
    }  
}
```

**o/p:**

**start**

**Invalid**

**Inside Finally Block**

**end**

**4.**

```
package org;
```

```
public class Solution {
```

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

```
        try
```

```
        {
```

```
            try
```

```
            {
```

```
            }
```

```
            catch (Exception e)
```

```
            {
```

```
            }
```

```
        }
```

```
        catch (Exception e)
```

5.

```
package org;

public class MainClass {

    public static void main(String[] args) {
        try {
            finally {
                }
            }
        catch (Exception e)
        {
            }
        }
    }
}
```

```
String s = "java";  
  
char[] c = s.toCharArray();  
  
for(int i=0; i<c.length; i++)  
{  
    System.out.print(c[i]);  
}  
  
System.out.println();  
  
for(int i=c.length-1; i>=0; i--)  
{  
    System.out.print(c[i]);  
}  
}
```

**o/p:**  
java  
avaj

# Java Notes



Prepared by Uday Pawar

throws

[udaypawar037@gmail.com](mailto:udaypawar037@gmail.com)

- 1. throws is an indication to the caller about the possibility of an Exception.**
- 2. throws is used to propagate an Exception.**
- 3. throws is generally used with Checked Exceptions.**
- 4. Typically we use throws with methods, and we can use throws w.r.t Constructors as well.**

**1.**

```
package jspiders;
```

```
public class A {
```

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

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

```
}
```

```
}
```

```
// ArithmeticException -> Unchecked Exception
```

**2.**

```
package jspiders;
```

```
public class B {
```

```
public static void main(String[] args) {  
  
    int[] a = {10, 20, 30};  
  
    System.out.println(a[1000]);  
  
}  
}  
  
// ArrayIndexOutOfBoundsException -> Unchecked Exception
```

3.

```
package jspiders;  
  
public class C {  
  
    public static void main(String[] args) {  
  
        for(int i=1; i<=5; i++)  
        {  
            System.out.println(i);  
  
            try  
            {  
            }  
        }  
    }  
}
```

```
        Thread.sleep(2000);

    }

    catch (InterruptedException e)

    {

        e.printStackTrace();

    }

}

}

// InterruptedException -> Checked Exception
```

4.

```
package jspiders;
```

```
import java.io.FileNotFoundException;
```

```
import java.io.FileReader;
```

```
public class D {
```

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

```
        try
```

```
    {
        FileReader f = new FileReader("demo.txt");
    }
    catch (FileNotFoundException e)
    {
        System.out.println("File Not Found");
    }
}
```

// FileNotFoundException -> Checked Exception

5.

```
package jspiders;

public class Demo {
```

```
    static void div() throws ArithmeticException {
```

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

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

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

try
{
    div();
}

catch (Exception e)
{
    System.out.println("Handled");
}

System.out.println("end");

}

}

6.

package jspiders;

public class Test {

    static void display() throws Exception
```

```
{  
    for (int i=1; i<=5; i++)  
    {  
        System.out.println(i);  
        Thread.sleep(3000);  
    }  
}
```

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

```
    try {  
        display();  
    }  
    catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

7.

```
package jspiders;
```

```
import java.io.FileNotFoundException;
import java.io.FileReader;

public class Example {

    void readData() throws FileNotFoundException
    {
        FileReader f = new FileReader("dinga.txt");
    }

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

        try {
            e.readData();
        } catch (FileNotFoundException e1) {
            System.out.println("File not present");
        }
    }
}
```

8a.

**package example;**

**public interface Calculator {**

**int add(int a, int b);**

**int sub(int a, int b);**

**int mul(int a, int b);**

**int div(int a, int b);**

**}**

8b.

**package example;**

**public class CalculatorImpl implements Calculator {**

**@Override**

**public int add(int a, int b) {**

**return a+b;**

**}**

```
@Override  
public int sub(int a, int b) {  
    return a-b;  
}
```

```
@Override  
public int mul(int a, int b) {  
    return a*b;  
}
```

```
@Override  
public int div(int a, int b) {  
    return a/b;  
}  
}
```

8c.  
**package example;**

```
import java.util.Scanner;
```

```
public class MainClass {
```

```
public static void main(String[] args) {  
  
    Scanner scan = new Scanner(System.in);  
    Calculator c = new CalculatorImpl();  
  
    System.out.println("---Welcome to Calculator Project---");  
  
    System.out.println("=====");  
  
    while(true) {  
  
        System.out.println("1:Addition\n2:Subtraction\n3:Multiplication");  
        System.out.println("4:Division\n5:Exit");  
        System.out.println("Enter Choice:");  
        int choice = scan.nextInt();  
  
        int a = 0;  
        int b = 0;  
  
        if(choice>=1 && choice<=4)  
        {
```

```
System.out.println("enter first number:");
a = scan.nextInt();
System.out.println("enter second number:");
b = scan.nextInt();
}
```

```
switch(choice) {
```

```
case 1:
```

```
    int sum = c.add(a, b);
    System.out.println("Sum: "+sum);
    break;
```

```
case 2:
```

```
    int diff= c.sub(a, b);
    System.out.println("Difference: "+diff);
    break;
```

```
case 3:
```

```
    System.out.println("Product: "+c.mul(a, b));
    break;
```

```
case 4:
```

```
    System.out.println("Quotient: "+c.div(a, b));
```

```
        break;

case 5:
    System.out.println("Thank You!!");
    System.exit(0);

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

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

}

}

o/p:
---Welcome to Calculator Project---
=====
1:Addition
2:Subtraction
3:Multiplication
4:Division
5:Exit
```

**Enter Choice:**

**1**

**enter first number:**

**20**

**enter second number:**

**30**

**Sum: 50**

---

**1:Addition**

**2:Subtraction**

**3:Multiplication**

**4:Division**

**5:Exit**

**Enter Choice:**

**2**

**enter first number:**

**5**

**enter second number:**

**3**

**Difference: 2**

---

**1:Addition**

**2:Subtraction**

**3:Multiplication**

**4:Division**

**5:Exit**

**Enter Choice:**

**3**

**enter first number:**

**4**

**enter second number:**

**6**

**Product: 24**

---

**1:Addition**

**2:Subtraction**

**3:Multiplication**

**4:Division**

**5:Exit**

**Enter Choice:**

**5**

**Thank You!!**

# Java Notes



Java<sup>TM</sup>

Prepared by Uday Pawar

## Custom Exception or User-Defined Exception

**1. Based on the project, it is sometimes necessary to create our own Exception and those exceptions which the user/Programmer creates are called as Custom Exception or User-Defined Exception.**

### Rules for working with Custom Exception.

- 1. Create a class with the Exception Name.**
- 2. The Exception class which we created should either inherit Exception(Checked) or RuntimeException(Unchecked) class.**
- 3. Create an object of the exception class and invoke/throw the object of that Exception.**
- 4. Handle it using try and catch block.**

### throw

- 1. throw is a keyword in order to invoke an object of Exception.**
- 2. throw is generally used with custom exception.**

**syntax:** `throw objectOfExceptionType ;`

`// throw new ExceptionName();`

**1a.**

**package customexception;**

```
public class InvalidPasswordException extends RuntimeException
{
```

```
}
```

1b.

```
package customexception;
```

```
import java.util.Scanner;
```

```
public class LoginForm {
```

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

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

```
        System.out.println("Enter User Id:");
```

```
        String id = scan.next();
```

```
        System.out.println("Enter Password:");
```

```
        int password = scan.nextInt();
```

```
        if(id.equals("admin")) {
```

```
            if(password == 123) {
```

```
                System.out.println("Login Successful");
```

```
}
```

```
else {  
  
    try {  
        // throw new  
        InvalidPasswordException();  
  
        InvalidPasswordException obj = new  
        InvalidPasswordException();  
  
        throw obj;  
    }  
  
    catch(InvalidPasswordException e) {  
        System.out.println("Invalid Password  
Entered!!");  
    }  
  
}  
  
scan.close();  
  
}  
  
}
```

**o/p:**

**Enter User Id:**

**admin**

**Enter Password:**

**126**

**Invalid Password Entered!!**

**2a.**

**package customexception;**

**public class InsufficientBalanceException extends Exception {**

**}**

**2b.**

**package customexception;**

**import java.util.Scanner;**

**public class ATM {**

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

**Scanner s = new Scanner(System.in);**

```
int balance = 10000;

System.out.println("Enter Amount to be Withdrawn:");
int amount = s.nextInt();

if(amount<=balance) {
    System.out.println("Withdrawal Successful");
}
else
{
    try
    {
        throw new InsufficientBalanceException();
    }
    catch (InsufficientBalanceException e)
    {
        System.out.println("Not Enough Balance to
Withdraw! :(");
    }
}

}
```

**o/p:**

**Enter Amount to be Withdrawn:**

**15263**

**Not Enough Balance to Withdraw! :(**

**3a.**

```
package customexception;
```

```
public class Employee {
```

```
    private String name;
```

```
    Employee(String name) {
```

```
        this.name = name;
```

```
    }
```

```
    public String getName() {
```

```
        return name;
```

```
    }
```

```
}
```

**3b.**

```
package customexception;

public class Test {

    public static void main(String[] args) {

        Employee e = new Employee("Tom");
        System.out.println(e.getName());

    }

}
```

**o/p:**

**Tom**

**4a.**

```
package customexception;

public class AgeInvalidException extends RuntimeException {

    private String message;

    AgeInvalidException(String message) {
```

```
    this.message = message;  
}  
  
@Override  
public String getMessage() {  
    return message;  
}  
  
}
```

4b.

```
package customexception;  
  
import java.util.Scanner;  
  
public class MatrimonyPortal {  
  
    public static void main(String[] args) {  
  
        Scanner s = new Scanner(System.in);  
  
        System.out.println("Enter Age:");  
        int age = s.nextInt();
```

```
if(age>=21) {  
    System.out.println("Get Married Soon!!!");  
}  
  
else  
{  
    try  
    {  
        throw new AgeInvalidException("Have  
Patience, You are not yet 21!!");  
    }  
    catch (Exception e)  
    {  
        System.out.println(e.getMessage());  
    }  
}  
}  
}
```

```
/* AgeInvalidException obj = new AgeInvalidException("Have  
Patience!!");  
throw obj; */
```

**o/p:**

**Enter Age:**

**12**

**Have Patience, You are not yet 21!!**

Uday Pawar

```
1.
package day1;

import java.util.ArrayList;

public class Demo {

    public static void main(String[] args) {

        ArrayList l = new ArrayList();

        /* add() is used to add an object into the collection */
        l.add(10);
        l.add(20.34);
        l.add(10);
        l.add(null);
        l.add("java");

        System.out.println(l);

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

        /* size() is used to return the length of the Collection */
        System.out.println(l.size());

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

        /* get() is used to return an object based on the index
position */
        System.out.println(l.get(2));

        // System.out.println(l.get(250)); IndexOutOfBoundsException

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

        /* contains() is used to check if the object is present in the
Collection or not */
        System.out.println(l.contains("java"));
        System.out.println(l.contains("Java"));

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

        System.out.println(l);

        /* remove() is used to remove an object based on the index
position */
        l.remove(2);

        System.out.println(l);

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

        /* isEmpty() is used to check if the Collection is Empty or
not */
        System.out.println(l.isEmpty());

        /* clear() will remove all the Objects from the Collection */
        l.clear();

        System.out.println(l.isEmpty());
```

```

        System.out.println("=====");
        ArrayList x = new ArrayList();
        x.add(20);
        x.add(10);
        x.add(30);
        x.add(10);
        x.add(30);
        x.add(10);
        x.add(60);

        System.out.println(x);
        System.out.println("-----");
        /* indexOf() is used to find the index position of an Object
and first occurrence in case of duplication */
        System.out.println(x.indexOf(30));
        System.out.println(x.indexOf(10));
        System.out.println(x.indexOf(100)); // -1 if the object is not
present

        System.out.println("-----");
        /* lastIndexOf() is used to find the last index position of an
Object */
        System.out.println(x.lastIndexOf(10));
    }
}

```

**o/p:**

```

[10, 20.34, 10, null, java]
-----
5
-----
10
-----
true
false
-----
[10, 20.34, 10, null, java]
[10, 20.34, null, java]
-----
false
true
-----
[20, 10, 30, 10, 30, 10, 60]
-----
2
1
-1
-----
5

```

ArrayList	LinkedList
-	
1. PDC in java.util	1. PDC in java.util
2. JDK 1.2	2. JDK 1.2
3. Initial Capacity is 10	3. NO Initial Capacity
4. Incremental Capacity is (currentcapacity*3/2) + 1	4. NO Incremental Capacity
5. Underlined Data Structure Resizeable Array / Growable Array	5. Underlined Data Structure Doubly LinkedList
6. Sequential Memory Allocation Allocation	6. Non-Sequential Memory
7. 3 Constructors	7. 2 Constructors
1.	
package day2;	
import java.util.LinkedList;	
public class Demo {	
public static void main(String[] args) {	
LinkedList l = new LinkedList();	
l.add(10);	
l.add(20);	
l.add(30);	
l.add(40);	
System.out.println(l);	
System.out.println("-----");	
for(int i=0; i<l.size(); i++) {	
System.out.println(l.get(i));	
}	
System.out.println("-----");	
for(int i=l.size()-1; i>=0; i--) {	
System.out.println(l.get(i));	
}	
System.out.println("-----");	
int[] x = {10, 20, 30};	
System.out.println(x.length);	
String y = "java";	
System.out.println(y.length());	
System.out.println(l.size());	
}	

```
}
```

o/p:

```
[10, 20, 30, 40]
```

```
-----
```

```
10
```

```
20
```

```
30
```

```
40
```

```
-----
```

```
40
```

```
30
```

```
20
```

```
10
```

```
-----
```

```
3
```

```
4
```

```
4
```

```
-----
```

```
2.
```

```
package day2;
```

```
import java.util.ArrayList;
import java.util.LinkedList;
```

```
public class Test {
```

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

```
        ArrayList al = new ArrayList();
```

```
        al.add(10);
```

```
        al.add(20);
```

```
        System.out.println("Objects inside ArrayList: "+al+" Size:  
"+al.size());
```

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

```
        LinkedList ll = new LinkedList();
```

```
        /**
```

```
         * addAll() is used to add all the objects of one collection  
into another collection.
```

```
        */
```

```
        ll.addAll(al);
```

```
        ll.add(30);
```

```
        System.out.println("Objects inside LinkedList: "+ll+" Size:  
"+ll.size());
```

```
}
```

```
}
```

o/p:

```
Objects inside ArrayList: [10, 20] Size: 2
```

```
-----
```

```
Objects inside LinkedList: [10, 20, 30] Size: 3
```

```

3.
package day2;

import java.util.*;

public class Example {

    public static void main(String[] args) {

        ArrayList al = new ArrayList();
        al.add(10);
        al.add(20);
        System.out.println("Objects inside ArrayList: "+al+"      Size: "+al.size());

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

        LinkedList ll = new LinkedList();

        ll.addAll(al);
        ll.add(30);
        System.out.println("Objects inside LinkedList: "+ll+"      Size: "+ll.size());

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

        System.out.println("Objects inside LinkedList: "+ll+"      Size: "+ll.size());

        /**
         * containsAll() is used to check if all the objects of one
        collection is present
         * inside another collection or not.
         */
        System.out.println(ll.containsAll(al)); // true

        ll.remove(1);

        System.out.println("Objects inside LinkedList: "+ll+"      Size: "+ll.size());
        System.out.println(ll.containsAll(al));
    }
}

```

o/p:  
Objects inside ArrayList: [10, 20] Size: 2  
-----  
Objects inside LinkedList: [10, 20, 30] Size: 3  
-----  
Objects inside LinkedList: [10, 20, 30] Size: 3  
true  
Objects inside LinkedList: [10, 30] Size: 2  
false

```

4.
package day2;

import java.util.*;

```

```

public class Runner {

    public static void main(String[] args) {

        ArrayList al = new ArrayList();

        al.add(10);
        al.add(20);
        System.out.println("Objects inside ArrayList: "+al+" Size: "+al.size());

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

        LinkedList ll = new LinkedList();

        ll.addAll(al);
        ll.add(30);
        System.out.println("Objects inside LinkedList: "+ll+" Size: "+ll.size());

        // removeAll() is used to remove all the objects of one
        collection from another collection
        ll.removeAll(al);

        System.out.println("Objects inside LinkedList: "+ll+" Size: "+ll.size());

    }

}

```

o/p:  
 Objects inside ArrayList: [10, 20] Size: 2  
 -----  
 Objects inside LinkedList: [10, 20, 30] Size: 3  
 Objects inside LinkedList: [30] Size: 1

5.

```

package day2;

import java.util.ArrayList;

public class Solution {

    public static void main(String[] args) {

        ArrayList l = new ArrayList();

        l.add(10);
        l.add(20);
        l.add(30);

        System.out.println(l); // [10, 20, 30]

        // add() is used to add an object based on the index position
        // already existing object gets shifted to the next position
        l.add(1, 50);

        System.out.println(l); // [10, 50, 20, 30]
    }
}

```

```
// set() is used to add an object based on the index position  
// already existing object gets overridden  
l.set(2, 70);  
  
System.out.println(l); // [10, 50, 70, 30]  
  
}  
}
```

o/p:  
[10, 20, 30]  
[10, 50, 20, 30]  
[10, 50, 70, 30]

6.

```
package day2;
```

```
import java.util.ArrayList;  
import java.util.Collections;
```

```
public class MainClass {
```

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

```
        ArrayList l = new ArrayList();
```

```
        l.add(30);  
        l.add(40);  
        l.add(10);  
        l.add(20);
```

```
        System.out.println("Before Sorting:");  
        for(int i=0 ;i<l.size(); i++) {  
            System.out.println(l.get(i));  
        }
```

```
        Collections.sort(l);
```

```
        System.out.println("After Sorting:");  
        for(int i=0 ;i<l.size(); i++) {  
            System.out.println(l.get(i));  
        }
```

```
    }  
}
```

o/p:  
Before Sorting:

30  
40  
10  
20

After Sorting:

10  
20  
30  
40

```
1.  
package day3;  
  
public class ForEachLoopDemo {  
  
    public static void main(String[] args) {  
  
        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) {  
            System.out.println(i);  
        }  
  
        System.out.println("-----");  
  
        double[] percentage = {1.2, 3.4, 5.6, 7.8};  
  
        for(double z : percentage) {  
            System.out.println(z);  
        }  
  
        System.out.println("-----");  
  
        String[] fruits = {"Apple", "Mango", "ButterFruit"};  
  
        for(String fruit : fruits) {  
            System.out.println(fruit);  
        }  
    }  
}
```

o/p:  
10  
20  
30  
-----  
10  
20  
30  
-----  
1.2  
3.4  
5.6  
7.8  
-----  
Apple  
Mango  
ButterFruit

```
2.  
package day3;
```

```
public class BoxingDemo {  
    public static void main(String[] args) {  
  
        // Primitive way of representing 10  
        int a = 10;  
  
        // Non-Primitive way of representing 10  
        Integer b = new Integer(10);  
  
        System.out.println(a+" "+b);  
  
        System.out.println("-----");  
  
        // Primitive way of representing 'A'  
        char x = 'A';  
  
        // Non-Primitive way of representing 'A'  
        Character y = new Character('A');  
  
        System.out.println(x+" "+y);  
  
        System.out.println("=====");  
  
        int i = 5;  
        Integer j = new Integer(i);  
        System.out.println(i+" "+j);  
  
        System.out.println("-----");  
  
        char p = 'z';  
        Character q = new Character(p);  
        System.out.println(p+" "+q);  
  
        System.out.println("=====");  
  
        Integer c = new Integer(50);  
        int d = c;  
        System.out.println(c+" "+d);  
  
        System.out.println("-----");  
  
        Double obj = new Double(5.2);  
        double u = obj;  
        System.out.println(obj+" "+u);  
  
    }  
}
```

// All Wrapper class are pre-defined classes in java.lang package  
// All Wrapper class have overridden toString()

o/p:  
10 10  
-----  
A A  
=====

5 5  
-----

```

z z
=====
50 50
-----
5.2 5.2

3.
package day3;

import java.util.ArrayList;
import java.util.LinkedList;

public class GenericsDemo {
    public static void main(String[] args) {
        ArrayList<String> l = new ArrayList<String>();
        l.add("20");
        l.add("sql");
        l.add("java");

        for(String s : l) {
            System.out.println(s);
        }

        System.out.println("-----");
        LinkedList<Integer> x = new LinkedList<Integer>();
        x.add(10);
        x.add(34);
        x.add(67);

        for(Integer i : x) {           //for(int i : x) {
            System.out.println(i);
        }

        System.out.println("-----");
        ArrayList<Double> z = new ArrayList();
        z.add(8.99);
        z.add(102.34);
        z.add(3.45);

        for(double j : z) {           // for(Double j : z) {
            System.out.println(j);
        }
    }
}

```

o/p:  
20  
sql  
java

```
-----
10
34
67
-----
8.99
102.34
3.45

4.
package day3;

import java.util.ArrayList;
import java.util.LinkedList;

public class Example {

    public static void main(String[] args) {

        int a = 10;
        char b = 'z';

        ArrayList l = new ArrayList();

        l.add(a);           // l.add(new Integer(a)); ->
l.add(new Integer(10));
        l.add(20.45);       //
l.add(new Double(20.45));
        l.add(b);           // l.add(new Character(b)); ->
l.add(new Character('z'));

        for(Object o : l) {
            System.out.println(o);
        }

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

        LinkedList ll = new LinkedList();

        ll.add(10);          // new Integer(10);
ll.add("java");      // new String("java");
        ll.add(1.2);         // new Double(1.2);

        for(Object obj : ll) {
            System.out.println(obj);
        }

    }

}

o/p:
10
20.45
z
-----
10
java
1.2
```

## Programs

1a.

```
package day4;

public class Student {

    int age ;
    String name;

    Student(int age, String name) {
        this.age = age;
        this.name = name;
    }

}
```

1b.

```
package day4;

import java.util.ArrayList;

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

        Student s1 = new Student(21, "Tom");
        Student s2 = new Student(22, "Jerry");
        Student s3 = new Student(23, "Smith");

        ArrayList<Student> l = new ArrayList<Student>();

        l.add(s1);
        l.add(s2);
        l.add(s3);

        for(Student obj : l) {
            System.out.println(obj);
            System.out.println("Name:"+obj.name+" Age:"+obj.age);
            System.out.println("-----");
        }
    }
}
```

o/p:

```
day4.Student@15db9742
Name:Tom Age:21
-----
day4.Student@6d06d69c
Name:Jerry Age:22
-----
day4.Student@7852e922
Name:Smith Age:23
-----
```

2a.

```
package day4;

public class Employee {
```

```

        int id;
        String name;

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

        @Override
        public String toString() {
            return "Id:" + id + " Name:" + name;
        }
    }

2b.

package day4;

import java.util.ArrayList;

public class Runner {

    public static void main(String[] args) {

        Employee e1 = new Employee(101, "Ambani");
        Employee e2 = new Employee(102, "Cook");
        Employee e3 = new Employee(103, "Sundar");

        ArrayList<Employee> l = new ArrayList<Employee>();

        l.add(e1);
        l.add(e2);
        l.add(e3);

        for(Employee emp : l) {           // Iterators
            System.out.println(emp);
        }

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

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

    }
}

```

o/p:

```

Id:101 Name:Ambani
Id:102 Name:Cook
Id:103 Name:Sundar
-----
Id:101 Name:Ambani
Id:102 Name:Cook
Id:103 Name:Sundar

```

3.

```

package day4;

import java.util.Vector;

```

```
public class Solution {  
    public static void main(String[] args) {  
        Vector v = new Vector();  
  
        v.add(10);  
        v.add(20.45);  
        v.add("dinga");  
  
        for(Object o : v) {  
            System.out.println(o);  
        }  
    }  
}
```

o/p:  
10  
20.45  
dinga

4.

```
package day4;  
  
import java.util.ArrayList;  
import java.util.LinkedList;  
import java.util.Vector;  
  
public class Demo {  
  
    public static void main(String[] args) {  
  
        ArrayList x = new ArrayList();  
        x.add(10);  
        System.out.println(x);  
  
        LinkedList y = new LinkedList(x);  
        y.add(20);  
        System.out.println(y);  
  
        Vector z = new Vector(y);  
        z.add(30);  
        System.out.println(z);  
  
        System.out.println("-----");  
  
        ArrayList a = new ArrayList();  
        ArrayList b = new ArrayList(50);  
        LinkedList c = new LinkedList();  
  
        Vector i = new Vector();  
        Vector j = new Vector(60);  
        Vector k = new Vector(20, 5);  
    }  
}
```

```
}
```

o/p:

```
[10]
[10, 20]
[10, 20, 30]
```

### Wrapper Class

- The Non-Primitive version of a Primitive Datatype.
- The Class/Object version of a Primitive Datatype.
- All Wrapper Classes are present inside `java.lang` package.
- `toString()` is overridden in all Wrapper Classes and String class.

Primitive Datatype	Wrapper Class (Non-Primitive Datatype)
--------------------	--

1. byte	-->	Byte
2. short	-->	Short
3. int	-->	Integer
4. long	-->	Long
5. float	-->	Float
6. double	-->	Double
7. char	-->	Character
8. boolean	-->	Boolean

```
*****
```

### Auto-Boxing and Auto Un-Boxing

-- Auto-Boxing is a process of converting a primitive Datatype into non-primitive datatype.

-- Auto Un-Boxing is a process of converting a non-primitive Datatype into primitive datatype.

```
*****
```

### For Each Loop or Enhanced For Loop

- Enhanced For loop or For Each Loop is a looping Statement in order to traverse the group of Objects.

syntax: `for(DatatypeToBeTraversed iterating/traversing variable : Array or Collection object)`

```
{  
    -----;  
    -----;  
    -----;  
}
```

```
*****
```

Generics

-----

Generics are used to specify the element type  
jdk 1.5  
<ElementType>

\*\*\*\*\*

Vector:

Difference between ArrayList and LinkedList

ArrayList | LinkedList

-----  
1. PDC in java.util | 1. PDC in java.util  
2. JDK 1.2 | 2. JDK 1.2  
  
3. IC = 10 | 3. No initial capacity concept  
4. IC = (CC\*3)/2 +1 | 4. No incremental capacity concept  
  
5. Resizeable/ Growable | 5. Doubly LinkedList  
    Array  
  
6. 3 Constructors | 6. 2 Constructors  
  
7. Sequential Memory | 7. Non Sequential Memory Allocation  
    Allocation

Difference between ArrayList and Vector

ArrayList | Vector

-----  
1. JDK 1.2 | 1. JDK 1.0  
  
2. IC = (CC\*3)/2 +1 | 2. IC = CC\*2  
  
3. 3 Constructors | 3. 4 Constructors  
  
4. Not Thread Safe | 4. Thread Safe (Synchronized)

-----  
Constructors  
=====

ArrayList()  
ArrayList(int initialCapacity)  
ArrayList(Collection c)

-----  
LinkedList()  
LinkedList(Collection c)

-----

```
Vector()  
Vector(int intialCapacity)  
Vector(int intialCapacity, int incrementalCapacity)  
Vector(Collection c)
```

```
1.
package day5;

import java.util.HashSet;

public class Demo {

    public static void main(String[] args) {

        HashSet h = new HashSet();

        h.add(20);
        h.add(20.67);
        h.add(null);
        h.add(20);
        h.add("java");

        for(Object obj : h) {
            System.out.println(obj);
        }

        System.out.println("-----");
        System.out.println(h);
        System.out.println("-----");
        System.out.println("Size: "+h.size());
    }
}
```

o/p:  
null  
java  
20  
20.67  
-----  
[null, java, 20, 20.67]  
-----  
Size: 4

```
2.
package day5;

import java.util.LinkedHashSet;

public class Test {

    public static void main(String[] args) {

        LinkedHashSet<String> lhs = new LinkedHashSet<String>();

        lhs.add("java");
        lhs.add("python");
        lhs.add("java");
        lhs.add("javascript");

        for(String subject : lhs) {
            System.out.println(subject);
        }
    }
}
```

```
}

o/p:
java
python
javascript

3.
package day5;

import java.util.TreeSet;

public class Runner {

    public static void main(String[] args) {

        TreeSet<Integer> t = new TreeSet();

        t.add(30);
        t.add(40);
        t.add(20);
        t.add(30);
        t.add(50);
        t.add(10);

        for(int i : t) {
            System.out.println(i);
        }

    }
}

o/p:
10
20
30
40
50

4.
package day5;

import java.util.TreeSet;

public class Solution {

    public static void main(String[] args) {

        Integer i;

        TreeSet<String> t = new TreeSet<String>();

        t.add("Banana");
        t.add("Cat");
        t.add("Apple");

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

```

        System.out.println("-----");
        for(String s : t) {
            System.out.println(s);
        }
    }

o/p:
[Apple, Banana, Cat]
-----
Apple
Banana
Cat

5.
package day5;

import java.util.TreeSet;

public class MainClass {

    public static void main(String[] args) {

        String a = "A";
        String b = "B";

        System.out.println(a.compareTo(b)); // -1
        System.out.println(b.compareTo(a)); // +1
        System.out.println(b.compareTo(b)); // 0

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

        Integer x = 10;
        Integer y = 20;

        System.out.println(x.compareTo(y)); // -1
        System.out.println(y.compareTo(x)); // 1
        System.out.println(x.compareTo(x)); // 0

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

        Double i = 2.4;
        Double j = 3.5;

        System.out.println(i.compareTo(j));
        System.out.println(i.compareTo(i));
        System.out.println(j.compareTo(i));

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

        TreeSet t = new TreeSet();

        t.add(10);
        t.add("dinga");
        t.add(10.4);
    }
}

```

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

o/p:  
-1  
1  
0  
-----  
-1  
1  
0  
-----  
-1  
0  
1  
-----  
[ ]

How can we compare User-Defined Object?

-----  
<<Comparable>>

1. Comparable is a pre-defined interface present in java.lang package.
2. Comparable was introduced from JDK 1.2 .
3. <<Comparable>> has an abstract method called as compareTo().
4. <<Comparable>> is used to compare objects and sort them.
5. Default Sorting

```
syntax : public int compareTo(Object o); // E e

compareTo() returns      -> +1 if it is greater than
                           -> -1 if it is lesser than
                           ->  0 if it same objects
```

Rules to make the objects as comparable/ Rules in order to work with Comparable interface.

1. class has to implement Comparable interface.
2. Specify the Generics Type of which objects we are going to compare.
3. Override the compareTo() by specifying the business logic of comparing or sorting.

1a.

```
package defaultsorting;

class Car implements Comparable<Car> {

    int cost;

    Car(int cost) {
        this.cost = cost;
    }

    @Override
    public String toString() {
        return "cost: "+cost;
    }

    @Override
    public int compareTo(Car c) {
        return this.cost - c.cost;
    }
}

/*
public static void main(String[] args) {
Car c = new Car(100);
System.out.println(c);

String s = new String("java");
System.out.println(s);

Integer z = new Integer(10);
System.out.println(z);
}
*/
```

1b.

```
package defaultsorting;

import java.util.TreeSet;

class SortCar {

    public static void main(String[] args) {

        Car c1 = new Car(200);
        Car c2 = new Car(300);
        Car c3 = new Car(100);

        TreeSet<Car> t = new TreeSet();

        t.add(c1);
        t.add(c2);
        t.add(c3);

        for(Car c : t) {
            System.out.println(c);
        }
    }
}
```

o/p:

```
cost: 100
cost: 200
cost: 300
```

2a.

```
package defaultsorting;

public class Student implements Comparable<Student> {

    int id;
    String name;

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

    @Override
    public String toString() {
        return id+" "+name;
    }

    @Override
    public int compareTo(Student s) {
        return this.name.compareTo(s.name);
    }

    /*@Override
    public int compareTo(Student s) {
        return this.id - s.id;
    }*/
}
```

```

}

2b.
package defaultsorting;

import java.util.TreeSet;

public class SortStudent {

    public static void main(String[] args) {

        Student s1 = new Student(103, "Brian");
        Student s2 = new Student(101, "Craig");
        Student s3 = new Student(102, "Alex");

        TreeSet<Student> t = new TreeSet<Student>();

        t.add(s1);
        t.add(s2);
        t.add(s3);

        for(Student obj : t) {
            System.out.println(obj);
        }

    }
}

```

o/p:  
102 Alex  
103 Brian  
101 Craig

```

3a.
package defaultsorting;

public class Employee implements Comparable<Employee> {

    int id;
    String name;
    double salary;

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

    @Override
    public String toString() {
        return "Employee [id=" + id + ", name=" + name + ", salary=" +
salary + "]";
    }

    /*@Override
    public int compareTo(Employee e) {
        return (int) (this.salary - e.salary);           //2.4 - 1.1 ->
1.3 -> Explore -> Tomorrow
    }*/
}

```

```

@Override
public int compareTo(Employee e) {
    return e.name.compareTo(this.name);
}

//    @Override
//    public int compareTo(Employee e) {
//        return e.id - this.id; //return this.id - e.id;
//    }
}

3b.

package defaultsorting;

import java.util.TreeSet;

public class TestEmployee {

    public static void main(String[] args) {

        Employee e1 = new Employee(2, "John", 45.67);
        Employee e2 = new Employee(1, "Aaron", 62.67);

        TreeSet<Employee> t = new TreeSet<Employee>();

        t.add(e1);
        t.add(e2);
        t.add(new Employee(4, "Tom", 12.67));
        t.add(new Employee(3, "Jerry", 99.41));

        for(Employee emp : t) {
            System.out.println(emp);
        }
    }
}

```

o/p:

```

Employee [id=4, name=Tom, salary=12.67]
Employee [id=2, name=John, salary=45.67]
Employee [id=3, name=Jerry, salary=99.41]
Employee [id=1, name=Aaron, salary=62.67]

```

Comparator

- 
1. Comparator is a pre-defined interface present in java.util package.
  2. It was introduced from JDK 1.2 .
  3. Comparator has an abstract method called as compare().

syntax : public int compare(Object o1, Object o2);

Rules to use compare Objects using <>Comparato<>

- 
1. Design a new class in such a way which implements the Comparator interface and specify the Generics type.
  2. import the Comparator interface.
  3. Override the compare() by specifying the business logic for comparing and sorting.
  4. Pass The Object of the Class which has the sorting logic to the constructor of the TreeSet.

1a.

```
package customsoring;

public class Student {

    int age;
    String name;

    Student(int age, String name) {
        this.age = age;
        this.name = name;
    }

    @Override
    public String toString() {
        return age+" "+name;
    }
}
```

1b.

```
package customsoring;

import java.util.Comparator;

public class SortStudentsByAge implements Comparator<Student> {

    @Override
    public int compare(Student x, Student y) {
        return      x.age - y.age;
    }
}
```

1c.

```
package customsoring;
```

```

import java.util.Comparator;

public class SortStudentsByName implements Comparator<Student> {

    @Override
    public int compare(Student x, Student y) {
        return x.name.compareTo(y.name);
    }
}

1d.
package customsorting;

import java.util.TreeSet;

public class SortStudent {

    public static void main(String[] args) {

        Student s1 = new Student(24, "B");
        Student s2 = new Student(20, "C");
        Student s3 = new Student(21, "A");

        SortStudentsByAge age = new SortStudentsByAge();
        SortStudentsByName name = new SortStudentsByName();

        TreeSet<Student> t = new TreeSet<Student>(name);

        t.add(s1);
        t.add(s2);
        t.add(s3);

        for(Student std : t) {
            System.out.println(std);
        }
    }
}

```

o/p:  
21 A  
24 B  
20 C

2a.

```

package customsorting;

public class Employee {

    Integer id;
    String name;
    Double salary;

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

```

```

    }

    @Override
    public String toString() {
        return id+" "+name+" "+salary;
    }
}

2b.
package customsorting;

import java.util.Comparator;

public class SortEmployeeById implements Comparator<Employee> {

    @Override // descending
    public int compare(Employee x, Employee y) {
        return y.id.compareTo(x.id);
    }

    /*@Override Ascending order
    public int compare(Employee x, Employee y) {
        return x.id - y.id;
    }*/
}

2c.
package customsorting;

import java.util.Comparator;

public class SortEmployeeByName implements Comparator<Employee>{

    @Override
    public int compare(Employee x, Employee y) {
        return y.name.compareTo(x.name);
    }

    /*@Override
    public int compare(Employee x, Employee y) {
        return x.name.compareTo(y.name);
    }*/
}

2d.
package customsorting;

import java.util.Comparator;

public class SortEmployeeBySalary implements Comparator<Employee> {

    @Override
    public int compare(Employee x, Employee y) {
        return x.salary.compareTo(y.salary);
    }
}

```

```
2e.  
package customsorting;  
  
import java.util.TreeSet;  
  
public class SortEmployee {  
  
    public static void main(String[] args) {  
  
        Employee e1 = new Employee(10, "Ambani", 18.45);  
        Employee e2 = new Employee(30, "Birla", 18.67);  
  
        SortEmployeeById id = new SortEmployeeById();  
        SortEmployeeByName name = new SortEmployeeByName();  
        SortEmployeeBySalary salary = new SortEmployeeBySalary();  
  
        TreeSet<Employee> t = new TreeSet<Employee>(id);  
  
        t.add(e1);  
        t.add(e2);  
        t.add(new Employee(20, "Tata", 18.98));  
        t.add(new Employee(15, "Kotak", 18.34));  
  
        for(Employee emp : t) {  
            System.out.println(emp);  
        }  
    }  
}
```

## Map

=====

1. Map is a part of Collection Framework which does not extend the Collection interface.
2. Map is used organize the data in terms of key and value pair.
  - i. Keys cannot be Duplicated
  - ii. Values can be Duplicated
3. Map is a pre-defined interface present in java.util package.
4. Map was introduced from JDK 1.2
5. The Implementation Classes of the <> are:
  - a. HashMap
  - b. LinkedHashMap
  - c. TreeMap
  - d. HashTable
6. One key and value together we call it as ENTRY. Therefore In order Map is a collection ENTRIES.

### Important methods used wrt Maps

-----

1. put()
2. get()
3. clear()
4. isEmpty()
5. remove()
6. containsKey()
7. containsValue()
8. keySet()

### HashMap

-----

1. Pre-defined class in java.util package.
2. Jdk 1.2
3. Insertion Order is not Maintained.
4. Underlined Data Structure is HashTable.

### LinkedHashMap

-----

1. Pre-defined class in java.util package.
2. Jdk 1.4
3. Insertion Order is Maintained.
4. Underlined Data Structure is LinkedList and HashTable.

### TreeMap

-----

1. Pre-defined class in java.util package.
2. Jdk 1.2
3. Maintains Sorted Order ie.(Sorting based on key in ascending order)
4. Underlined Data Structure is Binary Tree.

-----  
HashMap      -> JDK 1.2 -> Not Thread Safe(Not Synchronized)

```
HashTable -> JDK 1.0 -> Thread Safe (Synchronized)

*****  
  
1.  
package mapprograms;  
  
import java.util.HashMap;  
  
public class Demo {  
  
    public static void main(String[] args) {  
  
        HashMap h = new HashMap();  
  
        // put() is used to add key and value inside Map  
        h.put(10, "dinga");  
        h.put("guldu", 10.45);  
        h.put(1.2, 100);  
  
        System.out.println(h);  
  
        System.out.println("-----");  
  
        // get() is used to get an value based on the key specified  
        System.out.println(h.get(10));  
        System.out.println(h.get(230));  
  
        System.out.println("-----");  
  
        // containsKey() is used to check if the key is present or not  
        System.out.println(h.containsKey("Guldu"));  
        System.out.println(h.containsKey("guldu"));  
  
        System.out.println("-----");  
  
        // containsValue() is used to check if the Value is present or  
not  
        System.out.println(h.containsValue(10.45));  
        System.out.println(h.containsValue("Dinga"));  
  
        System.out.println("-----");  
  
        System.out.println(h);  
  
        // remove() is used to remove an value based on the key  
specified  
        h.remove(10);  
  
        System.out.println(h);  
  
        System.out.println("-----");  
  
        // isEmpty() is used to check if the Collection is empty or  
not  
        System.out.println(h.isEmpty());  
  
        // clear() is used to remove all the objects from the  
Collection
```

```

        h.clear();

        System.out.println(h.isEmpty());
        System.out.println("-----");
    }

}

o/p:
{1.2=100, 10=dinga, guldu=10.45}
-----
dinga
null
-----
false
true
-----
true
false
-----
{1.2=100, 10=dinga, guldu=10.45}
{1.2=100, guldu=10.45}
-----
false
true
-----

```

2.

```

package mapprograms;

import java.util.HashMap;
import java.util.LinkedHashMap;
import java.util.Set;
import java.util.TreeMap;

public class Test {

    public static void main(String[] args) {

        HashMap<String, Integer> hm = new HashMap();

        hm.put("tom", 22);
        hm.put("jerry", 21);
        hm.put("bheem", 23);

        Set<String> s1 = hm.keySet();

        for(String key : s1) {
            System.out.println(key+" is "+hm.get(key)+" years old");
        }

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

        LinkedHashMap<Integer, String> lhm = new
        LinkedHashMap<Integer, String>();

        lhm.put(10, "Java");
        lhm.put(20, "Sql");
    }
}
```

```

        lhm.put(30, "Web");

        Set<Integer> s = lhm.keySet();

        for(int key : s) {
            System.out.println(key+" --> "+lhm.get(key));
        }

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

        TreeMap<Integer, Double> t = new TreeMap<Integer, Double>();

        t.put(20, 1.5);
        t.put(30, 2.5);
        t.put(10, 4.5);

        Set<Integer> s2 = t.keySet();

        for(int key : s2) {
            System.out.println(key+" : "+t.get(key));
        }
    }
}

```

**o/p:**

```

bheem is 23 years old
tom is 22 years old
jerry is 21 years old
-----
10 --> Java
20 --> Sql
30 --> Web
-----
10 : 4.5
20 : 1.5
30 : 2.5

```

3.

```

package mapprograms;

import java.util.Set;
import java.util.TreeMap;

public class Fruits {

    public static void main(String[] args) {

        TreeMap<String, Integer> t = new TreeMap();

        t.put("Mango", 23);
        t.put("Apple", 20);
        t.put("Banana", 15);

        Set<String> s = t.keySet();

        for(String key: s) {
            System.out.println("Cost of 1Kg "+key+" is
"+t.get(key));
        }
    }
}

```

```
        }  
    }  
  
o/p:
```

```
Cost of 1Kg Apple is 20  
Cost of 1Kg Banana is 15  
Cost of 1Kg Mango is 23
```

4.

```
package mapprograms;  
  
import java.util.HashMap;  
  
public class Runner {  
  
    public static void main(String[] args) {  
  
        HashMap h = new HashMap();  
  
        h.put(1, "Sony");  
  
        System.out.println(h);  
  
        h.put(1, "Nokia");  
  
        System.out.println(h);  
    }  
}
```

o/p:

```
{1=Sony}  
{1=Nokia}
```