

History of Java

- James Gosling, Mike Sheridan and Patrick Naughton initiated the Java language project in June 1991.
- Java was originally developed by James Gosling at Sun Microsystems and released in 1995.
- The language was initially called Oak after an oak tree that stood outside Gosling's office.
- Later the project went by the name Green and was finally renamed Java, from Java coffee, a type of coffee from Indonesia.
- Gosling and his team did a brainstorm session and after the session, they came up with several names such as JAVA, DNA, SILK, RUBY, etc.
- Sun Microsystems released the first public implementation as Java 1.0 in 1996.

Version History:-

Version	Date
JDK Beta	1995
JDK 1.0	January 23, 1996 ^[39]
JKD 1.1	February 19, 1997
J2SE 1.2	December 8, 1998
J2SE 1.3	May 8, 2000
J2SE 1.4	February 6, 2002
J2SE 5.0	September 30, 2004
Java SE 6	December 11, 2006
Java SE 7	July 28, 2011
Java SE 8	March 18, 2014
Java SE 9	September 21, 2017
Java SE 10	March 20, 2018
Java SE 11	September 25, 2018 ^[40]
Java SE 12	March 19, 2019
Java SE 13	September 17, 2019
Java SE 14	March 17, 2020
Java SE 15	September 15, 2020 ^[41]
Java SE 16	March 16, 2021

- The first version was released on January 23, 1996.
- The acquisition of Sun Microsystems by Oracle Corporation was completed on January 27, 2010
- As of September 2020, Java 8 and 11 are supported as Long Term Support (LTS) versions
- In September 2017, Mark Reinhold, chief Architect of the Java Platform, proposed to change the release train to "one feature release every six months".
- OpenJDK (Open Java Development Kit) is a free and open source implementation of the (Java SE). It is the result of an effort Sun Microsystems began in 2006.

- Java Language and Virtual Machine Specifications: <https://docs.oracle.com/javase/specs/>

Java Platforms

1. Java SE
 - Java Platform Standard Edition.
 - It is also called as Core Java.
 - For general purpose use on Desktop PC's, servers and similar devices.
2. Java EE
 - Java Platform Enterprise Edition.

- It is also called as advanced Java / enterprise java / web java.
- Java SE plus various API's which are useful client-server applications.

3. Java ME

- Java Platform Micro Edition.
- Specifies several different sets of libraries for devices with limited storage, display, and power capacities.
- It is often used to develop applications for mobile devices, PDAs, TV set-top boxes and printers.

4. Java Card

- A technology that allows small Java-based applications (applets) to be run securely on smart cards and similar small-memory devices.

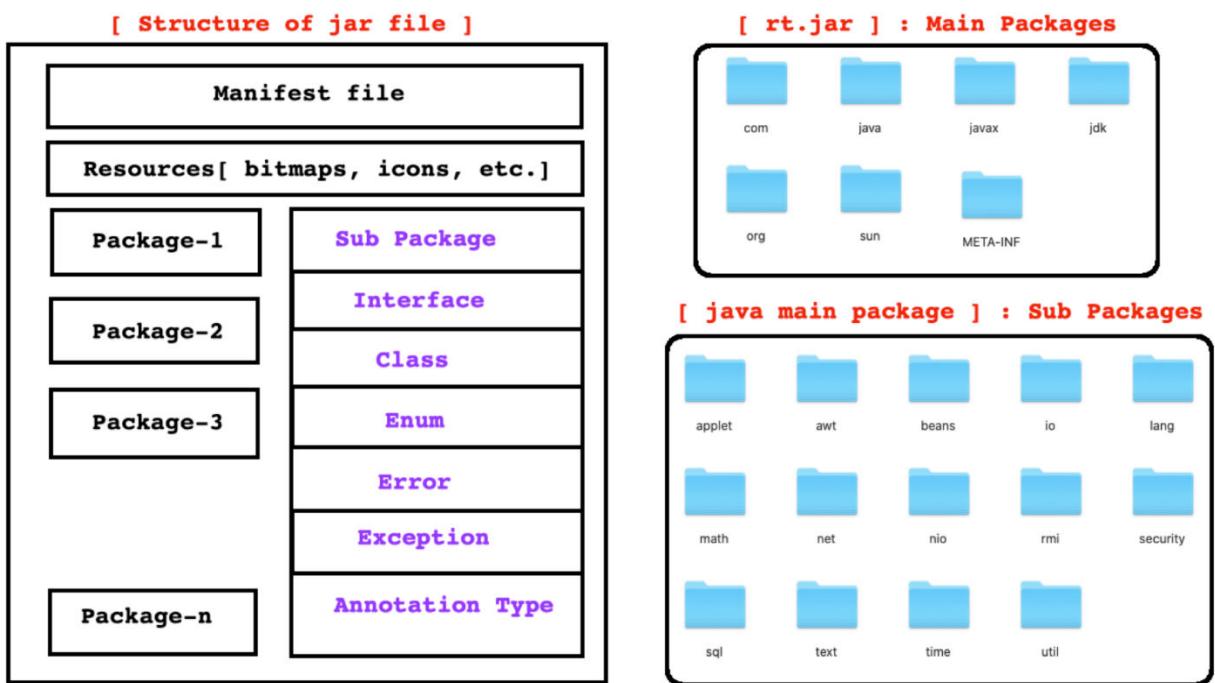
Components Of Java Platform:-

- **SDK** = Development Tools + Documentation + Libraries + Runtime Environment.
- **JDK** = Java Development Tools + Java Docs + rt.jar + JVM.
 - JDK : Java Development Kit.
 - It is a software, that need to be install on developers machine.
 - We can download it from oracle official website.
- **JDK** = Java Development Tools + Java Docs + JRE[rt.jar + JVM].
- **JRE** : Java Runtime Environment.
 - rt.jar and JVM are integrated part of JRE.
 - JRE is a software which comes with JDK. We can also download it separately.
 - To deploy application, we should install it on client's machine.
- rt.jar file contains core Java API in compiled form.
- **JVM** : An engine, which manages execution of Java application.

JDK installation Directory Structure:-

- 1.bin :- it contains java lang tools Ex. javac ,javah java etc
- 2.include :-JNI stands for java native interface.
- 3.lib :it contain library file which is required use java lang tool
- 4.src :-if we extract src.zip then src directory gets created
- 5.docs :-it contains documentation of java API
- 6.jre :-it contains JVM and rt.jar file
- 7.db :-it contains database specific files(driver ,sample,database)

Classification of Package and Types



Java terminologies:-

C++ terminologies	Java terminologies
Class	Class
Data member	Field
Member function	Method
Object	Instance
Pointer	Reference
Access specifier	Access Modifier
Base class	Super class
Derived class	Sub class
Derived from	Extended from
Runtime Polymorphism	Dynamic method dispatch

Note:-

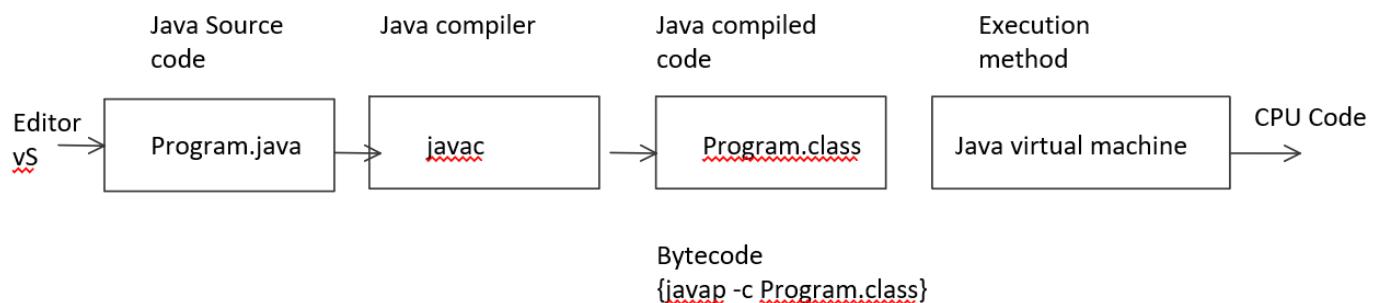
1. Jis class ka object create kr sakte hai use hum **concrete class** bolte hai.
2. jis class ka object create nhi kr sakte hai use hum **abstract class** bolte hai.
3. jis method ko body hoti hai use hum **concrete method** bolte hai
4. jis method ko body nhi hoti hai use hum **abstract method** bolte hai.

- 5.jis class ka child class nhi bana sakte use final class bolte hai
 - 6.jis method ko hum override nhi kr sakte aise method ko final method bolte hai.
- 1.A class from which, we can create object is called **concrete class**.
 - 2.A class from which, we can not create object is called **abstract class**.
 - 3.A method of class, which is having body is called **concreate method**
 - 4.A method of class, which do not have body is called **abstract method**.
 - 5.A class, that we can not extend is called **final class**
 - 6.A method that we can not redefine/override it inside sub class is called **final method**.

```
class Program_1 {  
    //Entry point method  
    public static void main(String args[])  
    {  
        //Printf("Hello world");           in C  
        //Std::cout<<"Hello world"<<std::endl;   in C++  
        System.out.println("Hello world");      //in Java  
  
        //system is final class declared in java.lang package  
        //out is public static final field of system class.Data type of  
out is                java.io.printstream.  
        //Println ka return type void hai  
        //public void println() is a non static method of  
java.io.printstream class.  
        //in java the return type of main is must be void.  
        //rm *class  is used to remove the class file  
        //By using Java D compiler you can convert .class file into java  
file.  
    }  
}
```

Compilation:- javac program_1.java
Execution:- java Program_1

Flow of Execution program:-



Component of JVM

1. ClassLoader Subsystem

- a. Bootstrap class loader
- b. Extension class loader
- c. Application class loader
- d. User defined class loader

2. Runtime Data Area

- a. Method Area
- b. Heap
- c. Java stack
- d. Pc register
- e. Native method stack

3. Execution Engine

- a. Interpreter
- b. Just in time compiler
- c. garbage collector

→ The Bytecode is object oriented assembly lang code design for JVM
→ If we create blank java file then it wont give me compile time error. But runtime error gave me Error: could not find or load main class program. calling main method is job of JVM not complier.

→ public is access modifier in java.

```
class Program_2 {  
public static void main(String[] args) {  
    System.out.println("Hello world");  
}  
}
```

→ It give me Error: Main method not found in class Program_2, please define the main method as:

```
class Program_2 {  
private static void main(String[] args) {  
    System.out.println("Hello world");  
}  
}//Error  
  
class Program_2 {  
protected static void main(String[] args) {  
    System.out.println("Hello world");  
}  
}//Error
```

→ main method mai access modifier compulsory public hi hona chahiye.

→ main method compulsory static hi hona chahiye.

```
class Program_2 {  
public void main(String[] args) {  
    System.out.println("Hello world");  
}  
}//Error
```

→ main method se compulsory return type void hi hona chahiye.

```
class Program_2 {  
public static main(String[] args) {  
    System.out.println("Hello world");  
}  
}//Error
```

→ main ka naam lower case mai hi hona chahiye.

```
class Program_2 {  
public static void MAIN(String[] args) {  
    System.out.println("Hello world");  
}  
} //Error
```

→ only one type in main method i.e String but args name you can give anything.

```
class Program_2 {  
public static void MAIN(int args, String[] args) {  
    System.out.println("Hello world");  
}  
} //Error
```

```
class Program_2 {  
    public static void main(String[] Ashok) {  
        System.out.println("Hello world");  
    }  
} //No error
```

Program:-

```
class A{  
    public static void main(String[] args) {  
        System.out.println("A.main");  
    }  
}  
class B{  
    public static void main(String[] args) {  
        System.out.println("B.main");  
    }  
}  
class Program_3 {  
    public static void main(String[] args) {  
        System.out.println("Program.main");  
    }  
}  
-->yaha pr three class likhe hai aur three main hai .class file pr  
particular class execute kr skte hai isame.
```

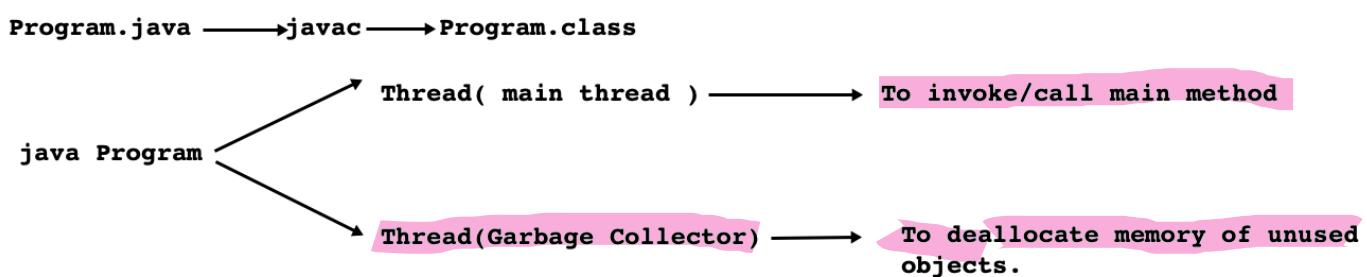
Q1. Can we defined two main method in class?

Ans:- no we can not defined more than one main method ,but we can
overload main method

```
class Program_4 {  
//Main method overloading  
    public static void main() {  
        System.out.println("Program.main");  
    }  
    public static void main(String[] args) {  
        System.out.println("Program.main(args)");  
        Program_4.main();  
    }  
}
```

→java mai two type ke thread hai first is for to invoke/call main
method

→Aur dusra thread to deallocate memory of unused objects.



- **Syntax:**

1. public static void main(String[] args)
2. public static void main(String... args)

- Java compiler do not check/invoke main method. **JVM invoke main method.**

- When we start execution of Java application then **JVM starts execution of two threads:**

1. **Main thread** : responsible for invoking main method.

2. **Garbage Collector** : responsible for deallocating memory of unused object.

- **We can overload main method in Java.**

- We can define main method per class. But only one main can be considered as entry point method.

Comments

- Comments should be used to give overviews of code and provide additional information that is not readily available in the code itself. Comments should contain only information that is relevant to reading and understanding the program.

- **Types of Comments:**

- **Implementation Comment**:- Implementation comments are those found in C++, which are delimited by /*...*/, and //.

1. **Single-Line Comment**

2. **Block Comment**(also called as **multiline comment**)

- **Documentation Comment** :- Documentation comments (known as "doc comments") are Java-only, and are delimited by /**...*/.
- Doc comments can be extracted to HTML files using the javadoc tool.

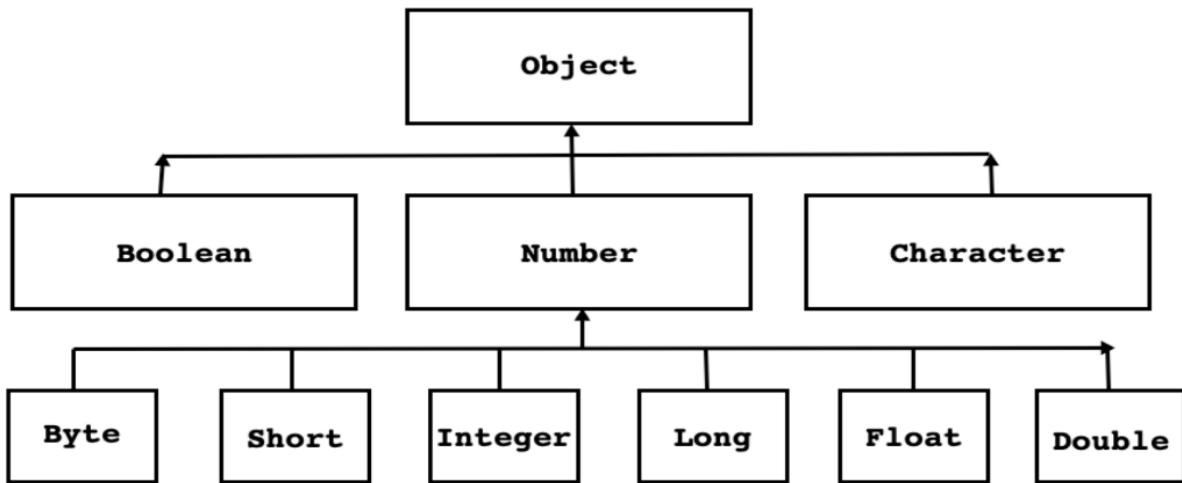
Data Types:-

- Data type of any variable decide following things:
 1. **Memory:** How much memory is required to store the data.
 2. **Nature:** Which kind of data is allowed to store inside memory.
 3. **Operation:** Which operations are allowed to perform on the data stored in memory.
 4. **Range:** Set of values that we can store inside memory.
- The Java programming language is a statically typed language, which means that every variable and every expression has a type that is known at compile time.
- o Types of data type:
 1. **Primitive type(also called as value type)**
 - boolean type
 - Numeric type
 1. **Integral types**(byte, char, short, int, long)
 2. **Floating point types**(float, double)
 - 2. **Non primitive type(also called as reference type)**
 - **Interface, Class, Type variable, Array**

Wrapper Class

- In Java, **primitive types are not classes**. But for **every primitive type, Java has defined a class**. It is called **wrapper class**.
 - All wrapper classes **are final**.
 - All wrapper classes are declared **in java.lang package**.
 - Uses of Wrapper class
 1. To parse string(i.e. to convert state of string into numeric type).
 2. To store value of primitive type into instance of generic class, type argument must be wrapper class.
 - Stack stk = new Stack(); //Not OK
 - Stack stk = new Stack(); //OK
- string convert into interger then used wrapper class
integer.parseInt();

```
int number1=Integer.parseInt("123");
float number2=Float.parseFloat("123.45f");
Double number3=Double.parseDouble("123.45f");
```



Sr.No.	Primitive Type	Size[In Bytes]	Default Value[For Field]	Wrapper Class
1	boolean	Isn't specified	FALSE	Boolean
2	byte	1	0	Byte
3	char	2	\u0000	Character
4	short	2	0	Short
5	int	4	0	Integer
6	float	4	0.0f	Float
7	double	8	0.0d	Double
8	long	8	0L	Long

```

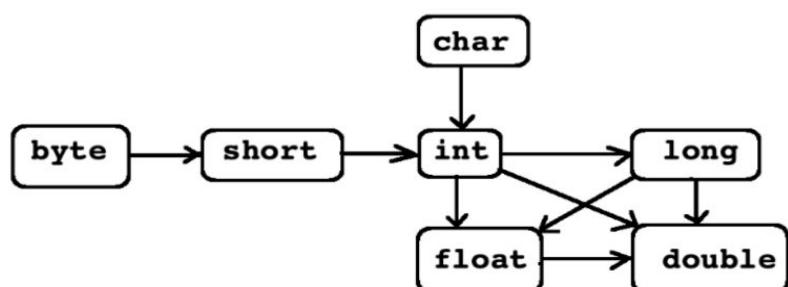
class Program{
    public static void main1(String[] args) {
        //Java is statically type checked language
        int num1 = 10;
        System.out.println( num1 );
    }
    public static void main2(String[] args) {
        int num1 = 10;
        //printf( "Num1 %d\n",num1); //in C
        //std::cout << "Num1 : " << num1 << std::endl; in C++
        System.out.println("Num1 : "+num1);
    }
    public static void main(String[] args) {
        //int num1;
        //System.out.println( num1 ); //error: variable num1 might
not have been initialized
    }
}
  
```

```
//int num1 = 10; //Initialization  
//System.out.println( num1 ); //OK  
  
int num1;  
num1 = 10; //Assignment  
System.out.println( num1 ); //OK  
}  
}  
  
-----  
  
class Program{  
  
    public static void main1(String[] args) {  
        //int num1; //declaration of Num1  
        //During declaration, process of storing values inside variable is  
        called initialization  
        int num1 = 10; //Initialization  
        int num2 = num1; //Initialization data type hai tho Initialization  
                        // num2=num1 this is assignment  
        System.out.println("Num2 : "+num2);  
    }  
}
```

Widening

- Process of converting value of variable of **narrower type** into **wider type** is called **widening**.
- In case of widening, **explicit type casting is optional**.

```
class Program{  
    public static void main(String[] args) {  
        int num1 = 10; //Initialization  
        //double num2 = (double)num1; //Widening //OK:10.0 typecasting  
        double num2 = num1; //Widening //OK:10.0  
        System.out.println("Num2 : "+num2);  
    }  
}
```

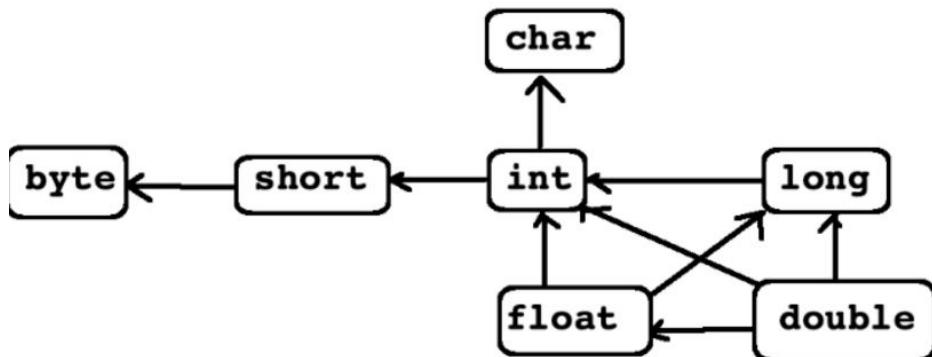


Widening Conversion

Narrowing

- Process of converting value of variable of wider type into narrower type is called narrowing.
- In case of narrowing, explicit type casting is mandatory.

```
class Program{  
    public static void main1(String[] args) {  
        double num1 = 10.5; //Initialization  
        double num2 = num1; //Initialization  
        System.out.println("Num2 : "+num2);  
    }  
    public static void main(String[] args) {  
        double num1 = 10.5; //Initialization  
        //int num2 = num1; //error: incompatible types: possible lossy  
conversion from double to int  
        int num2 = (int)num1; //Narrowing //OK : 10  
        System.out.println("Num2 : "+num2);  
    }  
}
```



Narrowing Conversion.

Note:-

```
//Static methods are designed to call on classname  
    //Example : ClassName.staticMethod( );  
//Non static methods are designed to call on instance/object  
    //Example : objectRef.nonStaticMethod( );
```

Unboxing :-

- Process of converting value of variable of non primitive type into primitive type is called unboxing.
- If string does not contain parseable numeric value then parseXXX() method throws NumberFormatException.

```
class Program{  
    public static void main1(String[] args) {  
        /*  
         Employee emp;           //Static memory allocation  
         Employee *ptr = new Employee(); //Dynamic memory allocation  
        */  
        //String is a class declared in java.lang package.  
  
        String s1 = new String("SunBeam"); //OK  
        System.out.println(s1);  
  
        String s2 = "Pune"; //OK  
        System.out.println(s2);  
    }  
  
    public static void main2(String[] args) {  
        String str = "123"; //String => Non primitive / Reference type  
        //int number = (int)str; //Not OK  
        int number = Integer.parseInt(str); //UnBoxing  
        System.out.println("Number : "+number);  
    }  
    public static void main3(String[] args) {  
        String str = "123.45f";  
        float number = Float.parseFloat(str); //UnBoxing  
        System.out.println("Number : "+number);  
    }  
    public static void main4(String[] args) {  
        String str = "123.45d";  
        double number = Double.parseDouble(str); //UnBoxing  
        System.out.println("Number : "+number);  
    }  
    public static void main5(String[] args) {  
        String str = "A1B2C3"; //it give error  
        int number = Integer.parseInt(str);  
        //UnBoxing : java.lang.NumberFormatException  
        System.out.println("Number : "+number);  
    }  
}
```

```
//NumberFormatException - if the string does not contain a  
parsable integer.  
}  
public static void main(String[] args) {  
    String str = "A";  
    char ch = str.charAt(0);  
    System.out.println("Char : " + ch);  
}  
}
```

charAt

```
public char charAt(int index)
```

Returns the char value at the specified index. An index ranges from 0 to length() - 1. The first char value of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

If the char value specified by the index is a surrogate, the surrogate value is returned.

Specified by:

charAt in interface CharSequence

Parameters:

index - the index of the char value.

Returns:

the char value at the specified index of this string. The first char value is at index 0.

Throws:

IndexOutOfBoundsException - if the index argument is negative or not less than the length of this string.

parseInt

```
public static int parseInt(String s)  
    throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign '-' ('\u002D') to indicate a negative value or an ASCII plus sign '+' ('\u002B') to indicate a positive value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the parseInt(java.lang.String, int) method.

Parameters:

s - a String containing the int representation to be parsed

Returns:

the integer value represented by the argument in decimal.

Throws:

NumberFormatException - if the string does not contain a parsable integer.

Boxing

- Process of converting value of variable of primitive type into not primitive type is called boxing

```
class Program{  
    //string is non-primitive & int is primitive  
    public static void main(String[] args) {  
        int number = 10; //primitive to non-primitive is known as boxing  
        String str = Integer.toString(number); //Boxing  
        System.out.println("Number : " + str);  
    }  
}
```

```
public static void main2(String[] args) {
    float number = 10.5f;
    String str = Float.toString(number); //Boxing
    System.out.println("Number : "+str);
}

public static void main3(String[] args) {
    double number = 20.5d;
    String str = Double.toString(number); //Boxing
    System.out.println("Number : "+str);
}

public static void main1(String[] args) {
    String str;

    int num1 = 10;
    str = String.valueOf(num1); //Boxing
    System.out.println("Num1 : "+str);

    float num2 = 20.5f;
    str = String.valueOf(num2); //Boxing
    System.out.println("Num2 : "+str);

    double num3 = 30.5d;
    str = String.valueOf(num3); //Boxing
    System.out.println("Num3 : "+str);
}

//Is program mai hum primitive to non primitive convert kar rahe
isliye Integer.toString,Float.toString etc kar skte hai par sabke
liye ek hi method chahiye tho fir valueof method use kr sakte hai.
```

Methods of `java.lang.String class.`

static String	<code>valueOf(boolean b)</code> Returns the string representation of the boolean argument.
static String	<code>valueOf(char c)</code> Returns the string representation of the char argument.
static String	<code>valueOf(char[] data)</code> Returns the string representation of the char array argument.
static String	<code>valueOf(char[] data, int offset, int count)</code> Returns the string representation of a specific subarray of the char array argument.
static String	<code>valueOf(double d)</code> Returns the string representation of the double argument.
static String	<code>valueOf(float f)</code> Returns the string representation of the float argument.
static String	<code>valueOf(int i)</code> Returns the string representation of the int argument.
static String	<code>valueOf(long l)</code> Returns the string representation of the long argument.
static String	<code>valueOf(Object obj)</code> Returns the string representation of the Object argument.

Core Java Notes By ASHOK PATE

toString

```
public String toString()
```

Returns a String object representing this Integer's value. The value is converted to signed decimal representation and returned as a string, exactly as if the integer value were given as an argument to the `toString(int)` method.

Overrides:

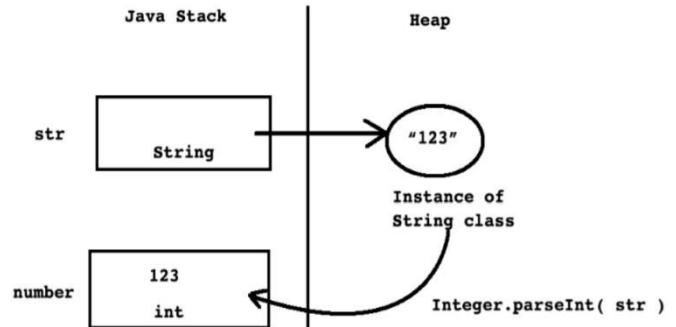
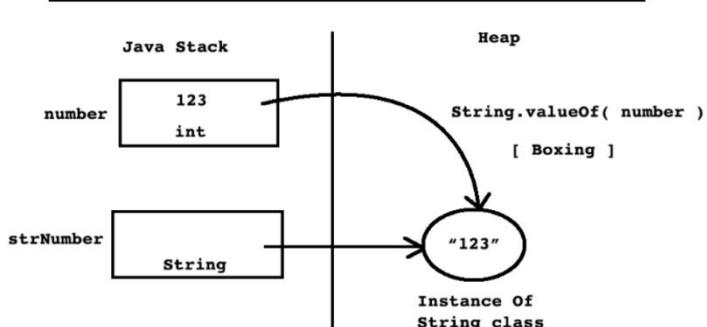
`toString` in class `Object`

Returns:

a string representation of the value of this object **in base 10.**

```
int number = 123;
String strNumber = String.valueOf( number ); //Boxing
```

```
String str = "123";
int number = Integer.parseInt( str ); //UnBoxing
```



```
class Program{
    public static void sum( int a, int b ){ //a, b  => Method Parameters / Parameters
        int result = num1 + num2;
        System.out.println("Result : "+result);
    }
    public static void main(String[] args) { //args => Method Parameter / Parameter
        Program.sum( 10, 20 ); //10, 20  => Method Arguments / Assignments
    }
}
```

```
int x = 10, y = 20;
Program.sum( x, y ); //x, y  => Method Arguments / Assignments
}
```

Command line arguments:-

```
class Program{
    public static void main1(String[] args) {
        String message = args[ 0 ];
        System.out.println(message);
```

```

    }
    public static void main2(String[] args) {
        //int num1 = args[ 0 ]; //NOT Ok //error: incompatible types:
String cannot be converted to int
        int num1 = Integer.parseInt(args[0]);

        //int num2 = args[ 1 ]; //NOT OK //error: incompatible types:
String cannot be converted to int
        int num2 = Integer.parseInt(args[1]);

        int result = num1 + num2;
        System.out.println("Result : "+result);
    }
    public static void main(String[] args) {
        int num1 = Integer.parseInt(args[0]);

        float num2 = Float.parseFloat(args[1]);

        double num3 = Double.parseDouble(args[2]);

        double result = num1 + num2 + num3;

        System.out.println("Result : "+result);
    }
}
//args ka type non primitive hai.so parsing karna padenga.
/*
argc: contains argument count passed from terminal
argv: array of string,contains address of arguments passed from cmd line
*/
//=> javac program.java //program classs
//=>java program           //Arrayindexoutofboundexception
//if you do not passed the argument then this error occurs
-----
```

Println & Print & Printf method:-

```

//println is non static overloaded method of java.io.printstream class
//and the return type is void
//print is non static overloaded method of java.io.printstream class
//and the return type is void
//printf is non static overloaded method of java.io.printstream class
//return type is printstream.
```

```
class Program{
    public static void main1(String[] args) {
        System.out.print("Hello"); //Helloworld!! ->all print in one
line
        System.out.print("World");
        System.out.print("!!");
        System.out.print("\n");
    }
    public static void main2(String[] args) {
        System.out.println("Hello"); //Hello
        System.out.println("World"); //World
        System.out.println("!!"); //!!
        System.out.println("\n"); //print in different line
    }
    public static void main(String[] args) {
        String name1 = "Amit Pol";
        int empid1 = 1122;
        float salary1 = 125345.50f;
        //System.out.println(name1+" "+empid1+" "+salary1);
        System.out.printf("%-20s%-5d%-10.2f\n", name1, empid1, salary1);

        String name2 = "Mukesh Salunkhe";
        int empid2 = 714;
        float salary2 = 37500.45f;
        //System.out.println(name2+" "+empid2+" "+salary2);
        System.out.printf("%-20s%-5d%-10.2f\n", name2, empid2, salary2);
    }
}
```

+ **println** is non static overloaded method of **java.io.PrintStream class.**

void	println() Terminates the current line by writing the line separator string.
void	println(boolean x) Prints a boolean and then terminate the line.
void	println(char x) Prints a character and then terminate the line.
void	println(char[] x) Prints an array of characters and then terminate the line.
void	println(double x) Prints a double and then terminate the line.
void	println(float x) Prints a float and then terminate the line.
void	println(int x) Prints an integer and then terminate the line.
void	println(long x) Prints a long and then terminate the line.
void	println(Object x) Prints an Object and then terminate the line.
void	println(String x) Prints a String and then terminate the line.

+ print is non static overloaded method of java.io.PrintStream class

void	print(boolean b) Prints a boolean value.
void	print(char c) Prints a character.
void	print(char[] s) Prints an array of characters.
void	print(double d) Prints a double-precision floating-point number.
void	print(float f) Prints a floating-point number.
void	print(int i) Prints an integer.
void	print(long l) Prints a long integer.
void	print(Object obj) Prints an object.
void	print(String s) Prints a string.

+ printf is non static overloaded method of java.io.PrintStream class.

PrintStream	printf(Locale l, String format, Object... args) A convenience method to write a formatted string to this output stream using the specified format string and arguments.
PrintStream	printf(String format, Object... args) A convenience method to write a formatted string to this output stream using the specified format string and arguments.

Console class:-

```
import java.io.Console;
class Program{
    public static void main(String[] args) {
        //java.io.Console console = System.console();
        Console console = System.console();
        System.out.print("Enter name : ");
        String name = console.readLine();
        System.out.print("Enter empid : ");
        int empid = Integer.parseInt(console.readLine());
        System.out.print("Enter salary : ");
        float salary = Float.parseFloat(console.readLine());

        System.out.println("Name      : "+name);
        System.out.println("Empid     : "+empid);
        System.out.println("Salary    : "+salary);
    }
}
//console is class declare in console.io is pacakage
//this code does not work in Eclipse.
```

Core Java Notes By ASHOK PATE

readLine

```
public String readLine()
```

Reads a single line of text from the console.

Returns:

A string containing the line read from the console, not including any line-termination characters, or null if an end of stream has been reached.

Throws:

IOException - If an I/O error occurs.

Scanner class:-

```
import java.util.Scanner;
class Program{
    //System.in  => Keyboard
    //System.out => Monitor
    //System.err => Error Stream => Monitor
    public static void main(String[] args) {
        //java.util.Scanner sc = new java.util.Scanner( System.in );
        Scanner sc = new Scanner( System.in );
        System.out.print("Enter name : ");
        String name = sc.nextLine();
        System.out.print("Enter empid : ");
        int empid = sc.nextInt();
        System.out.print("Enter salary : ");
        float salary = sc.nextFloat();

        System.out.println("Name      : " +name);
        System.out.println("Empid     : " +empid);
        System.out.println("Salary    : " +salary);
    }
}
```

+ Methods of java.util.Scanner class

boolean	<code>nextBoolean()</code>	Scans the next token of the input into a boolean value and returns that value.
byte	<code>nextByte()</code>	Scans the next token of the input as a byte.
byte	<code>nextByte(int radix)</code>	Scans the next token of the input as a byte.
double	<code>nextDouble()</code>	Scans the next token of the input as a double.
float	<code>nextFloat()</code>	Scans the next token of the input as a float.
int	<code>nextInt()</code>	Scans the next token of the input as an int.
int	<code>nextInt(int radix)</code>	Scans the next token of the input as an int.
String	<code>nextLine()</code>	Advances this scanner past the current line and returns the input that was skipped.
long	<code>nextLong()</code>	Scans the next token of the input as a long.
long	<code>nextLong(int radix)</code>	Scans the next token of the input as a long.
short	<code>nextShort()</code>	Scans the next token of the input as a short.
short	<code>nextShort(int radix)</code>	Scans the next token of the input as a short.

- Stream is an abstraction(object) which either produce (write) / consume (read) information from source to destination.
 - Standard stream objects of Java which is associated with console:
 1. System.in
 - It represents keyboard.
 2. System.out
 - It represents Monitor.
 3. System.err
 - Error stream which represents Monitor.
-

jOptionPane :-

```
//jOptionPane is class include in jvac.swing pacakge
//mostly used for GUI
import javax.swing.JOptionPane;
class Program{
    public static void main(String[] args) {
        String name = JOptionPane.showInputDialog("Enter name.");
        int empid = Integer.parseInt(JOptionPane.showInputDialog("Enter
empid."));
        float salary = Float.parseFloat(JOptionPane.showInputDialog("Enter
salary."));

        System.out.println("Name : "+name);
        System.out.println("Empid : "+empid);
        System.out.println("Salary : "+salary);
    }
}
```

showInputDialog

```
public static String showInputDialog(Component parentComponent,
                                     Object message,
                                     Object initialValue)
```

Shows a question-message dialog requesting input from the user and parented to parentComponent. The input value will be initialized to initialValue. The dialog is displayed on top of the Component's frame, and is usually positioned below the Component.

Parameters:

parentComponent - the parent Component for the dialog

message - the Object to display

initialSelectionValue - the value used to initialize the input field

Since:

1.4

BufferedReader:-

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
class Program{
    public static void main(String[] args) throws Exception {
        BufferedReader reader = new BufferedReader(new InputStreamReader(
System.in));

        System.out.print("Enter Name : ");
        String name = reader.readLine();
        System.out.print("Enter Empid : ");
        int empid = Integer.parseInt(reader.readLine());
        System.out.print("Enter salary : ");
        float salary = Float.parseFloat(reader.readLine());

        System.out.println("Name : "+name);
        System.out.println("Empid : "+empid);
        System.out.println("Salary : "+salary);
    }
}
/*error: unreported exception IOException; must be caught or declared to
be thrown
    int empid = Integer.parseInt(reader.readLine());
```

-->to avoid this error give throws Exception after args.

*/

readLine

```
public String readLine()
    throws IOException
```

Reads a line of text. A line is considered to be terminated by any one of a line feed ('\n'), a carriage return ('\r'), or a carriage return followed immediately by a linefeed.

Returns:

A String containing the contents of the line, not including any line-termination characters, or null if the end of the stream has been reached

Throws:

IOException - If an I/O error occurs

See Also:

Files.readAllLines(java.nio.file.Path, java.nio.charset.Charset)

Class:-

- Consider following examples:
 1. day, month, year - related to - Date
 2. hour, minute, second - related to - Time
 3. red, green, blue - related to Color
 4. real, imag - related to - Complex
 5. xPosition, yPosition - related to Point
 6. number, type, balance - related to Account
 7. name, id, salary - related to Employee
- If we want to group related data elements together then we should use/define class in Java.
- class is a non primitive/reference type in Java.
- If we want create object-instance of a class then it is mandatory to use new operator.
- If we create instance using new operator then it gets space on heap section.
- Only fields of the get space once per instance according to order of their declaration inside class.

```
//day, month, year are realated to Date
```

```
class Date{  
    //Variable declared inside class is called field  
    //int day, month, year;    //OK  
    int day;      //range 1 to 31  
    int month;    // Range 1 to 12  
    int year;     // Range greater than 1900  
}
```

```
//red, green, blue are related to Color
```

```
class Color{  
    int red;  
    int green;  
    int blue;  
}
```

```
//xPosition, yPosition related to Point
```

```
class Point{  
    int xPosition;  
    int yPosition;  
}
```

```
//real, imag related to Complex
```

```
class Complex{  
    int real;  
    int imag;
```

```
}

//name, rollNumber, marks are realated to Student
class Student{
    String name;
    int rollNumber;
    float marks;
}

//name, empid, salary are realated to Employee
class Employee{
    String name;
    int empid;
    float salary;
}

import java.util.Scanner;
class Program{
    public static void main(String[] args) {
        Scanner sc = new Scanner( System.in );

        System.out.print("Name : ");
        String name = sc.nextLine();
        System.out.print("Empid : ");
        int empid = sc.nextInt();
        System.out.print("Salary : ");
        float salary = sc.nextFloat();

        System.out.println(name+" "+empid+" "+salary);
    }
}

//only one employee record accepted .this is problem
//i want more employee record together
//thats why we used class in next program.
```

Reference/Instance:-

- **Field**
 - A variable declared inside class / class scope is called a field.
 - Field is also called as **attribute** or **property**.
- **Method**
 - A function implemented inside class/class scope is called as method.
 - Method is also called as operation, behavior or message.
- **Class**

- Class is a collection of fields and methods.
- Class can contain
 - 1. Nested Type 2. Field 3. Constructor 4. Method
- Instance
 - In Java, Object is also called as instance.
 - Process of creating instance/object from a class is called as instantiation.
 - Every instance on heap section is anonymous.

```
import java.util.Scanner;
class Program{
    public static void main(String[] args) {
        Scanner sc = new Scanner( System.in );

        class Employee{
            String name;      //Field => Default value null
            int empid;        //Field => Default value 0
            float salary;     //Field => Default value 0.0f
        }

        // If we want to create instance( in C++ instance is nothing but
        object ) of a class then
        // we must must use new operator.

        //Employee emp;    //In Java, emp is not a object/instance. "emp"
        is object reference / reference.
        //new Employee( );    //Anonymous Instance in Java

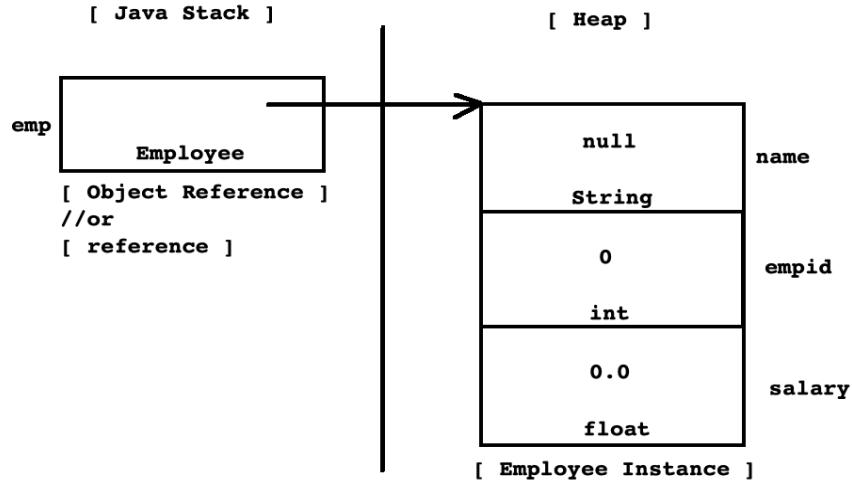
        //If we create instance using new operator then it gets space on
        Heap.
        //In C/C++/Java, if we allocate memory on Heap then it is
        considered as anonymous.

        // If we create instance of a class then:
            //1. Method parameter and method local variable do not get
            space inside it.
            //2. Methods(static/non static) /constructor defined inside
            class do not get space inside instance
            //3. Nested type( Interface/class/enum ) do not get space
            inside instance.
            //4. Static field do not get space instance instance.
            //5. Only non static field get space once per instance.
        // Since non static field get space once per instance, it is also
        called as instance variable.
```

```

        /* Employee emp;    //Object reference / reference
        emp = new Employee( ); //OK */
        //Employee emp = new Employee( ); //OK
    }
}

```



```

struct Employee *ptr = (struct Employee *)malloc( sizeof(struct Employee ) )

Employee *ptr = name Employee( );

Employee emp; //OK : reference declaration
emp = new Employee( ); //OK : Instantiation
//or
Employee emp = new Employee( );

+ Instantiation
//C
//1. struct TypeName identifier

//C++
//1. class TypeName identifier; //OK
//2. TypeName identifier; //OK
//3. TypeName *identifier = new TypeName( ); //OK

//Java
//1. TypeName identifier = new TypeName( );

```

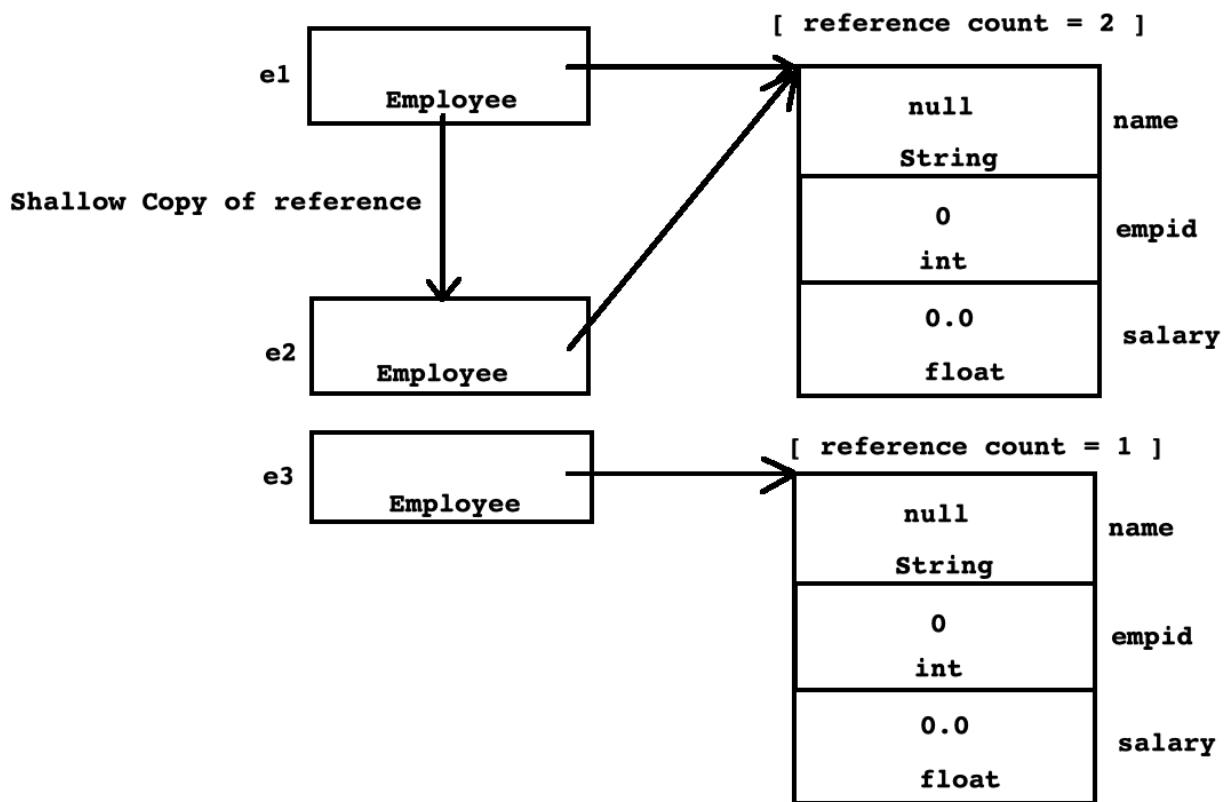
- Process of creating instance from a class is called instantiation.

- Example:

`Employee emp = new Employee(); //Instantiation of class Employee
* Here class Employee is instantiated.`

- A class from which we can create instance, is called concrete class.
- In other words, we can instantiate concrete class.

1. `Employee e1 = new Employee(); //Instance`
2. `Employee e2 = e1; //Shallow Copy copy of references.`
3. `Employee e3 = new Employee(); //Instance`



```

import java.util.Scanner;

class Program{
    public static void main(String[] args) {
        Scanner sc = new Scanner( System.in );
        class Employee{
            String name;      //Field => Default value null
            int empid;        //Field => Default value 0
            float salary;     //Field => Default value 0.0f
        }
        //Employee emp = new Employee();
        Employee emp;
        emp = new Employee();
        emp.name = "Sandeep";
        emp.empid = 33;
        emp.salary = 12345.50f;

        System.out.println("Name      : "+emp.name);
        System.out.println("Empid     : "+emp.empid);
        System.out.println("Salary    : "+emp.salary);
    }
}

```

```
}

-----
import java.util.Scanner;
class Program{
    public static void main(String[] args) {
        Scanner sc = new Scanner( System.in );
        class Employee{
            String name;      //Field => Default value null
            int empid;       //Field => Default value 0
            float salary;    //Field => Default value 0.0f
        }
        //Employee emp = new Employee();

        // Employee emp;//error: variable emp might not have been
        // initialized emp.name = "Sandeep";

        Employee emp;
        emp = new Employee();

        emp.name = "Sandeep";
        emp.empid = 33;
        emp.salary = 12345.50f;

        System.out.println("Name      : "+emp.name);
        System.out.println("Empid     : "+emp.empid);
        System.out.println("Salary    : "+emp.salary);
    }
}
```

This is not suitable method:-

```
import java.util.Scanner;
class Program{
    public static void main(String[] args) {
        //Scanner sc = new Scanner( System.in );
        class Employee{
            //Field / Attribute / Property
            String name;      //Field => Default value null
            int empid;       //Field => Default value 0
            float salary;    //Field => Default value 0.0f
        }

        Employee emp = new Employee();
```

```
//If we want to perform operation on instance then we should  
define method inside class.  
//Function implemented inside class is called method.  
  
/* System.out.print("Name : ");  
emp.name = sc.nextLine();  
System.out.print("Empid : ");  
emp.empid = sc.nextInt();  
System.out.print("Salary : ");  
emp.salary = sc.nextFloat(); */  
  
//process of calling method on instance is called as message  
passing  
  
/* System.out.println("Name : "+emp.name);  
System.out.println("Empid : "+emp.empid);  
System.out.println("Salary : "+emp.salary); */  
  
}  
}
```

Suitable method for above program:-

```
import java.util.Scanner;  
class Program{  
    public static void main(String[] args) {  
        //Scanner sc = new Scanner( System.in );  
        class Employee{  
            //Field / Attribute / Property  
            String name;      //Field => Default value null  
            int empid;        //Field => Default value 0  
            float salary;     //Field => Default value 0.0f  
  
            //Method / Operation / Behavior / Message  
            void acceptRecord( ){  
                Scanner sc = new Scanner( System.in );  
                System.out.print("Name : ");  
                name = sc.nextLine();  
                System.out.print("Empid : ");  
                empid = sc.nextInt();  
                System.out.print("Salary : ");  
                salary = sc.nextFloat();  
            }  
        }
```

```
        void printRecord( ){
            System.out.println("Name      :    "+name);
            System.out.println("Empid     :    "+empid);
            System.out.println("Salary    :    "+salary);
        }
    }

Employee emp = new Employee();
//If we want to perform operation on instance then we should
define method inside class.
//Function implemented inside class is called method.
//process of calling method on instance is called as message
passing

    emp.acceptRecord( );    //Message Passing
    //Here acceptRecord method is called on emp
    emp.printRecord( ); //Message Passing
    //Here printRecord method is called on emp.
}

```

Local Class:-

```
import java.util.Scanner;
class Program{ //Program.class
    public static void main(String[] args) {
        //Class is collection of fields and methods.
        /*
            In Java class can contain:
            1. Nested type( Interface/ Class / Enum )
            2. Fields( Static / non static )
            3. Methods( Static / non static/ final / abstract /
Synchronized )
            4. Constructor
        */
        //if we define class inside method then it is called method local
class
    class Employee{ //Program$1Employee.class
        String name;
        int empid;
        float salary;

        void acceptRecord( ){
            Scanner sc = new Scanner( System.in );

```

```
        System.out.print("Name : ");
        name = sc.nextLine();
        System.out.print("Empid : ");
        empid = sc.nextInt();
        System.out.print("Salary : ");
        salary = sc.nextFloat();
    }
    void printRecord( ){
        System.out.println("Name : "+name);
        System.out.println("Empid : "+empid);
        System.out.println("Salary : "+salary);
    }
}//End of class Employee
Employee emp = new Employee();
emp.acceptRecord( );
emp.printRecord( );
}
```

```
true : boolean status;

'A' : char ch;

123 : byte/sort/int/long num1;

123.45f : float num2;

123.45 : double num3;

"Java" : String str;

null : use to initialise reference variable.
```

- In C/C++ NULL is a macro which is designed to initialize pointer.
`#define NULL ((void*)0) //C
#define NULL 0 //C++`
 - Example
`int *ptr = NULL; //OK`
 - null is literal in Java, which is designed to initialise reference variable.
 - Example:
`int number = null; //Not OK : Compiler error`
- `Employee emp = null;`
`* emp => null reference variable`
`* emp => null object.`
- If reference contains null value means it doesn't contain reference of instance.

NullPointerException :-

```
import java.util.Scanner;
class Program{ //Program.class
    public static void main(String[] args) {
        //Non static field(s) declared inside class is called instance
variable
        //A method of a class which is designed to call on instance is
called instance method.
        class Employee{ //Concrete class
            String name; //instance variable
            int empid; //instance variable
            float salary; //instance variabl
            //A method of a class which is having body is called concrete
method
            //Concrete Method
            void acceptRecord( ){ //instance method
                Scanner sc = new Scanner( System.in );
                System.out.print("Name : ");
                name = sc.nextLine();
                System.out.print("Empid : ");
                empid = sc.nextInt();
                System.out.print("Salary : ");
                salary = sc.nextFloat();
            }
            //Concrete Method
            void printRecord( ){ //instance method
                System.out.println("Name : "+name);
                System.out.println("Empid : "+empid);
                System.out.println("Salary : "+salary);
            }
        }//End of class Employee
        //Employee emp = new Employee();
        Employee emp = null; //emp is null object
        //Instance methods are called on null object
        emp.acceptRecord( ); //java.lang.NullPointerException
        emp.printRecord( );
    /*

```

```
Exception in thread "main" java.lang.NullPointerException
at Program.main(Program.java:33)
*/
//Using null object, if we try to access any member of the class
then JVM throws NullPointerException
}

}
```

```
public class NullPointerException
extends RuntimeException
```

Thrown when an application attempts to use null in a case where an object is required. These include:

- Calling the instance method of a null object.
- Accessing or modifying the field of a null object.
- Taking the length of null as if it were an array.
- Accessing or modifying the slots of null as if it were an array.
- Throwing null as if it were a Throwable value.

Applications should throw instances of this class to indicate other illegal uses of the null object. NullPointerException objects may be constructed by the virtual machine as if suppression were disabled and/or the stack trace was not writable.

Since:

JDK1.0

See Also:

Serialized Form

To avoid NullPointerException Program :-

```
import java.util.Scanner;
class Program{ //Program.class
    //Employee e1;    //Error
    //Employee e2 = new Employee(); // Error
    public static void main(String[] args) {
        //We can not use reference or instance of local class outside
method.
        class Employee{ //Concrete class
            String name; //instance variable
            int empid; //instance variable
            float salary; //instance variable

            void acceptRecord( ){ //instance method
                Scanner sc = new Scanner( System.in );
                System.out.print("Name : ");
                name = sc.nextLine();
                System.out.print("Empid : ");
                empid = sc.nextInt();
            }
        }
    }
}
```

```
        System.out.print("Salary : ");
        salary = sc.nextFloat();
    }
//Concrete Method
void printRecord( ){ //instance method
    System.out.println("Name : "+name);
    System.out.println("Empid : "+empid);
    System.out.println("Salary : "+salary);
}
}//End of class Employee

/* Employee emp = null; //emp is null object
emp = new Employee( ); */

Employee emp = new Employee();
emp.acceptRecord( );
emp.printRecord( );
}
```

This reference:-

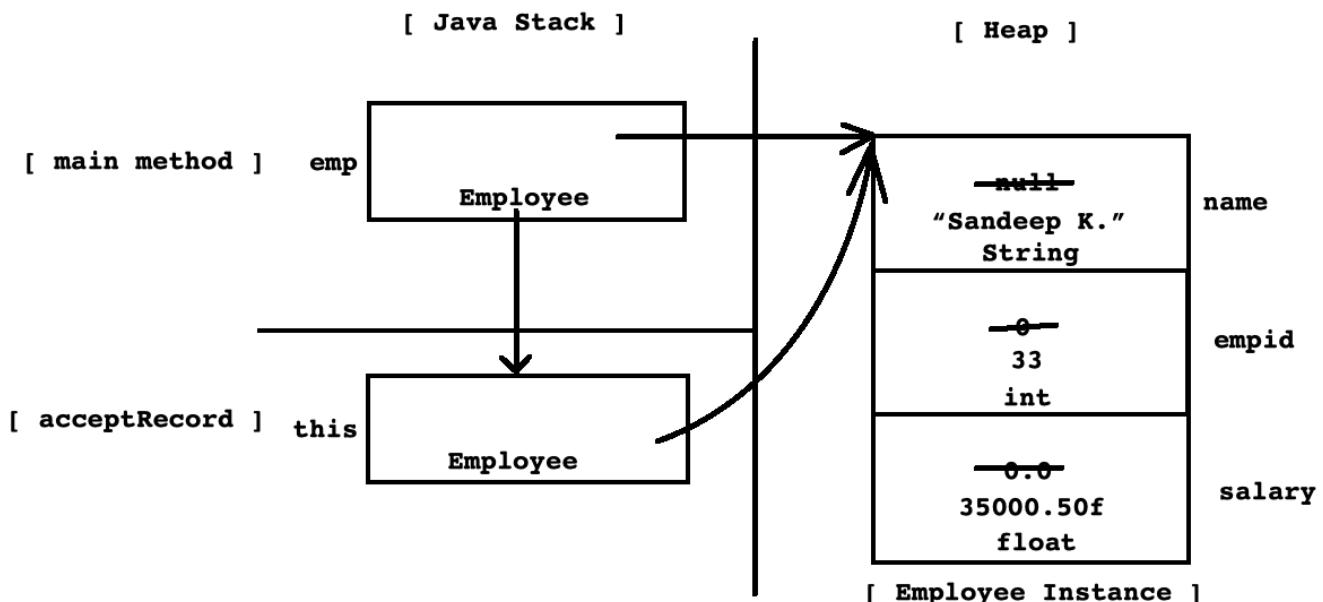
- If we call non static method on instance(actually object reference) then compiler implicitly pass, reference of current/calling instance as a argument to the method implicitly. To store reference of current/calling instance, compiler implicitly declare one reference as a parameter inside method. It is called this reference. • this is a keyword in Java which is designed to store reference of current/calling instance.
- Using this reference, non static fields and non static methods are communicating with each other. Hence this reference is considered as a link/connection between them.
- Definition
 - “this” is implicit reference variable that is available in every non static method of class which is used to store reference of current/calling instance.
- Inside method, to access members of same class, use this keyword is optional

Core Java Notes By ASHOK PATE

- If name of local variable/parameter and name of field is same then preference is always given to the local variable.

```
class Employee{  
    private String name;  
    private int empid;  
    private float salary;  
  
    public void initEmployee(String name, int empid, float salary ){  
        this.name = name;  
        this.empid = empid;  
        this.salary = salary;  
    }  
}
```

- Understood problem statement.
- According problem statement we decided classes and fields.
- Defined class and declared fields inside class.
- Created instance of a class. Here fields get space inside instance.
- To process state/value of the instance, called method on instance.
- Defined method inside class.



```
Import java.util.Scanner;  
  
class Program{ //Program.class  
    public static void main(String[] args) {  
  
        class Employee{  
            String name;  
            int empid;  
            float salary;
```

```
//this is implicit reference variable which is available in  
every non static method  
//of a class which is used to store reference of  
current/calling instance.  
//this is keyword in Java.  
//this is method parameter hence it doesn't get space inside  
inside.  
//It gets space once per method call.  
  
void acceptRecord( /* Employee this */ ){  
    Scanner sc = new Scanner( System.in );  
    System.out.print("Name : ");  
    //name = sc.nextLine();  
    this.name = sc.nextLine();  
    System.out.print("Empid : ");  
    //empid = sc.nextInt();  
    this.empid = sc.nextInt();  
    System.out.print("Salary : ");  
    //salary = sc.nextFloat();  
    this.salary = sc.nextFloat();  
}  
  
void printRecord( /* Employee this */ ){  
    //System.out.println("Name : "+name);  
    System.out.println("Name : "+this.name);  
    //System.out.println("Empid : "+empid);  
    System.out.println("Empid : "+this.empid);  
    //System.out.println("Salary : "+salary);  
    System.out.println("Salary : "+this.salary);  
}  
}  
  
Employee emp = new Employee();  
  
emp.acceptRecord( ); //emp.acceptRecord( emp );  
  
emp.printRecord( ); //emp.printRecord( emp );  
}  
}
```

Getters & Setters:-

→ Object ke ander ki salary asani se read kr pa rahe hai aur modify bhi kar pa rahe the upar wale program mai.pr object oriented dekhe tho aisa nhi hona chahiye is liye solution hai instance variable ko private kr do.
→ ab value directly print nhi honi chahiye method through hi hona chahiye.

→ sirf read krna hai change nhi krana tho getsalary method banayege.

→ SOP statement print on console but now we used getters they provide flexibility if you want print only name, only empid,only salary also then can print on console.

→ ab muje kuch change karna honga tho mai setter method use kr sakta hu.

→ setter method likhenge clas mai jakar.

```
import java.util.Scanner;
class Program{ //Program.class
    public static void main(String[] args) {

        class Employee{
            private String name;
            private int empid;
            private float salary; //Data hiding

            public String getName() {
                return this.name;
            }

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

            public int getEmpid() {
                return this.empid;
            }
            public void setEmpid(int empid) {
                this.empid = empid;
            }

            public float getSalary( ){
                return this.salary;
            }
        }
    }
}
```

```
public void setSalary(float salary) {
    this.salary = salary;
}

void acceptRecord( ){
    Scanner sc = new Scanner( System.in );
    System.out.print("Name : ");
    this.name = sc.nextLine();
    System.out.print("Empid : ");
    this.empid = sc.nextInt();
    System.out.print("Salary : ");
    this.salary = sc.nextFloat();
}

void printRecord( ){
    System.out.println("Name : "+this.name);
    System.out.println("Empid : "+this.empid);
    System.out.println("Salary : "+this.salary);
}
}

Employee emp = new Employee();
emp.acceptRecord( );      //Sandeep K 33. 35000.50
//emp.printRecord( );     //Sandeep K 33. 35000.50

emp.setName( "Sandeep Kulange" );
emp.setEmpid( 33 );
emp.setSalary( 25000.50f );

//String name = emp.getName();
//System.out.println("Name : "+name);
System.out.println("Name : "+emp.getName());
//ek line se kam ho raha tho two line kyu likhani

//int empid = emp.getEmpid();
//System.out.println("Empid : "+empid);
System.out.println("Empid : "+emp.getEmpid());

//float salary = emp.getSalary();
//System.out.println("Salary : "+salary);
System.out.println("Salary : "+emp.getSalary());
}

}
```

→ ab iss program mai user input lenge.

```
import java.util.Scanner;
class Program{ //Program.class
    public static void main(String[] args) {
        class Employee{
            private String name;
            private int empid;
            private float salary; //Data hiding

            public String getName() {
                return this.name;
            }
            public void setName(String name) {
                this.name = name;
            }
            public int getEmpid() {
                return this.empid;
            }
            public void setEmpid(int empid) {
                this.empid = empid;
            }
            public float getSalary( ){
                return this.salary;
            }
            public void setSalary(float salary) {
                this.salary = salary;
            }
        }
        Scanner sc = new Scanner( System.in );
        Employee emp = new Employee();
        System.out.print("Name : ");
        emp.setName( sc.nextLine() );
        System.out.print("Empid : ");
        emp.setEmpid( sc.nextInt() );
        System.out.print("Salary : ");
        emp.setSalary( sc.nextFloat() );

        System.out.println("Name : "+emp.getName());
        System.out.println("Empid : "+emp.getEmpid());
        System.out.println("Salary : "+emp.getSalary());
    }
}
```

Global Class :-

→ Ab muje iss program mai date program outside main used karna hai par local se kaam nhi chalenga isliye date class ko global mai likhana padenga.

→

```
import java.util.Scanner;
class Date{
    private int day;
    private int month;
    private int year;
    public int getDay() {
        return this.day;
    }
    public void setDay(int day) {
        this.day = day;
    }
    public int getMonth() {
        return this.month;
    }
    public void setMonth(int month) {
        this.month = month;
    }
    public int getYear() {
        return this.year;
    }
    public void setYear(int year) {
        this.year = year;
    }
}
class Program{
    public static void main(String[] args) {
        Date date = new Date();
        date.setDay(25);
        date.setMonth(03);
        date.setYear(2022);
        System.out.println(date.getDay()+" / "+date.getMonth()+" /
"+date.getYear());
    }
}
```

Constructor:-

```
import java.util.Scanner;
class Date{
    private int day;
    private int month;
    private int year;
    public void initDate( ){ //Constructor
        this.day = 11;
        this.month = 8;
        this.year = 2021;
    }
    public void acceptRecord( ){
        Scanner sc = new Scanner( System.in );
        System.out.print("Day :   ");
        this.day = sc.nextInt();
        System.out.print("Month :   ");
        this.month = sc.nextInt();
        System.out.print("Year :   ");
        this.year = sc.nextInt();
    }
    public void printRecord( ){
        System.out.println(this.day+ " / "+this.month+ " / "+this.year);
    }
}
class Program{
    public static void main(String[] args) {
        date = new Date( ); //0 / 0/ 0
        date.intiDate(); // 25/03/2022
        date.acceptRecord(); // 28/10/2022
        date.intiDate(); // 25/03/2022
        date.printRecord(); // 25/03/2022

    }
}
//Logically yaha galat ho raha hai initdate humne intialization ke likhe
//hai
//aur intilization one time hota hai aur intilization ek bar hona chahiye
//aur call bhi ek hi bar hona chahiye
//so this is not intialization
//now we change in next program then the intialization happend only once
```

```
//the change are class name is same as method name
//no return type
//when we create instance implicitly call ho rahi hai.

import java.util.Scanner;
class Date{
    private int day;
    private int month;
    private int year;
    public Date( ){ //Constructor
        this.day = 11;
        this.month = 8;
        this.year = 2021;
    }
    public void acceptRecord( ){
        Scanner sc = new Scanner( System.in );
        System.out.print("Day :   ");
        this.day = sc.nextInt();
        System.out.print("Month :   ");
        this.month = sc.nextInt();
        System.out.print("Year :   ");
        this.year = sc.nextInt();
    }
    public void printRecord( ){
        System.out.println(this.day+ " / "+this.month+ " / "+this.year);
    }
}
class Program{
    public static void main(String[] args) {
        Date date = null;
        date = new Date( );
        date.acceptRecord();
    }
}
```

- If we want to initialize instance then we should define constructor inside class.
- Constructor look like method but it is not considered as method.
- It is special because:
 1. Its name is same as class name.
 2. It doesn't have any return type.
 3. It is designed to call implicitly.

4. It gets called once per instance.

- We can not call constructor on instance explicitly Employee emp = new Employee(); emp.Employee(); //Not Ok

- Types of constructor:

1. Parameterless constructor

2. Parameterized constructor

3. Default constructor.

```
class Employee{  
    private String name;    //Fields  
    private int empid;  
    private float salary;  
    //parameterless constructor /zero argument constructor/user defined  
    default constructor  
    public Employee( ){  
        this.name = "";  
        this.empid = 0;  
        this.salary = 0;  
    }  
    //parameterized constructor  
    public Employee( String name, int empid, float salary){  
        this.name = name;  
        this.empid = empid;  
        this.salary = salary;  
    }  
    public void printRecord( ){  
        System.out.println("Name : "+this.name);  
        System.out.println("Empid : "+this.empid);  
        System.out.println("Salary : "+this.salary);  
    }  
}  
class Program{  
    public static void main(String[] args) {  
        Employee emp1 = new Employee();  
        //Here on instance parameterless constructor will call.  
        emp1.printRecord();  
        System.out.println("-----");  
  
        Employee emp2 = new Employee("Sandeep", 33, 12345.50f);  
        //Here on instance parameterized constructor will call.  
        emp2.printRecord( );  
    }  
}
```

Default Constructor:-

```
class Employee{  
    private String name;      //null  
    private int empid;        //0  
    private float salary;     //0.0  
    /*  
     * If we do not define any constructor inside class then compiler  
     * generates one constructor  
     * for the class by default. It is default constructor.  
     */  
    //Compiler generated default constructor  
    public Employee( ){ //Parameterless constructor  
        //Empty body  
    } /*  
    public void printRecord( ){  
        System.out.println("Name : "+this.name);  
        System.out.println("Empid : "+this.empid);  
        System.out.println("Salary : "+this.salary);  
    }  
}  
  
class Program{  
    public static void main(String[] args) {  
        Employee emp1 = new Employee( ); //OK  
        //Here on instance default constructor will call.  
        emp1.printRecord( );  
  
        Employee emp2 = new Employee( "Sandeep", 33, 45000.50f ); //NOT  
OK  
        //error: constructor Employee in class Employee cannot be applied  
        //to given types  
        //Compiler do not generate default parameterized constructor  
    }  
}
```

```
class Employee{  
    private String name;      //null  
    private int empid;        //0  
    private float salary;     //0.0  
  
    public Employee( String name, int empid, float salary ) {  
        this.name = name;  
        this.empid = empid;
```

```
this.salary = salary;
}
public void printRecord( ){
    System.out.println("Name : "+this.name);
    System.out.println("Empid : "+this.empid);
    System.out.println("Salary : "+this.salary);
}
}
class Program{
    public static void main(String[] args) {
        Employee emp1 = new Employee( );      //NOT OK : Compiler error
        emp1.printRecord( );

        Employee emp2 = new Employee( "Sandeep", 33, 45000.50f ); // OK
        emp2.printRecord( );
        //Here on instance parameterized constructor will call.
    }
}
```

Constructor Chaining:-

```
class Employee{
    private String name;      //null
    private int empid;        //0
    private float salary;     //0.0
    public Employee( ) {
        this( "Empty", -1, 85000.0f); //Constructor Chaining
    }
    /*
     --> To achieve reusability, we can call constructor from another
constructor.
     --> This process of calling constructor from another constructor is
called constructor chaining.
     --> For constructor chaining, we should use this statement.
     --> this statement must be first statement inside constructor body.
    */
    /*
     Advantages of reusability:
     1. We can reduce development time
     2. We can reduce development cost
     3. We can reduce developers effort.
    */
```

```
public Employee( String name, int empid, float salary ) {  
    this.name = name;  
    this.empid = empid;  
    this.salary = salary;  
}  
public void printRecord( ){  
    System.out.println("Name : "+this.name);  
    System.out.println("Empid : "+this.empid);  
    System.out.println("Salary : "+this.salary);  
}  
}  
class Program{  
    public static void main(String[] args) {  
        Employee emp = new Employee( ); //OK  
        emp.printRecord( );  
    }  
}
```

Coding and Naming convention:-

- + Coding / Naming Conventions
 - 1. Pascal coding convention
 - 2. Camel case coding convention
- + Pascal coding convention
 - Example
 - 1. System, String, Integer, Float, Double, Scanner
 - 2. BufferedReader, StringBuffer, StringBuilder, StringTokenizer
 - 3. InputStreamReader, OutputStreamWriter, NullPointerException
 - 4. IndexOutOfBoundsException
 - In this case, including first word, first character of each word must be in upper case.
 - We should use this convention for giving name to the:
 - 1. Types(Interface, class, enum, annotation)
 - 2. File name
- + Camel case coding convention
 - Example
 - 1. main(), run()
 - 2. parseInt(), acceptRecord(), printRecord()
 - 3. showInputDialog();
 - 4. addNumberOfDays();
 - In this case, excluding, first word, first character of each word must be in uppercase.
 - We should use this convention for giving name to the:
 - 1. Method parameter & method local variable
 - 2. fields and methods
 - 3. Object reference / reference

Coding Convention For Package Name & Final(const)variable & Enum:-

```
+ coding convention for package name.  
- generally package name is completely written in lowercase  
- Example:  
  1. java.lang  
  2. java.lang.reflect  
  3. java.util  
  4. com.sunbeaminfo.dac.test  
  5. com.mysql.cj.jdbc  
  
+ Coding convention for final(const) variable  
- Name of final variable should be in uppercase.  
- Example:  
  1. public static final int MIN_VALUE = -128  
  2. public static final int MAX_VALUE = 127  
  
+ Coding convention for enum  
enum Color{ //Pascal case convention  
    RED, GREEN, BLUE; //Enum constants  
}  
- Name of enum constant should be in uppercase.
```

Integer Class Constructor :-

```
class Program{  
    public static void main1(String[] args) {  
        //public Integer(int value)  
        Integer i1 = new Integer( 10 );  
        System.out.println(i1.toString());  
  
        int value = 20;  
        Integer i2 = new Integer( value );  
        System.out.println(i2.toString());  
    }  
    public static void main2(String[] args) {  
        //public Integer(String s)throws NumberFormatException  
        Integer i1 = new Integer("123");  
        System.out.println(i1.toString());  
  
        String s = "456";  
        Integer i2 = new Integer( s );  
        System.out.println(i2.toString());  
    }  
    public static void main3(String[] args) {  
        String s = "1a2b3c";  
        Integer i1 = new Integer( s ); //NumberFormatException
```

```
        System.out.println(i1.toString());
    }
    public static void main(String[] args) {
        Object obj1;
        Integer i1 = new Integer(); //error: no suitable constructor
found for Integer(no arguments)
        System.out.println(i1.toString());
    }
}
```

Integer

```
public Integer(int value)
```

Constructs a newly allocated Integer object that represents the specified int value.

Parameters:

value - the value to be represented by the Integer object.

Integer

```
public Integer(String s)
    throws NumberFormatException
```

Constructs a newly allocated Integer object that represents the int value indicated by the String parameter. The string is converted to an int value in exactly the manner used by the parseInt method for radix 10.

Parameters:

s - the String to be converted to an Integer.

Throws:

NumberFormatException - if the String does not contain a parsable integer.

See Also:

[parseInt\(java.lang.String, int\)](#)

Object Class:-

- It is a non final and concrete class declared in java.lang package.
- In java all the classes(not interfaces)are directly or indirectly extended from java.lang.Object class.
- In other words, java.lang.Object class is ultimate base class/super cosmic base class/root of Java class hierarchy.

- Object class do not extend any class or implement any interface.
- It doesn't contain nested type as well as field.
- It contains default constructor.
 - Object o = new Object("Hello"); //Not OK
 - Object o = new Object(); //OK
- Object class contains 11 methods.
- Consider the following code:
- In above code, java.lang.Object is direct super class of class Person.

```
class Person{  
}  
  
class Employee extends Person{  
}
```

- In case class Employee, class Person is direct super class and class Object is indirect super class.

- rt.jar file contains core Java API in compiled form.
- java.lang package contains fundamental classes of core Java.
- Object is a class declared in java.lang package(java.lang.Object).
- java.lang.Object class do not extend any class or do not implement any interface.
- In Java, every class(not interface)is directly or indirectly extended from java.lang.Object cla

class Employee{ //TODO }	is 100% same as	class Employee extends Object{ //TODO }
--------------------------------	-----------------	---

- java.lang.Object class is also called as Ultimate base class / super cosmic base class / root of Java class hierarchy.
- This class is introduced in JDK 1.0.
- Object class do not contain nested types and fields.
- Object class contain compiler defined default constructor.

1. Object obj = new Object(123); // NOT OK
2. Object obj = new Object("Sunbeam"); //Not OK

3. Object obj = new Object(); //OK

- There are 11 methods in java.lang.Object class.
 1. 5 non final methods(we can override it inside sub class)
 - 2 native + 3 non native methods
 2. 6 final methods(we can not override it inside sub class)
 - 4 native + 2 non native methods

Core Java Notes By ASHOK PATE

```
+ Methods of java.lang.Object class:  
- Non final methods( 5 )  
  1. public String toString( );  
  2. public boolean equals( Object obj );  
  3. public native int hashCode( );  
  4. protected native Object clone( )throws CloneNotSupportedException;  
  5. protected void finalize( )throws Throwable;  
- Final methods( 6 )  
  6. public final native Class<?> getClass( );  
  7. public final native void wait( )throws InterruptedException  
  8. public final native void wait( long timeout )throws InterruptedException  
  9. public final native void wait( long timeout, int nanos )throws InterruptedException  
10. Public final native void notify( );  
11. Public final native void notifyAll( );  
  
- native is keyword in Java.  
- If we want to invoke C++ member function in Java then we should use native keyword in Java.  
- Java Native Interface( JNI ) is a Java framework, which allows us to use native methods in Java.  
- For native methods  
  - use "Core Java VOL-II - Cay Horstman"  
  - We can not redefine/override final methods inside sub class.  
- If we want to see Documentation of any class from terminal then we should use javap tool.  
  - javap java.lang.Object ( press enter key )  
- Object class do not contain static method.  
- java.lang.Object class is non final and concrete class.  
  - i.e we can instantiate Object class  
  - i.e we can extend Object class.
```

Class Object

java.lang.Object

public class Object

Class Object is the root of the class hierarchy. Every class has Object as a superclass. All objects, including arrays, implement the methods of this class.

Since:

JDK1.0

See Also:

Class

java.lang.Object
java.lang.Class<T>

```
public final class Class<T>  
extends Object  
implements Serializable, GenericDeclaration, Type, AnnotatedElement
```

Difference between primitive and non primitive type?

- + What is the difference between primitive and non primitive type?
 1. Primitive type is also called as value type.
 1. Non primitive type is also called as reference type.
 2. There are 8 primitive/value types in Java
 2. There are 4 non primitive/reference types in Java
 3. boolean, byte, char, short, int, float, long, double are primitive types.
 3. Interface, class, array, type variable are non primitive types.
 4. Variable of primitive/value type contains value.
 4. Variable of non primitive/reference type contains reference.
 5. Variable of primitive/value type get space on Java stack.
 5. Instance of non primitive/reference type get space on heap.
 6. To create variable or instance of primitive/value type we can not use new operator.
 6. To create instance of non primitive/reference type we must use new operator.
 7. Variable of primitive/value type can not contain null value.
 7. Variable of non primitive/reference type can contain null value.
 8. Variable of primitive/value type by default contains 0 value.
 8. Variable of non primitive/reference type by default contains null value.

Initialization & Assignment:-

```
class Program{  
    public static void main1(String[] args) {  
        //During declaration, process of stroing value inside variable is  
        //called initialization.  
        int num1 = 10; //Initialization  
  
        //int num1 = 20; //Compiler error  
        //we can intialize any variable only once.  
  
        int num2 = num1; //Initialization  
        System.out.println("Num1 : "+num1); //10  
        System.out.println("Num2 : "+num2); //10  
    }  
    public static void main(String[] args) {  
        //After declaration, process of stroing value inside variable is  
        //called assignment.  
        int num1 = 10; //Initialization  
        num1 = 20; //Assignment  
        num1 = 30; //Assignment  
        //We can do assignment multiple times.
```

```
        System.out.println("Num1 : "+num1); //30
    }
}
-Constructor ek hi bar kyu call hota hai bcoz initialization only
once hi hota hai.
```

Final Keyword:- (Use in Local)

```
import java.util.Scanner;

class Program{
    public static void main1(String[] args) {
        int number = 10;
        number = number + 5;
        System.out.println("Number : "+number);//15
    }

    //ab muze initialization ke bad value modify nhi krna hai
    //tho use final keyword used krnege.
    public static void main2(String[] args) {
        final int number = 10; //Initialization
        //number = number + 5; //Not OK
        System.out.println("Number : "+number);//10
    }
    //final variable mai ek bar assignment kiya ya initialization
    //kiya uske bad modify nhi kar sakte
    //final variable ko initialization dena necessary nhi hai.
    public static void main3(String[] args) {
        final int number; //Ok
        number = 10; //Assignment //OK
        //number = number + 5; //Not OK
        System.out.println("Number : "+number);//15
    }
    public static void main4(String[] args) {
        final int number = 10; //Intialization //OK
        //number = 20; //Assignment //NOT OK
        //number = number + 5; //Not OK
        System.out.println("Number : "+number);//15
    }
}

public static void main(String[] args) {
    Scanner sc = new Scanner( System.in );
```

```
        System.out.print("Number : ");
        final int number = sc.nextInt();
        //number = number + 5; //Not OK
        System.out.println("Number : "+number);
    }
}
```

- In java we do not get `const` keyword. But we can use `final` keyword.
 - After storing value, if we don't want to modify it then we should declare variable `final`.
 - We can provide value to the final variable either at compile time or run time.
-

Final Keyword:- (Use In Global)

//agar field ka value class ke kisi bhi method mai modify
//nhi karna hai bas read krna hai use case mai field ko final declare
karna chahiye.

```
class Test{
    /* private final int number;
    public Test( ){
        this.number = 10;
        //this.number = 20; //Not OK
    }*/
    private final int number = 10; //OK
    public Test( ){
        //this.number = 10; //Not OK
        //this.number = 20; //Not OK
    }
    public void showRecord( ){
        //this.number = this.number + 5; //Not OK
        System.out.println("Number : "+this.number);
    }
    public void printRecord( ){
        //this.number = this.number + 5; //Not OK
        System.out.println("Number : "+this.number);
    }
}
class Program{
    public static void main(String[] args) {
        Test t = new Test( );
        t.showRecord( );
```

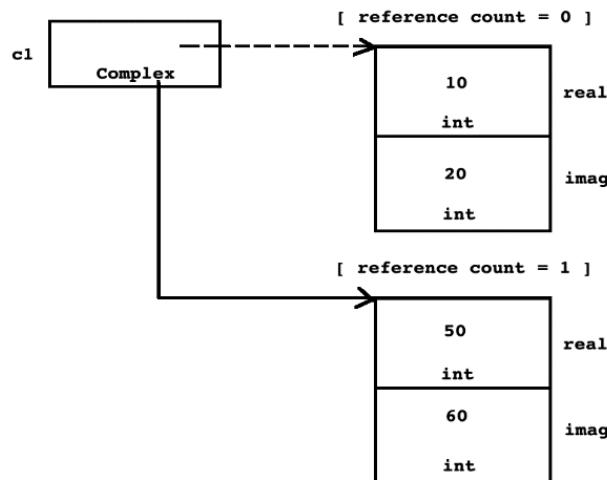
```
        System.out.println("-----");
        t.printRecord( );
    }
}
```

- Once initialized, if we don't want to modify state of any field inside any method of the class(including constructor body) then we should declare field final.
- If we want to declare any field final then we should declare it static also.

Final Keyword :- (use in object/reference)

```
class Complex{
    private int real;
    private int imag;
    public Complex(int real, int imag ){
        this.real = real;
        this.imag = imag;
    }
    public void setReal(int real) {
        this.real = real;
    }
    public void setImag(int imag) {
        this.imag = imag;
    }
    @Override
    public String toString() {
        return this.real + " " +this.imag;
    }
}
class Program{
    public static void main1(String[] args) {
        Complex c1 = new Complex( 10, 20 );
        c1 = new Complex( 50, 60 );
        System.out.println(c1.toString());
    }
    public static void main(String[] args) {
        //Object/reference is final
        //Instance is not final.
        final Complex c1 = new Complex( 10, 20 ); //Instance is not final
        c1.setReal(11);
        c1.setImag(22);
        //c1 = new Complex( 50, 60 ); //Not OK
    }
}
```

```
        System.out.println(c1.toString()); //11, 22
    }
}
• In Java, we can declare reference final but we can not declare instance final.
• We can declare method final. It is not allowed to override final method in sub class.
• We can declare class final. It is not allowed to extend final class.
```



References Memory allocation:-

```
class Date{
    //TODO
}

class Employee{
    private String name;
    private int empid;
    private float salary;
    private Date joinDate; //joinDate is non static field of the class.
    // Non static field get space inside instance and instance get space on heap.
    public Employee(){
        this.joinDate = new Date( );
    }
}

class Program{
    public static void main(String[] args) {
        Date joinDate = new Date( );
        //Date joinDate => joinDate is a method local variable. It will get space on Java Stack.
```

//new Date() => It is Date instance. It will get space on Heap.

}

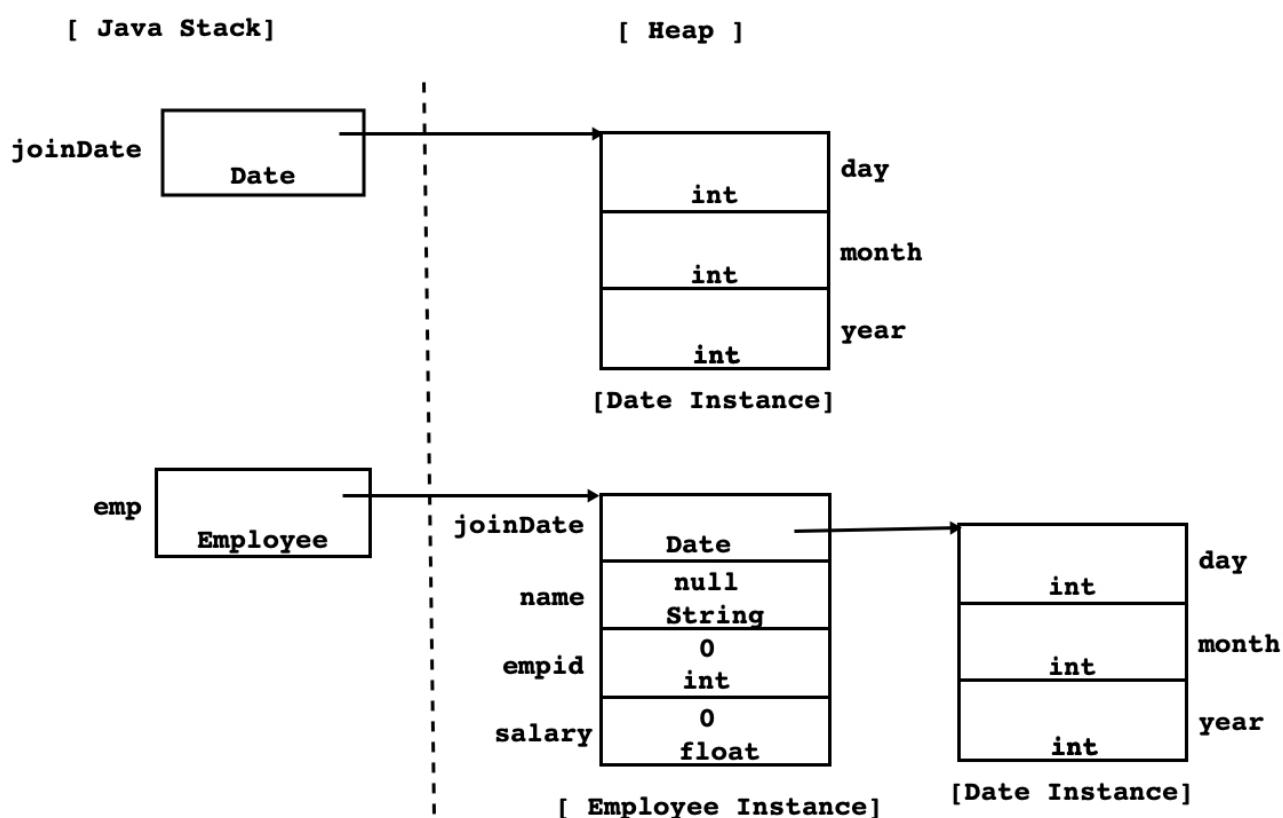
}

- Local reference variable get space on Java Stack.

- In above code joinDate is method local reference variable hence it gets space on Java Stack.

- Class scope reference variable get space on heap.

- In above code, emp is method local reference variable hence it gets space on Java Stack. But joinDate is field of Employee class hence it will get space inside instance on Heap.



Access Modifier:-

- If we want to control visibility of members of class then we should use access modifier.

```
import java.util.Scanner;
/*
- Access modifiers in Java
  1. private
  2. package level private / default
  3. protected
  4. public
```

```
- In Java, members of a class are by default considered as package level private i.e default.  
*/  
class Employee{  
    private String name;  
    private int empid;  
    private float salary;  
    //Parameterless constructor / zero argument constructor / user defined default constructor  
    //We can use any access modifier on constructor including the private  
  
    public Employee( ){  
    }  
  
    //Parameterized constructor  
    //If name of method parameter / Method local variable and name of field is same then preference is always given to the local variable/parameter  
    //In this case we should use this before members of class.  
    public Employee( String name, int empid, float salary){  
        this.name = name;  
        this.empid = empid;  
        this.salary = salary;  
    }  
    public void acceptRecord( ){  
        Scanner sc = new Scanner( System.in );  
        System.out.print("Name : ");  
        this.name = sc.nextLine();  
        System.out.print("Empid : ");  
        this.empid = sc.nextInt();  
        System.out.print("Salary : ");  
        this.salary =sc.nextFloat();  
    }  
  
public void printRecord( /* Employee this */){  
    System.out.println("Name : "+this.name);  
    System.out.println("Empid : "+this.empid);  
    System.out.println("Salary : "+this.salary);  
    }  
}  
class Program{  
    public static void main1(String[] args) {
```

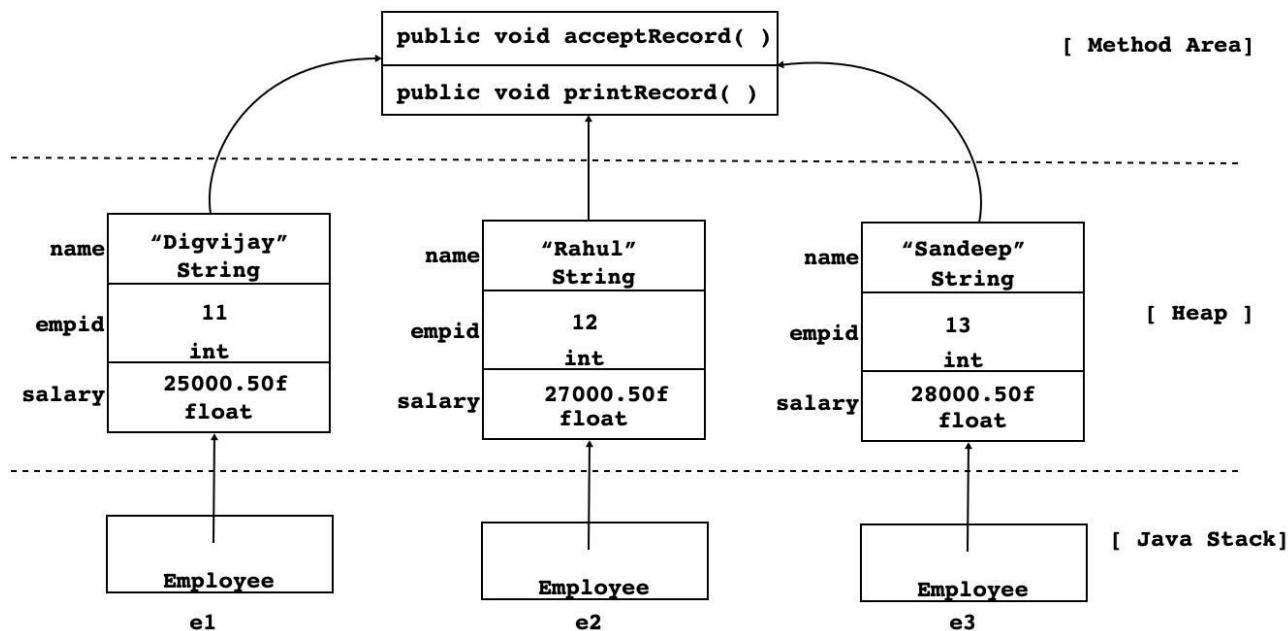
```
Employee emp1 = new Employee( "Sandeep", 33, 23000.50f );
//here on instance, parameterized constructor will call
emp1.printRecord( );
//this is looking for parameter less constructor.if you not
written then error give.
Employee emp2 = new Employee( );//Error:here on instance,
parameterized constructor will call
emp1.acceptRecord( );
emp1.printRecord( );
}
//Only Fields of the class gets space once per instance,
according to their order of declaration inside class.
//Method do not get space inside instance. Rather all the
instances of same class share single copy of it.
//By passing reference, all the instances of same class share
methods defined inside class.
public static void main(String[] args) {
    Employee e1 = new Employee( "Digvijay", 11, 25000.50f );
    Employee e2 = new Employee( "Rahul", 12, 27000.50f );
    Employee e3 = new Employee( "Sandeep", 13, 28000.50f );

    e1.printRecord( ); //e1.printRecord( e1 );
    e2.printRecord( ); //e2.printRecord( e2 );
    e3.printRecord( ); //e3.printRecord( e3 );
}
}
```

→Only Fields of the class gets space once per instance, according to their order of declaration inside class.

→Method do not get space inside instance. Rather all the instances of same class share single copy of it.

→By passing reference, all the instances of same class share methods defined inside class.



+ Characteristics of Instance

- State

- * Value stored inside instance is called as state of that instance.
 - * Value of the field represents state of instance.
- Behavior
- * Set of operations, which are allowed to perform on any instance is called its behavior.
 - * Methods defined inside class represents behaviour of the instance.
- Identity
- * Value of any field from instance which is used to identify instance uniquely is called identity of the instance.
 - * When state of instances are same in that case we can use reference as a identity.

+ Class

1. Class is a collection of fields and methods.
2. Structure and behaviour of instance depends on class hence class is considered as a template/model/blueprint for instance.
3. Class represents set/group of objects/instances that represents common structure and common behaviour.
4. Class is imaginary / logical entity.

5. Example:

- | | |
|-----------|---|
| 1. Book | + Object |
| 2. Laptop | - Object is instance of a class. |
| 3. Car | - An entity, which is having physical existence is called object. |
- An entity, which has state, behaviour and identity is called object.
- Object is real time / physical entity.
- Example:
1. Java Certification - Khalid Mughal
 2. MacBook Air
 3. Tata nano

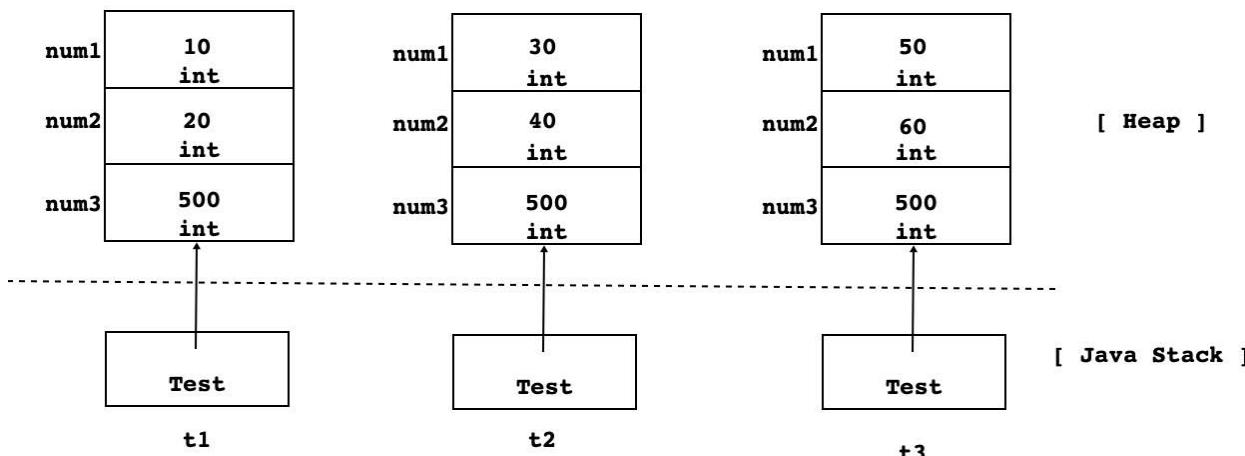
Static Field:-

```
import java.util.Scanner;
/*
    - A field of a class which get space inside instance is called
instance variable.
    - In other words, non static field declared inside class gets
space inside instance. Hence it is called instance variable.
    - Instance variable get space after instantiation( i.e after
creating instance ).
    - To access instance variable we must use object reference.

    - We can not call non static method on class name. Non static
method is designed to call on instance.
    - A method, which is designed to call on instance is called
instance method.
    - Since non static method is designed to call on instance, it
is also called as instance method.

    - In general to access instance members( instance variable,
instance method ) we must use instance.
*/
class Test{
    //Instance variables
    private int num1;      //Non static field / Instance variable
    private int num2;      //Non static field / Instance variable
    private int num3;      //Non static field / Instance variable
    public Test( int num1, int num2){
        this.num1 = num1;
        this.num2 = num2;
        this.num3 = 500;
    }
    //Concrete method
    public void printRecord( ){ //Non static method
        System.out.println("Num1 : "+this.num1);
        System.out.println("Num2 : "+this.num2);
        System.out.println("Num3 : "+this.num3);
        System.out.println("-----");
    }
}
class Program{
    public static void main(String[] args) {
        Test t1 = new Test( 10, 20 );
        Test t2 = new Test( 30, 40 );
    }
}
```

```
Test t3 = new Test( 40, 60 );  
  
t1.printRecord( );  
t2.printRecord( );  
t3.printRecord( );  
  
//Test.printRecord( ); //error: non-static method  
printRecord() cannot be referenced from a static context  
}  
}
```



- If we want to share value of any field inside all the instances of same class then we should declare that field static.
- Static field do not get space inside instance rather all the instances of same class share single copy of it. Hence size of static field is not considered inside size of instance.
- Non static Field is also called as instance variable. It gets space once per instance.
- Static Field is also called as class variable. It gets space once per class.
- Static Field gets space once per class during class loading on method area.
- Instance variables are designed to access using object reference.
- Class level variable can be accessed using object reference but it is designed to access using class name and dot operator.

Core Java Notes By ASHOK PATE

- static is modifier in Java.
- If want to share value of field inside all the instances of same class then we should declare that field static.
- Static field do not get space inside instance. Rather all the instances of same class share single copy it. Hence size of static field is not considered inside size of instance.
- A field of a class which do not get space inside instance is called class level variable. In short, static field is also called as class level variable.
- To access class level variable / static field we should use class name and dot operator(.)
- Non static field i.e instance variable get space after instantiation.
- Static field i.e class level variable get space during once per class during class loading on method area.
- Conclusion about fields:
 1. Non static field get space once per instance.
 2. Static field get space once per class.

```
class A{  
    private int a;  
    private int b;  
    private static int z;  
}
```

```
class B{  
    private int p;  
    private int q;  
    private static int z;  
}
```

```
class C{  
    private int x;  
    private int y;  
    private static int z;  
}
```

```
A a1 = new A();  
A a2 = new A();  
A a3 = new A();
```

```
B b1 = new B();  
B b2 = new B();  
B b3 = new B();
```

```
C c1 = new C();  
C c2 = new C();  
C c3 = new C();
```

- Here field z will get space once per class means total 3 times will get space.

Static Initialization Block :-

```
import java.util.Scanner;  
/*  
    - To initialize non static field / instance variable we should  
    use constructor.  
    - To initialize static field / class level variable we should  
    use static initializer block.  
    - JVM execute static initializer block during class loading.  
*/  
class Test{  
    private int num1;           //Non static field / Instance  
    variable  
    private int num2;           //Non static field / Instance  
    variable  
    private static int num3;     //static field / Class level  
    variable  
  
    //Static Initializer Block  
    static{  
        Test.num3 = 500;
```

```
}

//Constructor
public Test( int num1, int num2){
    this.num1 = num1;
    this.num2 = num2;
    //Test.num3 = 500; // OK: But not recommended
    //this.num3 = 500; //OK : Warning The static field
Test.num3 should be accessed in a static way
}
//Concrete method
public void printRecord( ){ //Non static method
    System.out.println("Num1      : "+this.num1);
    System.out.println("Num2      : "+this.num2);
    System.out.println("Num3      : "+Test.num3);
    //System.out.println("Num3      : "+this.num3); //OK :
Warning The static field Test.num3 should be accessed in a static
way
    System.out.println("-----");
}
}

class Program{
    public static void main(String[] args) {
        Test t1 = new Test( 10, 20 );
        Test t2 = new Test( 30, 40 );
        Test t3 = new Test( 50, 60 );

        t1.printRecord( );
        t2.printRecord( );
        t3.printRecord( );
    }
}
```

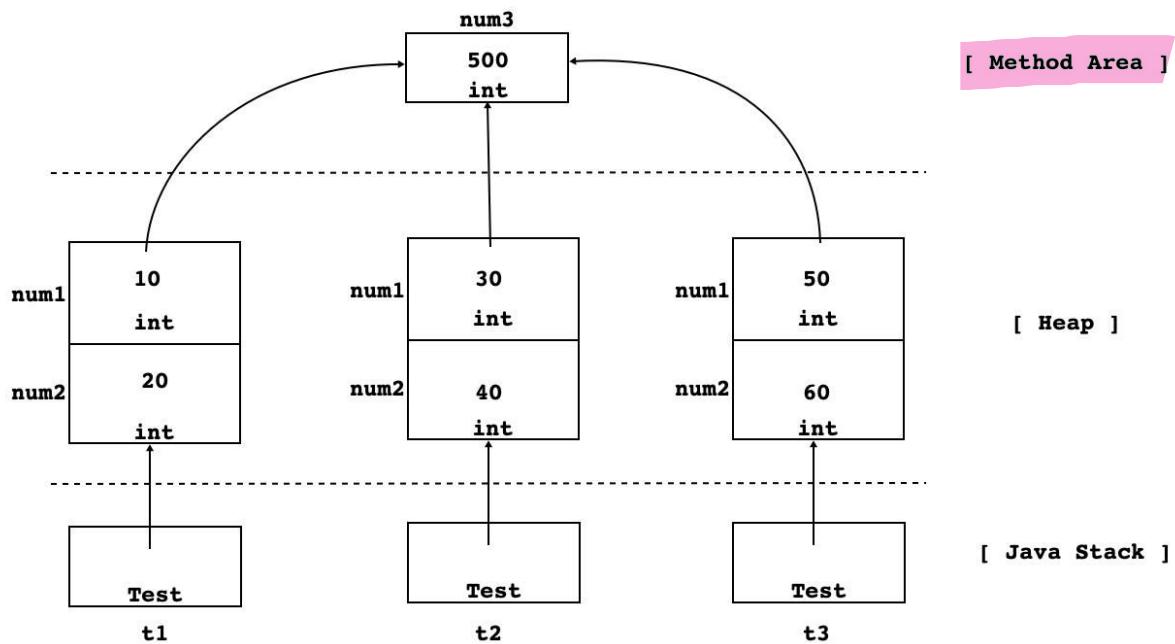
- A static initialization block is a normal block of code enclosed in braces, { }, and preceded by the static keyword.

Here is an example:

```
static{
    Test.num3 = 500;
}
```

- A class can have any number of static initialization blocks, and they can appear anywhere in the class body.
- The runtime system guarantees that static initialization blocks are called in the order that they appear in the source code.

- There is an alternative to static blocks – you can write a private static method:



Some modification for understanding purpose:-

```

import java.util.Scanner;
/*
    - To initialize non static field / instance variable we should
use constructor.
    - To initialize static field / class level variable we should
use static initializer block.
    - JVM execute static initializer block during class loading.
*/
class Test{
    private int num1;    //Non static field / Instance variable
    private int num2;    //Non static field / Instance variable
    private static int num3;//static field / Class level variable
    private static int num4;//static field / Class level variable
    private static int num5;//static field / Class level variable
    private static int num6;//static field / Class level variable
    private static int num7;//static field / Class level variable
    private static int num8;//static field / Class level variable
}
  
```

```
//Static Initializer Block
static{
    System.out.println("Static initlizer block#1");
    Test.num3 = 500;
    Test.num4 = 600;
    Test.num5 = 700;
}
static{
    System.out.println("Static initlizer block#2");
    Test.num6 = 800;
}

}
static{
    System.out.println("Static initlizer block#3");
    Test.num7 = 900;
}
static{
    System.out.println("Static initlizer block#4");
    Test.num8 = 1000;
}

//Constructor
public Test( int num1, int num2){
    System.out.println("constructor block#1");
    this.num1 = num1;
    this.num2 = num2;
    // Test.num3 = 500; // OK: But not recommended
    // this.num3 = 500; //OK : Warning The static field
Test.num3 should be accessed in a static way
}
//Concrete method
public void printRecord( ){ //Non static method

    System.out.println("Num1      : "+this.num1);
    System.out.println("Num2      : "+this.num2);
    System.out.println("Num3      : "+Test.num3);
    System.out.println("Num4      : "+Test.num4);
    System.out.println("Num5      : "+Test.num5);
    System.out.println("Num6      : "+Test.num6);
    System.out.println("Num7      : "+Test.num7);
    System.out.println("Num8      : "+Test.num8);
}
```

```
//System.out.println("Num3      :  "+this.num3); //OK :  
Warning The static field Test.num3 should be accessed in a static  
way  
        System.out.println("-----");  
    }  
}  
class Program{  
    public static void main(String[] args) {  
        Test t1 = new Test( 10, 20 );  
        Test t2 = new Test( 30, 40 );  
        Test t3 = new Test( 50, 60 );  
  
        t1.printRecord( );  
        t2.printRecord( );  
        t3.printRecord( );  
    }  
}
```

/*Output:-

```
Static initlizer block#1  
Static initlizer block#2  
Static initlizer block#3  
Static initlizer block#4  
constructor block#1  
constructor block#1  
constructor block#1  
Num1      :  10          |  30  |  50  
Num2      :  20          |  40  |  60  
Num3      :  500         | 500  | 500  
Num4      :  600         | 600  | 600  
Num5      :  700         | 700  | 700  
Num6      :  800         | 800  | 800  
Num7      :  900         | 900  | 900  
Num8      : 1000         |1000  |1000  
----- */
```

Sequence of execution observe the output:-

```
import java.util.Scanner;
/*
 - To intialize non static field / instance variable we should
use constructor.
 - To intialize static field / class level variable we should
use static initializer block.
 - JVM execute static initializer block during class loading.
*/
class Test{
    private int num1;           //Non static field / Instance
variable
    private int num2;           //Non static field / Instance
variable
    private static int num3;     //static field / Class level
variable

    //Static Initializer Block
    static{
        System.out.println("Inside static initialization block of
class Test");
        Test.num3 = 500;
    }

    //Constructor
    public Test( int num1, int num2){
        System.out.println("Inside constructor of class Test");
        this.num1 = num1;
        this.num2 = num2;
    }
    //Concrete method
    public void printRecord( ){ //Non static method
        System.out.println("Num1      : "+this.num1);
        System.out.println("Num2      : "+this.num2);
        System.out.println("Num3      : "+Test.num3);
        System.out.println("-----");
    }
}
class Program{
    static{
        System.out.println("Inside static initialization block of
class Program"); 1
    }
}
```

```
public Program() {
    System.out.println("Inside constructor of class Program");
}
public static void main(String[] args) throws Exception {
    System.out.println("Inside main method"); 2

    System.out.print("Press any key to continue..."); 2
    System.in.read();

    Test t1 = new Test( 10, 20 );
    Test t2 = new Test( 30, 40 );
    Test t3 = new Test( 50, 60 );

    t1.printRecord();
    t2.printRecord();
    t3.printRecord();
}
/*Output:-
```

```
Inside static initialization block of class Program
Inside main method
Press any key to continue...Inside static initialization block of
class Test
Inside constructor of class Test
Inside constructor of class Test
Inside constructor of class Test
Num1      :   10
Num2      :   20
Num3      :   500
-----
Num1      :   30
Num2      :   40
Num3      :   500
-----
Num1      :   50
Num2      :   60
Num3      :   500
-----*/
```

```
class Program{
    static{
        System.out.println("Inside static initialization block of
class Program");
        Program.myMain();
    }
    public static void myMain( ){
        System.out.println("Hello from my main");
    }
}
/*Error:- main method hona chahiye
static{
    System.out.println("Inside static initialization block of class
Program");
    Program.myMain();
}
*/


---


```

- /*
 - Non static methods are designed to call on instance. Hence it is also called as instance method.
 - Static methods are designed to call on class name. Hence it is called as class level method.
 - In General conclusion:
 1. To access non static members, method should be non static.
 2. To access static members method should be static.
 - Static method do not get this reference. Why?
 1. If we call, non static method on instance then method get this reference.
 2. Static methods are designed to call on class name.
 3. Static method is not designed to call on instance hence static method do not get this reference.
 - Since static method do not get this reference, we can not access non static members inside static method.
 - In other words, inside static method, we can access only static members;
 - Inside non static method, we can access static as well as non static members
 - Static method do not get this reference but we can create instance inside static method.
 - Using instance, we can access non static members inside static method.

```
/*
class Test{
    private int num1;          //Non static field / instance
variable
    private int num2;          //Non static field / instance
variable
    private static int num3;    //static field / class level
variable
    public Test() {
    }
    public void setNum1( /* Test this, */ int num1 ){ //Non static
/ Instance method
        this.num1 = num1;
    }
    public void setNum2( /* Test this, */ int num2 ){ //Non static
/ Instance method
        this.num2 = num2;
    }
    public static void setNum3( /* this reference is not required*/
int num3 ){ //Static Method / Class level method
        Test.num3 = num3;
    }
    public void printRecord( /* Test this */){
        System.out.println("Num1      : "+this.num1);
        System.out.println("Num2      : "+this.num2);
        System.out.println("Num3      : "+Test.num3);
    }
    public static void displayRecord( ){
        Test t1 = new Test();
        t1.setNum1(10); //t1.setNum1(t1, 10);
        t1.setNum2(20); //t1.setNum2(t1, 20);
        Test.setNum3(30);
        t1.printRecord( );
    }
}
class Program{
    public static void main(String[] args) {
        Test.displayRecord( );
    }
    public static void main1(String[] args) {
        Test t1 = new Test();
        t1.setNum1(10); //t1.setNum1(t1, 10);
        t1.setNum2(20); //t1.setNum2(t1, 20);
    }
}
```

```
Test.setNum3(30); //static method design to call on class
name.
    t1.printRecord( );
}
}

class Program{
// Using instance, we can access non static members inside static
method.
    public int num1 = 10;
    public static int num2 = 20;
    public static void main(String[] args) {
        //System.out.println("Num1 : "+num1); //error: non-
static variable num1 cannot be referenced from a static context
        Program p = new Program();
        System.out.println("Num1 : "+p.num1); //Ok : 10
        System.out.println("Num2 : "+num2); //OK : 20
    }
}

class B{
    public static void f2( ){
        System.out.println("B.f2()");
    }
    public static void f3( ){
        f2(); //OK : B.f2()
        B.f2(); //OK : B.f2()
    }
}
class Program{
    public static void main(String[] args) {
        //f3( ); // error: cannot find symbol
        B.f3(); //OK
    }
}
```

InstanceCounter:-

```
class InstanceCounter{
    private static int count;
    public InstanceCounter( ){
        ++ InstanceCounter.count;
    }
}
```

```
public static int getCount( ){
    return InstanceCounter.count;
}
}

class Program{
    public static void main(String[] args) {
        InstanceCounter c1 = new InstanceCounter( );
        InstanceCounter c2 = new InstanceCounter( );
        InstanceCounter c3 = new InstanceCounter( );
        System.out.println("Instance
Count : "+InstanceCounter.getCount());
    }
}

/*
 - If constructor is public then we can create instance of a
class inside method of same class
as well as different class
*/
class Complex{
    public Complex( ){
        System.out.println("Inside constructor");
    }
    public static void test( ){
        Complex c2 = new Complex( ); //OK
    }
}
class Program{
    public static void main(String[] args) {
        Complex c1 = new Complex( ); //OK
        Complex.test( );
    }
}

/*
 - We can declare constructor private.
 - If constructor is private then we can create instance of a
class inside method of same class only.
*/
class Complex{
    private Complex( ){
        System.out.println("Inside constructor");
    }
    public static void test( ){
```

```
Complex c2 = new Complex( ); //OK
}
}
class Program{
    public static void main(String[] args) {
        //Complex c1 = new Complex( ); //error: Complex() has
private access in Complex
        Complex.test( );
    }
}

/*
    - Inside method, if there is need to use this reference then
method should be non static.
    - Otherwise method should be static.
*/
class Calculator{
    public static int pow( int base, int index ){
        int result = 1;
        for( int count = 1; count <= index; ++ count )
            result = result * base;
        return result;
    }
}
class Program{
    public static void main(String[] args) {
        int result = Calculator.pow(10, 2);
        System.out.println("Result : "+result);
    }
}
//jis method mai this reference use krne ki jarurat hai use hi non
static karo agar nhi jarurat tho static kr do method ko.

//If we declare method static then local variable is not considered
static.
//Non static method local variable get space once per method call.
Hence it doesn't retain last updated value.
class Program{
    public static void print( ){
        int count = 0; //Non static method local variables
        count = count + 1;
        System.out.println("Count : "+count); //1
    }
}
```

```
public static void main(String[] args) {
    Program.print();      //1
    Program.print();      //1
    Program.print();      //1
}
}

//In Java, we can not declare method local variable static.
// Static field and static method is also called as class level
members.
//According to oops, class level members must in class scope. Hence
we can not declare local variable static.
//In Java we can declare field, method, nested type static.
//We can not delcare local variable, constructor and top level
class static.

class Program{
    public static void print( ){
        static int count = 0; // error: illegal start of expression
        count = count + 1;
        System.out.println("Count : "+count); //1
    }
    public static void main(String[] args) {
        Program.print();
        Program.print();
        Program.print();
    }
}
```

Singleton class

- A class from which, we can create only one instance is called singleton class.
- Steps to define singleton class:
 1. Define class and declare constructor private.
 2. Define factory method and return reference of instance from it.
 3. Use getter and setter to access members.

```
//1. Define clas
class Singleton{
    private int number;

    //2. Make constructor private
    private Singleton() {
```

```

}

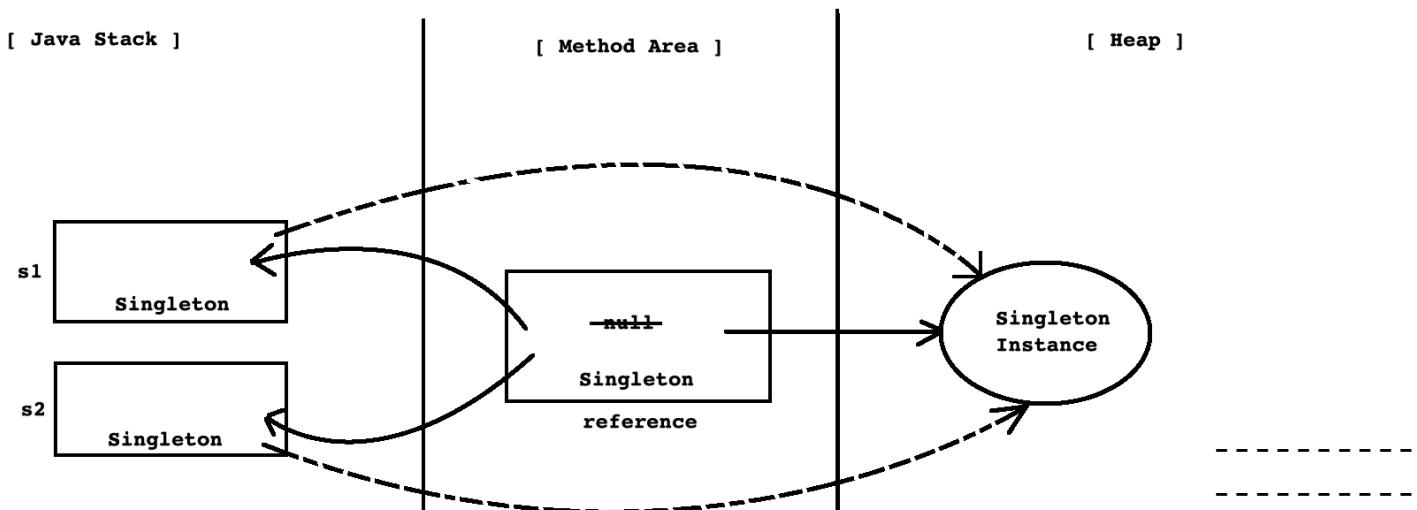
//4. Declare static field inside class
private static Singleton reference; //null

//3. Define a static method to create instance of a class
public static Singleton getReference( ){
    if( reference == null )
        reference = new Singleton( );
    return reference;
}

public int getNumber() {
    return this.number;
}
public void setNumber(int number) {
    this.number = number;
}
}

class Program{
    public static void main(String[] args) {
        Singleton s1 = Singleton.getReference( );
        s1.setNumber(123);
        System.out.println("Number : "+s1.getNumber());
    }
    public static void main1(String[] args) {
        Singleton s1 = Singleton.getReference( );
        Singleton s2 = Singleton.getReference( );
        System.out.println(s1 == s2 ); //true
    }
}

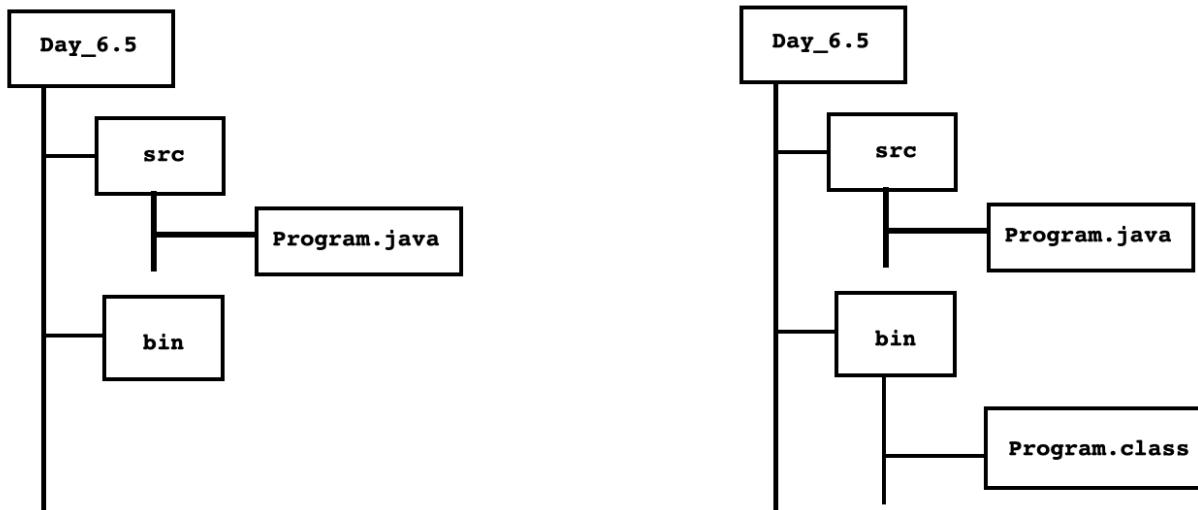
```



```
class Singleton{
    private static Singleton reference;
    static{
        reference = new Singleton();
    }
    private Singleton() {
    }
    public static Singleton getReference() {
        return reference;
    }
}
class Program{
    public static void main(String[] args) {
        Singleton s1 = Singleton.getReference();
        Singleton s2 = Singleton.getReference();
        System.out.println(s1 == s2 ); //true
    }
}
```

Now my requirement is I want to create dot java file in one folder and there dot class file in another folder and by this execute.

[Compile and execute code from Day_6.5 directory.]



Core Java Notes By ASHOK PATE

```
+ Absolute path
- A path of file from root directory is called absolute path
- Example:
  D:\SunBeam\Java\CJ06\Day6\Day_6.1\src\Program.java

+ Relative path
- A path of file from current directory is called relative path
- Example:
  cd D:\SunBeam\Java\CJ06\Day6\Day_6.1
  .\src\Program.java

+ Path
- Path is OS platforms environment variable which is used to locate development tools
- Java development tools are in ( jdk/bin ) directory.
- How to set path
  //In Windows
  set path="C:\Program Files (x86)\Java\jdk1.8.0_301\bin";
  set path="C:\Program Files\Eclipse Foundation\jdk-11.0.12.7-hotspot\bin"

  //In Linux/Mac OS
  export PATH=/usr/bin

+ Classpath
- It is environment variable of Java platform which help class loader to locate .class/.jar file.
- By default classpath is set to current directory.
- How to set classpath:
  //In Windows
  set classpath=.\bin;
  //Linux \ Mac OS
  export CLASSPATH=./bin
```

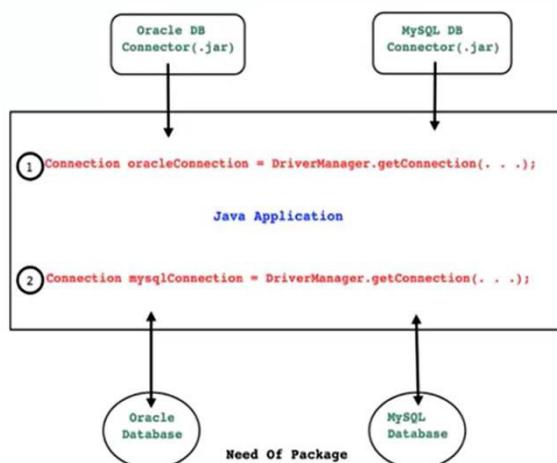
Package:-

- Package is java Language feature which helps developer to
 - 1.To group functionally equivalent or related types together.
 - 2.To avoid naming clashing/collision/conflict/ambiguity in source code.
 - 3.To control the access to types.
 - 4.To make types easier to find (from the perspective of java docs).

- consider following class:

➤ java.lang.object

Here java is main package,lang is sub package and object is type name.



- Not necessarily but as shown below, package can contain some or types.
1.Sub package 2.Interface 3.class 4.Enum 5.Exception 6.Error
7.Annotation Type.
- **package is a keyword in Java.**
- To define type inside package, it is mandatory write package declaration statement inside .java file.
- **Package declaration statement must be first statement inside.java file**
- If we define any type inside package then it is called as packaged type otherwise it will be unpackaged type.
- Any type can be member of single package only.

<pre>package p1; //OK class Program{ //TODO }</pre>	<pre>package p1, p2; //NOT OK class Program{ //TODO }</pre>	<pre>package p1; //OK package p2; //NOT OK class Program{ //TODO } package p3; //Not OK</pre>
---	---	---

Un-named Package

- If we define any type without package then it is considered as member of unnamed/default package.
- Unnamed packages are provided by the Java SE platform principally for convenience when developing small or temporary applications or when just beginning development.
- An unnamed package cannot have sub packages.
- In following code, class Program is a part of unnamed package.

Naming Conversion:-

- For small programs and casual development, a package can be unnamed or have a simple name, but if code is to be widely distributed, unique package names should be chosen using qualified names.
- Generally Package names are written in all lower case to avoid conflict with the names of classes or interfaces.

- Companies use their reserved internet domain name to begin their package names. For example: com.example.mypackage
- Following examples will help you in deciding name of package:
- java.lang.reflect.Proxy
- oracle.jdbc.driver.OracleDriver
- com.mysql.jdbc.cj.Driver
- org.cdac.sunbeam.dbda.utils.Date

How to use package members in different package?

- If we want to use types declared inside package anywhere outside the package then
 1. Either we should use fully qualified type name or
 2. import statement.
- If we are going to use any type infrequently then we should use fully qualified name.
- Let us see how to use type using package name.

```
class Program{  
    public static void main(String[] args) {  
        java.util.Scanner sc = new java.util.Scanner(System.in);  
  
    }  
}
```

- If we are going to use any type frequently then we should use import statement.
- Let us see how to import Scanner

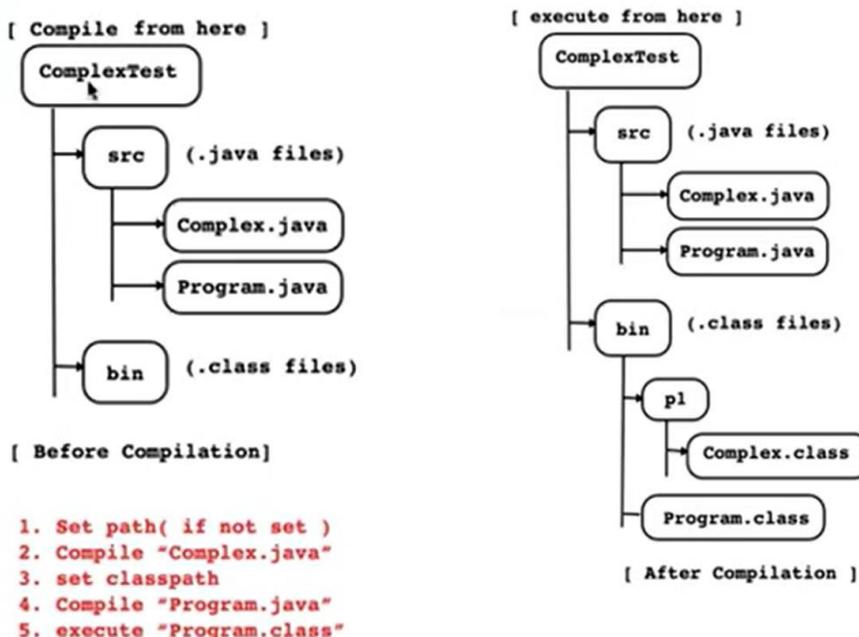
```
import java.util.Scanner;  
Class Program{  
    Public static void main(String[] args)  
    {  
        Scanner sc= new Scanner(System.in);  
    }  
}
```
- There can be any number of import statements after packagedeclaration statement.
- With the help of(*) we can import entire package.

```
import java.util.*;
class Program {
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
    }
}
```

- Another, less common form of import allows us to import the public nested classes of an enclosing class. Consider following code.

```
import java.lang.Thread.State;
class Program{
    public static void main(String[] args) {
        Thread thread=Thread.currentThread();
        State state = thread.getState();
    }
}
```

- Note: `java.lang` package contains fundamental types of core java. This package is by default imported in every .java file hence to use type declared in `java.lang` package, import statement is optional.



```
[ javac -d./bin ./src/Complex.java
  set classpath=./bin
  javac -d ./bin ./src/Program.java
  java Program  ]

package p1;
public class Complex{
//Packaged class agar yaha public nhi likha honga tho by default
access modifier is package level private/default. Package ke baher
used nhi kar sakte isliye usko public declare karte hai.

//name of the public file and dot java file must be same.
@Override
public String toString() {
    return "TComplex.toString()";
}
}

import p1.Complex;
class Program {
    public static void main(String[] args) {
        //p1.Complex c1 = new p1.Complex();    //Solution 1
        Complex c1 = new Complex();    //Solution 2
        System.out.println( c1.toString() );
    }
}
```

- Package name is physically mapped to the folder.
- Point to Remember: default access modifier of any type is package level private which is also called as default.
- If we want to use any type inside same package as well as in different package then access modifier of type must be public.
- Access modifier of type (class/interface) can be either package level private/public only. In other words, type can not be private or protected.

```
//Member of default package
public class Complex{ //Unpackaged class
    @Override
    public String toString() {
        return "TComplex.toString()";
    }
}
package p1;
class Program {
    public static void main(String[] args) {
        Complex c1 = new Complex(); //Can not access here
        System.out.println( c1.toString() );
    }
}
```

→ This cannot run error gives.

//Location : ./src/Complex.java public class Complex{ //Unpackaged Type @Override public String toString(){ return "Complex.toString()"; } }	//Location : ./src/Program.java package p1; public class Program{ public static void main(String[] args) { Complex c1 = new Complex(); System.out.println(c1.toString()); } }
javac -d ./bin ./src/Complex.java //OK:Complex.class export CLASSPATH=.:./bin javac -d ./bin ./src/Program.java //error : cannot find symbol	

- If we define any type without package then it is considered as a member of default package.
 - Conclusion : Since we can not import default package, it is not possible to use unpackaged type from packaged type.
-

```
package p1;
public class Complex{
    @Override
    public String toString() {
        return "TComplex.toString()";
    }
}
```

```
package p2;
import p1.Complex;
class Program {
```

```
public static void main(String[] args) {
    Complex c1 = new Complex();
    System.out.println( c1.toString() );
}
```

//Location : ./src/Complex.java	//Location : ./src/Program.java
package p1;	package p2;
public class Complex{	import p1.Complex;
@Override	public class Program{
public String toString(){	public static void main(String[] args) {
return "Complex.toString()";	Complex c1 = new Complex();
}	System.out.println(c1.toString());
}	}
	}
javac -d ./bin ./src/Complex.java //OK:p1/Complex.class	
export CLASSPATH=.:bin	
javac -d ./bin ./src/Program.java //OK:p2/Program.class	
//java Program //Error	
java p2.Program //Complex.toString()	

```
package p1;
public class Complex{
    public String toString(){
        return "Complex.toString";
    }
}

package p1;
//import p1.Complex; //Optional
class Program {
    public static void main(String[] args) {
        Complex c1 = new Complex( );
        System.out.println( c1.toString() );
    }
}
```

```
package p1.p2;
public class Complex{
    public String toString(){
        return "Complex.toString";
    }
}

package p1.p3;
import p1.p2.Complex;
class Program {
    public static void main(String[] args) {
        Complex c1 = new Complex( );
        System.out.println( c1.toString() );
    }
}

//import java.lang.*; //Optional-By default imported.
class Program {
    public static final double PI = 3.142;
    public static float pow( float base, int index ){
        float result = 1;
        for( int count = 1; count <= index; count++ )
            result = result * base;
        return result;
    }
    public static void main1(String[] args) {//hardcore value
        float radius = 10.5f;
        float area = ( float )( 3.142 * radius * radius );
        System.out.println("Area : "+area);
    }
    public static void main2(String[] args) {//class name pr call
kiya
        float radius = 10.5f;
        float area = ( float )( Program.PI * Program.pow(radius, 2)
);
        System.out.println("Area : "+area);
    }
    public static void main(String[] args) {//sab static hai tho
directly call kr sakte
}
```

```
    float radius = 10.5f;
    float area = ( float )(PI * pow(radius, 2) );
    System.out.println("Area      :      "+area);
}
}
```

Its present in the dot lang package and in Math class.
To access we can used Math.PI then for pow Math.Pow

Field Detail

E

```
public static final double E
```

The double value that is closer than any other to *e*, the base of the natural logarithms.

See Also:

[Constant Field Values](#)

PI

```
public static final double PI
```

The double value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter.

See Also:

[Constant Field Values](#)

```
//Use Static import for PI and pow

/* import static java.lang.Math.PI;
import static java.lang.Math.pow; */

import static java.lang.System.*;
import static java.lang.Math.*;
class Program {
    public static void main(String[] args) {
        float radius = 10.5f;
        float area = ( float )( PI * pow(radius, 2) );
        out.println("Area      :      "+area);
    }
}
```

```
public static void main1(String[] args) {
    float radius = 10.5f;
    float area = ( float )( Math.PI * Math.pow(radius, 2) );
    System.out.println("Area      :      "+area);
}
}

// If we want to access any public type(interface/class/enum etc)
// outside package the we should use import statement.
// Without type name, if we want to use any static member of type
// outside class then we should use static member.
```

Eclipse basic Program practise and short cut :-

```
package com.sunbeaminfo.cj06.test;
public class Program {
    private static final double PI = 3.142;
    private static double pow(float base, int index) {
        float result = 1;
        for( int count = 1; count <= index; ++ count )
            result = result * base;
        return result;
    }
    public static void main(String[] args) {
        float radius = 10.5f;
        float area = (float) (PI * pow(radius, 2));
        System.out.println("Area      :      "+area);
    }
}

//Eclipse shortcut
//Ctrl+ space for suggestion
//ctrl+ 1      for error soln typecast
//click on PI create constant PI
//ctrl + 1 pow and create method
```

This written in Employee java file

```
package com.sunbeaminfo.cj06.model;
public class Employee {
    private String name;
    private int empid;
    private float salary;
    public Employee() {
        // TODO Auto-generated constructor stub
    }
    public Employee(String name, int empid, float salary) {
        this.name = name;
        this.empid = empid;
        this.salary = salary;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getEmpid() {
        return empid;
    }
    public void setEmpid(int empid) {
        this.empid = empid;
    }
    public float getSalary() {
        return salary;
    }
    public void setSalary(float salary) {
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "Employee [name=" + name + ", empid=" + empid + ",
salary=" + salary + "]";
    }
}

//Emp type and ctrl + space and generate constructor
//right click then source then generate constructor using fields
```

```
//member initialized then access modifier choose and hit enter your  
contructor may get created  
//now i want getter and setters right click source and then  
generate getter and setters.  
//now to override right click source and then click generate  
toString.
```

This written in Program file

```
package com.sunbeaminfo.cj06.test;  
import com.sunbeaminfo.cj06.model.Employee;  
  
public class Program {  
    public static void main1(String[] args) {  
        Employee emp = new Employee();  
        System.out.println(emp); //Employee [name=null, empid=0,  
salary=0.0]  
    }  
    public static void main2(String[] args) {  
        Employee emp = new Employee();  
        emp.setName("Sandeep");  
        emp.setEmpid(33);  
        emp.setSalary(35000.50f);  
  
        // System.out.println(emp.toString());  
        System.out.println("Name : "+emp.getName());  
        System.out.println("Empid : "+emp.getEmpid());  
        System.out.println("Salary : "+emp.getSalary());  
    }  
    public static void main3(String[] args) {  
        Employee emp = new Employee("Sandeep", 33, 35000.50f );  
        System.out.println(emp); //Employee [name=Sandeep,  
empid=33, salary=35000.5]  
    }  
}  
  
//ctrl +1 and import will be add  
//or you can do partial coding Emp Ctrl +space hint enter  
//Sysout ctrl+space and SOP statement gave.  
//for tostring --> emp. Ctrl+space hit ent tostring()
```

Scanner used kiya bas.

```
package com.sunbeaminfo.cj06.model;

public class Employee {
    private String name;
    private int empid;
    private float salary;
    public Employee() {
        // TODO Auto-generated constructor stub
    }
    public Employee(String name, int empid, float salary) {
        this.name = name;
        this.empid = empid;
        this.salary = salary;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getEmpid() {
        return empid;
    }
    public void setEmpid(int empid) {
        this.empid = empid;
    }
    public float getSalary() {
        return salary;
    }
    public void setSalary(float salary) {
        this.salary = salary;
    }
    @Override
    public String toString() {
        return "Employee [name=" + name + ", empid=" + empid + ",
salary=" + salary + "]";
    }
}

package com.sunbeaminfo.cj06.test;
import java.util.Scanner;
```

```
import com.sunbeaminfo.cj06.model.Employee;
public class Program {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        Employee emp = new Employee();
        System.out.print("Enter name:    ");
        emp.setName( sc.nextLine() );
        System.out.print("Enter empid :    ");
        emp.setEmpid( sc.nextInt() );
        System.out.print("Enter salary :    ");
        emp.setSalary( sc.nextFloat() );

        System.out.println("Name : " +emp.getName());
        System.out.println("Empid : " +emp.getEmpid());
        System.out.println("Salary : " +emp.getSalary());
    }
}
//Sca ctrl+space hit enter.
```

AcceptRecord PrintRecord method use kiye bas:-

```
package com.sunbeaminfo.cj06.model;

public class Employee {
    private String name;
    private int empid;
    private float salary;
    public Employee() {
        // TODO Auto-generated constructor stub
    }
    public Employee(String name, int empid, float salary) {
        this.name = name;
        this.empid = empid;
        this.salary = salary;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getEmpid() {
        return empid;
    }
}
```

```
}

public void setEmpid(int empid) {
    this.empid = empid;
}
public float getSalary() {
    return salary;
}
public void setSalary(float salary) {
    this.salary = salary;
}
@Override
public String toString() {
    return "Employee [name=" + name + ", empid=" + empid +",
salary=" + salary + "]";
}
}

package com.sunbeaminfo.cj06.test;
import java.util.Scanner;

public class Program {
    public static void acceptRecord( Employee emp ) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter name: ");
        emp.setName( sc.nextLine() );
        System.out.print("Enter empid : ");
        emp.setEmpid( sc.nextInt() );
        System.out.print("Enter salary : ");
        emp.setSalary( sc.nextFloat() );
    }

    public static void printRecord( Employee emp ) {
        System.out.println("Name : "+emp.getName());
        System.out.println("Empid : "+emp.getEmpid());
        System.out.println("Salary : "+emp.getSalary());
    }

    public static void main(String[] args) {
        Employee emp = new Employee();
        Program.acceptRecord(emp);
        Program.printRecord(emp);
    }
}

//this used nhī kiya tho static karna method.
```

Menu driven program make in this:-

```
package com.sunbeaminfo.cj06.model;

public class Employee {
    private String name;
    private int empid;
    private float salary;
    public Employee() {
        // TODO Auto-generated constructor stub
    }
    public Employee(String name, int empid, float salary) {
        this.name = name;
        this.empid = empid;
        this.salary = salary;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getEmpid() {
        return empid;
    }
    public void setEmpid(int empid) {
        this.empid = empid;
    }
    public float getSalary() {
        return salary;
    }
    public void setSalary(float salary) {
        this.salary = salary;
    }
    @Override
    public String toString() {
        return "Employee [name=" + name + ", empid=" + empid +",
salary=" + salary + "]";
    }
}
```

```
package com.sunbeaminfo.cj06.test;
import java.util.Scanner;
import com.sunbeaminfo.cj06.model.Employee

/*
Stream is abstraction which produces(write) or consume(read)
information to & from source & destnation.
// Following are standard stream object of Java associated with
console:
    1. System.in => Keyborad
    2. System.out => Monitor
    3. System.err => Monitor ( It is error stream. It should be
used to display errors)
*/

public class Program {
    //private static Scanner sc = new Scanner(System.in);
    private static Scanner sc;
    static {
        sc = new Scanner(System.in);
    }
    public static void acceptRecord( Employee emp ) {
        System.out.print("Enter name:    ");
        sc.nextLine(); //To clear the buffer
        emp.setName( sc.nextLine() );
        System.out.print("Enter empid    :    ");
        emp.setEmpid( sc.nextInt() );
        System.out.print("Enter salary    :    ");
        emp.setSalary( sc.nextFloat() );
    }
    public static void printRecord( Employee emp ) {
        System.out.println("Name    :    "+emp.getName());
        System.out.println("Empid   :    "+emp.getEmpid());
        System.out.println("Salary  :    "+emp.getSalary());
    }
    public static int menuList( ) {
        System.out.println("0.Exit.");
        System.out.println("1.Accept Record.");
        System.out.println("2.Print Record.");
        System.out.print("Enter choice    :    ");
        return sc.nextInt();
    }
    public static void main(String[] args) {
        int choice;
```

```
Employee emp = new Employee();
while( ( choice = Program.menuList( ) ) != 0 ) {
    switch( choice ) {
        case 1:
            Program.acceptRecord(emp);
            break;
        case 2:
            Program.printRecord(emp);
            break;
    }
}
}
```

Correct way to write a OOPS program.

```
package com.sunbeaminfo.cj06.test;

public class Program {
    public static void main(String[] args) {
        int choice;
        EmployeeTest test = new EmployeeTest();
        while( ( choice = EmployeeTest.menuList( ) ) != 0 ) {
            switch( choice ) {
                case 1:
                    test.acceptRecord();
                    break;
                case 2:
                    test.printRecord();
                    break;
            }
        }
    }
}
```

```
package com.sunbeaminfo.cj06.test;
import java.util.Scanner;
import com.sunbeaminfo.cj06.model.Employee;

public class EmployeeTest {
    /* private Employee emp;
    public EmployeeTest( ) {
        this.emp = new Employee();
```

```
    } */  
  
    private Employee emp = new Employee();  
    static Scanner sc = new Scanner(System.in);  
    public void acceptRecord( ) {  
        System.out.print("Enter name:    ");  
        sc.nextLine(); //To clear the buffer  
        this.emp.setName( sc.nextLine() );  
        System.out.print("Enter empid :    ");  
        this.emp.setEmpid( sc.nextInt() );  
        System.out.print("Enter salary :    ");  
        this.emp.setSalary( sc.nextFloat() );  
    }  
    public void printRecord( ) {  
        System.out.println("Name : "+this.emp.getName());  
        System.out.println("Empid : "+this.emp.getEmpid());  
        System.out.println("Salary : "+this.emp.getSalary());  
    }  
    public static int menuList( ) {  
        System.out.println("0.Exit.");  
        System.out.println("1.Accept Record.");  
        System.out.println("2.Print Record.");  
        System.out.print("Enter choice : ");  
        return sc.nextInt();  
    }  
}
```

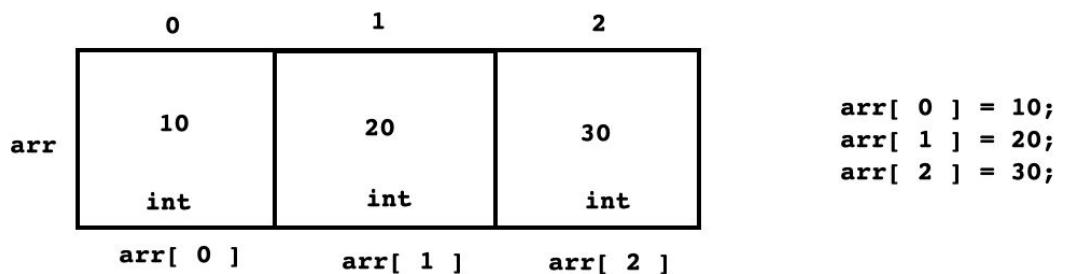
Instead of writing code you can used lambok plugins also

```
package com.sunbeaminfo.cj06.model  
import lombok.AllArgsConstructor;  
import lombok.Getter;  
import lombok.NoArgsConstructor;  
import lombok.Setter;  
import lombok.ToString;  
  
@NoArgsConstructor //Annotation  
@AllArgsConstructor //Annotation  
@Getter @Setter //Annotation  
@ToString //Annotation
```

```
public class Employee {
    private String name;
    private int empid;
    private float salary;
}
```

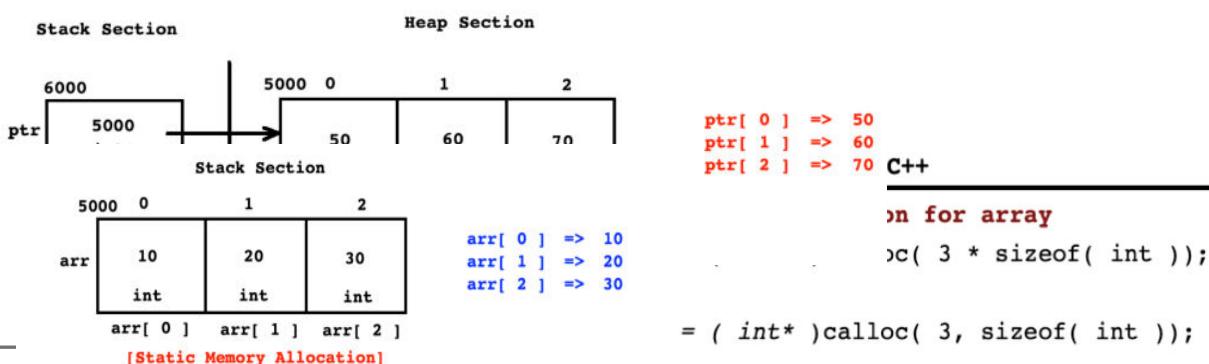
Array

- Data structure is a branch of computer science that we can use to organise data efficiently in RAM.
- Types of data structure:
 1. Linear data structure
 2. Non linear data structure
- Linear/Sequential Data structures:
 - Array, Stack, Queue, LinkedList
- Non linear data structures:
 - Tree, Graph, Hashtable etc
- In Java, data structure is also called as collection.
- data/value stored inside data structure/collection is called element.



- Here 10, 20 and 30 are elements.
- Array is a linear collection in which we can store multiple elements of same type in continuous memory location.

Static vs Dynamic memory allocation in C/C++



Array Declaration and Initialization In C

- int arr[3]; //OK : Declaration
- int arr[3] = { 10, 20, 30 }; //OK : Initialization
- int arr[] = { 10, 20, 30 }; //OK
- int arr[3] = { 10, 20 }; //OK : Partial Initialization
- int arr[3] = { }; //OK : Partial Initialization
- int arr[3] = { 10, 20, 30, 40, 50 }; //Not recommended

Accessing Elements Of Array in C

- If we want to access elements of array then we should use integer index.
- Array index always begins with 0.

```
int arr[ 3 ] = { 10, 20, 30 };
printf("%d\n", arr[ 0 ] );
printf("%d\n", arr[ 1 ] );
printf("%d\n", arr[ 2 ] );
```

```
int arr[ 3 ] = { 10, 20, 30 };
int index;
for( index = 0; index < 3; ++ index )
    printf("%d\n", arr[ index ] );
```

- **Advantage Of Array**

1. We can access elements of array randomly.

- **Disadvantage Of Array**

1. We can not resize array at runtime.
2. It requires continuous memory.
3. Insertion and removal of element from array is a time consuming job.
4. Using assignment operator, we can not copy array into another array.
5. Compiler do not check array bounds(min and max index).

Array In Java

- Array is a reference type in Java. In other words, to create instance of array, new operator is required. It means that array instance get space on heap.

- There are 3 types of array in Java:

1. Single dimensional array
2. Multi dimensional array
3. Ragged array

- Types of loop in Java:

1. do-while loop
2. while loop
3. for loop
4. for-each loop

- To perform operations on array we can use following classes:

1. java.util.Arrays
2. org.apache.commons.lang
3. ArrayUtils(download .jar file)

Methods of java.util.Arrays Class

Following are the methods of java.util.Arrays class.(try javap java.util.Arrays)

```
- public static <T> List<T> asList(T... a)
- public static int binarySearch(int[] a, int key) //Overloaded
- public static int binarySearch(Object[] a, Object key)
- public static int[] copyOf(int[] original, int newLength)
- public static <T> T[] copyOf(T[] original, int newLength)
- public static int[] copyOfRange(int[] original, int from, int to)
- public static <T> T[] copyOfRange(T[] original, int from, int to)
- public static void fill(int[] a, int val)
- public static void fill(Object[] a, Object val)
- public static void fill(Object[] a, int fromIndex, int toIndex, Object val)
- public static void sort(int[] a) //Overloaded
- public static void sort(Object[] a)
- public static void parallelSort(int[] a)
- public static <T extends Comparable<? super T>> void parallelSort(T[] a)
- public static String toString(Object[] a) //Overloaded
- public static String deepToString(Object[] a)
- public static IntStream stream(int[] array) //Overloaded
- public static <T> Stream<T> stream(T[] array)
```

```
public class Program {  
    public static void main1(String[] args) {  
        //int arr[ 3 ]; //Static memory allocation in C/C++  
  
        //int arr[ 3 ]; //Not OK : In Java  
  
        int arr1[ ] = null; //OK : Reference  
  
        //int [ arr2 ] = null; //NOT OK  
  
        int[ ] arr3 = null; //OK : Reference  
    }  
    public static void main2(String[] args) {  
        int arr1[ ] = null; //OK  
        arr1 = new int[ 3 ]; //OK  
  
        int[] arr2 = null; //OK  
        arr2 = new int[ 3 ]; //OK  
  
        int[] arr3 = new int[ 3 ]; //OK  
    }  
    public static void main(String[] args) {  
        int size = -3;  
        int[] arr = new int[ size ]; //NegativeArraySizeException  
        //Thrown if an application tries to create an array with  
negative size.  
    }  
}
```

public class NegativeArraySizeException
extends RuntimeException

Thrown if an application tries to create an array with negative size.

Since:

JDK1.0

See Also:

Serialized Form

Initialization

```
int[] arr = new int[ size ]{ 10, 20, 30 }; //Not OK  
int[] arr = new int[ ]{ 10, 20, 30 }; //OK  
int[] arr = { 10, 20, 30 }; //OK
```

package com.sunbeaminfo.cj06.test;

```
/*
 If this class object represents a class of arrays, then the
 internal form of the name consists of the name of the element type
 preceded by one or more '[' characters representing the depth of
 the array nesting. The encoding of element type names is as
 follows:
Element Type          Encoding
 boolean               Z
 byte                 B
 char                 C
 class or interface   Lclassname;
 double               D
 float                F
 int                  I
 long                 J
 short               S

*/
public class Program {
    public static void main1(String[] args) {
        int[] arr = new int[ 3 ];      //OK
        System.out.println( arr[ 0 ] ); //0
        System.out.println( arr[ 1 ] ); //0
        System.out.println( arr[ 2 ] ); //0
    }
    public static void main2(String[] args) {
        int[] arr = new int[ 3 ];      //OK
        for( int index = 0; index < 3; ++ index )
            System.out.println( arr[ index ] ); //0
    }
    public static void main(String[] args) {
        boolean[] arr1 = new boolean[ 3 ];
        System.out.println(arr1.toString()); // [Z@6d06d69c
    }
}

byte[] arr2 = new byte[ 3 ];
System.out.println(arr2.toString()); // [B@7852e922

char[] arr3 = new char[ 3 ];
System.out.println(arr3.toString()); // [C@4e25154f

short[] arr4 = new short[ 3 ];
System.out.println(arr4.toString()); // [S@70dea4e
```

```
int[] arr5 = new int[ 3 ]; //OK
System.out.println(arr5.toString()); //[@5c647e05

int[][] arr6 = new int[ 3 ][ 3 ]; //OK
System.out.println(arr6.toString()); //[[I@33909752

int[][][] arr = new int[ 3 ][ 3 ][ 3 ]; //OK
System.out.println(arr.toString()); //[[[I@55f96302

float[] arr7 = new float[ 3 ]; //OK
System.out.println(arr7.toString()); //[@55f96302

double[] arr8 = new double[ 3 ]; //OK
System.out.println(arr8.toString()); //[@3d4eac69

long[] arr9 = new long[ 3 ]; //OK
System.out.println(arr9.toString()); //[@42a57993
}
}
```

static String	toString(boolean[] a)
	Returns a string representation of the contents of the specified array.
static String	toString(byte[] a)
	Returns a string representation of the contents of the specified array.
static String	toString(char[] a)
	Returns a string representation of the contents of the specified array.
static String	toString(double[] a)
	Returns a string representation of the contents of the specified array.
static String	toString(float[] a)
	Returns a string representation of the contents of the specified array.
static String	toString(int[] a)
	Returns a string representation of the contents of the specified array.
static String	toString(long[] a)
	Returns a string representation of the contents of the specified array.
static String	toString(Object[] a)
	Returns a string representation of the contents of the specified array.
static String	toString(short[] a)
	Returns a string representation of the contents of the specified array.

```
package com.sunbeaminfo.cj06.test;

import java.util.Arrays;
import java.util.Objects;
import java.util.Scanner;

public class Program {
    private static Scanner sc = new Scanner(System.in);
    /* private static void acceptRecord(int[] arr) {
        if( arr != null ) {
            for( int index = 0; index < 3; ++ index ) {
                System.out.print("Enter element :   ");
                arr[ index ] = sc.nextInt();
            }
        }
    }
    private static void printRecord(int[] arr) {
        if( arr != null ) {
            for( int index = 0; index < 3; ++ index )
                System.out.println(arr[ index ]);
        }
    } */ //isse comment nhi kiya tho code run hi nhi hota kyuki
humne null kiya hai so solution niche hai iska
    private static void acceptRecord(int[] arr) {
        //in java 7 the new feature objects naam ka class hai usme
requireNonNull ki method hai
        //ye khudse check krengi ki null hai ya nhi.
        //ye line nhi likhi tho program run honga aur element dalne
ke baad error ayenga.
        arr = Objects.requireNonNull(arr, "inside acceptRecord
method, arr must not be null");
        for( int index = 0; index < 3; ++ index ) {
            System.out.print("Enter element :   ");
            arr[ index ] = sc.nextInt();
        }
    }
    private static void printRecord(int[] arr) {
        arr = Objects.requireNonNull(arr, "inside printRecord
method, arr must not be null");
        for( int index = 0; index < 3; ++ index )
```

```
        System.out.println(arr[ index ]);  
    }  
    public static void main1(String[] args) { //default value  
        int[] arr = new int[ 3 ];  
        String str = Arrays.toString(arr);  
        System.out.println(str); // [0, 0, 0]  
    }  
    public static void main2(String[] args) {  
        int[] arr = new int[ 3 ];  
        arr[ 0 ] = 10; //hardcore value  
        arr[ 1 ] = 20;  
        arr[ 2 ] = 30;  
  
        System.out.println( Arrays.toString(arr));  
  
        /* for( int index = 0; index < 3; ++ index )  
            System.out.println(arr[ index ]); */  
  
        /* System.out.println(arr[ 0 ]);  
        System.out.println(arr[ 1 ]);  
        System.out.println(arr[ 2 ]); */  
  
    }  
    public static void main3(String[] args) {  
        int[] arr = new int[ 3 ]; //user input value  
  
        for( int index = 0; index < 3; ++ index ) {  
            System.out.print("Enter element : ");  
            arr[ index ] = sc.nextInt();  
        }  
  
        for( int index = 0; index < 3; ++ index )  
            System.out.println(arr[ index ]);  
    }  
    public static void main4(String[] args) { //two method written  
and just called them  
        int[] arr = new int[ 3 ];  
        Program.acceptRecord( arr );  
        Program.printRecord( arr );  
    }  
}
```

```
public static void main(String[] args) {  
    int[] arr = null;  
    Program.acceptRecord( arr );  
    Program.printRecord( arr );  
}  
}
```

requireNonNull

```
public static <T> T requireNonNull(T obj,  
                                    String message)
```

Checks that the specified object reference is not null and throws a customized `NullPointerException` if it is. This method is designed primarily for doing parameter validation in methods and constructors with multiple parameters, as demonstrated below:

```
public Foo(Bar bar, Baz baz) {  
    this.bar = Objects.requireNonNull(bar, "bar must not be null");  
    this.baz = Objects.requireNonNull(baz, "baz must not be null");  
}
```

Type Parameters:

T - the type of the reference

Parameters:

obj - the object reference to check for nullity

message - detail message to be used in the event that a `NullPointerException` is thrown

Returns:

obj if not null

Throws:

`NullPointerException` - if obj is null

Mavan used in eclipse like lambok jar file.

Using `toString()`Method :-

```
package com.sunbeaminfo.cj06.test;
```

```
import java.util.Arrays;  
import java.util.Objects;  
import java.util.Scanner;
```

```
public class Program {  
    public static void main(String[] args) {  
        //int[] arr = new int[ ] { 10, 20, 30 }; //OK  
  
        int[] arr = { 10, 20, 30 }; //OK  
        System.out.println(Arrays.toString(arr));  
    }  
    public static void main2(String[] args) {  
        //int[] arr = new int[ 3 ] { 10, 20, 30 }; //Not OK
```

```
//"{ 10, 20, 30 }" is called as initializer

    int[] arr = new int[ ] { 10, 20, 30 }; //OK
    System.out.println(Arrays.toString(arr));
}

public static void main1(String[] args) {
    int[] arr = new int[ 3 ]; //OK
    System.out.println(Arrays.toString(arr)); // [0, 0, 0]
}

```

ArrayIndexOutOfBoundsException :-

```
package com.sunbeaminfo.cj06.test;
public class Program {
    public static void main1(String[] args){
        int[ ] arr = new int[ ] { 10, 20, 30, 40, 50 };
        System.out.println(arr.length); //5

        int element = arr[ 0 ];
        System.out.println( element ); //10

        element = arr[ 2 ];
        System.out.println( element ); //30

        element = arr[ 4 ];
        System.out.println( element ); //50
    }

    public static void main(String[] args){
        int[ ] arr = new int[ ] { 10, 20, 30, 40, 50 };
        System.out.println(arr.length); //5

        //int element = arr[ -1 ]; //ArrayIndexOutOfBoundsException
        //System.out.println( element );

        //int element = arr[ 5 ]; //ArrayIndexOutOfBoundsException
        //System.out.println( element );

        int element = arr[ arr.length ]; //ArrayIndexOutOfBoundsException
        System.out.println( element );
    }
}
```

```
//Thrown to indicate that an array has been accessed with an
illegal index.

//The index is either negative or greater than or equal to the
size of the array.

}

}

public class ArrayIndexOutOfBoundsException
extends IndexOutOfBoundsException
```

Thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

Since:

JDK1.0

See Also:

[Serialized Form](#)

Reference Copy and Instance copy:-

copyOf

```
public static int[] copyOf(int[] original,
                           int newLength)
```

Copies the specified array, truncating or padding with zeros (if necessary) so the copy has the specified length. For all indices that are valid in both the original array and the copy, the two arrays will contain identical values. For any indices that are valid in the copy but not the original, the copy will contain 0. Such indices will exist if and only if the specified length is greater than that of the original array.

Parameters:

original - the array to be copied

newLength - the length of the copy to be returned

Returns:

a copy of the original array, truncated or padded with zeros to obtain the specified length

Throws:

[NegativeArraySizeException](#) - if newLength is negative

[NullPointerException](#) - if original is null

Since:

1.6

copyOfRange

```
public static int[] copyOfRange(int[] original,
                               int from,
                               int to)
```

Copies the specified range of the specified array into a new array. The initial index of the range (from) must lie between zero and original.length, inclusive. The value at original[from] is placed into the initial element of the copy (unless from == original.length or from == to). Values from subsequent elements in the original array are placed into subsequent elements in the copy. The final index of the range (to), which must be greater than or equal to from, may be greater than original.length, in which case 0 is placed in all elements of the copy whose index is greater than or equal to original.length - from. The length of the returned array will be to - from.

Parameters:

original - the array from which a range is to be copied

from - the initial index of the range to be copied, inclusive

to - the final index of the range to be copied, exclusive. (This index may lie outside the array.)

Returns:

a new array containing the specified range from the original array, truncated or padded with zeros to obtain the required length

Throws:

[ArrayIndexOutOfBoundsException](#) - if from < 0 or from > original.length

[IllegalArgumentException](#) - if from > to

[NullPointerException](#) - if original is null

Since:

1.6

```

package com.sunbeaminfo.cj06.test;
import java.util.Arrays;

public class Program {

    public static void main1(String[] args) {
        int[] arr1 = new int[] { 11, 22, 33, 44, 55 };

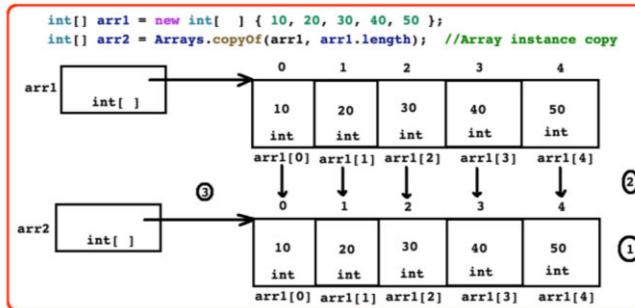
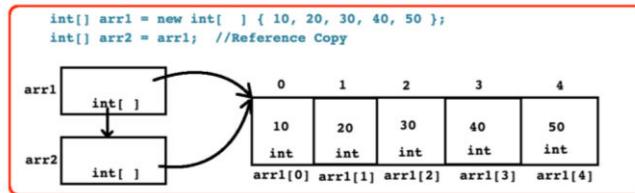
        int[] arr2 = Arrays.copyOf(arr1, arr1.length);
        System.out.println(Arrays.toString(arr2));

        arr2 = Arrays.copyOf(arr1, arr1.length + 2);
        System.out.println(Arrays.toString(arr2));
    }

    public static void main(String[] args) {
        int[] arr1 = new int[] { 11, 22, 33, 44, 55, 66, 77, 88, 99 };
        int[] arr2 = Arrays.copyOfRange(arr1, 2, 5);
        System.out.println(Arrays.toString(arr2));
    }
}

```

<pre>Array Reference copy</pre> <pre> int[] arr1 = new int[] { 10, 20, 30, 40, 50 }; int[] arr2 = arr1; //Reference Copy </pre>
<pre>Array Instance Copy(Using Arrays.copyOf())</pre> <pre> int[] arr1 = new int[] { 10, 20, 30, 40, 50 }; int[] arr2 = Arrays.copyOf(arr1, arr1.length); //Array instance copy </pre>



Sorting Arrays Elements:-

Core Java Notes By ASHOK PATE

```
package com.sunbeaminfo.cj06.test;
import java.util.Arrays;

public class Program {
    public static void main(String[] args) {
        int[] arr = new int[] { 5, 1, 4, 2, 3 };
        System.out.println(Arrays.toString(arr)); // [5, 1, 4, 2, 3]
        Arrays.sort(arr); // DualPivotQuicksort
        System.out.println(Arrays.toString(arr)); // [1, 2, 3, 4, 5]
    }
}
```

sort

```
public static void sort(int[] a)
```

Sorts the specified array into ascending numerical order.

Implementation note: The sorting algorithm is a Dual-Pivot Quicksort by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch. This algorithm offers $O(n \log(n))$ performance on many data sets that cause other quicksorts to degrade to quadratic performance, and is typically faster than traditional (one-pivot) Quicksort implementations.

Parameters:

a - the array to be sorted

sort

```
public static void sort(int[] a,
                      int fromIndex,
                      int toIndex)
```

Sorts the specified range of the array into ascending order. The range to be sorted extends from the index `fromIndex`, inclusive, to the index `toIndex`, exclusive. If `fromIndex == toIndex`, the range to be sorted is empty.

Implementation note: The sorting algorithm is a Dual-Pivot Quicksort by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch. This algorithm offers $O(n \log(n))$ performance on many data sets that cause other quicksorts to degrade to quadratic performance, and is typically faster than traditional (one-pivot) Quicksort implementations.

Parameters:

a - the array to be sorted

fromIndex - the index of the first element, inclusive, to be sorted

toIndex - the index of the last element, exclusive, to be sorted

Throws:

`IllegalArgumentException` - if `fromIndex > toIndex`

`ArrayIndexOutOfBoundsException` - if `fromIndex < 0` or `toIndex > a.length`

For each Loop:-

```
package com.sunbeaminfo.cj06.test;

import java.util.Arrays;

public class Program {
    public static void main1(String[] args) {
        int[] arr = new int[ ] { 10, 20, 30 };
        int index = 0;
        do {
            System.out.println(arr[ index ] );
        }
    }
}
```

```
        ++ index;
    }while( index < arr.length );
}
public static void main2(String[] args) {
    int[] arr = new int[ ] { 10, 20, 30 };
    int index = 0;
    while( index < arr.length ){
        System.out.println(arr[ index ] );
        ++ index;
    }
}
public static void main3(String[] args) {
    int[] arr = new int[ ] { 10, 20, 30 };
    for( int index = 0; index < arr.length; ++ index )
        System.out.println(arr[ index ] );
}
public static void main(String[] args) {
    int[] arr = new int[ ] { 10, 20, 30 };

    //foreach element in arr
    for( int element : arr ) { //For each loop
        System.out.println(element);
    }
    //for each loop is also called as iterator
    //for each loop is read-only and forward-only
}
}
```

Arrays Of Primitive Values:-

```
package com.sunbeaminfo.cj06.test;

import java.util.Arrays;

public class Program {
    public static void main(String[] args) {
        boolean[] arr1 = new boolean[ 3 ];
        System.out.println(Arrays.toString(arr1)); // [false,
false, false]
```

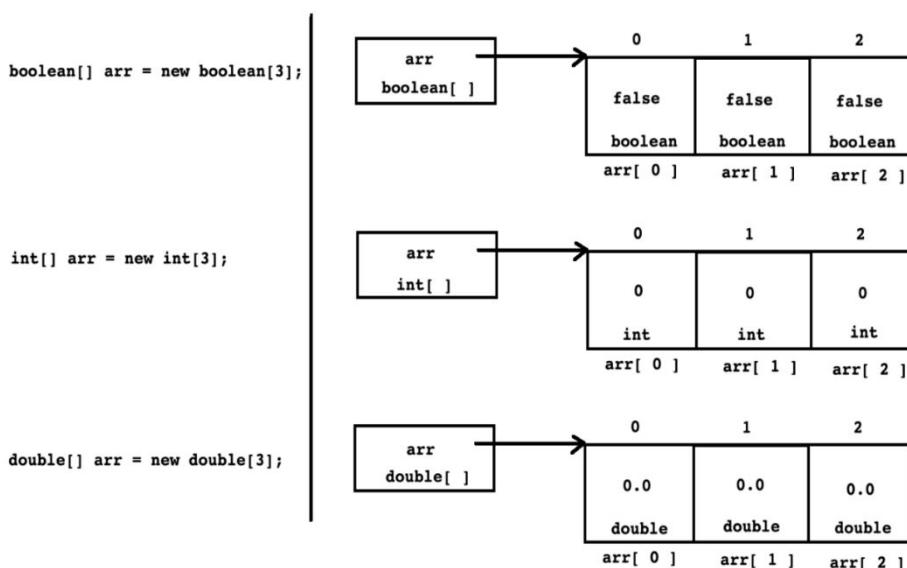
```

int[] arr2 = new int[ 3 ];
System.out.println(Arrays.toString(arr2)); // [0, 0, 0]

double[] arr3 = new double[ 3 ];
System.out.println(Arrays.toString(arr3)); // [0.0, 0.0,
0.0]
}

//if we create arr to store value of primitive type then their
default value
//depends on default type of data type.

```



If we create array of primitive values then it's default value depends of default value of data type.

Array Of References and Instances :-

```
package com.sunbeaminfo.cj06.test;
```

```
import java.util.Arrays;
```

```

class Date{
    private int day, month, year;
    public void printRecord( ) {
        System.out.println(this.day+ " / "+this.month+ " / "+this.year);
    }
}
public class Program {
    public static void main1(String[] args) {
        Date dt1 = null;

```

```
Date dt2 = null;
Date dt3 = null;

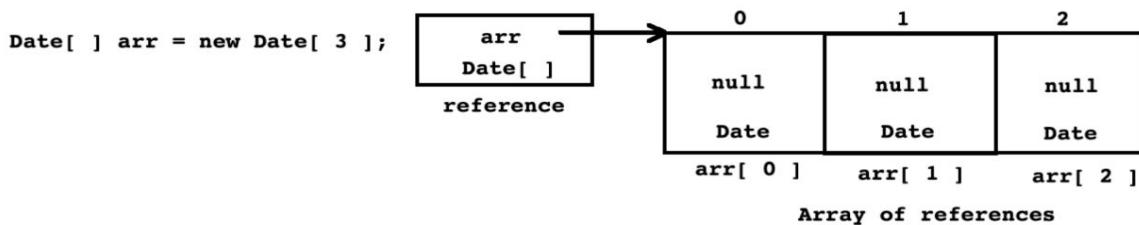
Date[] arr = new Date[ 3 ]; //Array of references
System.out.println(Arrays.toString(arr)); // [null, null,
null]
}

public static void main2(String[] args) {
    Date[] arr = new Date[ 3 ]; //Array of references
    for( int index = 0; index < arr.length; ++ index )
        arr[ index ].printRecord(); //NullPointerException
}
public static void main3(String[] args) {
    Date[] arr = new Date[ 3 ]; //Array of references
    arr[ 0 ] = new Date( );
    arr[ 1 ] = new Date( );
    arr[ 2 ] = new Date( );

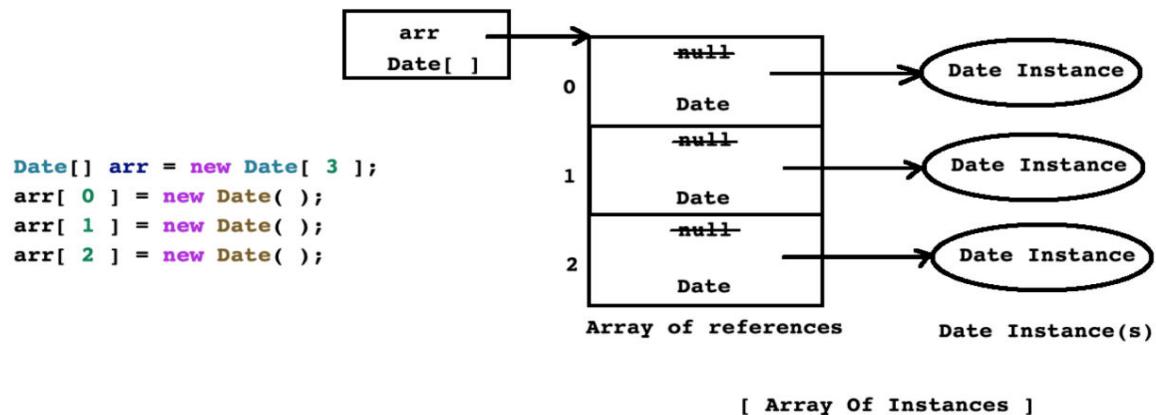
    for( int index = 0; index < arr.length; ++ index )
        arr[ index ].printRecord(); //OK
}
public static void main(String[] args) {
    //Array of instances

    Date[] arr = new Date[ 3 ]; //Array of references
    for( int index = 0; index < arr.length; ++ index )
        arr[ index ] = new Date( );

    for( int index = 0; index < arr.length; ++ index )
        arr[ index ].printRecord(); //OK
}
}
```



If we create an array of references then by default it contains null.



Arrays Of references

```
public class Program {
    public static void main(String[] args) {
        Date[] arr = new Date[ 3 ]; //Contains all null
    }
}
```

- Let us see how to create array of instances of non primitive type

```
public class Program {
    public static void main(String[] args) {
        Date[] arr = new Date[ 3 ];
        arr[ 0 ] = new Date( );
        arr[ 1 ] = new Date( );
        arr[ 2 ] = new Date( );
    }
    //or
    public static void main(String[] args) {
        Date[] arr = new Date[ 3 ];
        for( int index = 0; index < arr.length; ++ index )
            arr[ index ] = new Date( );
    }
}
```

Example 1:-

```
package com.sunbeaminfo.cj06.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@NoArgsConstructor
@AllArgsConstructor
@Getter @Setter
public class Employee {
    private String name;
    private int empid;
    private float salary;
    @Override
    public String toString() {
        return String.format("%-30s%-5d%-10.2f", this.name,
this.empid, this.salary);
    }
}

package com.sunbeaminfo.cj06.test;

import java.util.Arrays;

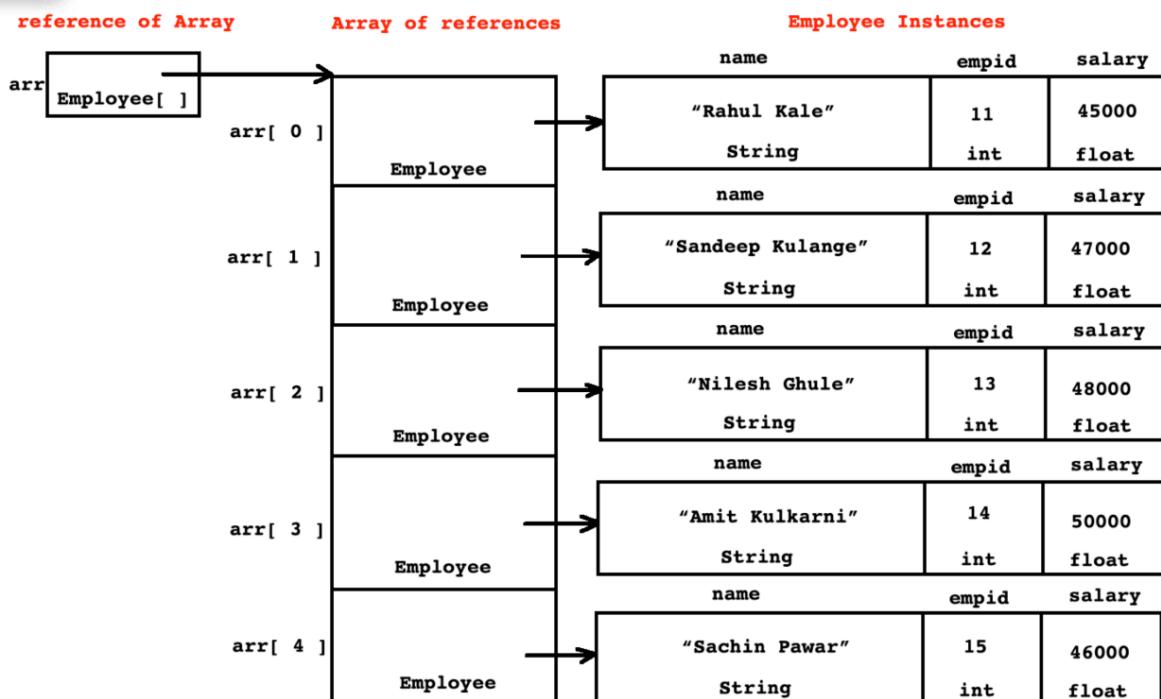
import com.sunbeaminfo.cj06.model.Employee;

public class Program {
    private static Employee[] getEmployees() {
        Employee[] arr = new Employee[ 5 ];
        arr[ 0 ] = new Employee("Rahul Kale",11, 45000);
        arr[ 1 ] = new Employee("Sandeep Kulange",12,47000);
        arr[ 2 ] = new Employee("Nilesh Ghule",13,48000);
        arr[ 3 ] = new Employee("Amit Kulkarni",14,50000);
        arr[ 4 ] = new Employee("Sachin Pawar",15,46000);
        return arr;
    }
}
```

```

    }
    private static void printRecord(Employee[] arr) {
        if( arr != null )
            for (Employee emp : arr) {
                System.out.println(emp.toString());
            }
    }
    public static void main(String[] args) {
        Employee[] arr = Program.getEmployees();
        Program.printRecord( arr );
    }
}

```



Multi Dimensional Array :-

- Array of elements where each element is array of same column size is called as multi dimensional array.

```

package com.sunbeaminfo.cj06.test;

import java.util.Objects;
import java.util.Scanner;

```

```
public class Program {

    static Scanner sc = new Scanner(System.in);
    private static void acceptRecord(int[][] arr) {
        arr = Objects.requireNonNull(arr, "arr must not be null");
        for( int row = 0; row < arr.length; ++ row ) {
            for( int col = 0; col < arr[ row ].length; ++ col ) {
                System.out.print("Enter element :   ");
                arr[ row ][ col ] = sc.nextInt();
            }
        }
    }
    private static void printRecord(int[][] arr) {
        arr = Objects.requireNonNull(arr, "arr must not be null");
        for( int row = 0; row < arr.length; ++ row ) {
            for( int col = 0; col < arr[ row ].length; ++ col ) {
                System.out.print(arr[ row ][ col ]+ "   ");
            }
            System.out.println();
        }
    }
    public static void main1(String[] args) {
        int arr1[][] = null;      //OK

        int[] arr2[] = null;      //OK

        int[][] arr3 = null;      //OK : Recommended
    }
    public static void main2(String[] args) {
        int[][] arr1 = null;
        arr1 = new int[ 4 ][ 3 ];  //OK

        int[][] arr2 = new int[ 4 ][ 3 ];
    }
    public static void main3(String[] args) {
        int[][] arr = new int[ 4 ][ 3 ];

        for( int row = 0; row < 4; ++ row ) {
            for( int col = 0; col < 3; ++ col ) {
                System.out.print(arr[ row ][ col ]+ "   ");
            }
        }
    }
}
```

```

        }
        System.out.println();
    }
}

public static void main4(String[] args) {
    int[][] arr = new int[ 4 ][ 3 ];

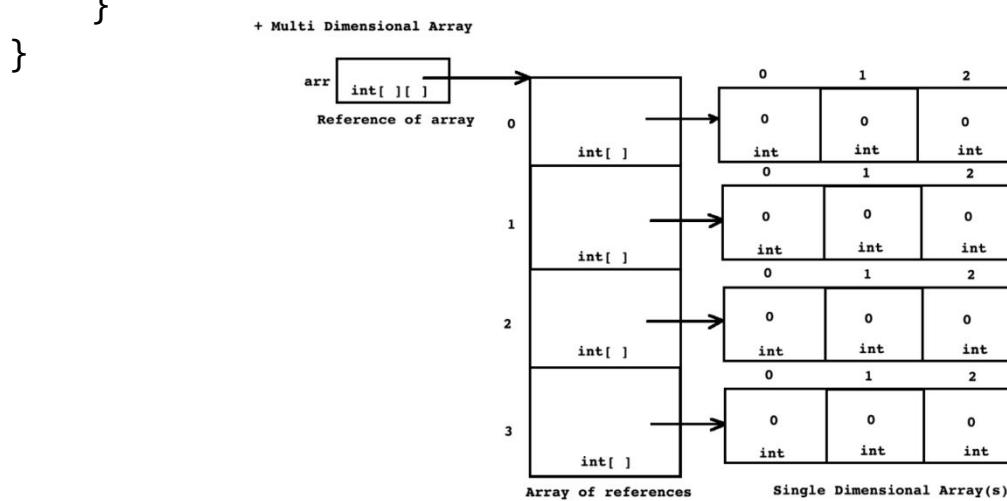
    for( int row = 0; row < arr.length; ++ row ) {
        for( int col = 0; col < arr[ row ].length; ++ col ) {
            System.out.print("Enter element :   ");
            arr[ row ][ col ] = sc.nextInt();
        }
    }

    for( int row = 0; row < arr.length; ++ row ) {
        for( int col = 0; col < arr[ row ].length; ++ col ) {
            System.out.print(arr[ row ][ col ]+ "   ");
        }
    }
    System.out.println();
}

public static void main(String[] args) {
    int[][] arr = new int[ 4 ][ 3 ];
    Program.acceptRecord( arr );
    Program.printRecord( arr );

}

```



```
package com.sunbeaminfo.cj06.test;

import java.util.Arrays;
import java.util.Objects;
import java.util.Scanner;

public class Program {
    public static void main1(String[] args) {
        //int[][] arr = new int[ 4 ][ 3 ]; //OK
        //int[][] arr = new int[ ][ ] {
{1,2,3},{4,5,6},{7,8,9},{10,11,12}}; //OK
        //int[][] arr = { {1,2,3},{4,5,6},{7,8,9},{10,11,12}}; //OK
        //int[][] arr = new int[ 4 ][ 3 ] {
{1,2,3},{4,5,6},{7,8,9},{10,11,12}}; //Not OK
    }

    public static void main2(String[] args) {
        int[][] arr = new int[ ][ ] {
{1,2,3},{4,5,6},{7,8,9},{10,11,12}}; //OK

        for( int row = 0; row < arr.length; ++ row ) {
            for( int col = 0; col < arr[ row ].length; ++ col ) {
                System.out.print(arr[ row ][ col ]+ "   ");
            }
            System.out.println();
        }
    }

    public static void main3(String[] args) {
        int[][] arr = new int[ ][ ] {
{1,2,3},{4,5,6},{7,8,9},{10,11,12}}; //OK
        for( int[] row : arr ) {
            for( int col : row ) {
                System.out.print(col+ "   ");
            }
            System.out.println();
        }
    }

    public static void main4(String[] args) {
```

```
int[][] arr = new int[ ][ ] {  
{1,2,3},{4,5,6},{7,8,9},{10,11,12}}; //OK  
//System.out.println(arr.toString()); //[[I@6d06d69c  
  
//System.out.println(Arrays.toString(arr)); //[[I@6d06d69c,  
[I@7852e922, [I@4e25154f, [I@70dea4e]  
  
/* for( int row = 0; row < arr.length; ++ row )  
    System.out.println(Arrays.toString(arr[ row ] ) ); */  
  
//System.out.println(Arrays.deepToString(arr)); //This method  
is designed for converting multidimensional arrays to strings.
```

```
for( int[] row : arr ) {  
    System.out.println(Arrays.toString( row ));  
}  
}  
public static void main(String[] args) {  
    int[] arr = new int[ 3 ];  
    System.out.println(arr.getClass().getSuperclass().getName());  
//java.lang.Object  
}  
}
```

deepToString

public static String deepToString(Object[] a)

Returns a string representation of the "deep" contents" of the specified array. If the array contains other arrays as elements, the string representation contains their contents and so on. This method is designed for converting multidimensional arrays to strings.

The string representation consists of a list of the array's elements, enclosed in square brackets ("[]"). Adjacent elements are separated by the characters ", " (a comma followed by a space). Elements are converted to strings as by `String.valueOf(Object)`, unless they are themselves arrays.

If an element e is an array of a primitive type, it is converted to a string as by invoking the appropriate overloading of `Arrays.toString(e)`. If an element e is an array of a reference type, it is converted to a string as by invoking this method recursively.

To avoid infinite recursion, if the specified array contains itself as an element, or contains an indirect reference to itself through one or more levels of arrays, the self-reference is converted to the string "[...]" . For example, an array containing only a reference to itself would be rendered as "[[...]]".

This method returns "null" if the specified array is null.

Parameters:

a - the array whose string representation to return

Returns:

a string representation of a

Since:

1.5

See Also:

`toString(Object[])`

Ragged Array :-

- A multidimensional array where column size of every array is different.

```
package com.sunbeaminfo.cj06.test;
import java.util.Scanner;

public class Program {
    static Scanner sc = new Scanner(System.in);
    public static int[][] getArrayInstance() {
        int[][] arr = new int[ 4 ][ ];//Array of references
        arr[ 0 ] = new int[ 3 ];
        arr[ 1 ] = new int[ 2 ];
        arr[ 2 ] = new int[ 4 ];
        arr[ 3 ] = new int[ 3 ];
        return arr;
    }
    private static void acceptRecord(int[][] arr) {
        for( int row = 0; row < arr.length; ++ row ) {
            for( int col = 0; col < arr[ row ].length; ++ col ) {
                System.out.print("Enter element :   ");
                arr[ row ][ col ] = sc.nextInt();
            }
        }
    }
    private static void printRecord(int[][] arr) {
        for( int row = 0; row < arr.length; ++ row ) {
            for( int col = 0; col < arr[ row ].length; ++ col ) {
                System.out.printf("%-5d",arr[ row ][ col ]);
            }
            System.out.println();
        }
    }
    public static void main1(String[] args) {
        int arr1[][] = null; //OK
        int []arr2[] = null; //OK
        int [][]arr3 = null; //OK
    }
}
```

```
}

public static void main2(String[] args) {
    int[][] arr = new int[ 4 ][  ] ; //Array of references
    arr[ 0 ] = new int[ 3 ];
    arr[ 1 ] = new int[ 2 ];
    arr[ 2 ] = new int[ 4 ];
    arr[ 3 ] = new int[ 3 ];
}

public static void main3(String[] args) {
    int[][] arr = new int[ 4 ][  ] ; //Array of references
    arr[ 0 ] = new int[ 3 ];
    arr[ 1 ] = new int[ 2 ];
    arr[ 2 ] = new int[ 4 ];
    arr[ 3 ] = new int[ 3 ];

    for( int row = 0; row < arr.length; ++ row ) {
        for( int col = 0; col < arr[ row ].length; ++ col ) {
            System.out.printf("%-5d",arr[ row ][ col ]);
        }
        System.out.println();
    }
}

public static void main4(String[] args) {
    int[][] arr = new int[ 4 ][  ] ; //Array of references
    arr[ 0 ] = new int[ 3 ];
    arr[ 1 ] = new int[ 2 ];
    arr[ 2 ] = new int[ 4 ];
    arr[ 3 ] = new int[ 3 ];

    for( int row = 0; row < arr.length; ++ row ) {
        for( int col = 0; col < arr[ row ].length; ++ col ) {
            System.out.print("Enter element :   ");
            arr[ row ][ col ] = sc.nextInt();
        }
    }

    for( int row = 0; row < arr.length; ++ row ) {
        for( int col = 0; col < arr[ row ].length; ++ col ) {
            System.out.printf("%-5d",arr[ row ][ col ]);
        }
    }
}
```

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

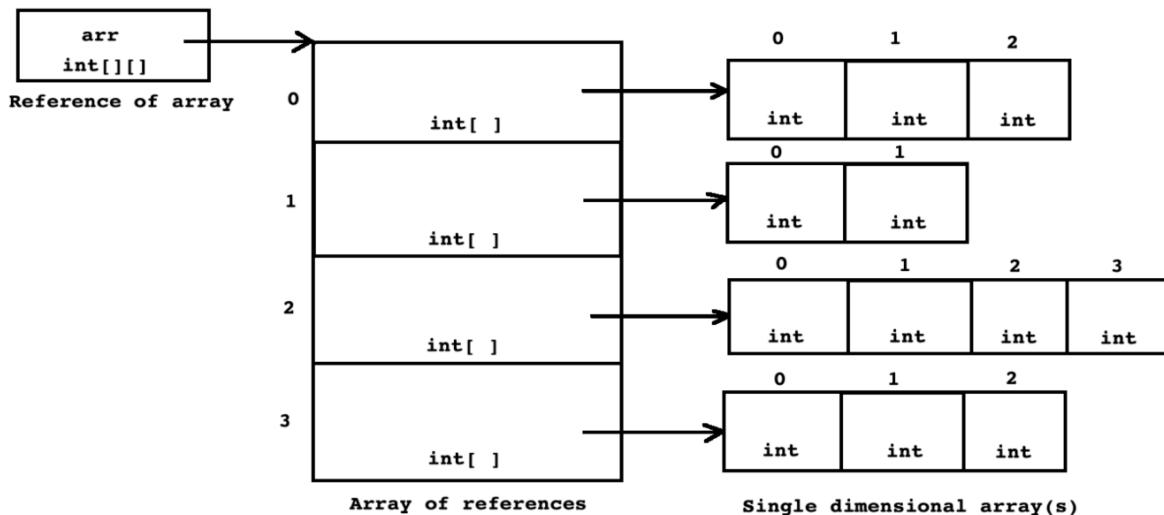
public static void main(String[] args) {

    int[][] arr = Program.getArrayInstance();

    Program.acceptRecord( arr );

    Program.printRecord( arr );
}
}
```

+ Ragged Array



```
package com.sunbeaminfo.cj06.test;

import java.util.Arrays;
import java.util.Scanner;

public class Program {
    public static void main1(String[] args) {
        int[][] arr = new int[4][];
        arr[0] = new int[] { 1, 2, 3 };
        arr[1] = new int[] { 4, 5 };
        arr[2] = new int[] { 6, 7, 8, 9 };
        arr[3] = new int[] { 10, 11, 12 };
    }
}
```

```
        System.out.println(Arrays.deepToString(arr));
    }

    public static void main(String[] args) {
        //int[][] arr = new int[ ][ ] {
{1,2,3},{4,5},{6,7,8,9},{10,11,12}};
        int[][] arr = { {1,2,3},{4,5},{6,7,8,9},{10,11,12} };
        System.out.println(Arrays.deepToString(arr));
    }
}
```

deepToString

public static String deepToString(Object[] a)

Returns a string representation of the "deep" contents of the specified array. If the array contains other arrays as elements, the string representation contains their contents and so on. This method is designed for converting multidimensional arrays to strings.

The string representation consists of a list of the array's elements, enclosed in square brackets ("[]"). Adjacent elements are separated by the characters ", " (a comma followed by a space). Elements are converted to strings as by `String.valueOf(Object)`, unless they are themselves arrays.

If an element e is an array of a primitive type, it is converted to a string as by invoking the appropriate overloading of `Arrays.toString(e)`. If an element e is an array of a reference type, it is converted to a string as by invoking this method recursively.

To avoid infinite recursion, if the specified array contains itself as an element, or contains an indirect reference to itself through one or more levels of arrays, the self-reference is converted to the string "[...]" . For example, an array containing only a reference to itself would be rendered as "[[...]]".

This method returns "null" if the specified array is null.

Parameters:

a - the array whose string representation to return

Returns:

a string representation of a

Since:

1.5

See Also:

`toString(Object[])`

Argument Passing Methods :-

- In C programming language, we can pass argument to the function using 2 ways:
 1. By value.
 2. By address
- In C++ programming language, we can pass argument to the function using 3 ways:
 1. By value.
 2. By address
 3. By reference

- In Java programming language, we can pass argument to the method using a way:

1. By value.

➤ In other word, every variable of primitive type/non primitive type is pass to the method by value only.

```
package com.sunbeaminfo.cj06.test;
public class Program {
    public static void swap( int a, int b ) { //a & b are method
parameters / parameters
        int temp = a;
        a = b;
        b = temp;
    }
    public static void main(String[] args) {
        int x = 10;
        int y = 20;

        Program.swap(x, y); //x & y are method arguments / arguments

        System.out.println("x : "+x); //10
        System.out.println("y : "+y); //20
    }
}
//In this no swapping done only by using array we can swap Pass by
reference code is below.
```

Simulation Of Pass By Reference in Java:-

```
package com.sunbeaminfo.cj06.test;
public class Program {
    public static void swap( int[] arr ) { //a & b are method
parameters / parameters
        int temp = arr[ 0 ];
        arr[ 0 ] = arr[ 1 ];
        arr[ 1 ] = temp;
    }
    public static void main(String[] args) {
        int x = 10;
        int y = 20;

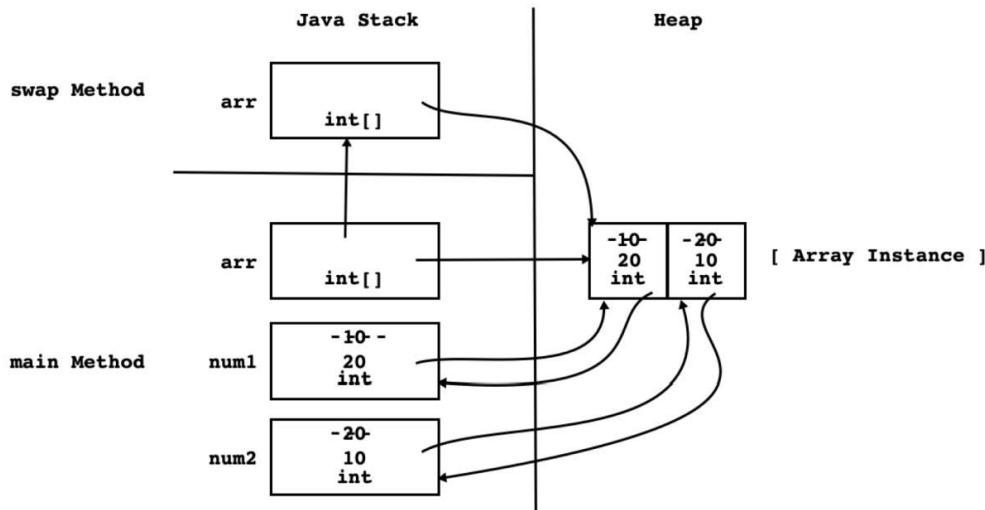
        int[] arr = new int[ ] { x, y };
    }
}
```

```

        Program.swap(arr);
        x = arr[ 0 ]; y = arr[ 1 ];

        System.out.println("x : "+x); //20
        System.out.println("y : "+y); //10
    }
}

```



```

package com.sunbeaminfo.cj06.test;
import java.util.Scanner;

public class Program {
    private static Scanner sc = new Scanner(System.in);
    public static void acceptRecord( int[] number ) {
        System.out.print("Enter number : ");
        number[ 0 ] = sc.nextInt();
    }
    public static void printRecord( int[] number ) {
        System.out.println("Number : "+number[ 0 ]);
    }
    public static void main(String[] args) {
        int[] number = new int[1];
        Program.acceptRecord( number );
        Program.printRecord( number ); //123
    }
}

```

```
package com.sunbeaminfo.cj06.test;
/*
 - C Programming language is procedure oriented programming language.
 - In C, we can not give same name to the methods.
 - According to oops theory, if implementation of methods are
 logically same / equivalent
     then we should give same name to the method.
*/
public class Program {
    private static void add(int num1, int num2) {
        int result = num1 + num2;
        System.out.println("Result : "+result);
    }
    private static void sum(double num1, double num2) {
        double result = num1 + num2;
        System.out.println("Result : "+result);
    }
    public static void main(String[] args) {
        add( 10, 20 );

        sum( 10.2, 20.5 );
    }
}
//Solution of above problem is next concept
```

Method Overloading:-

Type 1:

```
package com.sunbeaminfo.cj06.test;
/*
 - If we want to give same name to the method and if type of all
 the parameters are same
     then number of parameters passed to the method must be
 different.
*/
public class Program {
    //Type of all the params are same( here int )
    private static void sum(int num1, int num2) { // 2 params
        int result = num1 + num2;
        System.out.println("Result : "+result);
    }
}
```

```
private static void sum(int num1, int num2, int num3) { // 3
params
    int result = num1 + num2 + num3;
    System.out.println("Result : "+result);
}
public static void main(String[] args) {
    Program.sum( 10, 20 ); //OK
    Program.sum( 10, 20, 30 ); //OK
}
}
```

Type 2:

```
package com.sunbeaminfo.cj06.test;
/*
 - If we want to give same name to the method and if number of
parameters are same
    then type of at least one parameter must be different.
*/
public class Program {
    //Type of all at least one parameter must be different
    private static void sum(int num1, int num2) { //2 params
        int result = num1 + num2;
        System.out.println("Result : "+result);
    }
    private static void sum(int num1, double num2) { //2 params
        double result = num1 + num2;
        System.out.println("Result : "+result);
    }
    public static void main(String[] args) {
        Program.sum( 10, 20 ); //OK
        Program.sum( 10, 20.5 ); //OK
    }
}
```

Type 3:

```
package com.sunbeaminfo.cj06.test;
/*
 - If we want to give same name to the method and if number of
parameters are same
    then order of type of parameters must be different.

```

```
/*
public class Program {
    //Order of type of params is different
    private static void sum(int num1, float num2) { //2 params
        float result = num1 + num2;
        System.out.println("Result : "+result);
    }
    private static void sum(float num1, int num2) { //2 params
        float result = num1 + num2;
        System.out.println("Result : "+result);
    }
    public static void main(String[] args) {
        Program.sum( 10, 20.2f ); //OK
        Program.sum( 10.1f, 20 ); //OK
    }
}
```

Error if we don't follow 3 type rule :-

```
package com.sunbeaminfo.cj06.test;
/* Only one the basis of different return type we can not give same
name to the methods,
*/
public class Program {
    //Only return type is different
    private static int sum(int num1, int num2) { //2 params
        int result = num1 + num2;
        return result;
    }
    private static void sum(int num1, int num2) { //2 params
        int result = num1 + num2;
        System.out.println("Result : "+result);
    }
    public static void main(String[] args) {
        int result = Program.sum(10, 20);
        System.out.println("Result : "+result);
        Program.sum(50, 60);
        //catching value from method is optional
    }
}
```

+ Method Overloading

1. If we want to give same name to the method and if type of all the parameters are same then number of parameter passed to the method must be different.
2. If we want to give same name to the method and if number of parameters are same then type of at least one parameter passed must be different.
3. If we want to give same name to the method and if number of parameters are same then order of type parameters passed must be different.
4. Only on the basis of different return type, we can not give same name to the method.

+ Process of defining methods using above rules is called method overloading. In other words, process of defining method with same name and different signature is called as method overloading.

+ Methods which are taking part in overloading are called as overloaded methods.

+ If implementation of methods are logically same/equivalent then we should overload method.

+ According to last rule, return type is not considered in method overloading.

+ In Java, we can overload static as well as non static method.

Variable Arity/Argument Method:-

```
package com.sunbeaminfo.cj06.test;
public class Program {
    //Variable argument method / variable arity method
    public static void sum( int... args ) {
        int result = 0;
        for( int element : args )
            result = result + element;
        System.out.println("Result : " +result);
    }
    public static void main(String[] args) {
        Program.sum();
        Program.sum( 10, 20 );
        Program.sum( 10, 20, 30, 40, 50 );
        Program.sum( 10, 20, 30, 40, 50, 60, 70 );
        Program.sum( 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 );
    }
}
```

Debugging:-

```
package com.sunbeaminfo.cj06.test;
/*
 - Logical error is called as bug.
 - In other words, syntactically valid but logically invalid
statement represents bug.
 - Technique of finding bug is called as debugging.
 - If we want to debug code then perspective should be Debug.
 - To debug the code it is mandatory to use breakpoint( also
called as stop point )
 - If we debug code without breakpoint then it is same as run.

 - To start debugging press - F11 key
 - For line by line debugging / step over, press F6 key
 - To stop debugging press - F8 key.
 - If we want to jump on called method / step into - press F5
key.
 - To return from called method / step return - press F7 key
*/
public class Program {
    public static void main(String[] args) {
        System.out.println("Debugging is started");
        int x = 10;
        Program.printRecord();
        x = ++x + x ++ - ++x + x++;
        if( x > 55 )
            System.out.println("Valid");
        else
            System.out.println("Invalid");
        System.out.println("Debugging is about to finish");
    }
    public static void printRecord( ) {
        System.out.println("Inside printRecord");
        int y = 8;
        y = ++y + ++y + y ++ + y++;
        System.out.println("Result : " + y);
        System.out.println("About to leave printRecord");
    }
}
```

Enum In C/C++ Programming Language.:-

- According ANSI C standard, if we want to assign name to the integer constant then we should use enum.
- Enum helps developer to improve readability of source code.
- enum is keyword in C. Let us consider syntax of enum:

```
enum Identifier
{
    //enumerator-list
};

enum Color
{
    RED, GREEN, BLUE
    //RED = 0, GREEN = 1, BLUE = 2
};
```

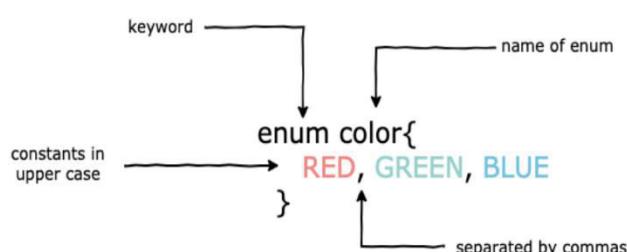
- By default, the first enumeration-constant is associated with the value 0. The next enumeration-constant in the list is associated with the value of (constant-expression + 1), unless you explicitly associate it with another value.
- constant-expression must have int type and can be negative.

```
enum Channel
{
    FOX = 11,
    CNN = 25,
    ESPN = 15,
    HBO = 22,
    MAX = 30,
    NBC = 32
};

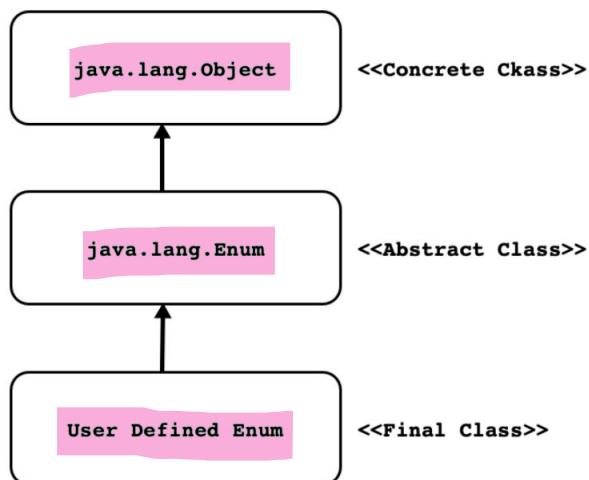
enum Suit { Diamonds = 1, Hearts, Clubs, Spades };
```

Enum In Java Programming language:-

- An enum is a special class that represents a group of constants.
- Enum keyword is used to create an enum. The constants declared inside are separated by a comma and should be in upper case.



Enum Class Hierarchy:-



Enum API:-

- **Enum class is introduced in JDK 1.5.**
- Following are the methods declared in `java.lang.Enum` class:
 1. `public final String name()`
 2. `public final int ordinal()`
 3. `public final Class getDeclaringClass()`
 4. `public static > T valueOf(Class enumType, String name)`
- **Sole constructor:** o Programmers cannot invoke this constructor. It is for use by code emitted by the compiler in response to enum type declarations.

Java Source Code	Compiled Code
<pre>enum Color{ RED, GREEN, BLUE } class Program{ public static void main(String[] args) { Color color = Color.GREEN; } }</pre>	<pre>final class Color extends Enum<Color> { public static final Color RED; public static final Color GREEN; public static final Color BLUE; public static Color[] values(); public static Color valueOf(String name); }</pre>

Properties of enum:-

1. Similar to a class, an enum can have objects and methods. The only difference is that enum constants are public, static and final by default. Since it is final, we can't extend enums
2. It cannot extend other classes since it already extends the java.lang.Enum class.
3. It can implement interfaces.
4. The enum objects cannot be created explicitly and hence the enum constructor cannot be invoked directly.
5. It can only contain concrete methods and no abstract methods.

Application of enum
1. enum is used for values that are not going to change e.g. names of days, colors in a rainbow, number of cards in a deck etc.
2. enum is commonly used in switch statements and below is an example of it:

```
class Program {  
    enum color {  
        RED, GREEN, BLUE  
    }  
    public static void main(String[] args) {  
        color x = color.GREEN; // storing value  
        switch(x) {  
            case RED:  
                System.out.println("x has RED color");  
                break;  
            case GREEN:  
                System.out.println("x has GREEN color");  
                break;  
            case BLUE:  
                System.out.println("x has BLUE color");  
                break;  
        }  
    }  
}
```

Core Java Notes By ASHOK PATE

Class Enum<E extends Enum<E>>

java.lang.Object
java.lang.Enum<E>

Type Parameters:

E - The enum type subclass

All Implemented Interfaces:

Serializable, Comparable<E>

```
public abstract class Enum<E extends Enum<E>>
extends Object
implements Comparable<E>, Serializable
```

This is the common base class of all Java language enumeration types. More information about enums, including descriptions of the implicitly declared methods synthesized by the compiler, can be found in section 8.9 of *The Java™ Language Specification*.

Note that when using an enumeration type as the type of a set or as the type of the keys in a map, specialized and efficient set and map implementations are available.

Since:

1.5

See Also:

Class.getEnumConstants(), EnumSet, EnumMap, Serialized Form

Example 1:-

```
// package com.sunbeaminfo.cj06.test;
import java.util.Arrays;

enum Color{
    RED, GREEN, BLUE
    //RED=0, GREEN=1, BLUE=2
}
/*
    final class Color extends Enum<Color> {
        public static final Color RED;

        public static final Color GREEN;

        public static final Color BLUE;

        public static Color[] values();

        public static Color valueOf(String str );
    }
*/
public class Program {

    public static void main1(String[] args) {
        System.out.println("Name : "+Color.RED.name());
        System.out.println("Ordinal : "+Color.RED.ordinal());
```

```
System.out.println("Name : "+Color.GREEN.name());
System.out.println("Ordinal : "+Color.GREEN.ordinal());

System.out.println("Name : "+Color.BLUE.name());
System.out.println("Ordinal : "+Color.BLUE.ordinal());
}

public static void main2(String[] args) {
    Color color = Color.RED;
    System.out.println("Name : "+color.name());
    System.out.println("Ordinal : "+color.ordinal());

    color = Color.GREEN;
    System.out.println("Name : "+color.name());
    System.out.println("Ordinal : "+color.ordinal());

    color = Color.BLUE;
    System.out.println("Name : "+color.name());
    System.out.println("Ordinal : "+color.ordinal());
}

public static void main3(String[] args) {
    Color[] colors = Color.values();
    for (Color color : colors) {
        System.out.println("Name : "+color.name());
        System.out.println("Ordinal : "+color.ordinal());
    }
}

public static void main4(String[] args) {
    Color[] colors = Color.values();
    for (Color color : colors)
        System.out.println(color.name()+" "+color.ordinal());
}

public static void main5(String[] args) {
    Color[] colors = Color.values();
    System.out.println(Arrays.toString(colors));
}

-----
```

Example 2:-

```
package com.sunbeaminfo.cj06.test;

import java.util.Arrays;
import java.util.Scanner;

enum Color{
    RED, GREEN, BLUE
    //RED=0, GREEN=1, BLUE=2
}
public class Program {
    private static Scanner sc = new Scanner(System.in);
    public static void main(String[] args) {
        System.out.print("Enter color name(RED/GREEN/BLUE): ");
        String colorName = sc.nextLine();
        Color color = Color.valueOf(colorName.toUpperCase());
        System.out.println(color.name()+" "+color.ordinal());
    }
}
/*
Enter color name(RED/GREEN/BLUE): PURPLE
Exception in thread "main" java.lang.IllegalArgumentException: No enum constant com.sunbeaminfo.cj06.test.Color.PURPLE
    at java.lang.Enum.valueOf(Enum.java:238)
    at com.sunbeaminfo.cj06.test.Color.valueOf(Program.java:1)
    at com.sunbeaminfo.cj06.test.Program.main(Program.java:16)
*/
```

Example 3:-

```
// package com.sunbeaminfo.cj06.test;

import java.util.Arrays;
import java.util.Scanner;

enum Day{
    //MON=1,TUES=2,WED=3      //In C/C++
    MON(1),TUES(2),WED(3); //It must be first statement inside enum
    //if you assign name means 1 you have to write construction in
    class
    private int dayNumber;
```

```
private Day(int dayNumber) {
    this.dayNumber = dayNumber;
}
public int getDayNumber() {
    return dayNumber;
}
@Override
public String toString() {
    return String.valueOf(dayNumber);
}
}

public class Program {
    public static void main(String[] args) {
        Day day = Day.MON;
        System.out.println(day.name());          //MON
        System.out.println(day.ordinal());       //0
        System.out.println(day.getDayNumber());   //1
        System.out.println(day.toString());       //1
    }
}
```

Example 4:-

```
package com.sunbeaminfo.cj06.test;

import java.util.Arrays;
import java.util.Scanner;

enum Day{
    MON("MonDay"), TUES("TuesDay"), WED("WednesDay");
    private String dayName;
    private Day(String dayName) {
        this.dayName = dayName;
    }
    public String getDayName() {
        return dayName;
    }
    @Override
    public String toString() {
```

```
        return this.dayName;
    }
}

public class Program {
    public static void main(String[] args) {
        Day day = Day.MON;
        System.out.println(day.name());           //MON
        System.out.println(day.ordinal());        //0
        System.out.println(day.getDayName());      //MonDay
        System.out.println(day.toString());        //MonDay
    }
}
```

Example 5:-

```
package com.sunbeaminfo.cj06.test;

import java.util.Arrays;
import java.util.Scanner;

enum Day{
    MON(1, "MonDay"), TUES(2, "TuesDay"), WED(3, "WednesDay");
    private int dayNumber;
    private String dayName;
    private Day(int dayNumber, String dayName) {
        this.dayNumber = dayNumber;
        this.dayName = dayName;
    }
    public int getDayNumber() {
        return dayNumber;
    }
    public String getDayName() {
        return dayName;
    }
    @Override
    public String toString() {
        return this.dayNumber+" "+this.dayName;
    }
}
```

```
public class Program {
    public static void main(String[] args) {
        Day day = Day.MON;
        System.out.println(day.name());           //MON
        System.out.println(day.ordinal());        //0

        System.out.println(day.getDayName());      //MonDay
        System.out.println(day.getDayNumber());     //1

        System.out.println(day.toString());

    }
}
```

Example 6:-

```
package com.sunbeaminfo.cj06.test;

import java.util.Arrays;
import java.util.Scanner;

enum Day{
    MON(1, "MonDay"), TUES("TuesDay"), WED(3);
    private int dayNumber;
    private String dayName;
    private Day(int dayNumber) {
        this(dayNumber, null );
    }
    private Day(String dayName) {
        this( 0, dayName );
    }
    private Day(int dayNumber, String dayName) {
        this.dayNumber = dayNumber;
        this.dayName = dayName;
    }

    public int getDayNumber() {
        return dayNumber;
    }
}
```

```
public String getDayName() {
    return dayName;
}
@Override
public String toString() {
    return this.dayNumber+" "+this.dayName;
}
}

public class Program {
    public static void main(String[] args) {
        Day[] days = Day.values();
        for (Day day : days) {
            System.out.println(day.name()+""
"+day.ordinal()+" "+day.getDayName()+" "+day.getDayNumber());
        }
    }
}
```

OOPS

Major and Minor pillars of oops :-

- **4 Major pillars**

1. Abstraction
2. Encapsulation
3. Modularity
4. Hierarchy

- **3 Minor Pillars**

1. Typing
2. Concurrency
3. Persistence

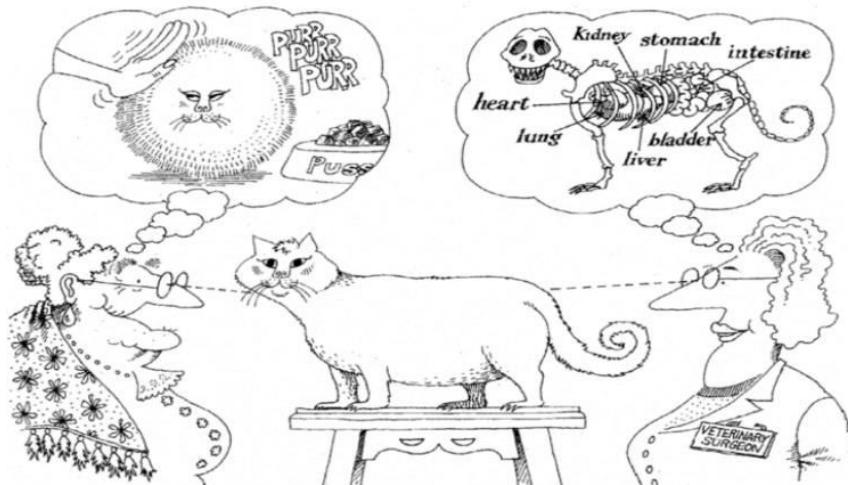
Advantages of OOPS

1. To achieve simplicity
2. To achieve data hiding and data security.
3. To minimize the module dependency so that failure in single part should not stop complete system.
4. To achieve reusability so that we can reduce development time/cost/efforts.
5. To reduce maintenance of the system.
6. To fully utilize hardware resources.
7. To maintain state of object on secondary storage so that failure in system should not impact on data.

Abstraction

- It is a major pillar of oops.
- It is a process of getting essential things from object.
- It describes outer behaviour of the object.
- Abstraction focuses on some essential characteristics of object relative to the perspective of viewer. In other words, abstraction changes from user to user.
- Using abstraction, we can achieve simplicity.
- Abstraction in Java Complex c1 = new Complex();
c1.acceptRecord(); c1.printRecord();

Abstraction



Abstraction focuses on the essential characteristics of some object, relative to the perspective of the viewer.

```
package com.sunbeaminfo.cj06.test;

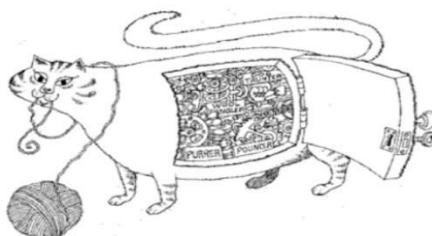
import java.util.Scanner;

class Complex{ //class
    //Data
    private int real;    //Field
    private int imag;   //Field
    public Complex( ) {
        this.real = 0;
        this.imag = 0;
    }
    //Code
    public void acceptRecord( ) { //Method
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter real number : ");
        this.real = sc.nextInt();
        System.out.print("Enter imag number : ");
        this.imag = sc.nextInt();
    }
    public void printRecord( ) { //Method
        System.out.println("Real Number : "+this.real);
        System.out.println("Imag Number : "+this.imag);
    }
} //This are the encapsulation
// To achieve abstraction, we should provide some implementation. It
is called encapsulation.
public class Program {
    public static void main(String[] args) {
        //Abstraction
        Complex c1 = new Complex( );
        c1.acceptRecord( );
        c1.printRecord( );
    }
}
```

Encapsulation

- It is a major pillar of oops.
- Definition:
 1. Binding of data and code together is called encapsulation.
 2. To achieve abstraction, we should provide some implementation. It is called encapsulation.
- Encapsulation represents, internal behaviour of the object.
- Using encapsulation we can achieve data hiding.
- Abstraction and encapsulation are complementary concepts: Abstraction focuses on the observable behaviour of an object, whereas encapsulation focuses on the implementation that gives rise to this behaviour.

Encapsulation



Encapsulation hides the details of the implementation of an object.

Abstraction and encapsulation are complementary concepts: Abstraction focuses on the observable behavior of an object, whereas encapsulation focuses on the implementation that gives rise to this behavior.

Modularity

- It is a major pillar of oops.
- It is the process of developing complex system using small parts.
- Using modularity, we can reduce module dependency.
- We can implement modularity by creating library files.
 - .lib/.a, .dll / .so files
 - .jar/.war/.ear in java

Hierarchy

- It is a major pillar of oops.
- Level / order / ranking of abstraction is called hierarchy.
- Main purpose of hierarchy is to achieve reusability.
- Advantages of code reusability
 1. We can reduce development time.

- 2. We can reduce development cost.
- 3. We can reduce developers effort.

- Types of hierarchy:

1. Has-a / Part-of => Association
2. Is-a / Kind-of => Inheritance / Generalization
3. Use-a => Dependency
4. Creates-a => Instantiation

Typing

- It is a minor pillar of oops.
- Typing is also called as polymorphism.
- Polymorphism is a Greek word. Polymorphism = Poly(many) + morphism(forms).
- An ability of object to take multiple forms is called polymorphism. • Using polymorphism, we can reduce maintenance of the system.
- Types of polymorphism:

1. Compile time polymorphism

- It is also calling static polymorphism / Early binding / Weak Typing / False polymorphism.
- We can achieve it using:

1. Method Overloading

2. Run time polymorphism

- It is also calling dynamic polymorphism / Late binding / Strong Typing / True polymorphism.
- We can achieve it using:

1. Method Overriding.

Concurrency

- It is a minor pillar of oops.
- In context of operating system, it is called as multitasking. • It is the process of executing multiple task simultaneously.
- Main purpose of concurrency is to utilise CPU efficiently. • In Java, we can achieve concurrency using thread.

Persistence

- It is a minor pillar of oops.
- It is process of maintaining state of object on secondary storage.
- In Java, we can achieve Persistence using file and database.

Association:-

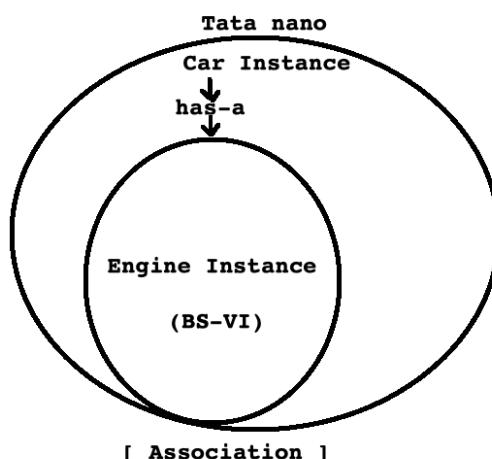
- If has-a relationship is exist between the types then we should use association.
- Example 1. Car has a engine
2. Room has a chair
- Let us consider example of car and engine:
 1. Car has a engine
 2. Engine is part of Car.
- If object-instance is a part/component of another instance then it is called as association.
- To implement association, we should declare instance of a class as a field inside another class.

+ How can we implement relationship between Car and Engine?

1. Car has a Engine.
2. Engine is a part of car

Dependency Instance : Engine Instance
Dependent Instance : Car Instance

```
class Engine{  
    //TODO  
}  
  
class Car{  
    //Engine e = new Engine( ); //Association  
    Engine e; //Association  
    public Car( ){  
        e = new Engine( );  
    }  
}  
  
Car c = new Car( );
```

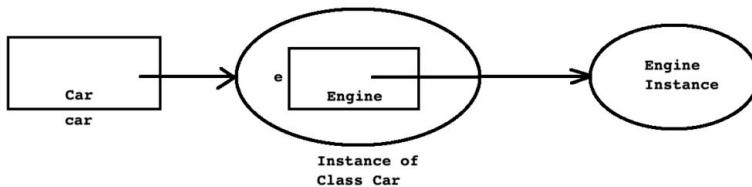


- Car has a engine. In other words engine is a part of car.
- It means that if we create instance of class Car then Engine instance must get space inside it.

Core Java Notes By ASHOK PATE

- Engine is part of Car

```
class Engine{
    //TODO
}
class Car{
    Engine e = new Engine(); //Association
}
Car c = new Car();
```



- Every class should contain

1. Fields
2. Constructor
3. Getter and Setter
4. `toString` method

- Date class

```
- int day, month, year;
```

- Address

```
- String cityName, stateName, pincode;
```

- Person

```
- String name; //Association
- Date birthDate; //Association
- Address currentAddress; //Association
```

- Program

```
- inside main() test functionality
```

+ Java Archive

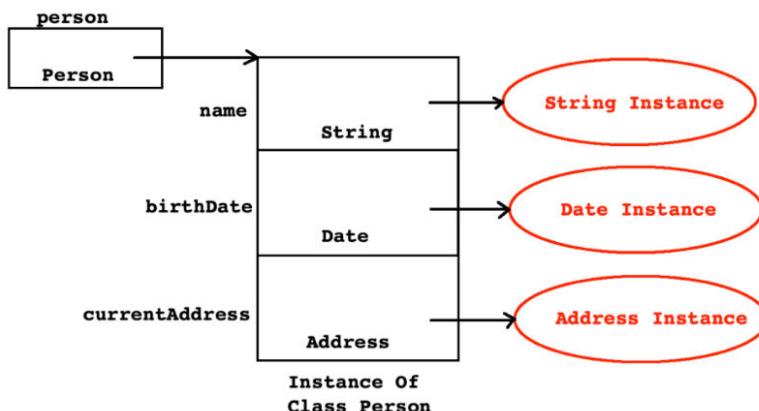
- If we want to create reusable library of classes then we should create jar file(.jar).
- jar is a tool which is used to create .jar file.
- Jar file can contain
 1. Manifest file(.mf)
 2. Resources
 3. Packages

```
+ AssociationLib( compiled to associationlib.jar )
- com.sunbeaminfo.cj06.association.lib
  1. Date
  2. Address
  3. Person
```



```
+ AssociationTest( include associationlib.jar
  into build-path / Runtime classpath /
  classpath )
- com.sunbeaminfo.cj06.association.test
  1. Program
```

Problem statement diagram:



Solution by program:

```
package com.sunbeaminfo.cj06.association.lib;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@NoArgsConstructor
@AllArgsConstructor
@Getter @Setter
@ToString
public class Person {
    private String name;          //Association
    private Date birthDate;       //Association
    private Address currentAddress; //Association
}

package com.sunbeaminfo.cj06.association.lib;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@NoArgsConstructor
@AllArgsConstructor
@Getter @Setter
@ToString
public class Date {
    private int day;
    private int month;
    private int year;
}

package com.sunbeaminfo.cj06.association.lib;
```

```
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@NoArgsConstructor
@AllArgsConstructor
@Getter @Setter
@ToString
public class Address {
    private String cityName;
    private String stateName;
    private String pincode;
}

//We create a jar file from this above three java file and used in
another so we can achieve modularity.
```

```
package com.sunbeaminfo.cj06.association.test;
import java.util.Objects;
import java.util.Scanner;
import com.sunbeaminfo.cj06.association.lib.Address;
import com.sunbeaminfo.cj06.association.lib.Date;
import com.sunbeaminfo.cj06.association.lib.Person;

public class Program {
    private static Scanner sc = new Scanner(System.in);
    private static void acceptRecord(Date birthDate) {
        System.out.print("Day : ");
        birthDate.setDay(sc.nextInt());
        System.out.print("Month : ");
        birthDate.setMonth(sc.nextInt());
        System.out.print("Year : ");
        birthDate.setYear(sc.nextInt());
    }
    private static void acceptRecord(Address currentAddress) {
        System.out.print("City Name : ");
        currentAddress.setCityName(sc.nextLine());
        System.out.print("State Name : ");
        currentAddress.setStateName(sc.nextLine());
    }
}
```

```
System.out.print("Pincode      :    ");
currentAddress.setPincode(sc.nextLine());
}

private static void acceptRecord(Person person) {
    System.out.print("Name      :    ");
    person.setName(sc.nextLine());

    //Create Date instance, accept record for it and then set it
    //to person instance
    Date birthDate = new Date();
    Program.acceptRecord(birthDate);
    person.setBirthDate(birthDate);

    //Create Address instance, accept record for it and then set it
    //to person instance
    Address currentAddress = new Address();
    sc.nextLine();
    Program.acceptRecord( currentAddress );
    person.setCurrentAddress(currentAddress);
}

private static void printRecord(Date birthDate) {
    birthDate = Objects.requireNonNull(birthDate, "Date should not
be null");
    System.out.println("BirthDate      :    "+birthDate.getDay()+
/ "+birthDate.getMonth()"+ / "+birthDate.getYear());
}

private static void printRecord(Address currentAddress) {
    currentAddress = Objects.requireNonNull(currentAddress,
"Address should not be null");
    System.out.println("Current Address
:    "+currentAddress.getCityName()+"    "+currentAddress.getStateName()+
"    "+currentAddress.getPincode());
}

private static void printRecord(Person person) {
    person = Objects.requireNonNull(person, "Person should not be
null");
    System.out.println("Name      :    "+person.getName());

    Date birthDate = person.getBirthDate();
    Program.printRecord(birthDate);
```

```
    Address currentAddress = person.getCurrentAddress();
    Program.printRecord(currentAddress);
}
public static void main(String[] args) {
    Person person = new Person( );
    Program.acceptRecord( person );
    Program.printRecord( person );
}
public static void main2(String[] args) {
    Address currentAddress = new Address();
    Program.acceptRecord( currentAddress );
    Program.printRecord( currentAddress );
}
public static void main1(String[] args) {
    Date birthDate = new Date();
    Program.acceptRecord( birthDate );
    Program.printRecord( birthDate );
}
}
```

+ Aggregation

- Let us take example of department and faculty?

```
- Department has a faculty.
class Faculty{
    //TODO
}
class Department{
    Faculty f = new Faculty( ); //Association -> Aggregation
}
* Dependency Instance: Faculty Instance
* Dependent Instance : Department Instance
```

- In case of association, if dependency instance can exist without dependent instance then it is called as aggregation.

- Aggregation is special form of association which represents loose coupling,

+ Composition

- Let us consider example of Human and Heart

```
- Human has a heart
class Heart{
    //TODO
}
class Human{
    Heart hrt = new Heart( ); //Association -> Composition
}
* Dependency Instance: Heart instance
* Dependent Instance : Human instance
```

- In case of association, if dependency instance do not exist without dependent instance then it is called as composition.

- Composition is a special form of association which represents tight coupling.

```
package com.sunbeaminfo.cj06.date.test;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.Calendar;
import java.util.Date;

public class Program {
    public static void main(String[] args) {
        /* LocalDateTime lt = LocalDateTime.now();
        lt.getHour();
        lt.getMinute();
        lt.getSecond();
        lt.getNano(); */
    }
    public static void main3(String[] args) {
        LocalDate ldt = LocalDate.now();
        System.out.println("Day : " + ldt.getDayOfMonth());
        System.out.println("Month : " + ldt.getMonthValue() );
        System.out.println("Year : " + ldt.getYear());
    }
    public static void main2(String[] args) {
        Calendar c = Calendar.getInstance();

        System.out.println("Day : " + c.get( Calendar.DAY_OF_MONTH ) );
        System.out.println("Month : " + ( c.get( Calendar.MONTH ) +
1 ) );
        System.out.println("Year : " + ( c.get( Calendar.YEAR ) ) );
    }
    public static void main1(String[] args) {
        Date date = new Date();
        System.out.println("Day : " + date.getDate());
        System.out.println("Month : " + ( date.getMonth() + 1 ) );
        System.out.println("Year : " + ( date.getYear() + 1900 ) );
    }
}
//java.util mai date naam ka class hai usme ye method hai.
//java.util mai calender naam ka class hai usme ye fields hai.
//java.time naam ke package mai localdate ka class hai usme method hai
now();
```

Core Java Notes By ASHOK PATE

getDate

```
@Deprecated  
public int getDate()
```

Deprecated. As of JDK version 1.1, replaced by `Calendar.get(Calendar.DAY_OF_MONTH)`.

Returns the day of the month represented by this Date object. The value returned is between 1 and 31 representing the day of the month that contains or begins with the instant in time represented by this Date object, as interpreted in the local time zone.

Returns:

the day of the month represented by this date.

See Also:

[Calendar](#)

getMonth

```
@Deprecated  
public int getMonth()
```

Deprecated. As of JDK version 1.1, replaced by `Calendar.get(Calendar.MONTH)`.

Returns a number representing the month that contains or begins with the instant in time represented by this Date object. The value returned is between 0 and 11, with the value 0 representing January.

Returns:

the month represented by this date.

See Also:

[Calendar](#)

getYear

```
@Deprecated  
public int getYear()
```

Deprecated. As of JDK version 1.1, replaced by `Calendar.get(Calendar.YEAR) - 1900`.

Returns a value that is the result of subtracting 1900 from the year that contains or begins with the instant in time represented by this Date object, as interpreted in the local time zone.

Returns:

the year represented by this date, minus 1900.

See Also:

[Calendar](#)

DAY_OF_MONTH

```
public static final int DAY_OF_MONTH
```

Field number for get and set indicating the day of the month. This is a synonym for DATE. The first day of the month has value 1.

See Also:

[DATE, Constant Field Values](#)

DAY_OF_YEAR

```
public static final int DAY_OF_YEAR
```

Field number for get and set indicating the day number within the current year. The first day of the year has value 1.

See Also:

[Constant Field Values](#)

DAY_OF_WEEK

```
public static final int DAY_OF_WEEK
```

Field number for get and set indicating the day of the week. This field takes values SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, and SATURDAY.

See Also:

[SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, Constant Field Values](#)

Core Java Notes By ASHOK PATE

now

```
public static LocalDate now()
```

Obtains the current date from the system clock in the default time-zone.

This will query the system clock in the default time-zone to obtain the current date.

Using this method will prevent the ability to use an alternate clock for testing because the clock is hard-coded.

Returns:

the current date using the system clock and default time-zone, not null

now

```
public static LocalDate now(ZoneId zone)
```

Obtains the current date from the system clock in the specified time-zone.

This will query the system clock to obtain the current date. Specifying the time-zone avoids dependence on the default time-zone.

Using this method will prevent the ability to use an alternate clock for testing because the clock is hard-coded.

Parameters:

zone - the zone ID to use, not null

Returns:

the current date using the system clock, not null

getHour

```
public int getHour()
```

Gets the hour-of-day field.

Returns:

the hour-of-day, from 0 to 23

getMinute

```
public int getMinute()
```

Gets the minute-of-hour field.

Returns:

the minute-of-hour, from 0 to 59

getSecond

```
public int getSecond()
```

Gets the second-of-minute field.

Returns:

the second-of-minute, from 0 to 59

getNano

```
public int getNano()
```

Gets the nano-of-second field.

Returns:

the nano-of-second, from 0 to 999,999,999

```
package com.sunbeaminfo.cj06.date.test;
import lombok.Builder;
import lombok.Setter;
import lombok.ToString;
@Setter @ToString
class Employee{
    private String name;
    private int empid;
    private float salary;
}
public class Program {
    public static void main1(String[] args) {
        Employee emp = new Employee();
        emp.setName("Sandeep");
        emp.setEmpid(33);
        emp.setSalary(45000.50f);
        System.out.println(emp.toString());
    }
}
```

```
package com.sunbeaminfo.cj06.date.test;
import lombok.Builder;
import lombok.Setter;
import lombok.ToString;
@Builder @ToString
class Employee{
    private String name;
    private int empid;
    private float salary;
}
public class Program {
    public static void main(String[] args) {
        Employee emp = Employee.builder()
            .name("Sandeep")
            .empid(33)
            .salary(45000.56f)
            .build();
        System.out.println(emp);
    }
}
```

Inheritance:-

- If "is-a" relationship is exist between the types then we should use inheritance.
- Inheritance is also called as generalization.
- Example
 - 1. Manager is a employee
 - 2. Book is a product
 - 3. Triangle is a shape
 - 4. SavingAccount is a account.

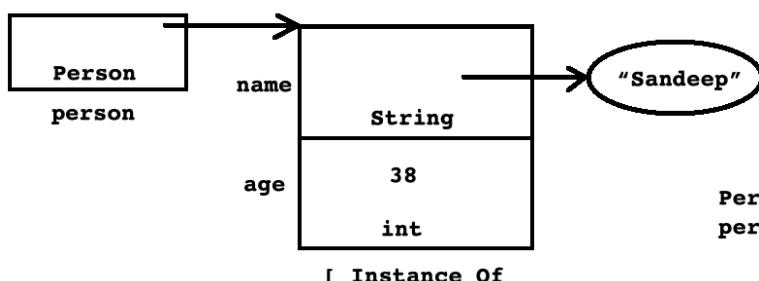
```
class Employee{ //Parent class
    //TODO
}
class Manager extends Employee{ //Child class
    //TODO
}
//Here class Manager is extended from class Employee.
```

- If we want to implement inheritance then we should use extends keyword.
- In Java, parent class is called as super class and child class is called as sub class.
- Java do not support private and protected mode of inheritance
- If Java, class can extend only one class. In other words, multiple class inheritance is not allowed.
- Consider following code:

```
class A{ }
class B{ }
class C extends A, B{ //Not OK
}
```

Example 1:

```
// package com.sunbeaminfo.cj06.inheritance.test;
class Person{
    String name;
    int age;
    public Person( ) {
    }
    public Person( String name, int age ) {
        this.name = name;
        this.age = age;
    }
    public void showRecord( ) {
        System.out.println("Name : "+this.name);
        System.out.println("Age : "+this.age);
    }
}
public class Program {
    public static void main1(String[] args) {
        //Person p1 = new Person( );
        Person p1 = new Person( "Sandeep", 38 );
        p1.showRecord();
    }
    public static void main(String[] args) {
        Person p1 = new Person( );
        p1.name = "Sandeep";
        p1.age = 38;
        System.out.println("Name : "+p1.name);
        System.out.println("Age : "+p1.age);
    }
}
```



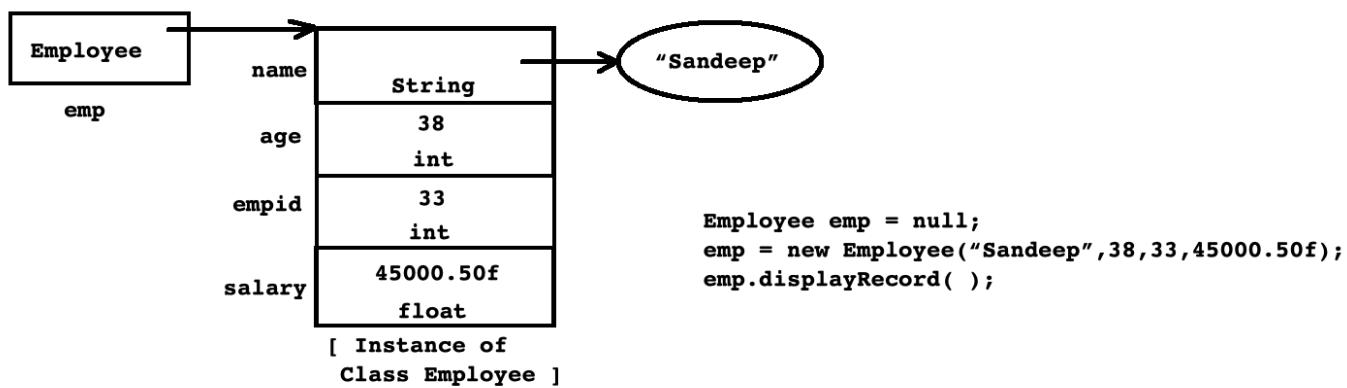
```
Person person = new Person("Sandeep", 38 );
person.showRecord(); // "Sandeep", 38
```

Example 2:

```
package com.sunbeaminfo.cj06.inheritance.test;
class Employee{
```

```
String name;
int age;
int empid;
float salary;
public Employee() {
}
public Employee( String name, int age, int empid, float salary ) {
    this.name = name;
    this.age = age;
    this.empid = empid;
    this.salary = salary;
}
public void displayRecord( ) {
    System.out.println("Name      : "+this.name);
    System.out.println("Age       : "+this.age);
    System.out.println("Empid     : "+this.empid);
    System.out.println("Salary    : "+this.salary);
}
public class Program {
    public static void main(String[] args) {
        Employee emp = new Employee("Sandeep", 38, 33, 45000.50f );
        emp.displayRecord();
    }
    public static void main1(String[] args) {
        Employee emp = new Employee();
        emp.name = "Sandeep";
        emp.age = 38;
        emp.empid = 33;
        emp.salary = 45000.50f;

        System.out.println("Name      : "+emp.name);
        System.out.println("Age       : "+emp.age);
        System.out.println("Empid     : "+emp.empid);
        System.out.println("Salary    : "+emp.salary);
    }
}
```



Example 3 :- very very important according concept point of view.

```

// package com.sunbeaminfo.cj06.inheritance.test;

import java.sql.PseudoColumnUsage;

class Person{ //Parent Class / Super Class
    //Fields / Properties
    String name;
    int age;
    public Person( ) {
        //super( ); //OK
        System.out.println("public Person( )");
    }
    //in java except constructor all the member are inherite into
    //subclass
    public Person( String name, int age ) {
        System.out.println("public Person( String name, int age )");
        this.name = name;
        this.age = age;
    }
    //Method / Behavior
    public void showRecord( ) {
        System.out.println("Name      :      "+this.name);
        System.out.println("Age :      "+this.age);
    }
    static void test(){
        System.out.println("Super class method");
    }
}
  
```

```
}

class Employee extends Person{ // Child Class / Sub Class
    int empid;
    float salary;
    public Employee() {
        super( ); //
        System.out.println("public Employee()");
    }
    public Employee( String name, int age, int empid, float salary ) {
        //super statement must be in first statement.
        super(name, age ); //Super Statement
        System.out.println("public Employee( String name, int age, int
empid, float salary )");
        this.empid = empid;
        this.salary = salary;
    }
    /*public void showRecord( ) {
        System.out.println("Name      : "+this.name);
        System.out.println("Age : "+this.age);
    }*/
    public void displayRecord( ) {
        this.showRecord();
        System.out.println("Empid      : "+this.empid);
        System.out.println("Salary      : "+this.salary);
    }
}
public class Program {
    public static void main1(String[] args) {
        Person person = new Person();
        person.name = "Sandeep";
        person.age = 38;

        System.out.println("Name      : "+person.name);
        System.out.println("Age : "+person.age);
    }
    public static void main2(String[] args) {
        Person person = new Person("Sandeep", 38);
        person.showRecord();
    }
}
```

```
public static void main3(String[] args) {
    Employee emp = new Employee();
    emp.name = "Sandeep";
    emp.age = 38;
    emp.empid = 33;
    emp.salary = 45000.50f;

    System.out.println("Name : "+emp.name);
    System.out.println("Age : "+emp.age);
    System.out.println("Empid : "+emp.empid);
    System.out.println("Salary : "+emp.salary);
}

public static void main4(String[] args) {
    //Employee emp = new Employee("Sandeep", 38, 33, 45000.50f );
    Employee emp = new Employee();
    emp.showRecord();
    emp.displayRecord();
}
public static void main5(String[] args) {
    //Employee emp = new Employee("Sandeep", 38, 33, 45000.50f );
    Employee emp = new Employee();
    emp.displayRecord();
}
public static void main6(String[] args) {
    // Person.test(); //super class method
    Employee.test(); //super class method
}
public static void main7(String[] args) {
    //Person p1 = new Person();
    //p1.showRecord(); //only parameterless constructor called
    //super class ka instance banaya tho sirf super class ka hi
    // constructor called hota hai

    //Person p2 = new Person("Sandeep", 38);
    //p2.showRecord(); //only parametrized constructor called.

    //Employee emp1 = new Employee();
    //emp1.displayRecord();
    /*sub class ka instance banate hai tho pahle super class ka
```

```
constructor called hota hai badme child class ka constructor
*/
Employee emp = new Employee("Sandeep", 38, 33, 45000.50f);
emp.displayRecord();
/*yaha pr arguments pass kiye hai fir bhi sub class ka
instance
banane ke bad ye pahle super class ke parameterless
constructor
called hua by default pr muje parametrized called krna tha tho
iske liye
solution hai .....*/
/* Then we should use super statement */

}

public static void main8(String[] args) {
    Person person = new Person();
    person.name = "Sandeep";      //OK
    person.age = 38;      //OK
    //person.empid = 33;      //NOT OK
    //person.salary = 45000.50f; //NOT OK
    //sub class ke member super class mai inherite nhi hote

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

- If we create instance of sub class then all the non static fields declared in super class and sub class get space inside it. In other words, non static fields of super class inherit into sub class.
- Static field do not get space inside instance. It is designed to share among all the instances of same class.
- Using sub class, we can access static fields declared in super class. In other words, static fields of super class inherit into sub class.

- All the fields of super class inherit into sub class but only non static fields gets space inside instance of sub class.
 - Fields of sub class, do not inherit into super class. Hence if we create instance of super class then only non static fields declared in super class get space inside it.
 - If we declare field in super class static then, all the instances of super class and sub class share single copy of static field declared in super class.
 - We can call/invoke, non static method of super class on instance of sub class. In other words, non static method inherit into sub class.
 - We can call static method of super class on sub class. In other words, static method inherit into sub class.
 - Except constructor, all the methods of super class inherit into sub class.
 - If we create instance of super class then only super class constructor gets called. But if we create instance of sub class then JVM first give call to the super class constructor and then sub class constructor.
 - From any constructor of sub class, by default, super class's parameterless constructor gets called.
 - Using super statement, we can call any constructor of super class from constructor of sub class.
 - Super statement, must be first statement inside constructor body.
-

Nested class :

Ex:4

```
package com.sunbeaminfo.cj06.inheritance.test;

class Person{ //Parent Class / Super Class
//Nested class
```

```
public class Date{
    public void printRecord( ) {
        System.out.println("Inside Date.printRecord");
    }
}
class Employee extends Person{ // Child Class / Sub Class

}
public class Program {
    public static void main(String[] args) {
        Employee.Date dt1 = new Employee().new Date();
        dt1.printRecord();
        //super class ka nested class sub class mai inherite ho jata
jai.
    }
    public static void main1(String[] args) {
        Person.Date dt1 = new Person().new Date();
        dt1.printRecord();
    }
}
```

Ex:5

```
package com.sunbeaminfo.cj06.inheritance.test;

class Person{
    private String name;
    private int age;
    public void setName(String name) {
        this.name = name;
    }
    public void setAge(int age) {
        this.age = age;
    }
    public void showRecord( ) {
        System.out.println("Name : "+this.name );
        System.out.println("Age : "+this.age );
    }
}
```

```
class Employee extends Person{
    private int empid;
    private float salary;
    public void setEmpid(int empid) {
        this.empid = empid;
    }
    public void setSalary(float salary) {
        this.salary = salary;
    }
    public void displayRecord( ) {
        this.showRecord();
        System.out.println("Empid : "+this.empid);
        System.out.println("Salary : "+this.salary);
    }
}
public class Program {
    public static void main1(String[] args) {
        Person p = new Person( );
        p.setName("Sandeep");
        p.setAge(38);
        p.showRecord();
    }
    public static void main(String[] args) {
        Employee emp = new Employee();
        emp.setName("Sandeep");
        emp.setAge(38);
        emp.setEmpid(33);
        emp.setSalary(45000.50f);
        emp.displayRecord();
    }
    //private member sub class mai inherite ho jate hai.
}
```

Ex: 6 same program above concept

```
// Testing if a subclass can access the private members of a
superclass
```

```
class Class1 {
    private String name;
```

```
public void setName(String name) {
    this.name = name;
    System.out.println("The name has been set successfully.");
}

public void showName() {
    System.out.println("The name is: " + name);
}
}

class Class2 extends Class1 {
    private int age;

    public void setAge(int age) {
        this.age = age;
        System.out.println("The age has been set successfully.");
    }

    public void showAge() {
        System.out.println("The age is: " + age);
    }

    public void displayName() {
        //Accessing the private member of superclass here
        //System.out.println("The name is: " + name); //error, can't
        compile because access to the private member name of the superclass
        Class1 is not permitted here.
    }
}

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

        Class1 c1 = new Class1();
        Class2 c2 = new Class2();

        c1.setName("Name_C1");
        c2.setName("Name_C2"); //No error, setName() is a public
        member of the superclass which indirectly gives access to the private
        member "name".
    }
}
```

```
c1.showName();
c2.showName(); //No error, showName() is a public member of
the superclass which indirectly gives access to the private member
"name".
c2.setAge(25);
c2.showAge();

//c2.displayName(); //error
}

}
```

Conclusion

Yes, a subclass can indirectly access the private members of a superclass. A subclass can't directly access the private members of a superclass.

All the public, private and protected members (i.e. all the fields and methods) of a superclass are inherited by a subclass but the subclass can directly access only the public and protected members of the superclass. If an inherited member from a superclass gives access to a private member of the superclass then the subclass can use this inherited member to access the private member of the superclass.

Ex.7

```
package p1;
public class A {
    private int num1 = 10;
    int num2 = 20;
    protected int num3 = 30;
    public int num4 = 40;
    public void showRecord( ) {
        //System.out.println("Num1 : "+this.num1); //OK
        //System.out.println("Num2 : "+this.num2); //OK
        //System.out.println("Num3 : "+this.num3); //OK
        //System.out.println("Num4 : "+this.num4); //OK
    }
}
```

```
package p
public class B extends A{
    public void displayRecord( ) {
        //System.out.println("Num1 : "+this.num1); //The field
A.num1 is not visible
        // private member inhirate hote hai pr access nhi hote

        //System.out.println("Num2 : "+this.num2); //OK
        //System.out.println("Num3 : "+this.num3); //OK
        //System.out.println("Num4 : "+this.num4); //OK
    }
}

package p2;
import p1.A;
public class C extends A{
    public void printRecord( ) {
        //System.out.println("Num1 : "+this.num1); //The field
A.num1 is not visible
        //System.out.println("Num1 : "+this.num2); //The field
A.num1 is not visible
        //System.out.println("Num3 : "+this.num3); //OK
        System.out.println("Num4 : "+this.num4); //OK
    }
}

package p2;
import p1.A;
public class D {
    public void printRecord( ) {
        A a = new A();
        //System.out.println("Num1 : "+a.num1); //The field A.num1
is not visible
        //System.out.println("Num2 : "+a.num2); //The field A.num1
is not visible
        //System.out.println("Num3 : "+a.num3); //The field A.num3
is not visible
        System.out.println("Num4 : "+a.num4); //OK
    }
}
```

```
}

}

package p1;
import p2.C;
import p2.D;

public class Program { //Non Sub Class
    public static void main(String[] args) {
        //C c = new C();
        //c.printRecord();

        D d = new D();
        d.printRecord();
    }

    public static void main2(String[] args) {
        //Private
        /* A a = new A();
           System.out.println("Num1      :      "+a.num1); */ //The field
        A.num1 is not visible

        // Default/package level private
        //System.out.println("Num2      :      "+num2); //NOT OK

        /* A a = new A();
           System.out.println("Num2      :      "+a.num2); //OK */

        //System.out.println("Num3      :      "+num3); //Not OK
        /* A a = new A();
           System.out.println("Num3      :      "+a.num3); //OK */

        //System.out.println("Num4      :      "+num4); //NOT OK
        A a = new A();
        System.out.println("Num4      :      "+a.num4); //OK
    }

    public static void main1(String[] args) {
        //A a = new A();
        //a.showRecord();
        B b = new B();
        b.displayRecord();
    }
}
```

}

Accessibility in Inheritance.

Access Modifier	Same Package			Different Package	
	Same Class	Sub Class	Non Sub Class	Sub Class	Non Sub Class
private	A	NA	NA	NA	NA
package level private	A	A	A	NA	NA
protected	A	A	A	A	NA
public	A	A	A	A	A

- Default == Package level private.

- In Java, class members are by default considered as package level private.

Ex.8

```
package com.sunbeam.cj06.inheritance.parent;

public class Person{
    private String name;
    private int age;
    public Person( ) {
    }
    public Person( String name, int age ) {
        this.name = name;
        this.age = age;
    }
    public void printRecord( ) {
        System.out.println("Name : "+this.name);
        System.out.println("Age : "+this.age);
    }
}
//jar file created from above program.

package com.sunbeaminfo.cj06.inheritance.child;

import com.sunbeam.cj06.inheritance.parent.Person;
/*
 - According to clients requirement , if implementation of super class
 method is
     partially complete / logically incomplete then we should redefine /
     override that method
```

inside sub class.

- Process of redefining method of super class inside sub class is called method overriding.

```
/*
class Employee extends Person{
    private int empid;
    private float salary;
    public Employee() {
    }
    public Employee(String name, int age, int empid, float salary) {
        super(name, age);
        this.empid = empid;
        this.salary = salary;
    }
    //super class aur sub class method ka naam same ho tho
    //super keyword pr method call karna pdta hai
    public void printRecord( ) {
        super.printRecord();
        System.out.println("Empid : "+this.empid);
        System.out.println("Salary : "+this.salary);
    }
}

public class Program {
    public static void main(String[] args) { //Client <= SunBeam
        Employee emp = new Employee("Sandeep", 38, 33, 45000.50f );
        emp.printRecord();
    }
}
```

Ex.9

```
package com.sunbeaminfo.cj06.inheritance.test;
```

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

```
class Person{
    private String name;
    private int age;
    public Person() {
    }
```

```
public Person(String name, int age) {
    this.name = name;
    this.age = age;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public int getAge() {
    return age;
}
public void setAge(int age) {
    this.age = age;
}
}
class Employee extends Person{
    private int empid;
    private float salary;
    public Employee() {
    }
    public Employee(String name, int age, int empid, float salary) {
        super(name, age);
        this.empid = empid;
        this.salary = salary;
    }
    public int getEmpid() {
        return empid;
    }
    public void setEmpid(int empid) {
        this.empid = empid;
    }
    public float getSalary() {
        return salary;
    }
    public void setSalary(float salary) {
        this.salary = salary;
    }
}
```

```
class InheritanceTest{
    private static Scanner sc = new Scanner(System.in);
    Person person = new Person();
    Employee emp = new Employee();
    public void acceptPersonRecord( ) {
        System.out.print("Name : ");
        person.setName(sc.nextLine());
        System.out.print("Age : ");
        person.setAge(sc.nextInt());
    }
    public void acceptEmployeeRecord( ) {
        System.out.print("Name : ");
        emp.setName(sc.nextLine());
        System.out.print("Age : ");
        emp.setAge(sc.nextInt());
        System.out.print("Empid : ");
        emp.setEmpid(sc.nextInt());
        System.out.print("Salary : ");
        emp.setSalary(sc.nextFloat());
    }
    public void printPersonRecord( ) {
        System.out.println("Name : "+person.getName());
        System.out.println("Age : "+person.getAge());
    }
    public void printEmployeeRecord( ) {
        System.out.println("Name : "+emp.getName());
        System.out.println("Age : "+emp.getAge());
        System.out.println("Empid : "+emp.getEmpid());
        System.out.println("Salary : "+emp.getSalary());
    }
}
public class Program {
    public static void main(String[] args) {
        InheritanceTest test = new InheritanceTest();
        //test.acceptPersonRecord();
        //test.printPersonRecord();

        test.acceptEmployeeRecord();
        test.printEmployeeRecord();
    }
}
```

```
}

//I don't want multiple printzRecord and acceptRecord. Here only two
class but if we have more.

-----
package com.sunbeaminfo.cj06.inheritance.test;
import java.util.Scanner;
class Person{
    private String name;
    private int age;
    public Person() {
    }
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
class Employee extends Person{
    private int empid;
    private float salary;
    public Employee() {
    }
    public Employee(String name, int age, int empid, float salary) {
        super(name, age);
        this.empid = empid;
        this.salary = salary;
    }
    public int getEmpid() {
        return empid;
    }
}
```

```
}

public void setEmpid(int empid) {
    this.empid = empid;
}

public float getSalary() {
    return salary;
}

public void setSalary(float salary) {
    this.salary = salary;
}

}

class InheritanceTest{
    private static Scanner sc = new Scanner(System.in);
    private Person person; //null
    public void setPerson(Person person) {
        this.person = person;
    }
    public void acceptRecord( ) {
        if( this.person != null ) {
            //TODO
        }
    }
    public void printRecord( ) {
        if( this.person != null ) {
            //TODO
        }
    }
}

public class Program {
    public static void main(String[] args) {
        InheritanceTest test = new InheritanceTest();
        test.setPerson( new Person() );
        //test.acceptRecord();
        //test.printRecord();

        test.setPerson( new Employee() );
        //test.acceptRecord();
        //test.printRecord();
    }
}
```

Ex.10

```
package com.sunbeaminfo.cj06.inheritance.test;

import java.awt.image.RePLICATEScaleFilter;
import java.util.Scanner;

class Person{
    private String name;
    private int age;
    public Person() {
    }
    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public void showRecord( ) {
        System.out.println("Name : "+this.name);
        System.out.println("Age : "+this.age);
    }
    public void printRecord( ) {
        System.out.println("Name : "+this.name);
        System.out.println("Age : "+this.age);
    }
}
class Employee extends Person{
    private int empid;
    private float salary;
    public Employee() {
    }
    public Employee(String name, int age, int empid, float salary) {
        super(name, age);
        this.empid = empid;
        this.salary = salary;
    }
    public void printRecord( ) {
        super.printRecord();
        System.out.println("Empid : "+this.empid);
        System.out.println("Salary : "+this.salary);
    }
}
```

```
public void displayRecord( ) {  
    //this.showRecord(); //OK  
    super.showRecord();  
    System.out.println("Empid : "+this.empid);  
    System.out.println("Salary : "+this.salary);  
}  
}  
  
public class Program {  
    /* • During inheritance, members of sub class do not inherit into  
       super class.  
       Hence using super class instance, we can access members of super class  
       only.  
    */  
    public static void main1(String[] args) {  
        Person person = new Person();  
        person.showRecord(); //OK  
        //person.displayRecord(); //Not OK  
    }  
    /* • During inheritance, members of super class, inherit into sub  
       class. Hence  
       using sub class instance, we can access members of super class as well  
       as sub  
    */  
    public static void main2(String[] args) {  
        Employee emp = new Employee();  
        //emp .showRecord(); //OK  
        emp.displayRecord(); //OK  
    }  
    public static void main3(String[] args) {  
        Person p1 = null; //OK  
  
        Person p2 = new Person(); //OK  
  
        Employee e3 = new Employee(); //OK  
        Person p3 = e3; //OK => Upcasting  
  
        Person p4 = new Employee(); //OK => Upcasting  
    }  
    public static void main4(String[] args) {
```

```
//Employee emp1 = null; //OK

//Employee emp2 = new Employee(); //OK

//Person p3 = new Person();
//Employee emp3 = (Employee) p3; //NOT Ok: ClassCastException

Employee emp4 = (Employee) new Person(); //NOT OK:
ClassCastException
}

public static void main5(String[] args) {
    Person p1 = null; //OK

    Person p2 = new Person(); //OK

    Person p3 = new Employee(); //OK: Upcasting
    //Super class reference can contain reference of instance of
    sub class.

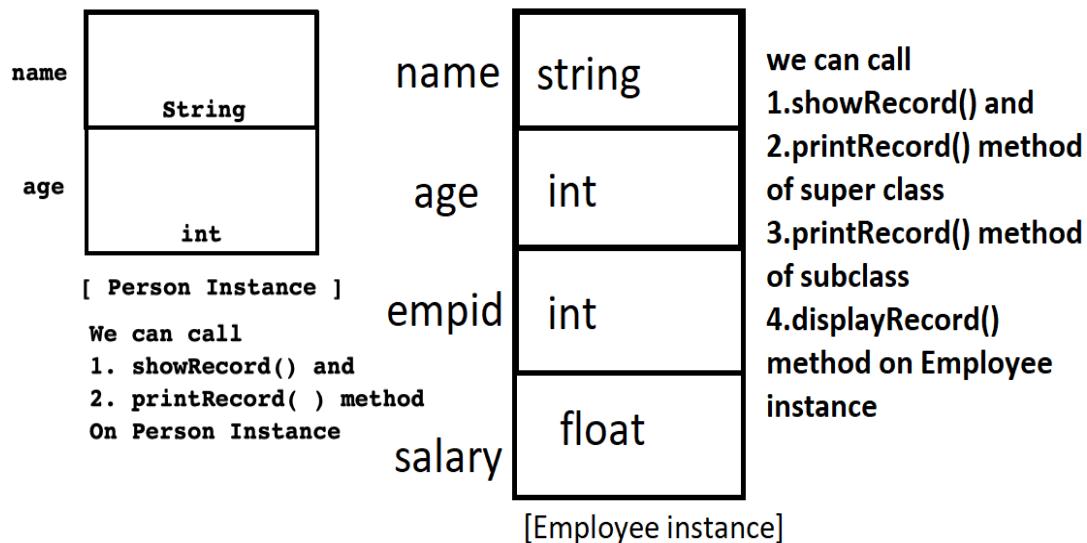
    Employee emp1 = null; //OK

    Employee emp2 = new Employee(); //OK

    //Employee emp3 = new Person(); //NOT OK
    //Sub class reference can not contain reference of instance of
    super class.
}
```

- If name of super class and sub class method is same and if we try to call such method on instance of sub class then preference is given to the method of sub class. This is called shadowing.
- Using super keyword, we can access members of super class inside method of sub class.
- During inheritance, members of sub class do not inherit into super class. Hence using super class instance, we can access members of super class only.
- During inheritance, members of super class, inherit into sub class. Hence using sub class instance, we can access members of super class as well as sub class.

- Members of super class, inherit into sub class. Hence we can consider, sub class instance as a super class instance.
- Example : Employee is a Person
 - Since Employee contains all the properties and behaviour of person hence Employee is a person.
- Since sub class instance can be considered as super class instance, we can use it in place super class instance.
 - During inheritance, members of sub class, do not inherit into super class. Hence using super class instance we can access members of super class only.
 - During inheritance, members of super class, inherit into sub class. Hence using Sub class instance we can access m



- Since members of super class, inherit into sub class hence we can consider sub class instance as a super class instance. Example: Employee is a person.
- Since sub class instance can be considered as super class instance, we can use it in place of super class instance.
- Since members of sub class, do not inherit into super class, we can not consider super class instance as as sub class instance. Example: Every Person is not a Employee.
- Since super class instance can not be considered as sub class instance, we can not use it in place sub class instance.

Upcasting & Downcasting :-

```
package com.sunbeaminfo.cj06.inheritance.test;
class Person{
    String name;
    int age;
}
class Employee extends Person{
    int empid;
    float salary;
}
public class Program {

    public static void main1(String[] args) {
        //Super class ke instance ke through sirf super class
        // ke member hi access ho rahe hai
        Person p = new Person();
        p.name = "Sandeep"; //OK
        p.age = 38; //OK
        //p.empid = 33; //Not OK
        //p.salary = 45000.50f; //NOT OK

        System.out.println("Name : "+p.name); //OK
        System.out.println("Age : "+p.age); //OK
        //System.out.println("Empid : "+p.empid); //Not OK
        //System.out.println("Salary : "+p.salary); //Not OK
    }
    public static void main2(String[] args) {
        //sub class ke instance ke through super class as well as
        // subclass
        // ke member ko bhi access kr sakte hai
        Employee emp = new Employee();
        emp.name = "Sandeep"; //OK
        emp.age = 38; //OK
        emp.empid = 33; //OK
        emp.salary = 45000.50f; //OK

        System.out.println("Name : "+emp.name); //OK
        System.out.println("Age : "+emp.age); //OK
    }
}
```

```
System.out.println("Empid : "+emp.empid); //OK
System.out.println("Salary : "+emp.salary); //OK
}
public static void main3(String[] args) {
    //Super class reference can contain reference of sub class
    instance. It is called upcasting.
    Person p = new Employee(); //Upcasting
    p.name = "Sandeep"; //OK
    p.age = 38; //OK
    //p.empid = 33; //Not OK
    //p.salary = 45000.50f; //NOT OK

    Employee emp = (Employee) p; //Downcasting
    emp.empid = 33;
    emp.salary = 45000.50f;

    System.out.println("Name : "+p.name); //OK
    System.out.println("Age : "+p.age); //OK
    //System.out.println("Empid : "+p.empid); //NOT OK
    //System.out.println("Salary : "+p.salary); //NOT OK
    System.out.println("Empid : "+emp.empid); //OK
    System.out.println("Salary : "+emp.salary); //OK
}
public static void main4(String[] args) {
    Employee emp = new Employee();
    emp.name = "Sandeep"; //OK
    emp.age = 38; //OK
    emp.empid = 33; //OK
    emp.salary = 45000.50f; //OK

    //Person p = ( Person)emp; //Upcasting
    Person p = emp; //Upcasting
    System.out.println("Name : "+p.name); //OK
    System.out.println("Age : "+p.age); //OK
    //System.out.println("Empid : "+p.empid); //Not OK
    //System.out.println("Salary : "+p.salary); //Not OK
}
}
```

- Process of converting reference of sub class into reference of super class is called as upcasting.
- Using upcasting, we can minimize object dependency in the code. In other words, we can reduce maintenance of the code.
- During upcasting, explicit type casting is optional.
- Super class reference variable can contain reference of sub class instance. It is also called as upcasting.

➤ `Person p = new Employee(); //Upcasting`

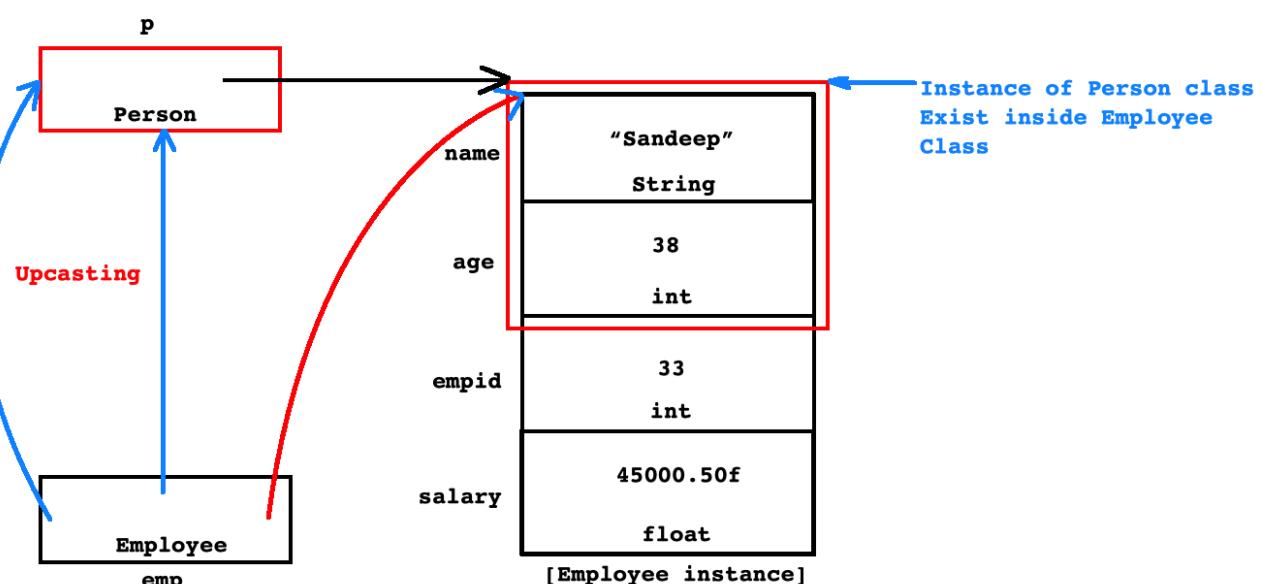
- In case of upcasting, using super class reference variable, we can access:

1. Fields of super class 2. Methods of super class 3. overridden methods of sub class

- In case of upcasting, using super class reference variable, we can not access:

1. fields of sub class 2. Non overridden methods of sub class.

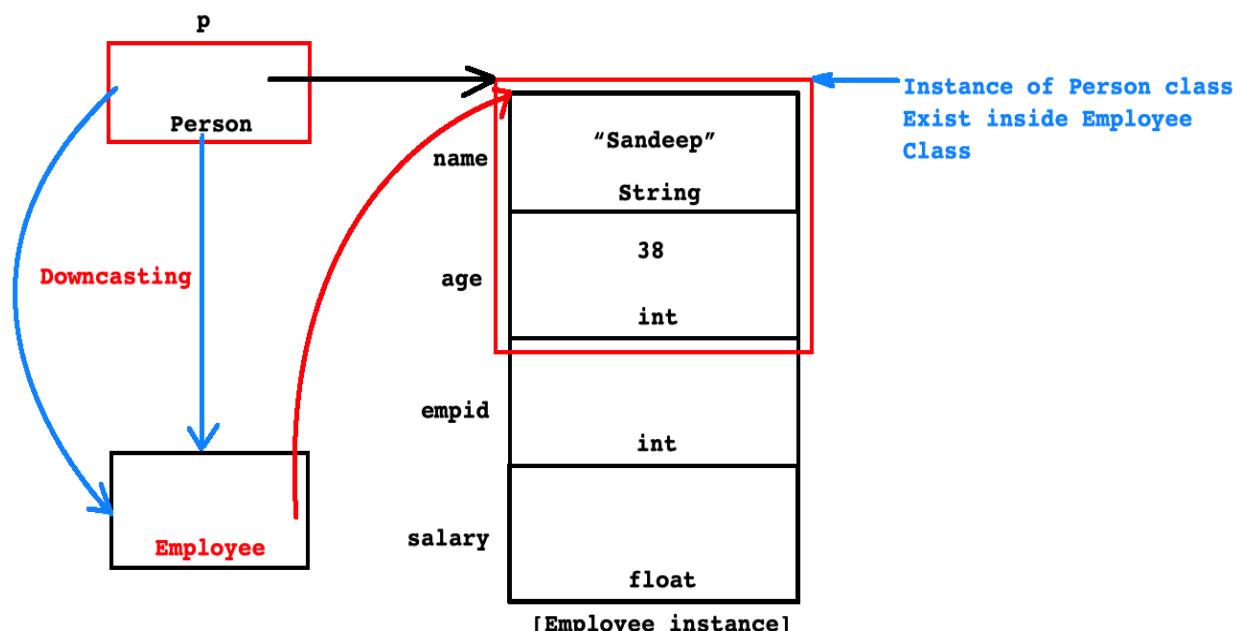
- If we want to access fields and non overridden methods of sub class then we should do down casting.



- + Upcasting
1. Process of converting, reference of sub class into reference of super class is called as upcasting.
 2. Super class reference can contain reference of instance of sub class. It is called upcasting.
 3. In case of upcasting, explicit type casting is optional.

- Process of converting reference of super class into reference of sub class is called down casting.
- Only in case of upcasting, we should do down casting. Otherwise JVM will throw ClassCastException.
- In case of upcasting, using super class reference variable, we can access overridden method of sub class. It is also called as dynamic method dispatch.

<pre>Person p = new Employee(); //Upcasting Employee emp = (Employee)p; //Downcasting : OK</pre>	<pre>Person p = null; Employee emp = (Employee)p; //OK : Downcasting //p = null //emp = null</pre>
<pre>Person p = new Person(); Employee emp = (Employee)p; //Downcasting : ClassCastException</pre>	



- Process of converting reference of super class into reference of sub class is called down-casting.
- In case of down-casting, explicit type casting is mandatory,

```
Person p = new Employee(); //Upcasting
Employee emp = (Employee)p; //Downcasting
```

Ex.

```
// package com.sunbeaminfo.cjo6.inheritance.test;
/*
```

- Process of redefining method of super class inside sub class is called method overriding

- In case of upcasting, process of calling method of sub class using reference of super class
is called dynamic method dispatch.

```
/*
class A{    //Super Class
    public void f1( ) {
        System.out.println("A.f1");
    }
    public void f2( ) {
        System.out.println("A.f2");
    }
}
class B extends A{ //Sub Class
    public void f2( ) {
        System.out.println("B.f2");
    }
    public void f3( ) {
        System.out.println("B.f3");
    }
}
public class Program {
    public static void main(String[] args) {
        A a = new B( ); //Upcasting
        a.f2(); //OK      :  B.f2    => Dynamic Method Dispatch
    }
    public static void main1(String[] args) {
        A a = new B( ); //Upcasting
        a.f1(); //OK      :  A.f1

        //a.f3(); //Not OK
        B b = (B) a; //Downcasting
        b.f3(); //OK      :  B.f3
    }
}
```

Ex.

```
package com.sunbeaminfo.cjo6.inheritance.test;
//In Java, all the methods are by default virtual.
class A{    //Super Class
    public void f2( ) {
```

```
        System.out.println("A.f2");
    }
}
class B extends A{ //Sub Class
    public void f2( ) {
        System.out.println("B.f2");
    }
    public void f3( ) {
        System.out.println("B.f3");
    }
}
public class Program {
    public static void main(String[] args) {
        A a = new B();
        a.f2(); //OK : B.f2
        //a.f3( ); //NOT OK
    }
}
```

Overriding:-

- Process of redefining method of super class inside sub class is called method overriding.
- Method redefined in sub class is called overridden method.
- When we call method of sub class using reference of super class then it is called dynamic method dispatch.
- If implementation of super class method is partially complete then we should redefine/override method inside sub class.
- Rules of method overriding
 1. Access modifier in sub class method should be same or it should be wider.
 2. Return type of sub class method should be same or it should be sub type. In other words, it should be covariant.
 3. Name of the method, number of parameters and type of parameters inside sub class method must be same.
 4. Checked exception list inside sub class method should be same or it should be sub set.
- Override is annotation declared in java.lang package.
- We can use this annotation on method only.
- It helps developer to override method inside sub class.

- + Method Overriding
 - If implementation of super class method is logically incomplete then we should override method inside sub class.
 - Process of redefining method of super class, inside sub class is called method overriding.
- Rules of method overriding:
 1. Access modifier of sub class method should be same or its should be wider.
 2. Return type of sub class method should be same or it should be sub type. In other words it should be co-variant.
 3. Name of sub class method, number of parameters and type of parameters must be same.
 4. Checked exception list in sub class method should be same or it should be sub set.
- Override is annotation declared in java.lang package.
- To verify whether method is overridden properly not then we should use @Overrode annotation.

```
class A{  
    public void print( ){  
        //TODO  
    }  
}  
class B extends A{  
    @Override  
    public void print( ){  
        //TODO  
    }  
}
```

```
package com.sunbeaminfo.cjo6.inheritance.test;
```

```
abstract class A{  
    public abstract void f1( );  
    public abstract void f2( );  
    public abstract void f3( );  
}  
abstract class Helper extends A{  
    @Override public void f1( ) { }  
    @Override public void f2( ) { }  
    @Override public void f3( ) { }  
}  
class B extends Helper{  
    @Override  
    public void f1() {  
        System.out.println("A.f1");  
    }  
}  
class C extends Helper{  
    @Override
```

```
public void f2() {
    System.out.println("B.f2");
}
}

class D extends Helper{
    @Override
    public void f3() {
        System.out.println("C.f3");
    }
}

public class Program {
    public static void main(String[] args) {
        A a = null;

        a = new B();
        a.f1();

        a = new C();
        a.f2();

        a = new D();
        a.f3();

    }
}

-----
```

```
package com.sunbeaminfo.cjo6.inheritance.test;

abstract class A{
    private int num1;
    private int num2;
    public A( int num1, int num2 ) { //Sole Constructor
        this.num1 = num1;
        this.num2 = num2;
    }
    public void print( ) {
        System.out.println("Num1 : "+this.num1);
        System.out.println("Num2 : "+this.num2);
    }
}
```

```
class B extends A{
    private int num3;
    public B(int num1, int num2, int num3) {
        super(num1, num2); //Look Here
        this.num3 = num3;
    }
    @Override
    public void print() {
        super.print();
        System.out.println("Num3 : "+this.num3);
    }
}
public class Program {
    public static void main(String[] args) {
        A a = new B( 10, 20, 30 );
        a.print();
    }
}
```

```
package com.sunbeaminfo.cjo6.inheritance.test;
class Account{
    private String name;
    private int number;
    private String type;
    private float balance;
    public Account() {
    }
    public Account(String name, int number, String type, float
balance) {
        this.name = name;
        this.number = number;
        this.type = type;
        this.balance = balance;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
```

```
        this.name = name;
    }
    public int getNumber() {
        return number;
    }
    public void setNumber(int number) {
        this.number = number;
    }
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
    public float getBalance() {
        return balance;
    }
    public void setBalance(float balance) {
        this.balance = balance;
    }
    @Override
    public String toString() {
        return "Account [name=" + name + ", number=" + number +",
type=" + type + ", balance=" + balance + "]";
    }
}

abstract class Service{
    public abstract void sendMessage( String message );
}

class EmailService extends Service{
    @Override
    public void sendMessage( String message ){
        System.out.println( "Email: "+message );
    }
}

class SMSService extends Service{
    @Override
    public void sendMessage( String message ){
        System.out.println( "SMS: "+message );
    }
}
```

```
    }
}

class Bank{
    private Account[] accounts;
    public Bank( ) {
        this.accounts = new Account[ 3 ]; //Array of references
        this.accounts[ 0 ] = new Account("Amit", 123,"Saving",
50000.50f);
        this.accounts[ 1 ] = new Account("Digvijay",456,"Current",
42000.50f);
        this.accounts[ 2 ] = new Account("Rajiv",789,"Loan",
59000.50f);
    }
//private EmailService service = new EmailService();
//private SMSservice service = new SMSservice();

    private Service service; //null
    public void setService(Service service) {
        this.service = service;
    }
    public boolean deposit( int number, float amount ){
        for( int index = 0; index < this.accounts.length; ++ index ) {
            if( this.accounts[ index ].getNumber() == number ) {
                this.accounts[ index ].setBalance( this.accounts[
index ].getBalance() + amount );
                if( this.service != null ) {
                    service.sendMessage( amount+" is deposited.
Current balance is: "+this.accounts[ index ].getBalance());
                }
                return true;
            }
        }
        return false;
    }
}
public class Program {
    public static void main(String[] args) {
        Bank bank = new Bank();
        //bank.setService(new SMSservice());
        //bank.setService(new EmailService());
```

```
        boolean status = bank.deposit(456, 3000);

    }
}

package com.sunbeaminfo.cjo6.inheritance.test;

class Rectangle{
    private float area;
    private float length;
    private float breadth;
    public Rectangle() {
    }
    public void setLength(float length) {
        this.length = length;
    }
    public void setBreadth(float breadth) {
        this.breadth = breadth;
    }
    public void calculateArea( ) {
        this.area = this.length * this.breadth;
    }
    public float getArea() {
        return this.area;
    }
}
class Circle{
    private float area;
    private float radius;
    public Circle() {
    }
    public void setRadius(float radius) {
        this.radius = radius;
    }
    public void calculateArea( ) {
        this.area = (float) (Math.PI * Math.pow(this.radius, 2));
    }
}
```

```
public float getArea() {
    return this.area;
}
}
public class Program {
    public static void main(String[] args) {
        Circle c = new Circle();
        c.setRadius(10);
        c.calculateArea();
        System.out.println("Area : "+c.getArea());
    }
    public static void main1(String[] args) {
        Rectangle rect = new Rectangle();
        rect.setLength(10);
        rect.setBreadth(20);
        rect.calculateArea();
        System.out.println("Area : "+rect.getArea());
    }
}
```

```
package com.sunbeaminfo.cjo6.inheritance.test;
class Shape{
    protected float area;
    public Shape() {
    }
    public final float getArea() {
        return this.area;
    }
}
class Rectangle extends Shape{
    private float length;
    private float breadth;
    public Rectangle() {
    }
    public void setLength(float length) {
        this.length = length;
    }
    public void setBreadth(float breadth) {
```

```
        this.breadth = breadth;
    }
    public void calculateArea( ) {
        this.area = this.length * this.breadth;
    }
}
class Circle extends Shape{
    private float radius;
    public Circle() {
    }
    public void setRadius(float radius) {
        this.radius = radius;
    }
    public void calculateArea( ) {
        this.area = (float) (Math.PI * Math.pow(this.radius, 2));
    }
}
public class Program {
    public static void main1(String[] args) {
        Circle c = new Circle();
        c.setRadius(10);
        c.calculateArea();
        System.out.println("Area : "+c.getArea());
    }
    public static void main2(String[] args) {
        Rectangle rect = new Rectangle();
        rect.setLength(10);
        rect.setBreadth(20);
        rect.calculateArea();
        System.out.println("Area : "+rect.getArea());
    }
}

package com.sunbeaminfo.cjo6.inheritance.test;

import java.util.Scanner;

class Shape{
    protected float area;
    public Shape() {
```

```
    }
    public final float getArea() {
        return this.area;
    }
}
class Rectangle extends Shape{
    private float length;
    private float breadth;
    public Rectangle() {
    }
    public void setLength(float length) {
        this.length = length;
    }
    public void setBreadth(float breadth) {
        this.breadth = breadth;
    }
    public void calculateArea( ) {
        super.area = this.length * this.breadth;
    }
}
class Circle extends Shape{
    private float radius;
    public Circle() {
    }
    public void setRadius(float radius) {
        this.radius = radius;
    }
    public void calculateArea( ) {
        super.area = (float) (Math.PI * Math.pow(this.radius, 2));
    }
}
public class Program {
    private static Scanner sc = new Scanner(System.in);
    private static void acceptRecord(Rectangle rect) {
        System.out.print("Length : ");
        rect.setLength(sc.nextFloat());
        System.out.print("Breadth : ");
        rect.setBreadth(sc.nextFloat());
    }
    private static void acceptRecord(Circle c) {
```

```
System.out.print("Radius : ");
c.setRadius(sc.nextFloat());
}
private static void printRecord(Rectangle rect) {
    System.out.println("Area : "+rect.getArea());
}
private static void printRecord(Circle c) {
    System.out.println("Area : "+c.getArea());
}
public static int menuList( ) {
    System.out.println("0.Exit");
    System.out.println("1.Rectangle");
    System.out.println("2.Circle");
    System.out.print("Enter choice : ");
    return sc.nextInt();
}
public static void main(String[] args) {
    int choice;
    while( ( choice = Program.menuList( ) ) != 0 ) {
        switch( choice ) {
            case 1:
                Rectangle rect = new Rectangle();
                Program.acceptRecord( rect );
                rect.calculateArea();
                Program.printRecord( rect );
                break;
            case 2:
                Circle c = new Circle();
                Program.acceptRecord( c );
                c.calculateArea();
                Program.printRecord( c );
                break;
        }
    }
}
-----  
package com.sunbeaminfo.cjo6.inheritance.test;  
  
import java.util.Scanner;
```

```
abstract class Shape{
    protected float area;
    public Shape() {
    }
    public abstract void calculateArea( );
    public final float getArea() {
        return this.area;
    }
}
class Rectangle extends Shape{
    private float length;
    private float breadth;
    public Rectangle() {
    }
    public void setLength(float length) {
        this.length = length;
    }
    public void setBreadth(float breadth) {
        this.breadth = breadth;
    }
    @Override
    public void calculateArea( ) {
        super.area = this.length * this.breadth;
    }
}
class Circle extends Shape{
    private float radius;
    public Circle() {
    }
    public void setRadius(float radius) {
        this.radius = radius;
    }
    @Override
    public void calculateArea( ) {
        super.area = (float) (Math.PI * Math.pow(this.radius, 2));
    }
}
public class Program {
    private static Scanner sc = new Scanner(System.in);
```

```
private static void acceptRecord(Shape shape) {  
    if( shape instanceof Rectangle ) {  
        Rectangle rect = (Rectangle) shape; //Downcasting  
        System.out.print("Length : ");  
        rect.setLength(sc.nextFloat());  
        System.out.print("Breadth : ");  
        rect.setBreadth(sc.nextFloat());  
    }else {  
        Circle c = (Circle) shape; //Downcasting  
        System.out.print("Radius : ");  
        c.setRadius(sc.nextFloat());  
    }  
}  
  
private static void printRecord(Shape shape) {  
    System.out.println("Area : "+shape.getArea());  
}  
  
public static int menuList( ) {  
    System.out.println("0.Exit");  
    System.out.println("1.Rectangle");  
    System.out.println("2.Circle");  
    System.out.print("Enter choice : ");  
    return sc.nextInt();  
}  
  
public static void main(String[] args) {  
    int choice;  
    while( ( choice = Program.menuList( ) ) != 0 ) {  
        Shape shape = null;  
        switch( choice ) {  
            case 1:  
                shape = new Rectangle(); //Upcasting  
                break;  
            case 2:  
                shape = new Circle(); //Upcasting  
                break;  
        }  
        if( shape != null ) {  
            Program.acceptRecord( shape );  
            shape.calculateArea(); //Dynamic Method Dispatch  
            Program.printRecord( shape );  
        }  
    }  
}
```

```
        }
    }
}

-----
package com.sunbeaminfo.cjo6.inheritance.test;

import java.util.Scanner;

abstract class Shape{
    protected float area;
    public Shape() {
    }
    public abstract void calculateArea( );
    public final float getArea() {
        return this.area;
    }
}
class Rectangle extends Shape{
    private float length;
    private float breadth;
    public Rectangle() {
    }
    public void setLength(float length) {
        this.length = length;
    }
    public void setBreadth(float breadth) {
        this.breadth = breadth;
    }
    @Override
    public void calculateArea( ) {
        super.area = this.length * this.breadth;
    }
}
class Circle extends Shape{
    private float radius;
    public Circle() {
    }
    public void setRadius(float radius) {
        this.radius = radius;
    }
}
```

```
}

@Override
public void calculateArea( ) {
    super.area = (float) (Math.PI * Math.pow(this.radius, 2));
}
}

class ShapeTest{
    private Shape shape;
    public void setShape(Shape shape) {
        this.shape = shape;
    }
    private static Scanner sc = new Scanner(System.in);
    public void acceptRecord( ) {
        if( this.shape != null ) {
            if( shape instanceof Rectangle ) {
                Rectangle rect = (Rectangle) shape; //Downcasting
                System.out.print("Length : ");
                rect.setLength(sc.nextFloat());
                System.out.print("Breadth : ");
                rect.setBreadth(sc.nextFloat());
            }else {
                Circle c = (Circle) shape; //Downcasting
                System.out.print("Radius : ");
                c.setRadius(sc.nextFloat());
            }
            this.shape.calculateArea(); //DMD
        }
    }
    public void printRecord( ) {
        if( this.shape != null ) {
            System.out.println("Area : "+shape.getArea());
        }
    }
    public static int menuList( ) {
        System.out.println("0.Exit");
        System.out.println("1.Rectangle");
        System.out.println("2.Circle");
        System.out.print("Enter choice : ");
        return sc.nextInt();
    }
}
```

```
}

public class Program {
    public static void main(String[] args) {
        int choice;
        ShapeTest test = new ShapeTest();
        while( ( choice = ShapeTest.menuList( ) ) != 0 ) {
            switch( choice ) {
                case 1:
                    test.setShape( new Rectangle());
                    break;
                case 2:
                    test.setShape(new Circle());
                    break;
            }
            test.acceptRecord();
            test.printRecord();
        }
    }
}
```

```
package com.sunbeaminfo.cjo6.inheritance.test;
```

```
import java.util.Scanner;

abstract class Shape{
    protected float area;
    public Shape() {
    }
    public abstract void calculateArea( );
    public final float getArea() {
        return this.area;
    }
}
class Rectangle extends Shape{
    private float length;
    private float breadth;
    public Rectangle() {
    }
    public void setLength(float length) {
```

```
        this.length = length;
    }
    public void setBreadth(float breadth) {
        this.breadth = breadth;
    }
    @Override
    public void calculateArea( ) {
        super.area = this.length * this.breadth;
    }
}
class Circle extends Shape{
    private float radius;
    public Circle() {
    }
    public void setRadius(float radius) {
        this.radius = radius;
    }
    @Override
    public void calculateArea( ) {
        super.area = (float) (Math.PI * Math.pow(this.radius, 2));
    }
}
class ShapeFactory{
    public static Shape getInstance( int choice ) {
        Shape shape = null;
        switch( choice ) {
            case 1:
                shape = new Rectangle();
                break;
            case 2:
                shape = new Circle( );
                break;
        }
        return shape;
    }
}
class ShapeTest{
    private Shape shape;
    public void setShape(Shape shape) {
        this.shape = shape;
    }
}
```

```
}

private static Scanner sc = new Scanner(System.in);
public void acceptRecord( ) {
    if( this.shape != null ) {
        if( shape instanceof Rectangle ) {
            Rectangle rect = (Rectangle) shape; //Downcasting
            System.out.print("Length : ");
            rect.setLength(sc.nextFloat());
            System.out.print("Breadth : ");
            rect.setBreadth(sc.nextFloat());
        }else {
            Circle c = (Circle) shape; //Downcasting
            System.out.print("Radius : ");
            c.setRadius(sc.nextFloat());
        }
        this.shape.calculateArea(); //DMD
    }
}
public void printRecord( ) {
    if( this.shape != null ) {
        System.out.println("Area : "+shape.getArea());
    }
}
public static int menuList( ) {
    System.out.println("0.Exit");
    System.out.println("1.Rectangle");
    System.out.println("2.Circle");
    System.out.print("Enter choice : ");
    return sc.nextInt();
}
}

public class Program {
    public static void main(String[] args) {
        int choice;
        ShapeTest test = new ShapeTest();
        while( ( choice = ShapeTest.menuList( ) ) != 0 ) {
            Shape shape = ShapeFactory.getInstance(choice);
            test.setShape(shape);
            test.acceptRecord();
            test.printRecord();
        }
    }
}
```

```
        }
    }
}

package com.sunbeaminfo.cjo6.inheritance.test;

/*
 - If we want to compare state of instances then we should use
equals method.
 - equals is non final method of java.lang.Object class.
 - If we do not define equals method inside class then its super
class's equals method gets called.
 - If any class do not contain equals method then Object class's
equals method gets called.
 - Consider code of Object class's equals method:
    public boolean equals(Object obj) {
        return (this == obj);
    }
 - equals method of Object class compares state of references.
 - If we want to compare state of instances of non primitive type
then we should override equals method inside class.
*/
class Employee{
    private String name;
    private int empid;
    private float salary;
    public Employee(String name, int empid, float salary) {
        this.name = name;
        this.empid = empid;
        this.salary = salary;
    }
}
public class Program {
    public static void main1(String[] args) {
        int num1 = 10;
        int num2 = 10;
        if( num1 == num2 )
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
    }
}
```

```
//Output : Equal
//If we want to compare state of variable of primitive type
then we should use operator ==
}
public static void main2(String[] args) {
    /* int num1 = 10;
    int num2 = 10;
    if( num1.equals(num2) ) //Cannot invoke equals(int) on the
primitive type int
        System.out.println("Equal");
    else
        System.out.println("Not Equal"); */

    //In Java, primitive types are not classes.
    //Hence we can not call equals method on variable of primitive
type.
}
public static void main3(String[] args) {
    Employee emp1 = new Employee("Sandeep", 33, 45000.50f );
    Employee emp2 = new Employee("Sandeep", 33, 45000.50f );
    if( emp1 == emp2 )
        System.out.println("Equal");
    else
        System.out.println("Not Equal");

    //Output : Not Equal
    //We can use operator == with the variables of non primitive
types.
    //If we use operator == with variables of non primitive type
then
    //it doesn't compare state of instances. It compares state of
references.
}
public static void main4(String[] args) {
    Employee emp1 = new Employee("Sandeep", 33, 45000.50f );
    Employee emp2 = new Employee("Sandeep", 33, 45000.50f );
    if( emp1.equals(emp2))
        System.out.println("Equal");
    else
        System.out.println("Not Equal");
```

```
//Output: Not Equal
}
}

package com.sunbeaminfo.cjo6.inheritance.test;
class Employee{
    private String name;
    private int empid;
    private float salary;
    public Employee(String name, int empid, float salary) {
        this.name = name;
        this.empid = empid;
        this.salary = salary;
    }

    //Employee this = emp1;
    //Object obj = emp2;      //Upcasting
    @Override
    public boolean equals(Object obj) {
        if( obj != null ) {
            Employee other = (Employee) obj;      //Downcasting
            if( this.empid == other.empid )
                return true;
        }
        return false;
    }
}
public class Program {
    public static void main(String[] args) {
        Employee emp1 = new Employee("Sandeep", 33, 45000.50f );
        Employee emp2 = new Employee("Sandeep", 33, 45000.50f );
        //Employee emp2 = null;
        if( emp1.equals(emp2))
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output: Equal
    }
}
```

```
package com.sunbeaminfo.cjo6.inheritance.test;

//class Test => Resource Type
class Test implements AutoCloseable{
    public Test( ) {
        System.out.println("Inside constructor");
    }
    public void print( ) {
        System.out.println("Inside print method");
    }
    @Override
    public void close() throws Exception {
        System.out.println("Closing resource");
    }
}
public class Program {
    public static void main(String[] args) {
        Test t = null;
        t = new Test( );      //instance => Resource
        t.print();
    }
}
```

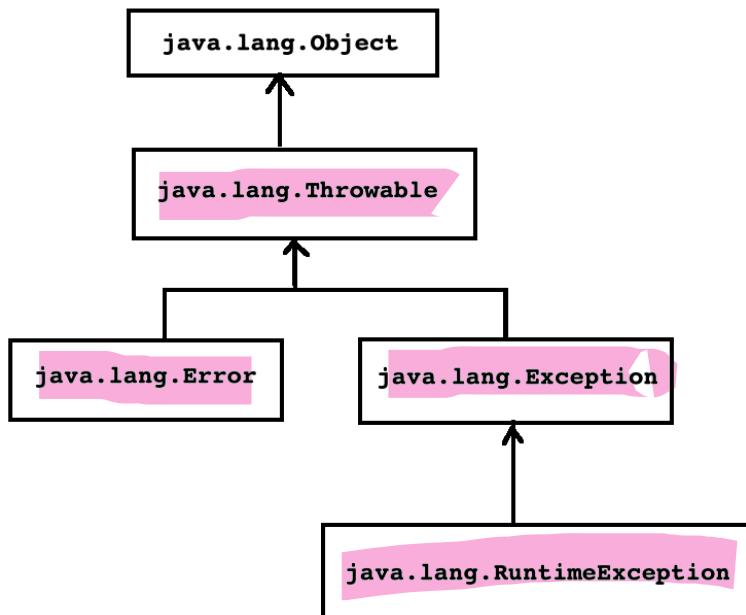
EXCEPTION HANDLING

-Basic of Exception Handling:-

Core Java Notes By ASHOK PATE

- + Operating System resources / non java resources / unmanaged resources:
 1. Memory (In Java, its JVM resource)
 2. File
 3. Thread
 4. Socket
 5. Connection
 6. H/W resources(CPU, Keyboard, Monitor)
- + Since operating system resources are limited, we should use it carefully. In other words, we should avoid their leakage.
- + To handle OS resources, carefully in the code, we should do exception handling.
- + To handle exception, we should use five keywords in Java.
 1. try
 2. catch
 3. throw
 4. throws
 5. finally,
- + What is exception?
 - Exception is an instance that we can use to send notification to the end user of system if any exceptional situation occurs in the program.
- + `java.lang.Throwable` is a sub class of `java.lang.Object` class.
- + `java.lang.Error` and `java.lang.Exception` are sub classes of `java.lang.Throwable` class.

-Hierarchy of Exception Handling:-



Throwable Class

- It is a class declared in `java.lang` package.
- The `Throwable` class is the super class of all errors and exceptions in the Java language.

- Only instances that are instances of Throwable class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement.

```
// package com.sunbeaminfo.cj06.exception.test;

import java.util.Scanner;

public class Program {
    public static void main(String[] args) {
        System.out.println("Opening resource");
        Scanner sc = null;
        sc = new Scanner(System.in);

        System.out.print("Num1 : ");
        int num1 = sc.nextInt();
        System.out.print("Num2 : ");
        int num2 = sc.nextInt();

        try {
            int result = num1 / num2;
            System.out.println("Result : "+result);
        }catch(ArithmaticException ex ) {
            //System.out.println(ex.getMessage());
            ex.printStackTrace(); //developer used this for message
        }
        System.out.println("Closing resource");
        sc.close();
    }
}
```

```
package com.sunbeaminfo.cj06.exception.test;

import java.util.InputMismatchException;
import java.util.Scanner;

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

```
System.out.println("Opening resource");
Scanner sc = null;
try {
    sc = new Scanner(System.in);
    System.out.print("Num1 : ");
    int num1 = sc.nextInt();
    System.out.print("Num2 : ");
    int num2 = sc.nextInt();
    int result = num1 / num2;
    System.out.println("Result : "+result);
} catch(ArithmetcException ex) {
    //System.out.println(ex.getMessage());
    //ex.printStackTrace();
    System.out.println("ArithmetcException");
} catch( InputMismatchException ex) {
    System.out.println("InputMismatchException");
}
System.out.println("Closing resource");
sc.close();
}
```

```
package com.sunbeaminfo.cj06.exception.test;

import java.util.InputMismatchException;
import java.util.Scanner;

public class Program {
    public static void main(String[] args) {
        System.out.println("Opening resource");
        Scanner sc = null;
        try {
            sc = new Scanner(System.in);
            System.out.print("Num1 : ");
            int num1 = sc.nextInt();
            System.out.print("Num2 : ");
            int num2 = sc.nextInt();
            int result = num1 / num2;
```

```
        System.out.println("Result : "+result);
    }catch(ArithmetricException | InputMismatchException ex ) {
        //Multi Catch Block
        System.out.println(ex.getClass().getSimpleName());
    }
    System.out.println("Closing resource");
    sc.close();
}
}
```

Class ArithmetricException

```
java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        java.lang.ArithmetricException
```

All Implemented Interfaces:
Serializable

```
public class ArithmetricException
extends RuntimeException
```

Thrown when an exceptional arithmetic condition has occurred. For example, an integer "divide by zero" throws an instance of this class. ArithmetricException objects may be constructed by the virtual machine as if suppression were disabled and/or the stack trace was not writable.

```
package com.sunbeaminfo.cj06.exception.test;
import java.util.Scanner;

public class Program {
    public static void main(String[] args) {
        System.out.println("Opening resource");
        Scanner sc = null;
        try {
            sc = new Scanner(System.in);
            System.out.print("Num1 : ");
            int num1 = sc.nextInt();
            System.out.print("Num2 : ");
            int num2 = sc.nextInt();
            int result = num1 / num2;
            System.out.println("Result : "+result);
        }catch( Exception ex ) {      //Generic Catch Block
            System.out.println(ex.getClass().getSimpleName());
        }
    }
}
```

```
        }
        System.out.println("Closing resource");
        sc.close();
    }
}

package com.sunbeaminfo.cj06.exception.test;
import java.util.Scanner;

public class Program {
    public static void main(String[] args) {
        System.out.println("Opening resource");
        Scanner sc = null;
        try {
            sc = new Scanner(System.in);
            System.out.print("Num1 : ");
            int num1 = sc.nextInt();
            System.out.print("Num2 : ");
            int num2 = sc.nextInt();
            int result = num1 / num2;
            System.out.println("Result : "+result);
        }catch( ArithmeticException ex ) {
            System.out.println("ArithmetcException");
        }catch( RuntimeException ex ) {
            System.out.println("RuntimeException");
        }catch( Exception ex ) { //Generic Catch Block
            System.out.println("Exception");
        }
        System.out.println("Closing resource");
        sc.close();
    }
}
```

- In case of hierarchy, It is necessary to handle all sub type of exception first.

```
try {
    //TODO
}catch (ArithmaticException e) {
    e.printStackTrace();
}catch (RuntimeException e) {
    e.printStackTrace();
}catch (Exception e) {
    e.printStackTrace();
}
```

- **throw**

- It is a keyword in Java.
- If we want to generate new exception then we should use throw keyword.
- Only objects that are instances of Throwable class (or one of its subclasses) are thrown by the Java Virtual Machine or can be thrown by the Java throw statement.
- throw statement is a jump statement.

```
package com.sunbeaminfo.cj06.exception.test;
import java.util.Scanner;

public class Program {
    public static void main(String[] args) {
        System.out.println("Opening resource");
        Scanner sc = null;
        try {
            sc = new Scanner(System.in);
            System.out.print("Num1 : ");
            int num1 = sc.nextInt();
            System.out.print("Num2 : ");
            int num2 = sc.nextInt();
            if( num2 == 0 ) {
                ArithmaticException ex = new
ArithmaticException("Divide by zero exception");
                throw ex;
            }else {
                int result = num1 / num2;
                System.out.println("Result : "+result);
            }
        } catch(ArithmaticException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

```
        }
    }catch( Exception ex ) {      //Generic Catch Block
        ex.printStackTrace();
    }
    System.out.println("Closing resource");
    sc.close();
}
-----  
  
package com.sunbeaminfo.cj06.exception.test;
import java.util.Scanner;

public class Program {
    public static void main(String[] args) {
        System.out.println("Opening resource");
        Scanner sc = null;
        try {
            sc = new Scanner(System.in);
            System.out.print("Num1 : ");
            int num1 = sc.nextInt();
            System.out.print("Num2 : ");
            int num2 = sc.nextInt();
            if( num2 == 0 )
                throw new ArithmeticException("Divide by zero
exception");
            int result = num1 / num2;
            System.out.println("Result : "+result);

        }catch( Exception ex ) {      //Generic Catch Block
            ex.printStackTrace();
        }
        System.out.println("Closing resource");
        sc.close();
    }
}
-----  
  
// package com.sunbeaminfo.cj06.exception.test;
import java.util.Scanner;
```

```
public class Program {  
    public static void main(String[] args) {  
        System.out.println("Opening resource");  
        Scanner sc = null;  
        try {  
            sc = new Scanner(System.in);  
            System.out.print("Num1 : ");  
            int num1 = sc.nextInt();  
            System.out.print("Num2 : ");  
            int num2 = sc.nextInt();  
            if( num2 == 0 )  
                //throw 0; //No exception of type int can be thrown;  
                //an exception type must be a subclass of Throwable  
                //throw new Program(); //No exception of type Program  
                //can be thrown; an exception type must be a subclass of Throwable  
                throw new ArithmeticException("Divide by zero  
exception");  
            int result = num1 / num2;  
            System.out.println("Result : "+result);  
  
        }catch( Exception ex ) {      //Generic Catch Block  
            ex.printStackTrace();  
        }  
        System.out.println("Closing resource");  
        sc.close();  
    }  
}
```

```
package com.sunbeaminfo.cj06.exception.test;  
import java.util.Scanner;  
  
public class Program {  
    public static void main(String[] args) {  
        System.out.println("Opening resource");  
        Scanner sc = null; //Scanner => Resource Type  
        try {  
            System.out.println("Inside try block");  
            sc = new Scanner(System.in); //Resource  
            System.out.print("Num1 : ");
```

```
        int num1 = sc.nextInt();
        System.out.print("Num2 : ");
        int num2 = sc.nextInt();
        if( num2 == 0 )
            //throw 0; //No exception of type int can be thrown;
an exception type must be a subclass of Throwable
            //throw new Program(); //No exception of type Program
can be thrown; an exception type must be a subclass of Throwable
            throw new ArithmeticException("Divide by zero
exception");
        int result = num1 / num2;
        System.out.println("Result : "+result);

    }catch( ArithmeticException ex ) {
        System.out.println("Inside catch block");
        ex.printStackTrace();
    }finally {
        System.out.println("Inside finally block");
        System.out.println("Closing resource");
        sc.close();
    }
}
}
```

```
package com.sunbeaminfo.cj06.exception.test;
import java.util.Scanner;

public class Program {
    public static void main(String[] args) {
        try( Scanner sc = new Scanner(System.in) ){ //Try-With-
Resource
            System.out.print("Num1 : ");
            int num1 = sc.nextInt();
            System.out.print("Num2 : ");
            int num2 = sc.nextInt();
            if( num2 == 0 )
                throw new ArithmeticException("Divide by zero
exception");
    }
}
```

```
        int result = num1 / num2;
        System.out.println("Result : "+result);
    }catch( ArithmeticException ex ) {
        ex.printStackTrace();
    }
}
```

```
package com.sunbeaminfo.cj06.exception.test;
import java.util.Scanner;

public class Program {
    public static void main(String[] args) {
        try( Scanner sc1 = new Scanner(System.in);
             Scanner sc2 = new Scanner(System.in);
             Scanner sc3 = new Scanner(System.in); ){ //Try-
With-Resource
        //TODO
    }catch( ArithmeticException ex ) {
        ex.printStackTrace();
    }
}
}
```

compact1, compact2, compact3
java.lang

Class NumberFormatException

```
java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.RuntimeException
        java.lang.IllegalArgumentException
          java.lang.NumberFormatException
```

All Implemented Interfaces:
Serializable

```
public class NumberFormatException
extends IllegalArgumentException
```

Thrown to indicate that the application has attempted to convert a string to one of the numeric types, but that the string does not have the appropriate format.

Since:
JDK1.0

See Also:
Integer.parseInt(String), Serialized Form

```
package com.sunbeaminfo.cj06.exception.test;
import java.util.Iterator;
import java.util.Scanner;

public class Program {
    //public static void sleep(long millis) throws InterruptedException
    public static void showRecord( ) {
        try {
            for( int count = 1; count <= 10; ++ count ) {
                System.out.println("Count : "+count);
                Thread.sleep(350);
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    public static void displayRecord( ) throws InterruptedException {
        for( int count = 1; count <= 10; ++ count ) {
            System.out.println("Count : "+count);
            Thread.sleep(350);
        }
    }
    public static void main1(String[] args) {
        //public static int parseInt(String s) throws
        NumberFormatException

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter number : ");
        String strNumber = sc.nextLine();
        int number = Integer.parseInt(strNumber);
        System.out.println("Number : "+number);

    }
    public static void main2(String[] args) {
        //Program.showRecord();

        try {
            Program.displayRecord();
        } catch (InterruptedException e) {
```

Core Java Notes By ASHOK PATE

```
        e.printStackTrace();
    }
}
```

```
}
```

```
sleep
```

```
public static void sleep(long millis,
                        int nanos)
                throws InterruptedException
```

Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds plus the specified number of nanoseconds, subject to the precision and accuracy of system timers and schedulers. The thread does not lose ownership of any monitors.

Parameters:

millis - the length of time to sleep in milliseconds

nanos - 0-999999 additional nanoseconds to sleep

Throws:

IllegalArgumentException - if the value of millis is negative, or the value of nanos is not in the range 0-999999

InterruptedException - if any thread has interrupted the current thread. The *interrupted status* of the current thread is cleared when this exception is thrown.

Checked Exception : A,B,C

```
class Program{
    public static void f1( )throws A, B, C{
        if( cnd1 is failed )
            throw new A( );
        else if( cnd2 is failed )
            throw new B( );
        else if( cnd3 is failed )
            throw new C( );
        else
            //TODO
    }
    public static void f2( )throws Exception{
        if( cnd1 is failed )
            throw new A( );
        else if( cnd2 is failed )
            throw new B( );
        else if( cnd3 is failed )
            throw new C( );
        else
            //TODO
    }
}
```

Reproduction or transfer of any part

compact1, compact2, compact3
java.lang

Class InterruptedException

```
java.lang.Object
  java.lang.Throwable
    java.lang.Exception
      java.lang.InterruptedException
```

All Implemented Interfaces:
Serializable

```
public class InterruptedException
extends Exception
```

Thrown when a thread is waiting, sleeping, or otherwise occupied, and the thread is interrupted, either before or during the activity. Occasionally a method may wish to test whether the current thread has been interrupted, and if so, to immediately throw this exception. The following code can be used to achieve this effect:

```
if (Thread.interrupted()) // Clears interrupted status!
    throw new InterruptedException();
```

Since:
JDK1.0

See Also:
Object.wait(), Object.wait(long), Object.wait(long, int), Thread.sleep(long), Thread.interrupt(), Thread.interrupted(), Serialized Form

Custom Exception:-

```
package com.sunbeaminfo.cj06.exception.test;
/* @SuppressWarnings("serial")
class StackUnderflowException extends RuntimeException{ //UnChecked
Exception
    public StackUnderflowException(String message) {
        super(message);
    }
}
@SuppressWarnings("serial")
class StackOverflowflowException extends
RuntimeException{ //UnChecked Exception
    public StackOverflowflowException(String message) {
        super(message);
    }
}
 */
```

```
@SuppressWarnings("serial")
class StackUnderflowException extends Exception{ //Checked
Exception
    public StackUnderflowException(String message) {
        super(message);
    }
}
@SuppressWarnings("serial")
class StackOverflowflowException extends Exception{ //Checked
Exception
    public StackOverflowflowException(String message) {
        super(message);
    }
}
```

```
package com.sunbeaminfo.cj06.exception.test;

import java.util.Scanner;

@SuppressWarnings("serial")
class StackUnderflowException extends Exception{ //Checked
Exception
    public StackUnderflowException(String message) {
        super(message);
    }
}

@SuppressWarnings("serial")
class StackOverflowException extends Exception{ //Checked Exception
public StackOverflowException(String message) {
    super(message);
}
}

class Stack{
    private int top = -1;
    private int[] arr;
    public Stack( ) {
        this( 5 );
    }
    public Stack( int size ) {
        this.arr = new int[ size ];
    }
    public boolean empty( ) {
        return this.top == -1;
    }
    public boolean full( ) {
        return this.top == this.arr.length - 1 ;
    }
    public void push(int element) throws StackOverflowException {
        if( this.full() )
            throw new StackOverflowException("Stack is full");
        this.arr[ ++ this.top ] = element;
    }
    public int peek( ) throws StackUnderflowException {
        if( this.empty() )
            throw new StackUnderflowException("Stack is empty");
        return this.arr[ this.top ];
    }
}
```

```
        throw new StackUnderflowException("Stack is empty");
    return this.arr[ this.top ];
}
public void pop( ) throws StackUnderflowException {
    if( this.empty() )
        throw new StackUnderflowException("Stack is empty");
    -- this.top;
}
}
public class Program {
    static Scanner sc = new Scanner(System.in);
    private static void acceptRecord(int[] element) {
        System.out.print("Enter element :   ");
        element[ 0 ] = sc.nextInt();
    }
    private static void printRecord(int[] element) {
        System.out.println("Removed element :   "+element[ 0 ]);
    }
    public static int menuList( ) {
        System.out.println("0.Exit");
        System.out.println("1.Push");
        System.out.println("2.Pop");
        System.out.print("Enter choice :   ");
        return sc.nextInt();
    }
    public static void main(String[] args) {
        int choice;
        int[] element = new int[ 1 ];
        Stack stk = new Stack( 5 );
        while( ( choice = Program.menuList( ) ) != 0 ) {
            try {
                switch( choice ) {
                    case 1:
                        Program.acceptRecord( element );
                        stk.push( element[ 0 ] );
                        break;
                    case 2:
                        element[ 0 ] = stk.peek();
                        Program.printRecord(element);
                        stk.pop();
                }
            }
        }
    }
}
```

```
        break;
    }
} catch (StackOverflowException |  
StackUnderflowException e) {  
    System.out.println(e.getMessage());  
}  
}  
}  
}
```

Day- 14

```
package com.sunbeaminfo.cj06.test;

import java.util.Date;

//T      :      Type

//E      :      Element

//N      :      Number

//K      :      Key

//V      :      Value

//USR    :      Second Type Parameters Names
```

```
//class Box<T> => Parameterized Type
class Box<T>{    //T is called as Type parameter
    private T object;
    public T getObject() {
        return this.object;
    }
    public void setObject(T object) {
        this.object = object;
    }
}
```

```
}

public class Program {
    public static void main1(String[] args) {
        Box<Date> b1 = new Box<Date>(); //Date is called as Type
argument

        b1.setObject(new Date());

        Date date = b1.getObject();

        System.out.println(date.toString());
    }
    public static void main2(String[] args) {
        //Box<Date> b1 = new Box<Date>();

        Box<Date> b1 = new Box<>(); //OK

        //Box<> b1 = new Box<Date>(); //Not OK

        //Box<Object> b1 = new Box<Date>(); //Not OK

        //Box<Date> b1 = new Box<Object>(); //Not OK

        b1.setObject(new Date());

        Date date = b1.getObject();

        System.out.println(date.toString());
    }

    public static void main3(String[] args) {
        Box b1 = new Box(); //Box => Raw type
        //Box<Object> b1 = new Box<Object>(); //OK

        b1.setObject(new Date());
        String str = (String) b1.getObject(); //ClassCastException
        System.out.println(str);
    }
}
```

```
public static void main(String[] args) {
    //Box<int> b1 = new Box<int>(); //NOT OK
    Box<Integer> b1 = new Box<>(); //OK
}
```

```
package com.sunbeaminfo.cj06.test;
class Pair<K, V>{
    private K key;
    private V value;
    public void put( K key, V value ) {
        this.key = key;
        this.value = value;
    }
    public K getKey() {
        return key;
    }
    public V getValue() {
        return value;
    }
}
public class Program {
    public static void main(String[] args) {
        Pair<Integer, String> p = new Pair<>();
        p.put(1, "DAC");
        System.out.println("Key : "+p.getKey());
        System.out.println("Value : "+p.getValue());
    }
}
```

```
package com.sunbeaminfo.cj06.test;

import java.util.Date;
class Box<T extends Number>{//T => Bounded Type Parameter
    private T object;
    public T getObject() {
        return object;
    }
    public void setObject(T object) {
        this.object = object;
    }
}
```

```
    }
}

class Box1<T>{
    private T object;
    public T getObject() {
        return object;
    }
    public void setObject(T object) {
        this.object = object;
    }
}

public class Program {
    public static void main1(String[] args) {
        Box1<Number> b1 = new Box1<>(); //OK

        Box1<Boolean> b2 = new Box1<>(); //OK

        Box1<Character> b3 = new Box1<>(); //OK

        Box1<Integer> b4 = new Box1<>(); //OK

        Box1<Double> b5 = new Box1<>(); //OK

        Box1<String> b6 = new Box1<>(); //OK

        Box1<Date> b7 = new Box1<>(); //OK
    }

    public static void main2(String[] args) {
        Box<Number> b1 = new Box<>(); //OK

        Box<Boolean> b2 = new Box<>(); //Not OK

        Box<Character> b3 = new Box<>(); //Not OK

        Box<Integer> b4 = new Box<>(); //OK

        Box<Double> b5 = new Box<>(); //OK

        Box<String> b6 = new Box<>(); //Not OK
    }
}
```

```
Box<Date> b7 = new Box<>(); //Not OK
}

-----
package com.sunbeaminfo.cj06.test;

import java.util.ArrayList;

public class Program {
    private static ArrayList<Integer> getIntegerList() {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(20);
        list.add(30);
        return list;
    }
    private static void printIntegerList( ArrayList<Integer> list ) {
        for( Integer element : list )
            System.out.println(element );
    }
    public static void main(String[] args) {
        ArrayList<Integer> list = Program.getIntegerList();
        Program.printIntegerList( list );
    }

    public static void main1(String[] args) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(20);
        list.add(30);
        for( Integer element : list )
            System.out.println(element );
    }
}

-----
package com.sunbeaminfo.cj06.test;
import java.util.ArrayList;

public class Program {
```

```
private static ArrayList<Integer> getIntegerList( ) {
    ArrayList<Integer> list = new ArrayList<>();
    list.add(10);
    list.add(20);
    list.add(30);
    return list;
}
private static ArrayList<Double> getDoubleList( ) {
    ArrayList<Double> list = new ArrayList<>();
    list.add(10.1);
    list.add(20.2);
    list.add(30.3);
    return list;
}
private static ArrayList<String> getStringList( ) {
    ArrayList<String> list = new ArrayList<>();
    list.add("Java");
    list.add("Python");
    list.add("Ruby");
    return list;
}
private static void printIntegerList( ArrayList<Integer> list ) {
    for( Integer element : list )
        System.out.println(element );
}
private static void printDoubleList( ArrayList<Double> list ) {
    for( Double element : list )
        System.out.println(element );
}
private static void printStringList( ArrayList<String> list ) {
    for( String element : list )
        System.out.println(element );
}
public static void main(String[] args) {
    ArrayList<Integer> integerList = Program.getIntegerList( );
    Program.printIntegerList( integerList );

    ArrayList<Double> doubleList = Program.getDoubleList();
    Program.printDoubleList( doubleList );
```

```
        ArrayList<String> stringList = Program.getStringList();
        Program.printStringList( stringList );

    }
}
```

Wild Card

- In generics "?" is called as wild card which represents unknown type.
- Types of wild card:
 1. Unbounded wild card
 2. Upper bounded wild card
 3. Lower bounded wilds card.

```
package com.sunbeaminfo.cj06.test;
import java.util.ArrayList;
public class Program {
    private static ArrayList<Integer> getIntegerList( ) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(20);
        list.add(30);
        return list;
    }
    private static ArrayList<Double> getDoubleList( ) {
        ArrayList<Double> list = new ArrayList<>();
        list.add(10.1);
        list.add(20.2);
        list.add(30.3);
        return list;
    }
    private static ArrayList<String> getStringList( ) {
        ArrayList<String> list = new ArrayList<>();
        list.add("Java");
        list.add("Python");
        list.add("Ruby");
        return list;
    }
    //UnBounded Wild Card
    private static void printList( ArrayList<?> list ) {
```

```
        System.out.println(list);
    }
    public static void main(String[] args) {
        ArrayList<Integer> integerList = Program.getIntegerList( );
        Program.printList( integerList );

        ArrayList<Double> doubleList = Program.getDoubleList();
        Program.printList( doubleList );

        ArrayList<String> stringList = Program.getStringList();
        Program.printList( stringList );

    }
}

package com.sunbeaminfo.cj06.test;

import java.util.ArrayList;

public class Program {
    private static ArrayList<Integer> getIntegerList( ) {
        ArrayList<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(20);
        list.add(30);
        return list;
    }
    private static ArrayList<Double> getDoubleList( ) {
        ArrayList<Double> list = new ArrayList<>();
        list.add(10.1);
        list.add(20.2);
        list.add(30.3);
        return list;
    }
    private static ArrayList<String> getStringList( ) {
        ArrayList<String> list = new ArrayList<>();
        list.add("Java");
        list.add("Python");
        list.add("Ruby");
        return list;
    }
}
```

```
}

//Upper bounded Wild Card
private static void printList( ArrayList<? extends Number> list )
{
    System.out.println(list);
}

public static void main(String[] args) {
    ArrayList<Integer> integerList = Program.getIntegerList();
    Program.printList( integerList ); //OK

    ArrayList<Double> doubleList = Program.getDoubleList();
    Program.printList( doubleList ); //OK

    ArrayList<String> stringList = Program.getStringList();
    Program.printList( stringList ); //Not OK
}

}
```

```
package com.sunbeaminfo.cj06.test;

import java.util.ArrayList;
import java.util.Date;

public class Program {
    private static ArrayList<Object> getObjectList() {
        ArrayList<Object> list = new ArrayList<>();
        list.add(true);
        list.add('A');
        list.add("Sandeep");
        list.add(123);
        list.add(new Date());
        return list;
    }
    private static ArrayList<Number> getNumberList() {
        ArrayList<Number> list = new ArrayList<>();
        list.add( new Integer(123) );
        list.add( new Float(123.45f) );
        list.add( new Double(123.45d) );
        return list;
    }
}
```

```
}

private static ArrayList<Integer> getIntegerList( ) {
    ArrayList<Integer> list = new ArrayList<>();
    list.add(10);
    list.add(20);
    list.add(30);
    return list;
}

private static ArrayList<Double> getDoubleList( ) {
    ArrayList<Double> list = new ArrayList<>();
    list.add(10.1);
    list.add(20.2);
    list.add(30.3);
    return list;
}

private static ArrayList<String> getStringList( ) {
    ArrayList<String> list = new ArrayList<>();
    list.add("Java");
    list.add("Python");
    list.add("Ruby");
    return list;
}

//Lower Bounded Wild Card
private static void printList( ArrayList< ? super Integer > list )
{
    System.out.println(list);
}

public static void main(String[] args) {
    ArrayList<Integer> integerList = Program.getIntegerList( );
    Program.printList( integerList ); //OK

    ArrayList<Double> doubleList = Program.getDoubleList();
    //Program.printList( doubleList ); //NOT OK

    ArrayList<String> stringList = Program.getStringList();
    //Program.printList( stringList ); //Not OK

    ArrayList<Number> numberList = Program.getNumberList();
    Program.printList(numberList); //OK
```

```
    ArrayList<Object> objectList = Program.getObjectList();
    Program.printList(objectList); //OK
}
}
```

Unbounded Wild Card

```
private static void printRecord(ArrayList<?> list) {
    for (Object element : list) {
        System.out.println(element);
    }
}
```

- In above code, list will contain reference of ArrayList which can contain any type of element.

```
public static void main(String[] args) {
    ArrayList<Integer> integerList = Program.getIntegerList();
    Program.printRecord( integerList ); //OK

    ArrayList<Double> doubleList = Program.getDoubleList();
    Program.printRecord( doubleList ); //OK

    ArrayList<String> stringList = Program.getStringList();
    Program.printRecord( stringList ); //OK
}
```

Upper Bounded Wild Card

```
private static void printRecord(ArrayList<? extends Number> list) {
    for (Object element : list) {
        System.out.println(element);
    }
}
```

- In above code, list will contain reference of ArrayList which can contain Number and its sub type of elements.

```
public static void main(String[] args) {
    ArrayList<Integer> integerList = Program.getIntegerList();
    Program.printRecord( integerList ); //OK

    ArrayList<Double> doubleList = Program.getDoubleList();
    Program.printRecord( doubleList ); //OK

    ArrayList<String> stringList = Program.getStringList();
    Program.printRecord( stringList ); //Not OK
}
```

Lower Bounded Wild Card

```
private static void printRecord(ArrayList< ? super Integer > list) {  
    for (Object element : list) {  
        System.out.println(element);  
    }  
}
```

- In above code, list will contain reference of ArrayList which can contain Integer and its super type of elements.

```
public static void main(String[] args) {  
    ArrayList<Integer> integerList = Program.getIntegerList();  
    Program.printRecord( integerList ); //OK  
  
    ArrayList<Double> doubleList = Program.getDoubleList();  
    Program.printRecord( doubleList ); //NOT OK  
  
    ArrayList<String> stringList = Program.getStringList();  
    Program.printRecord( stringList ); //NOT OK  
}
```

```
package com.sunbeaminfo.cj06.test;  
import java.util.Date;  
public class Program {  
    //Generic Method  
    private static void print(Object obj) {  
        System.out.println(obj);  
    }  
    public static void main(String[] args) {  
        Program.print( true );  
        Program.print( 'A' );  
        Program.print( 123 );  
        Program.print( 3.142 );  
        Program.print( "Hello" );  
        Program.print( new Date() );  
    }  
}
```

```
package com.sunbeaminfo.cj06.test;  
import java.util.Date;  
public class Program {  
    /* private static <T> void print(T obj) {
```

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

private static <T extends Number> void print(T obj) {
    System.out.println(obj);
}

public static void main(String[] args) {
    //Program.print( true );      //Not OK
    //Program.print( 'A' ); //Not OK
    Program.print( 123 );      //OK
    Program.print( 3.142 ); //OK
    //Program.print( "Hello" ); //Not OK
    //Program.print( new Date() ); //Not OK
}
}
```

Restrictions on Generics

1. Cannot Instantiate Generic Types with Primitive Types
2. Cannot Create Instances of Type Parameters
3. Cannot Declare Static Fields Whose Types are Type Parameters
4. Cannot Use Casts or instanceof with Parameterized Types
5. Cannot Create Arrays of Parameterized Types
6. Cannot Create, Catch, or Throw Objects of Parameterized Types
7. Cannot Overload a Method Where the Formal Parameter Types of Each Overload Erase to the Same Raw Type

```
package com.sunbeaminfo.cj06.test;
import java.util.ArrayList;
import java.util.List;

class MyException<T> extends RuntimeException{ //NOT OK: The generic
class MyException<T> may not subclass java.lang.Throwable
}
class Box<T>{
    //private static T ref; //NOT OK: Cannot make a static reference to
the non-static type T
    public void print( ) {
        //T t = new T();      //NOT OK: Cannot instantiate the type T
    }
}
```

```
}

public class Program {
    public static void main(String[] args) {
        ArrayList<Integer>[ ] arr = new ArrayList<Integer>[ 3 ]; //Not
OK: Cannot create a generic array of ArrayList<Integer>
    }
    public static void main2(String[] args) {
        List<Integer> list = new ArrayList<Integer>(); //Upcasting
        if( list instanceof ArrayList<Integer> ) { //NOT OK: cannot
perform instanceof check against parameterized type
ArrayList<Integer>.
        //TODO
    }
}
public static void main1(String[] args) {
    //ArrayList<int> list = new ArrayList<>(); //Not OK
    ArrayList<Integer> list = new ArrayList<>(); //OK
}

```

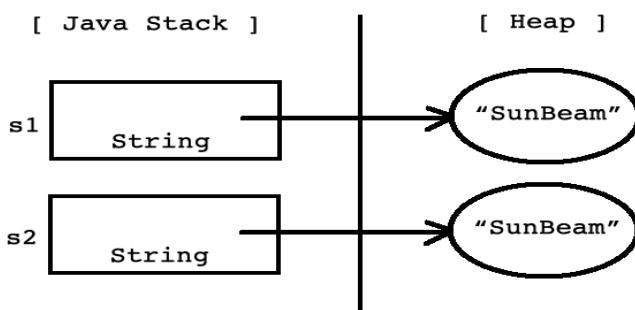
```
// package com.sunbeaminfo.cj06.test;
public class Program {
    public static void main(String[] args) {
        int count = 1;
        //unicode table char printing
        for( char ch = 2304; ch <= 2431; ++ ch ) {
            System.out.print(ch+ " ");
            ++ count;
            if( count == 10 ) {
                System.out.println();
                count = 1;
            }
        }
    }
}
```

```
package com.sunbeaminfo.cj06.test;
public class Program {
    //String is a class hence it is non primitive type
    //It is a final class declared in java.lang package.
    // String is collection of Character objects

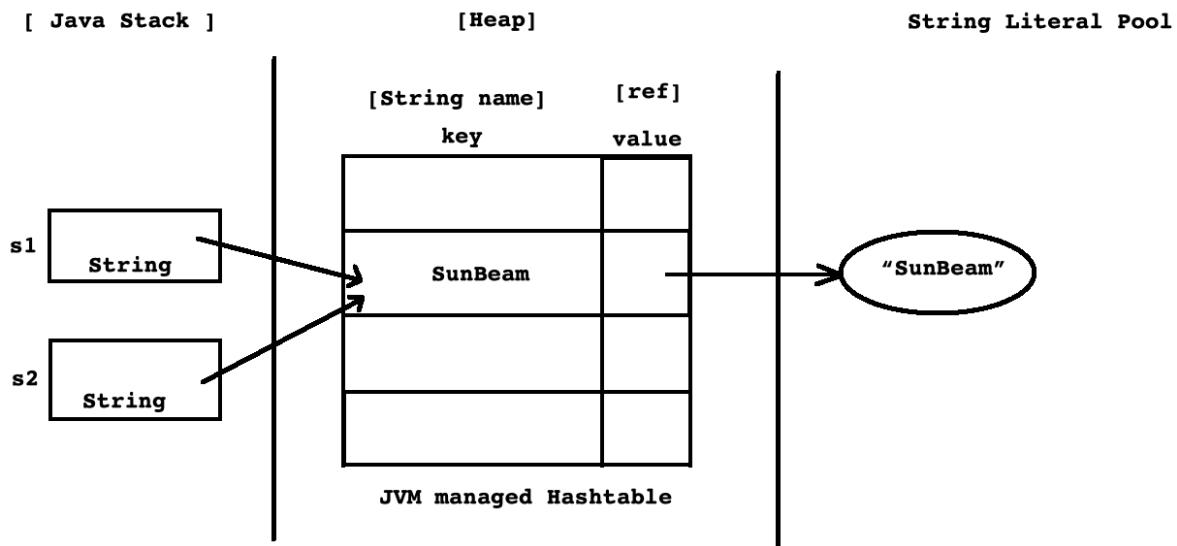
    public static void main1(String[] args) {
        //We can create String using new operator.
        //if we create String using new operator then it is called as
        string instance.
        //String instance get space on heap
        String s1 = new String("SunBeam"); //new String("SunBeam") =>
String instance
        System.out.println(s1); //SunBeam
    }
    public static void main2(String[] args) {
        String s1 = new String("SunBeam");
        String s2 = new String("SunBeam");
        if( s1 == s2 )
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output: Not Equal
    }
    public static void main3(String[] args) {
        String s1 = new String("SunBeam");
        String s2 = new String("SunBeam");
        if( s1.equals(s2) )
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output: Equal
    }
    public static void main4(String[] args) {
        //We can create String without new operator.
        //if we create String without new operator then it is called
        as string literal.
        //String literal get space on String literal pool
        String s1 = "SunBeam"; //"SunBeam" => String literal
```

```
/* char[] data = { 'S', 'u', 'n', 'B', 'e', 'a', 'm'}; //internally conversion
String s1 = new String(data); */
}

public static void main5(String[] args) {
    String s1 = "SunBeam";
    String s2 = "SunBeam";
    if( s1 == s2 )
        System.out.println("Equal");
    else
        System.out.println("Not Equal");
    //Output: Equal
}
public static void main(String[] args) {
    String s1 = "SunBeam";
    String s2 = "SunBeam";
    if( s1.equals(s2))
        System.out.println("Equal");
    else
        System.out.println("Not Equal");
    //Output: Equal
}
}
```

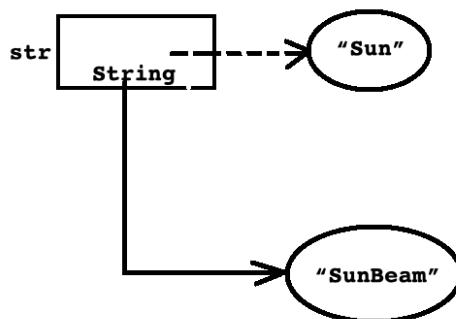


```
String s1 = new String("SunBeam");
String s2 = new String("SunBeam");
if( s1 == s2 )
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output: Not Equal
```



```
package com.sunbeaminfo.cj06.test;
public class Program {
    public static void main(String[] args) {
        String s1 = "Sun";
        String s2 = s1 + "Beam";
        System.out.println( s1 == s2 ); //false : Because String
instances are immutable
    }
}
```

```
String str = "Sun";
str = str + "Beam";
System.out.println( str );
```



- In java, `String` objects/instances are immutable.
 - In other words, if we try to modify `String`, then JVM create new `String` instance.
-

```
package com.sunbeaminfo.cj06.test;

import java.util.Date;

public class Program {
    public static void main1(String[] args) {
        String s1 = "Sun";
        String s2 = "Beam";
        s1 = s1.concat(s2);
        System.out.println(s1); //output:-SunBeam
    }
    public static void main2(String[] args) {
        String s1 = "Sun";
        int number = 123;
        //s1 = s1.concat( number ); //Error
        s1 = s1.concat( String.valueOf(number) ); //OK Sun123
        System.out.println(s1);
    }
    public static void main3(String[] args) {
        String s1 = "Sun";
        String s2 = "Beam";
        s1 = s1 + s2;
        System.out.println(s1); SunBeam
    }
    public static void main4(String[] args) {
        String s1 = "Sun";
        int number = 123;
        s1 = s1 + number;
        System.out.println(s1); sun123
    }
    public static void main5(String[] args) {
        String s1 = "Sun";
        s1 = s1 + new Date();
        System.out.println(s1); imp: System.out.println(args.length()); //0
    }
    public static void main6(String[] args) {
        String str = "";
        System.out.println(str.length()); //0
    }
    public static void main7(String[] args) {
```

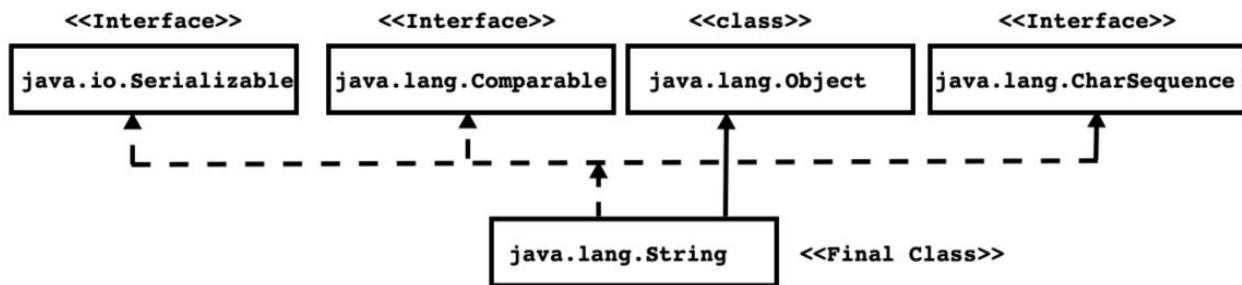
```
String str = "Pune";
//char ch = str.charAt(0); //P
//char ch = str.charAt(3); //e
//char ch = str.charAt(str.length() - 1 ); //e
char ch = str.charAt(str.length());
//StringIndexOutOfBoundsException
System.out.println(ch);
}
public static void main8(String[] args) {
    String s1 = " Sandeep ";
    String s2 = "Kulange";
    s1 = s1.trim(); //to remove spaces
    System.out.println(s1+"<"+s2);
}
public static void main9(String[] args) {
    String s1 = new String("Hello");
    String s2 = new String("Hello");
    System.out.println(s1.equals(s2)); //true
}
public static void main10(String[] args) {
    String s1 = new String("Hello");
    String s2 = new String("Hello");
    System.out.println(s1.hashCode()); //69609650
    System.out.println(s2.hashCode()); //69609650
}
-----
```

A Strategy for Defining Immutable Objects

1. Don't provide "setter" methods — methods that modify fields referred to by fields or objects
2. Make all fields final and private.
3. Don't allow subclasses to override methods. The simplest way to do this is to declare the constructor class as final.
A more sophisticated approach is to make the private and construct instances in factory methods.
4. If the instance fields include references to mutable objects, don't allow those objects to be changed:
 - Don't provide methods that modify the mutable objects.

-
- Don't share references to the mutable objects. Never store references to external, mutable objects passed to the constructor; if necessary, create copies, and store references to the copies. Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your methods.
-

String Class Hierarchy



String Introduction

- Strings, which are widely used in Java programming, are a sequence of characters. to manipulate string:
- In the Java programming language, **strings are objects.**
- We can use following classes
 1. `java.lang.String` : immutable character sequence
 2. `java.lang.StringBuffer` : mutable
 3. `java.lang.StringBuilder` : mutable character sequence
 4. `java.util.StringTokenizer`
 5. `java.util.regex.Pattern`
 6. `java.util.regex.Matcher`

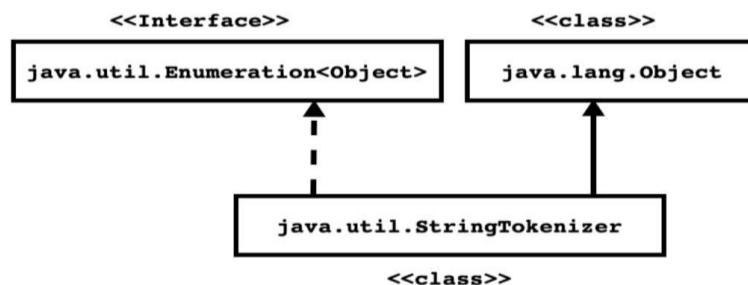
java.lang.Character

- It is a final class declared in `java.lang` package.
- The `Character` class wraps a value of the primitive type `char` in an object.
- This class provides a large number of static methods for character's category (lowercase letter, digit, etc.) and characters from uppercase to lowercase and vice versa.
- determining a for converting
- The fields and methods of class `Character` are defined in terms of character information from the Unicode Standard.
- The `char` data type are based on the original Unicode specification, which defined characters as fixed-width 16-bit entities.

- The range of legal code points is now U+0000 to U+10FFFF, known as Unicode scalar value.
 - The set of characters from U+0000 to U+FFFF is sometimes referred to as the Basic Multilingual Plane (BMP).
 - Characters whose code points are greater than U+FFFF are called supplementary characters.
 - The Java platform uses the UTF-16 representation in char arrays and in the String and StringBuffer classes.
 - A char value, therefore, represents Basic Multilingual Plane (BMP) code point
-

Split String:-

StringTokenizer



StringTokenizer

- The string tokenizer class allows an application to break a string into tokens.
- Methods of `java.util.Enumeration` interface
 - 1. boolean `hasMoreElements()`
 - 2. E `nextElement()`
- Methods of `java.util.StringTokenizer` int `countTokens()` boolean `hasMoreTokens()` String `nextToken()` class
 - 1. public
 - 2. public
 - 3. public
 - 4. public String `nextToken(String delim)`

```
package com.sunbeaminfo.cj06.test;

import java.util.StringTokenizer;

public class Program {
    public static void main(String[] args) {
        String str = "https://docs.oracle.com/javase/8/docs/api/";
        StringTokenizer stk = new StringTokenizer(str, "./:");
        int index = 0;
```

```
String[] arr = new String[ stk.countTokens() ];

while( stk.hasMoreTokens() ) {
    arr[ index ++ ] = stk.nextToken();
}
System.out.println(arr[ 0 ]);
System.out.println(arr[ arr.length - 1 ]);

}

public static void main5(String[] args) {
    String str = "https://docs.oracle.com/javase/8/docs/api/";
    StringTokenizer stk = new StringTokenizer(str, "./:");
    System.out.println(stk.countTokens());
    String token = null;
    while( stk.hasMoreTokens() ) {
        token = stk.nextToken();
        System.out.println(token);
    }
}

public static void main4(String[] args) {
    String str = "www.sunbeaminfo.com";
    StringTokenizer stk = new StringTokenizer(str, ".");
    System.out.println(stk.countTokens());
    String token = null;
    while( stk.hasMoreTokens() ) {
        token = stk.nextToken();
        System.out.println(token);
    }
}

public static void main3(String[] args) {
    String str = "SunBeam Infotech Pune";
    StringTokenizer stk = new StringTokenizer(str);
    String token = null;
    while( stk.hasMoreTokens() ) {
        token = stk.nextToken();
        System.out.println(token);
    }
}

public static void main2(String[] args) {
    String str = "SunBeam Infotech Pune";
    StringTokenizer stk = new StringTokenizer(str);
```

```
String token = null;
while( stk.hasMoreElements() ) {
    token = (String) stk.nextElement();
    System.out.println(token);
}
}

public static void main1(String[] args) {
    String str = "SunBeam Infotech Pune";
    StringTokenizer stk = new StringTokenizer(str);
    System.out.println(stk.countTokens()); //3
}
}
```

Regex Expression:-

Pattern and Matcher

- `java.util.regex.Pattern` and `Matcher` Classes are used for matching character sequences against patterns specified by regular expressions.
- An instance of the `Pattern` class represents a regular expression that is specified in string form in a syntax similar to that used by Perl.
- Instances of the `Matcher` class are used to match character sequences against a given pattern.

```
package com.sunbeaminfo.cj06.test;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

class Validator{
    public static final String NAME_PATTERN = "";
    public static boolean validateName( String name ) {
        return name.matches(NAME_PATTERN);
    }

    public static final String PHONE_PATTERN = "";
    public static boolean validatePhone( String phone ) {
        return phone.matches(PHONE_PATTERN);
    }
}
```

```
public static final String EMAIL_PATTERN = "";
public static boolean validateEmail( String email ) {
    return email.matches(EMAIL_PATTERN);
}
}

public class Program {
    public static void main1(String[] args) {
        String email = "sandeepkulange@gmail.com";
        String regex = "^[_A-Za-z0-9-\\\\+]+(\\.[_A-Za-z0-9-]+)*@[A-Za-
z0-9-]+(\\.[A-Za-z0-9]+)*(\\.\\{2,\\})$";

        Pattern pattern = Pattern.compile(regex,
Pattern.CASE_INSENSITIVE);
        Matcher matcher = pattern.matcher(email);
        if( matcher.matches())
            System.out.println(email);
        else
            System.out.println("Invalid email");
    }
    public static void main2(String[] args) {
        String email = "sandeepkulange@gmail.com";
        String regex = "^[_A-Za-z0-9-\\\\+]+(\\.[_A-Za-z0-9-]+)*@[A-Za-
z0-9-]+(\\.\\{2,\\})$";
        if( Pattern.matches(regex, email))
            System.out.println(email);
        else
            System.out.println("Invalid email");
    }
    public static void main3(String[] args) {
        String email = "sandeepkulange@gmail.com";
        String regex = "^[_A-Za-z0-9-\\\\+]+(\\.[_A-Za-z0-9-]+)*@[A-Za-
z0-9-]+(\\.\\{2,\\})$";
        if( email.matches(regex))
            System.out.println(email);
        else
            System.out.println("Invalid email");
    }
    public static void main4(String[] args) {
        String email = "sandeepkulange@gmail.com";
        if( Validator.validateEmail(email))
```

```
        System.out.println(email);
    else
        System.out.println("Invalid email");
}
-----
```

StringBuffer versus StringBuilder

- **StringBuffer and StringBuilder are final classes.**
- It is declared in **java.lang package.**
- It is used to **create mutable string instance.**
- **equals() and hashCode() method is not overridden inside it.**
- We can create instances of these classes using **new operator only.**
- **Instances get space on Heap.**
- **StringBuffer implementation is thread safe whereas StringBuilder is not.**
- **StringBuffer is introduced in JDK1.0 and StringBuilder is introduced in JDK 1.5.**

```
package com.sunbeaminfo.cj06.test;

import java.util.Scanner;

public class Program {
    public static void main1(String[] args) {
        //StringBuffer => Thread Safe / Synchronized.
        StringBuffer sb1 = new StringBuffer("SunBeam");
        StringBuffer sb2 = new StringBuffer("SunBeam");
        if( sb1 == sb2 )
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output : Not Equal
    }

    public static void main2(String[] args) {
        //equals and hashCode method is not overridden into
        StringBuffer class,
        StringBuffer sb1 = new StringBuffer("SunBeam");
        StringBuffer sb2 = new StringBuffer("SunBeam");
        if( sb1.equals(sb2))
```

```
        System.out.println("Equal");
    else
        System.out.println("Not Equal");
    //Output : Not Equal
}
public static void main3(String[] args) {
    StringBuffer sb1 = new StringBuffer("SunBeam");
    StringBuffer sb2 = new StringBuffer("SunBeam");
    if( sb1.toString().equals(sb2.toString()))
        System.out.println("Equal");
    else
        System.out.println("Not Equal");
    //Output : Equal
}
public static void main4(String[] args) {
    StringBuffer sb = new StringBuffer("SunBeam");
    sb.append("Pune/");
    sb.append("Karad");
    System.out.println(sb.toString());
}

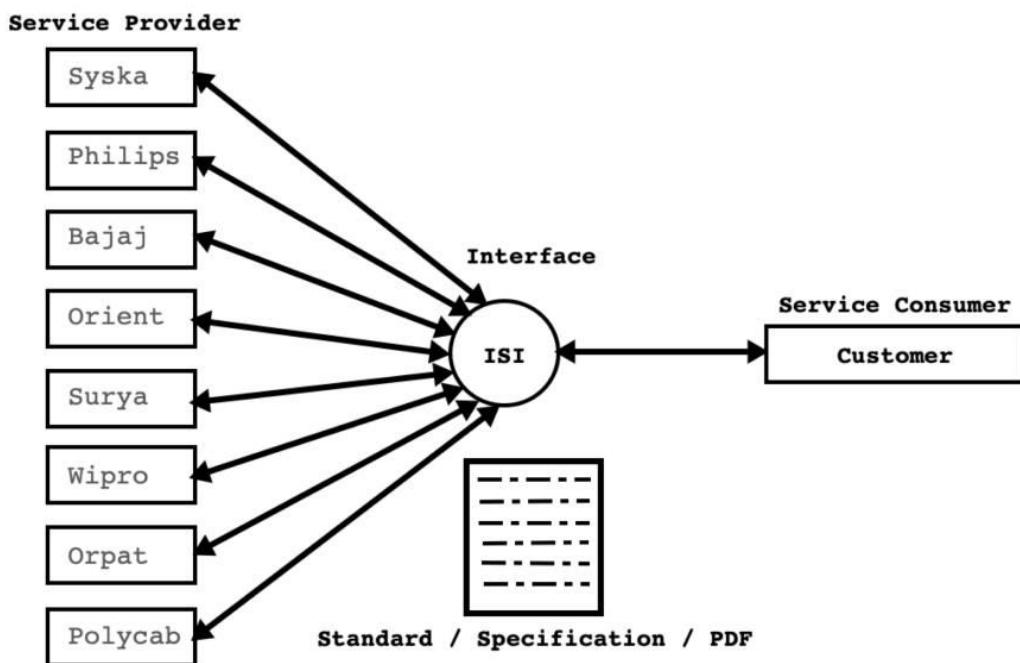
public static void main5(String[] args) {
    try( Scanner sc = new Scanner(System.in)){
        System.out.print("Enter number : ");
        int number = sc.nextInt();
        System.out.println("Number : "+number);

        String strNumber = String.valueOf(number);
        StringBuffer sb = new StringBuffer(strNumber);
        sb.reverse();
        strNumber = sb.toString();
        number = Integer.parseInt(strNumber);
        System.out.println("Number : "+number);
    }
}
public static void main6(String[] args) {
    try( Scanner sc = new Scanner(System.in)){
        System.out.print("Enter number : ");
        System.out.println("Number : "+Integer.parseInt(new
StringBuffer(String.valueOf(sc.nextInt())).reverse().toString()));
    }
}
```

```
        }
    }
}

package com.sunbeaminfo.cj06.test;
import java.text.SimpleDateFormat;
import java.util.Date;
public class Program {
    public static void main(String[] args) {
        Date date = new Date();
        //String pattern = "EEE, d MMM yyyy HH:mm:ss Z";
        String pattern = "dd MMMM,yyyy HH:mm:ss";
        SimpleDateFormat sdf = new SimpleDateFormat( pattern );
        String strDate = sdf.format(date);
        System.out.println(strDate);
    }
    public static void main1(String[] args) {
        Date date = new Date();
        System.out.println(date);
    }
}
```

Interface:-



- Set of rules are called specification/standard.
 - It is a contract between service consumer and service provider.
 - If we want to define specification for the sub classes then we should define interface.
 - Interface is non primitive type which helps developer:
 1. To build/develop trust between service provider and service consumer.
 2. To minimize vendor dependency.
 - interface is a keyword in Java.
-
- Interface can contain:
 1. Nested interface
 2. Field
 3. Abstract method
 4. Default method
 5. Static method
 - Interfaces cannot have constructors.
 - We can create reference of interface but we can not create instance of interface.
 - We can declare fields inside interface. Interface fields are by default public static and final.
 - We can write methods inside interface. Interface methods are by default considered as public and abstract.
-
- If we want to implement rules of interface then we should use implements keyword.
 - It is mandatory to override, all the abstract methods of interface otherwise sub class can be considered as abstract.

```
interface Printable{
    int number = 10;
    void print( );
}
```

```
* Solution 1
abstract class Test implements Printable{
}
```

```
* Solution 2
class Test implements Printable{
    @Override
    public void print( ){
        //TODO
    }
}
```

Interface Syntax

```
Interfaces      : I1, I2, I3
Classes        : C1, C2, C3
1. I2 implements I1          //Not OK
2. I2 extends I1           //OK : Interface Inheritance
3. I3 extends I1, I2       //OK : Multiple Interface Inheritance
4. I1 extends C1           //Not OK
5. I1 implements C1         //Not OK
6. C1 extends I1           //Not OK
7. C1 implements I1         //OK : Interface implementation Inheritance
8. C1 implements I1, I2     //OK : Multiple Interface implementation Inheritance
9. C2 implements C1         //Not OK
10. C2 extends C1          //OK : Implementation inheritance
11. C3 extends C1, C2       //NOT OK : Multiple Implementation inheritance
12. C2 implements I1 extends C1 //NOT OK
13. C2 extends C1 implements I1 //OK
14. C2 extends C1 implements I1, I2, I3 //OK
```



Types of inheritance

• Interface Inheritance

- During inheritance if super type and sub type is interface then it is called interface inheritance.

1. Single Inheritance(Valid in Java)
2. Multiple Inheritance(Valid in Java)
3. Hierarchical Inheritance(Valid in Java)
4. Multilevel Inheritance(Valid in Java)

• Implementation Inheritance

- During inheritance if super type and sub type is class then it is called implementation inheritance.

1. Single Inheritance(Valid in Java)
2. Multiple Inheritance(Invalid in Java)
3. Hierarchical Inheritance(Valid in Java)
4. Multilevel Inheritance(Valid in Java)

Interface Implementation Inheritance

```
interface Printable{
    int number = 10;
    //public static final int number = 10;
    void print( );
    //public abstract void print( );
}

class Test implements Printable{
    @Override
    public void print() {
        System.out.println("Number : "+Printable.number);
    }
}

public class Program {
    public static void main(String[] args) {
        Printable p = new Test(); //Upcasting
        p.print(); //Dynamic Method Dispatch
    }
}
```

-
1. If "is-a" relationship is not exist between super type and sub type and if we want same method design in all the sub types then super type must be interface.
 2. Using interface, we can group instances of unrelated type together.
 3. Interface can extend more than one interfaces.
 4. We can not define constructor inside interface.
 5. By default methods of interface are abstract.
 - Hint : In case of inheritance if state is not involved in super type then it should be interface.

Commonly Used Interfaces

1. java.lang.AutoCloseable
2. java.io.Closeable
3. java.lang.Cloneable
4. java.lang.Comparable
5. java.util.Comparator
6. java.lang.Iterable
7. java.util.Iterator
8. java.io.Serializable

Comparable

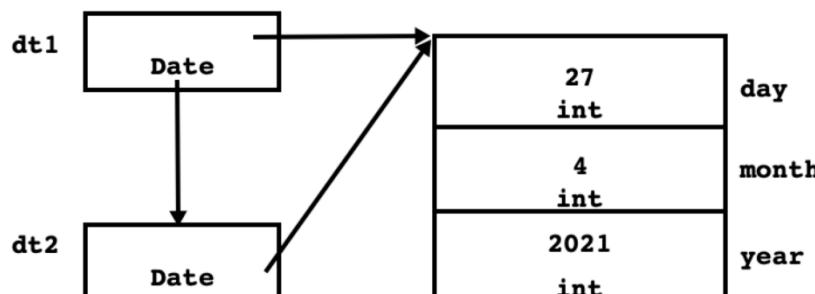
- It is interface declared in `java.lang` package.
- "`int compareTo(T other)`" is a method of `java.lang.Comparable` interface.
- If state of current object is less than state of other object then `compareTo()` method should return negative integer(-1).
- If state of current object is greater than state of other object then `compareTo()` method should return positive integer(+1).
- If state of current object is equal to state of other object then `compareTo()` method should return zero(0).
- If we want to sort, array of non primitive type which contains all the instances of same type then we should implement Comparable interface

Comparator

- It is interface declared in `java.util` package.
- "`int compare(T o1, T o2)`" is a method of `java.util.Comparator` interface.
- If state of current object is less than state of other object then `compare()` method should return negative integer(-1).
- If state of current object is greater than state of other object then `compare()` method should return positive integer(+1).
- If state of current object is equal to state of other object then `compare()` method should return zero(0).
- If we want to sort, array of instances of non primitive of different type then we should implement Comparator interface.

Cloneable Interface Implementation

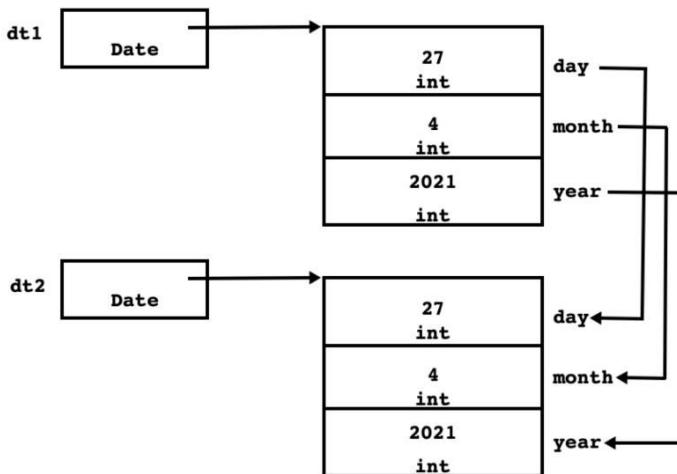
- `Date dt1 = new Date(27, 4, 2021);`
- `Date dt2 = dt1; //Shallow Copy Of References`



Cloneable Interface Implementation

- If we want to create new instance from existing instance then we should use clone method.
- clone() is non final native method of java.lang.Object class.
- Syntax: Ø protected native Object clone() throws CloneNotSupportedException
- Inside clone() method, if we want to create shallow copy instance then we should use super.clone() method.
- Cloneable is interface declared in java.lang package.
- Without implementing Cloneable interface, if we try to create clone of the instance then clone() method throws CloneNotSupportedException.

- Date dt1 = new Date(27, 4, 2021);
- Date dt2 = dt1.clone(); //Shallow Copy Of Instance



Marker Interface

- An interface which do not contain any member is called marker interface. In other words, empty interface is called as marker interface.
- Marker interface is also called as tagging interface.
- If we implement marker interface then Java compiler generates metadata for the JVM, which help JVM to clone/serialize or marshal state of object.
- Example:
 1. java.lang.Cloneable
 2. java.util.EventListener
 3. java.util.RandomAccess
 4. java.io.Serializable
 5. java.rmi.Remote.

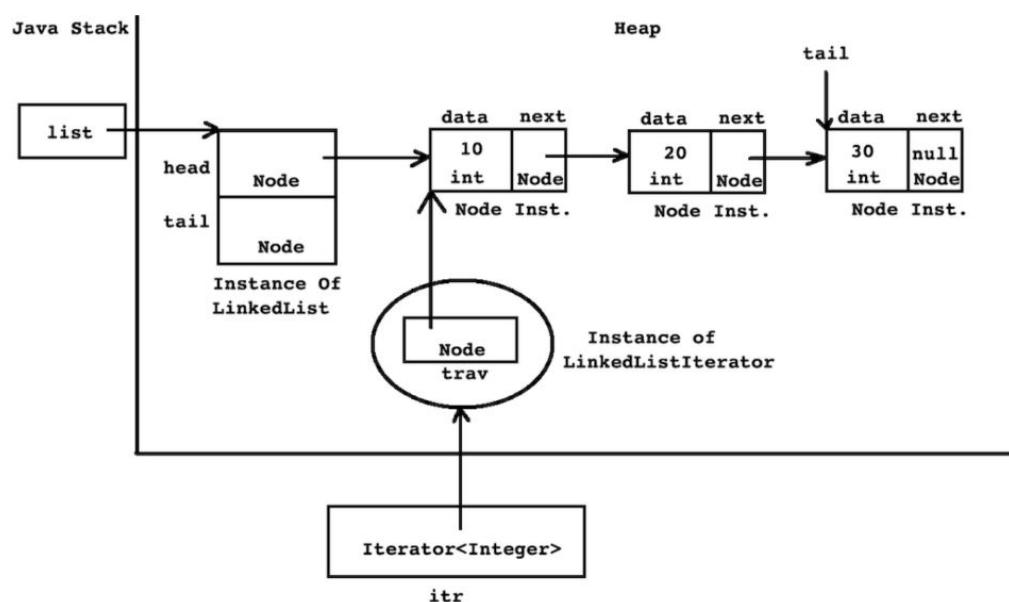
Iterable and Iterator Implementation

- Iterable is interface declared in `java.lang` package.
 - Implementing this interface allows an object to be the target of the "for-each" loop" statement.
 - It is introduced in `JDK 1.5`
 - Methods of `java.lang.Iterable` interface:
 1. `Iterator iterator()`
 2. `default Spliterator spliterator()`
 3. `default void forEach(Consumer action)`
 - Iterator is interface declared in `java.util` package.
 - It is used to traverse collection in forward direction only.
 - It is introduced in `JDK 1.2`
 - Methods of `java.util.Iterator` interface:
 1. `boolean hasNext()`
 2. `E next()`
 3. `default void remove()`
 4. `default void forEachRemaining(Consumer action).`
-

```
LinkedList<Integer> list = new LinkedList<>();
list.add(10);
list.add(20);
list.add(30);

for( Integer e : list )
    System.out.println(e);
```

```
foreach loop implicitly work as follows
Integer element = null;
Iterator<Integer> itr = list.iterator();
while( itr.hasNext() ) {
    element = itr.next();
    System.out.println(element);
}
```



Nested Class

- In Java, we can define class inside scope of another class. It is called nested class.
- Nested class represents encapsulation.

```
//Top-Level class
class Outer{    //Outer.class
    //Nested class
    class Inner{    //Outer$Inner.class
        //TODO
    }
}
```

- Access modifier of top level class can be either package level private or public only.
- We can use any access modifier on nested class.
- Types of nested class:
 1. Non static nested class / Inner class
 2. Static nested class

Non Static Nested Class

- Non static nested class is also called as inner class.
- If implementation of nested class depends on implementation of top level class then nested class should be non static.
- **Implementation Hint :** For the simplicity, consider non static nested class as non static method of class.

<pre>class Outer{ public class Inner{ //TODO } }</pre>	Instantiation of top level class: <code>Outer out = new Outer();</code>
* Instantiation of top level class: - method 1 <code>Outer out = new Outer();</code> <code>Outer.Inner in = out.new Inner();</code>	- method 2 <code>Outer.Inner in = new Outer().new Inner();</code>

Non Static Nested Class

- Top level class can contain static as well as non static members.
- Inside non static nested class we can not declare static members.
- If we want to declare any field static then it must be final.
- Using instance, we can access members of non static nested class inside method of top level class.
- Without instance, we can use all the members of top level class inside method of non static nested class.
- But if we want refer member of top level class, inside method of non static nested class then we should use "TopLevelClassName.this" syntax.

Static Nested Class

- If we declare nested class static then it is simply called as static nested class.
- We can declare nested class static but we can not declare top level class static.
- If implementation of nested class do not depends on implementation of top level class then we should declare nested class static.
- **Implementation hint:** For simplicity consider static nested class as a static method of a class.

```
class Outer{  
    public static class Inner{  
        //TODO  
    }  
}  
  
* Instantiation of top level class:  
Outer out = new Outer();  
  
* Instantiation of static nested class:  
Outer.Inner in = new Outer.Inner();
```

- Static nested class can contain static members.
- Using instance, we can access all the members of static nested class inside method of top level class.
- If we want to use non static members of top level class inside method of static nested class then it is mandatory to create instance of top level class.

Local Class

- In Java, we can define class inside scope of another method. It is called local class / method local class.
- Types of local class:
 1. Method local inner class
 2. Method local anonymous inner class.

Method Local Inner Class

- In Java, we can not declare local variable /class static hence local class is also called as local inner class.
- We can not use reference instance of method local class outside method.

```
public class Program { //Program.class
    public static void main(String[] args) {
        class Complex{ //Program$1Complex.class
            private int real = 10;
            private int imag = 20;
            public void print( ) {
                System.out.println("Real Number : "+this.real);
                System.out.println("Imag Number : "+this.imag);
            }
        }
        Complex c1 = new Complex();
        c1.print();
    }
}
```

Method local anonymous inner class.

- In java, we can create instance without reference. It is called anonymous instance. •

Example:

- new Object();
- We can define a class without name. It is called anonymous class.
- If we want to define anonymous class then we should use new operator.
- We can create anonymous class inside method only hence it is also called as method local anonymous class.
- We can not declare local class static hence it is also called as method local anonymous inner class.
- To define anonymous class, we need to take help of existing interface / abstract class / concrete class.

Method local anonymous inner class.

- Consider anonymous inner class using concrete class.

```
public static void main(String[] args) {
    //Object obj; //obj => reference
    //new Object( ); // new Object( ) => Anonymous instance
    //Object obj = new Object( );//Instance with reference
    Object obj = new Object( ) { //Program$1.class
        private String message = "Hello";
        @Override
        public String toString() {
            return this.message;
        }
    };
    String str = obj.toString();
    System.out.println(str);
}
```

Method local anonymous inner class.

- Consider anonymous inner class using abstract class:

```
abstract class Shape{  
    protected double area;  
    public abstract void calculateArea();  
    public double getArea() {  
        return area;  
    }  
}  
  
public class Program {  
    public static void main(String[] args) {  
        Shape sh = new Shape() {  
            private double radius = 10;  
            @Override  
            public void calculateArea() {  
                this.area = Math.PI * Math.pow(this.radius, 2);  
            }  
        };  
  
        sh.calculateArea();  
        System.out.println("Area : "+sh.getArea());  
    }  
}
```

Method local anonymous inner class.

- Consider anonymous inner class using interface.

```
interface Printable{  
    void print();  
}  
  
public class Program {  
    public static void main(String[] args) {  
        Printable p = new Printable() {  
            @Override  
            public void print() {  
                System.out.println("Hello");  
            }  
        };  
        p.print();  
    }  
}
```

Multithreading: -

-Process

1. Program in execution is called process.
2. Running instance of a program is called process.
3. Process is also called as task.

-Thread

1. Process may contain sub process. It is also called as thread.
2. Lightweight process is called thread.
3. Thread is a separate path of execution which runs independently.

-Single tasking: -

1. An ability of OS to execute single task/process at a time is called single tasking.
e.g MS DOS is single user and single tasking OS.

-Multi tasking: -

1. An ability of OS to execute multiple tasks/process at a time is called multi tasking.
e.g Mac OS, ubuntu 20.04, windows 10 etc.

-We can achieve it using.

1. Process
2. Thread

Main objective: -

1. To utilise hardware resources (CPU) efficiently.

-If we develop application using single thread then it is called single threaded application(STA).

-If we develop application using multiple threads then it is called multi threaded application.

-java is multithreaded programming language.

1. When JVM starts execution of java application it starts execution of main thread and garbage collector.

Because of these two threads every java application is multithreaded.

Main thread :-

1. It is user thread /non daemon thread.
2. It is responsible for invoking main method.
3. Its default priority is Thread.NORM_PRIORITY.

Garbage collector(GC) Finalizer

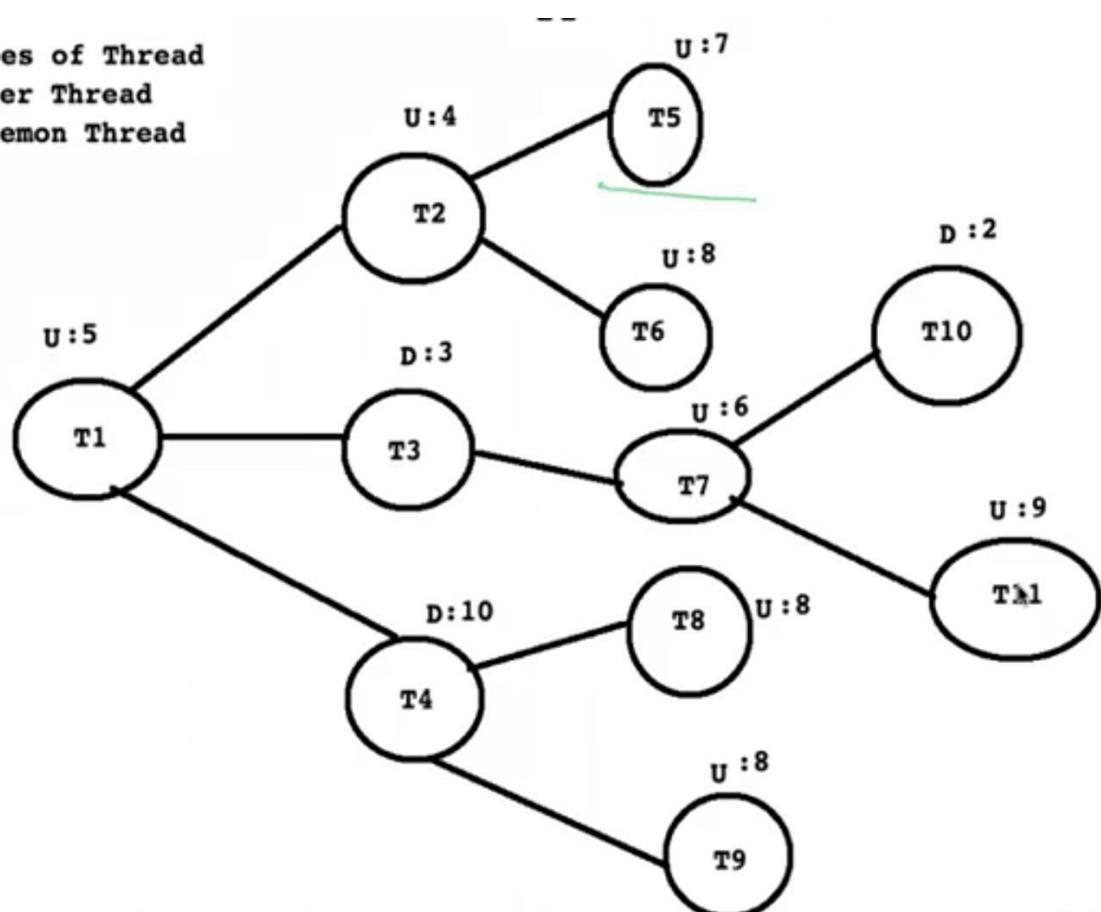
1. it is daemon thread / Background thread.
2. It is responsible for deallocating memory of used object.
3. Its default priority is 8 (Thread.NORM_PRIORITY + 3).

2. Thread is non java resource i.e OS resource.

To access OS thread java developer need not to use JNI framework rather developer can use readymade thread framework supplied by sun/oracle in other words java is multithreaded.

Type Of Thread :-

- 2 Types of Thread
 - 1. User Thread
 - 2. Daemon Thread



U: User thread terminate immediately. Do not wait for child threads.

D: Daemon Threads do not terminate immediately. It waits for child thread.

If we want to used thread in java program then it is necessary to use

-Types declared in `java.lang` package.

-Interface

1. `Runnable`

-Class(es)

1. `Thread`
2. `ThreadGroup`
3. `ThreadLocal`

-Enum

1. `Thread State`

-Exception

1. `IllegalThreadStateException`
2. `IllegalMonitorStateException`
3. `InterruptedException`.

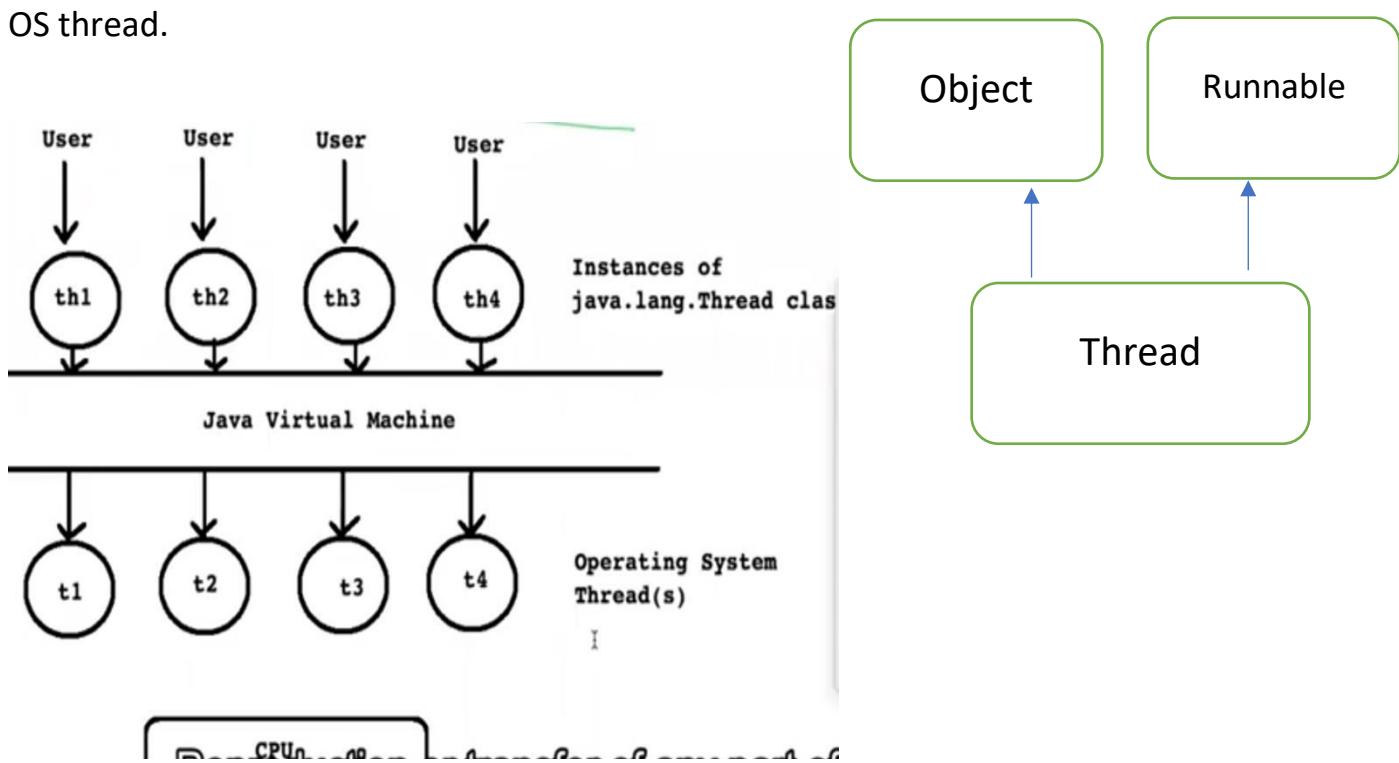
-An interface which is having single abstract method is called functional interface/SAM interface.

-Runnable

1. It is a functional interface declared in `java.lang` package
2. `void run()` is method of runnable interface
3. This method is called B.L method.

Thread is operating system resource.

`Java.lang.Thread` is java class whose instance represent OS thread.



-Nested Type

1.State [Enum]

-Fields

- 1.MIN_PRIORITY(1)
- 2.NORM_PRIORITY(5)
- 3.MAX_PRIORITY(10)

The screenshot shows a Java application running in an IDE. On the left, the code for Program.java is displayed:

```
1 package test;
2
3 public class Program
4 {
5     public static void main(String[] args)
6     {
7         Thread thread = Thread.currentThread();
8
9         String name = thread.getName();
10        System.out.println("Thread Name : "+name);
11
12        int priority = thread.getPriority();
13        System.out.println("Thread Priority : "+priority);
14
15        String threadGroup = thread.getThreadGroup().getName();
16        System.out.println("Thread Group : "+threadGroup);
17
18        String state = thread.getState().toString();
19        System.out.println("Thread State : "+state);
20
21        boolean threadType = thread.isDaemon();
22        System.out.println("Thread Type : "+( threadType ? "Daemon Thread" : "User Thread"));
23    }
24
25    public static void main1(String[] args)
26    {
27        Thread thread = Thread.currentThread();
28        System.out.println(thread.toString()); //Thread[main,5,main]
29    }
}
```

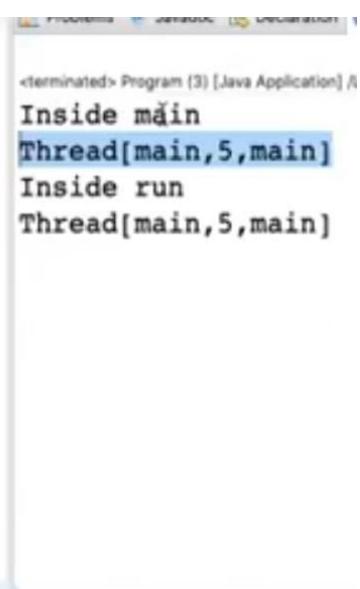
On the right, the output window shows the results of the program's execution:

Thread Name	:	main
Thread Priority	:	5
Thread Group	:	main
Thread State	:	RUNNABLE
Thread Type	:	User Thread

Reproduction or transfer of any part of this video

Core Java Notes By ASHOK PATE

```
1 package test;
2
3 class CThread implements Runnable{
4     @Override
5     public void run() {
6         System.out.println("Inside run");
7         Thread thread = Thread.currentThread();
8         System.out.println(thread.toString());
9     }
10 }
11 public class Program
12 {
13     public static void main(String[] args)
14     {
15         System.out.println("Inside main");
16
17         Thread thread = Thread.currentThread();
18         System.out.println(thread.toString());
19
20         Runnable target = new CThread();
21         target.run();
22     }
23 }
```



```
<terminated> Program (3) [Java Application] /I
Inside main
Thread[main,5,main]
Inside run
Thread[main,5,main]
```

```
1 package test;
2
3 class CThread implements Runnable{
4     @Override
5     public void run() {
6         System.out.println("Inside run");
7         System.out.println("Thread State : "+Thread.currentThread().getState().name());
8     }
9 }
10 public class Program
11 {
12     public static void main(String[] args) throws Exception
13     {
14         Runnable target = new CThread();
15         Thread th1 = new Thread(target); //NEW
16         System.out.println("Thread State : "+th1.getState().name());
17         th1.start(); //RUNNABLE
18         //th1.start(); //IllegalThreadStateException
19
20         while( true ) {
21             System.out.println("Thread State : "+th1.getState().name());
22             Thread.sleep( 100 );
23         }
24     }
25 }
```

```
1 package test;
2
3 class CThread implements Runnable{
4     private Thread thread;
5     public CThread( String name ) {
6         this.thread = new Thread(this, name);
7         this.thread.start();
8     }
9     @Override
10    public void run() {
11        System.out.println("Inside run");
12    }
13 }
14 public class Program
15 {
16     public static void main(String[] args) throws Exception
17     {
18         CThread th1 = new CThread("A");
19         CThread th = new CThread("B");
20         CThread th1 = new CThread("C");
21     }
22 }
```

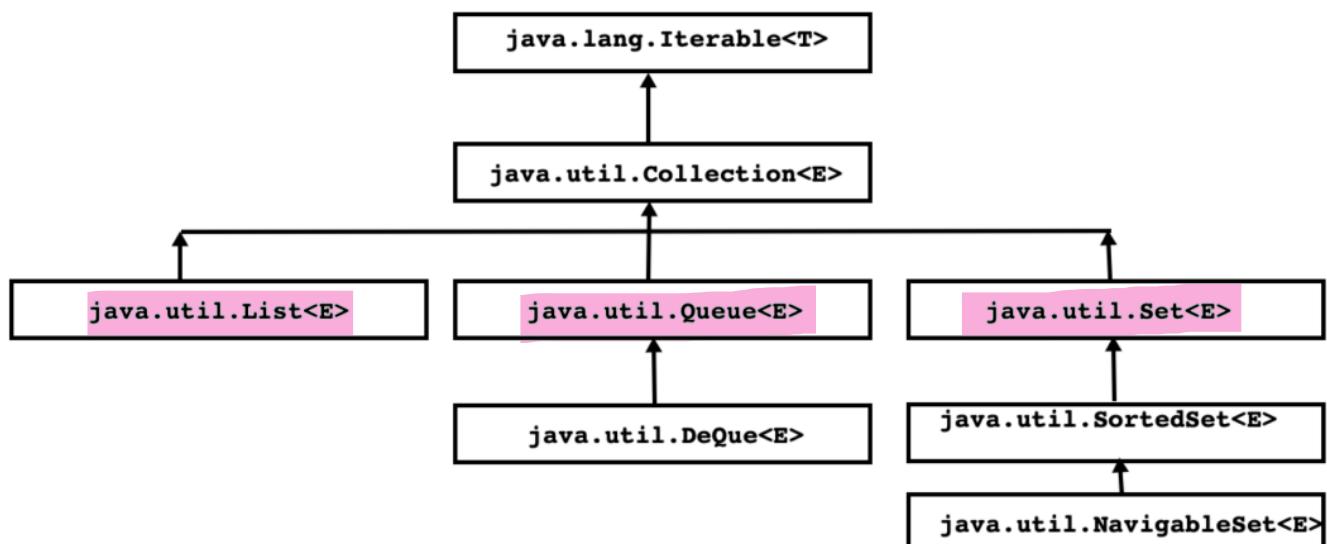
```
Program.java 23
1 package test;
2
3 class CThread extends Thread{
4     public CThread(String name ) {
5         //super( name );
6         this.setName(name);
7         this.start();
8     }
9     @Override
10    public void run() {
11        System.out.println("Inside run");
12    }
13 }
14 public class Program
15 {
16     public static void main(String[] args) throws Exception
17     {
18         CThread th1 = new CThread("A");
19         CThread th2 = new CThread("B");
20         CThread th3 = new CThread("C");
21
22     }
23 }
```

```
Problems Javadoc
<terminated> Program (5)
Inside run
Inside run
Inside run
```

Collection Framework: -

- Every value/data stored in data structure is called element.
- Framework is library of reusable classes/interfaces that is used to develop application.
- Library of reusable data structure classes that is used to develop java application is called collection framework.
- Main purpose of collection framework is to manage data in RAM efficiently.
- Consider following Example: 1. Person has-a birthdate 2. Employee is a person
- In java, collection instance do not contain instances rather it contains reference of instances.
- If we want to use collection framework them we should import `java.util` package.

Collection Interface Hierarchy



Iterable<T>

- It is a interface declared in `java.lang` package.
- All the collection classes implements Iterable interface hence we can traverse it using for each loop
- Methods of Iterable interface:
 1. `Iterator<T> iterator()`
 2. `default Spliterator<T> spliterator()`
 3. `default void forEach(Consumer<? Super T> action)`

Collection<E>

- Collection<E> is interface declared in java.util package.
- It is sub interface of Iterable interface.
- It is root interface in collection framework interface hierarchy.
- Default methods of Collection interface
 - 1. default Stream<E> stream()
 - 2. default Stream<E> parallelStream()
 - 3. default boolean removeIf(Predicate<? super E> filter)

• Abstract Methods of Collection Interface

1. boolean add(E e)
2. boolean addAll(Collection<? Extends E> c)
3. void clear()
4. boolean contains(Object o)
5. boolean containsAll(Collection <?> c)
6. boolean isEmpty()
7. boolean remove(Object o)
8. boolean removeAll(Collection<?> c)
9. boolean retainAll(Collection<?> c)
10. int size()
11. Object[] toArray()
12. <T> T[] toArray(T[] a)

List<E>

- It is sub interface of java.util.Collection interface.
- It is ordered/sequential collection.
- ArrayList, Vector, Stack, LinkedList etc. implements List interface. It generally referred as "List collections".
- List collection can contain duplicate element as well multiple null elements.
- Using integer index, we can access elements from List collection.
- We can traverse elements of List collection using Iterator as well as ListIterator.
- It is introduced in jdk 1.2.

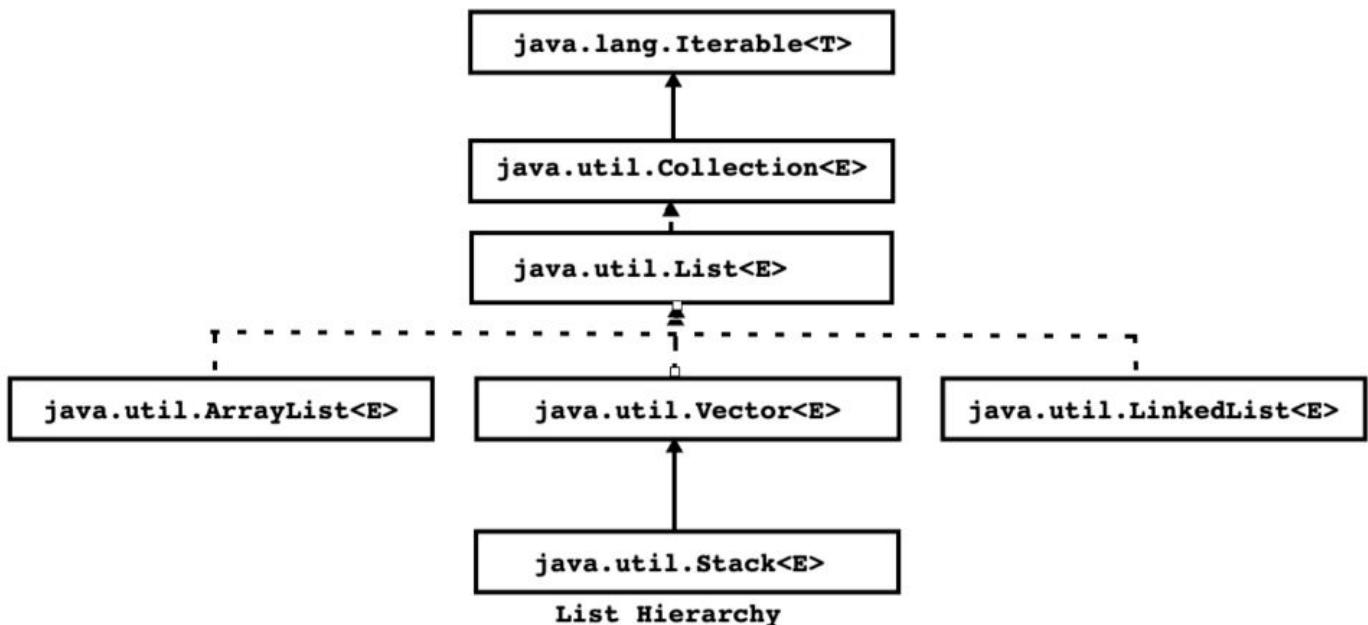
- Note: If we want to manage elements of non final type inside List collection then non final type should override "equals" method.

List<E>

- Abstract methods of List Interface

1. void add(int index, E element)
2. boolean addAll(int index, Collection<? Extends E> c)
3. E get(int index)
4. int indexOf(Object o)
5. int lastIndexOf(Object o)
6. ListIterator<E> listIterator()
7. ListIterator<E> listIterator(int index)
8. E remove(int index)
9. E set(int index, E element)
10. List<E> subList(int fromIndex, int toIndex)

List Interface Hierarchy:-

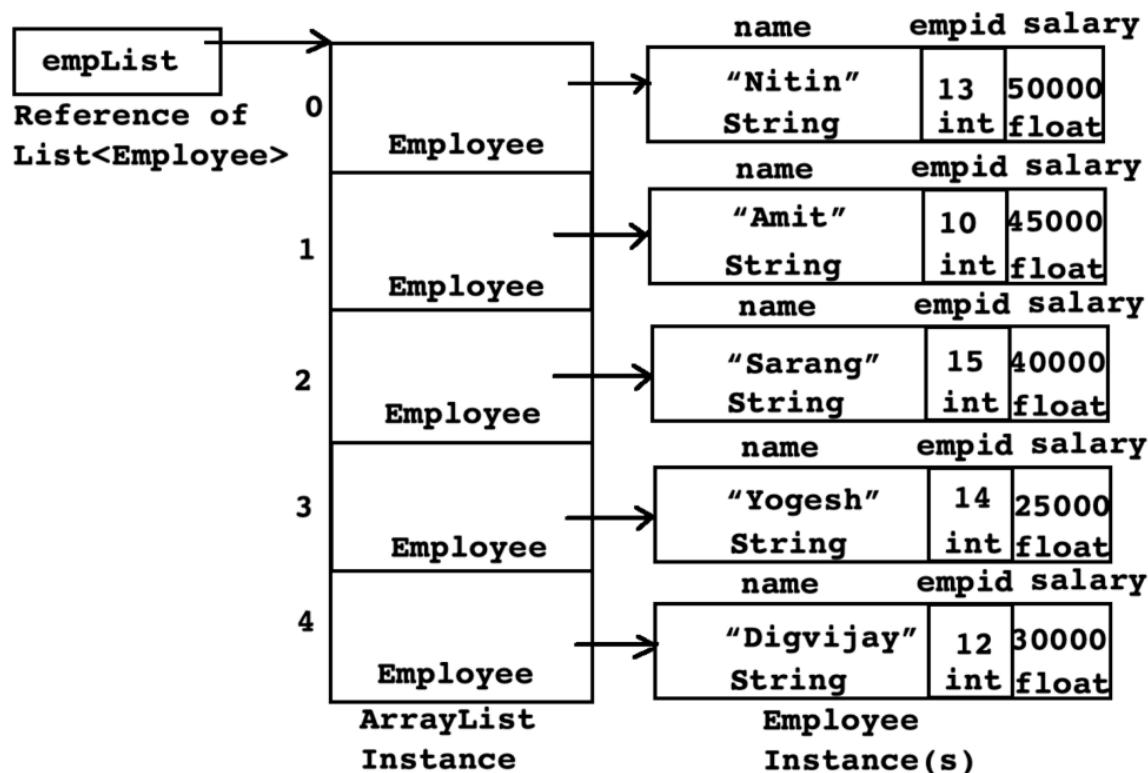


ArrayList <E>

- It is **resizable array**.
- It implements **List**, **RandomAccess**, **Cloneable**, **Serializable** interfaces.
- It is **List collection**.
- It is **unsynchronized** collection. Using "**Collections.synchronizedList**" method, we can make it synchronized.

➤ `List list = Collections.synchronizedList(new ArrayList(...));`

- Initial capacity of ArrayList is 10. If ArrayList is full then its capacity gets increased by half of its existing capacity.
- It is introduced in **jdk 1.2**
- Note: If we want to manage elements of non final type inside ArrayList then non final type should override "equals" method.



```
package com.sunbeaminfo.cj06.collection.test;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

public class Program {
    public static void main1(String[] args) {
        //ArrayList<Integer> list1 = new ArrayList<Integer>(); //OK
        ArrayList<Integer> list1 = new ArrayList<>(); //OK

        ArrayList<Integer> list2 = new ArrayList<>( 15 ); //OK

        ArrayList<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(20);
        ArrayList<Integer> list3 = new ArrayList<>( list ); //OK
        System.out.println( list3.size());//2

    public static void main2(String[] args) {
        List<Integer> list1 = new ArrayList<>();

        List<Integer> list2 = new ArrayList<>( 15 );

        List<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(20);
        List<Integer> list3 = new ArrayList<>( list );
    }

    public static void main3(String[] args) {
        Collection<Integer> list1 = new ArrayList<>();

        Collection<Integer> list2 = new ArrayList<>( 15 );

        Collection<Integer> list = new ArrayList<>();
        list.add(10);
        list.add(20);
        Collection<Integer> list3 = new ArrayList<>( list );
    }
}
```

```
package com.sunbeaminfo.cj06.collection.test;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Program {
    public static void main1(String[] args) {
        // java.util.List<E> is interface
        // java.util.ArrayList<E> is interface implementation class

        List<Integer> list = new ArrayList<>(); // Upcasting
        list.add(10);
        list.add(20);
        list.add(30);
        list.add(40);
        list.add(50);
        //also written like this.
        //List<Integer> list = Arrays.asList(10,20,30,40,50);
    }

    public static void main2(String[] args) {
        List<Integer> list = Arrays.asList(10, 20, 30, 40, 50 );
        for( int index = 0; index < list.size(); ++ index ) {
            Integer element = list.get(index);
            System.out.println(element);
        }
    }

    public static void main3(String[] args) {
        List<Integer> list = Arrays.asList(10, 20, 30, 40, 50 );
        System.out.println(list.getClass().getName());
        //java.util.Arrays$ArrayList
        int index = list.size();
        Integer element = list.get(index);
        //ArrayIndexOutOfBoundsException
        System.out.println(element);
    }

    public static void main4(String[] args) {
        List<Integer> list = new ArrayList<>(); // Upcasting
        list.add(10);
        list.add(20);
```

```
list.add(30);
list.add(40);
list.add(50);
int index = list.size();
Integer element = list.get(index); //IndexOutOfBoundsException
System.out.println(element);
}

}

-----
```

```
package com.sunbeaminfo.cj06.collection.test;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;
import java.util.ListIterator;

public class Program {

    public static void main1(String[] args) {
        List<Integer> list = Arrays.asList(10, 20, 30, 40, 50 );
        for( int index = 0; index < list.size(); ++ index ) {
            Integer element = list.get(index);
            System.out.println(element);
        }
    }

    public static void main2(String[] args) {
        List<Integer> list = Arrays.asList(10, 20, 30, 40, 50 );

        Iterator<Integer> itr = list.iterator();
        while( itr.hasNext() ) {
            Integer element = itr.next();
            System.out.println(element);
        }
    }

    public static void main3(String[] args) {
        List<Integer> list = Arrays.asList(10, 20, 30, 40, 50 );
        for( Integer element : list )
            System.out.println(element);
    }
}
```

```
public static void main4(String[] args) {
    List<Integer> list = Arrays.asList(10, 20, 30, 40, 50 );
    list.forEach( e -> System.out.println(e) ); //Using Lambda
}
Expression
public static void main5(String[] args) {
    List<Integer> list = Arrays.asList(10, 20, 30, 40, 50 );
    list.forEach( System.out::println ); //Using method reference
}

public static void main6(String[] args) {
    List<Integer> list = Arrays.asList(10, 20, 30, 40, 50 );
    ListIterator<Integer> itr = list.listIterator();
    Integer element = null;
    //forward direction print
    while( itr.hasNext() ) {
        element = itr.next();
        System.out.print(element+"  ");
    }
    System.out.println();
    //backward direction print
    while( itr.hasPrevious() ) {
        element = itr.previous();
        System.out.print(element+"  ");
    }
}
public static void main7(String[] args) {
    Arrays.asList(10, 20, 30, 40, 50 )
        .stream()
        .forEach(System.out::println);
}
}

package com.sunbeaminfo.cj06.collection.test;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.List;
```

```
public class Program {
    public static List<Integer> getList( ){
        List<Integer> list = new ArrayList<>();
        list.add( 10 );
        list.add( 20 );
        list.add( 50 );
        return list;
    }
    public static void main1(String[] args) {
        List<Integer> list = Program.getList();
        list.add(60);
        list.add(70);
        list.forEach(System.out::println);
    }
    public static void main2(String[] args) {
        List<Integer> list = Program.getList();
        Collection<Integer> c = new ArrayList<Integer>();
        c.add(60);
        c.add(70);
        list.addAll( c );
        list.forEach(System.out::println);
    }
    public static void main3(String[] args) {
        List<Integer> list = Program.getList();
        list.addAll( Arrays.asList(60,70) );
        list.forEach(System.out::println);
    }
    public static void main4(String[] args) {
        List<Integer> list = Program.getList();
        list.add(2, 30);
        list.add(3, 40);
        list.forEach(System.out::println);
    }
    public static void main5(String[] args) {
        List<Integer> list = Program.getList();
        list.addAll(2, Arrays.asList(30,40));
        list.forEach(System.out::println);
    }
}
```

```
package com.sunbeaminfo.cj06.collection.test;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Program {
    public static List<Integer> getList( ){
        List<Integer> list = new ArrayList<>();
        list.add( 10 );
        list.add( 20 );
        list.add( 30 );
        list.add( 40 );
        list.add( 50 );
        list.add( 60 );
        list.add( 70 );
        return list;
    }

    public static void main1(String[] args) {
        List<Integer> list = Program.getList();
        Integer key = new Integer(400);
        if( list.contains(key)) {
            int index = list.indexOf(key);
            System.out.println(key+ " : "+index);
        }else
            System.out.println(key+ " not found ");
    }

    public static void main2(String[] args) {
        List<Integer> list = Program.getList();
        Integer key = new Integer(40);
        if( list.contains(key)) {
            int index = list.indexOf(key);
            Integer element = list.remove(index);
            System.out.println(element+ " is removed");
        }else
            System.out.println(key+ " not found ");
    }

    public static void main3(String[] args) {
        List<Integer> list = Program.getList();
```

```
List<Integer> keys = Arrays.asList(30,50,70);
if( list.containsAll(keys) ) {
    for( int index = 0; index < keys.size(); ++ index ) {
        System.out.println(keys.get(index)+" : "+list.indexOf(keys.get(index)));
    }
}else
    System.out.println(keys+" not found ");
}

public static void main4(String[] args) {
List<Integer> list = Program.getList();
List<Integer> keys = Arrays.asList(30,50,70);
if( list.containsAll(keys) ) {
    boolean removedStatus = list.removeAll(keys);
    System.out.println(removedStatus);
}else
    System.out.println(keys+" not found ");
}

public static void main5(String[] args) {
List<Integer> list = Program.getList();
List<Integer> keys = Arrays.asList(30,50,70);
if( list.containsAll(keys) ) {
    list.retainAll(keys);
//list.removeAll(keys);
    System.out.println(keys);
    System.out.println(list);
}else
    System.out.println(keys+" not found ");
}
}
```

```
package com.sunbeaminfo.cj06.collection.test;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class Program {
    public static List<Integer> getList( ){
        List<Integer> list = new ArrayList<>();

```

```
        list.add( 50 );
        list.add( 10 );
        list.add( 40 );
        list.add( 20 );
        list.add( 30 );
        return list;
    }
    public static void main(String[] args) {
        List<Integer> list = Program.getList();
        Collections.sort(list);
        //list.sort(null);
        //list.stream().sorted().forEach(System.out::println);
    }
}
```

```
package com.sunbeaminfo.cj06.collection.model;

import lombok.AllArgsConstructor;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@NoArgsConstructor
@AllArgsConstructor
@Getter @Setter
@EqualsAndHashCode
//@ToString
public class Employee {
    @EqualsAndHashCode.Exclude private String name;
    @EqualsAndHashCode.Include private int empid;
    @EqualsAndHashCode.Exclude private float salary;

    /*@Override
    public boolean equals(Object obj) {
        if( obj != null ) {
            Employee other = (Employee) obj;
            if( this.empid == other.empid)
                return true;
        }
    }
}
```

```
        return true;
    }
    return false;
}*/



@Override
public String toString() {
    return String.format("%-15s%-5d%-10.2f", this.name,
this.empid, this.salary);
}

package com.sunbeaminfo.cj06.collection.test;

import java.util.ArrayList;

import com.sunbeaminfo.cj06.collection.model.Employee;

public class ListTest {
    private ArrayList<Employee> empList = new ArrayList<Employee>();
    public void addRecord( Employee[] employees ) {
        if( employees != null ) {
            for( Employee emp : employees )
                this.empList.add(emp);
        }
    }
    /* public Employee findRecord(int empid) {
        for( Employee emp : this.empList ) {
            if( emp.getEmpid() == empid )
                return emp;
        }
        return null;
    } */

    public Employee findRecord(int empid) {
        Employee key = new Employee();
        key.setEmpid(empid);
        if( this.empList.contains(key)) {
            int index = this.empList.indexOf(key);

```

```
        Employee emp = this.empList.get(index);
        return emp;
    }
    return null;
}
/* public boolean removeRecord(int empid) {
    Employee key = new Employee();
    key.setEmpid(empid);
    if( this.empList.contains(key)) {
        int index = this.empList.indexOf(key);
        this.empList.remove(index); //method of List I/F
        return true;
    }
    return false;
}*/
```

```
public boolean removeRecord(int empid) {
    Employee key = new Employee();
    key.setEmpid(empid);
    if( this.empList.contains(key)) {
        this.empList.remove(key);    //method of Collection I/F
        return true;
    }
    return false;
}
public void printRecords( ) {
    for( Employee emp : this.empList )
        System.out.println(emp.toString());
}
```

```
}
```

```
package com.sunbeaminfo.cj06.collection.test;
```

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
```

```
page. 276
```

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

import javax.lang.model.element.Element;

import com.sunbeaminfo.cj06.collection.model.Employee;

public class Program {
    private static Scanner sc = new Scanner(System.in);
    public static void acceptRecord( int[] empid ) {
        System.out.print("Enter empid : ");
        empid[ 0 ] = sc.nextInt();
    }
    private static void printRecord(Employee emp) {
        if( emp != null )
            System.out.println(emp.toString());
        else
            System.out.println("Employee not found");
    }
    private static void printRecord(boolean removedStatus) {
        if( removedStatus )
            System.out.println("Employee is removed");
        else
            System.out.println("Employee not found");
    }
    public static Employee[] getEmployees( ) {
        Employee[] arr = new Employee[ 5 ];
        arr[ 0 ] = new Employee( "Nitin", 13, 50000 );
        arr[ 1 ] = new Employee( "Amit", 10, 45000 );
        arr[ 2 ] = new Employee( "Sarang", 15, 40000 );
        arr[ 3 ] = new Employee( "Yogesh", 14, 25000 );
        arr[ 4 ] = new Employee( "Digvijay", 12, 30000 );
        return arr;
    }
    public static int menuList( ) {
        System.out.println("0.Exit");
        System.out.println("1.Add Record");
        System.out.println("2.Find Record");
        System.out.println("3.Remove Record");
        System.out.println("4.Print Record");
    }
}
```

```
System.out.print("Enter choice : ");
return sc.nextInt();
}
public static void main(String[] args) {
    int choice;
    int[] empid = new int[ 1 ];
    ListTest test = new ListTest();
    while( ( choice = Program.menuList( ) ) != 0 ) {
        switch( choice ) {
            case 1:
                Employee[] employees = Program.getEmployees();
                test.addRecord(employees);
                break;
            case 2:
                Program.acceptRecord(empid);
                Employee emp = test.findRecord( empid[ 0 ] );
                Program.printRecord( emp );
                break;
            case 3:
                Program.acceptRecord(empid);
                boolean removedStatus = test.removeRecord( empid[ 0 ] );
                Program.printRecord(removedStatus);
                break;
            case 4:
                test.printRecords();
                break;
        }
    }
}
```

Compare by Name , Salary , Empid Program.

```
package com.sunbeaminfo.cj06.collection.model;

import lombok.AllArgsConstructor;
import lombok.EqualsAndHashCode;
import lombok.Getter;
```

```
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@NoArgsConstructor
@AllArgsConstructor
@Getter @Setter
@EqualsAndHashCode
//@ToString
public class Employee implements Comparable<Employee> {
    @EqualsAndHashCode.Exclude private String name;
    @EqualsAndHashCode.Include private int empid;
    @EqualsAndHashCode.Exclude private float salary;

    /*@Override
    public boolean equals(Object obj) {
        if( obj != null ) {
            Employee other = (Employee) obj;
            if( this.empid == other.empid)
                return true;
        }
        return false;
    }*/
    @Override
    public String toString() {
        return String.format("%-15s%-5d%-10.2f", this.name,
this.empid, this.salary);
    }

    @Override
    public int compareTo(Employee other) {
        return this.empid - other.empid;
    }
}

package com.sunbeaminfo.cj06.collection.test;

import java.util.ArrayList;
```

```
import java.util.Collections;
import java.util.Comparator;

import com.sunbeaminfo.cj06.collection.model.Employee;

public class ListTest {
    private ArrayList<Employee> empList = new ArrayList<Employee>();
    public void addRecord( Employee[] employees ) {
        if( employees != null ) {
            for( Employee emp : employees )
                this.empList.add(emp);
        }
    }
    /* public Employee findRecord(int empid) {
        for( Employee emp : this.empList ) {
            if( emp.getEmpid() == empid )
                return emp;
        }
        return null;
    }*/
    public Employee findRecord(int empid) {
        Employee key = new Employee();
        key.setEmpid(empid);
        if( this.empList.contains(key) ) {
            int index = this.empList.indexOf(key);
            Employee emp = this.empList.get(index);
            return emp;
        }
        return null;
    }
    /* public boolean removeRecord(int empid) {
        Employee key = new Employee();
        key.setEmpid(empid);
        if( this.empList.contains(key) ) {
            int index = this.empList.indexOf(key);
            this.empList.remove(index); //method of List I/F
            return true;
        }
        return false;
    }*/
}
```

```
 }*/  
  
 public boolean removeRecord(int empid) {  
 Employee key = new Employee();  
 key.setEmpid(empid);  
 if( this.empList.contains(key)) {  
     this.empList.remove(key);    //method of Collection I/F  
     return true;  
 }  
 return false;  
}  
 public void printRecords( Comparator<Employee> comparator ) {  
 //Collections.sort(this.empList, comparator);  
 this.empList.sort(comparator);  
 for( Employee emp : this.empList )  
     System.out.println(emp.toString());  
}  
}  
  
package com.sunbeaminfo.cj06.collection.util;  
import java.util.Comparator;  
import com.sunbeaminfo.cj06.collection.model.Employee;  
  
public class CompareByEmpid implements Comparator<Employee> {  
 @Override  
 public int compare(Employee e1, Employee e2) {  
     return e1.getEmpid() - e2.getEmpid();  
 }  
}  
  
package com.sunbeaminfo.cj06.collection.util;  
import java.util.Comparator;  
import com.sunbeaminfo.cj06.collection.model.Employee;  
  
public class CompareByName implements Comparator<Employee> {  
 @Override  
 public int compare(Employee e1, Employee e2) {  
     return e1.getName().compareTo(e2.getName());  
 }  
}
```

```
package com.sunbeaminfo.cj06.collection.util;
import java.util.Comparator;
import com.sunbeaminfo.cj06.collection.model.Employee;

public class CompareBySalary implements Comparator<Employee> {
    @Override
    public int compare(Employee e1, Employee e2) {
        return (int) (e1.getSalary() - e2.getSalary());
    }
}

package com.sunbeaminfo.cj06.collection.test;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.Scanner;

import javax.lang.model.element.Element;

import com.sunbeaminfo.cj06.collection.model.Employee;
import com.sunbeaminfo.cj06.collection.util.CompareByEmpid;
import com.sunbeaminfo.cj06.collection.util.CompareByName;
import com.sunbeaminfo.cj06.collection.util.CompareBySalary;

public class Program {
    private static Scanner sc = new Scanner(System.in);
    public static void acceptRecord( int[] empid ) {
        System.out.print("Enter empid : ");
        empid[ 0 ] = sc.nextInt();
    }
    private static void printRecord(Employee emp) {
        if( emp != null )
            System.out.println(emp.toString());
        else

```

```
        System.out.println("Employee not found");
    }
private static void printRecord(boolean removedStatus) {
    if( removedStatus )
        System.out.println("Employee is removed");
    else
        System.out.println("Employee not found");
}
public static Employee[] getEmployees( ) {
    Employee[] arr = new Employee[ 5 ];
    arr[ 0 ] = new Employee( "Nitin", 13, 50000 );
    arr[ 1 ] = new Employee( "Amit", 10, 45000 );
    arr[ 2 ] = new Employee( "Sarang", 15, 40000 );
    arr[ 3 ] = new Employee( "Yogesh", 14, 25000 );
    arr[ 4 ] = new Employee( "Digvijay", 12, 30000 );
    return arr;
}
public static int menuList( ) {
    System.out.println("0.Exit");
    System.out.println("1.Add Record");
    System.out.println("2.Find Record");
    System.out.println("3.Remove Record");
    System.out.println("4.Print Record");
    System.out.print("Enter choice : ");
    return sc.nextInt();
}
public static int subMenuList( ) {
    System.out.println("0.Exit");
    System.out.println("1.Sort By Name");
    System.out.println("2.Sort By Empid");
    System.out.println("3.Sort By Salary");
    System.out.print("Enter choice : ");
    return sc.nextInt();
}
public static void main(String[] args) {
    int choice;
    int[] empid = new int[ 1 ];
    ListTest test = new ListTest();
    while( ( choice = Program.menuList( ) ) != 0 ) {
        switch( choice ) {
```

```
case 1:  
    Employee[] employees = Program.getEmployees();  
    test.addRecord(employees);  
    break;  
case 2:  
    Program.acceptRecord(empid);  
    Employee emp = test.findRecord( empid[ 0 ] );  
    Program.printRecord( emp );  
    break;  
case 3:  
    Program.acceptRecord(empid);  
    boolean removedStatus = test.removeRecord( empid[ 0 ] )  
);  
    Program.printRecord(removedStatus);  
    break;  
case 4:  
    while( ( choice = Program.subMenuList( ) ) != 0 ) {  
        Comparator<Employee> comparator = null;  
        switch( choice ) {  
            case 1:  
                comparator = new CompareByName();  
                break;  
            case 2:  
                comparator = new CompareByEmpid();  
                break;  
            case 3:  
                comparator = new CompareBySalary();  
                break;  
        }  
        test.printRecords( comparator );  
    }  
    break;  
}  
}  
}
```

Generic Program of above code.

```
package com.sunbeaminfo.cj06.collection.model;

import lombok.AllArgsConstructor;
import lombok.EqualsAndHashCode;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

@NoArgsConstructor
@AllArgsConstructor
@Getter @Setter
@EqualsAndHashCode
//@ToString
public class Employee implements Comparable<Employee> {
    @EqualsAndHashCode.Exclude private String name;
    @EqualsAndHashCode.Include private int empid;
    @EqualsAndHashCode.Exclude private float salary;

    /*@Override
    public boolean equals(Object obj) {
        if( obj != null ) {
            Employee other = (Employee) obj;
            if( this.empid == other.empid)
                return true;
        }
        return false;
    }*/

    @Override
    public String toString() {
        return String.format("%-15s%-5d%-10.2f", this.name,
this.empid, this.salary);
    }

    @Override
    public int compareTo(Employee other) {
```

```
        return this.empid - other.empid;
    }
}

package com.sunbeaminfo.cj06.collection.test;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedList;
import java.util.List;
import java.util.Vector;

import com.sunbeaminfo.cj06.collection.model.Employee;

public class ListTest {
//private ArrayList<Employee> empList = new ArrayList<>();
    private List<Employee> empList;
    public void setEmpList(List<Employee> empList) {
        this.empList = empList;
    }
    public void addRecord( Employee[] employees ) {
        if( this.empList != null ) {
            if( employees != null ) {
                for( Employee emp : employees )
                    this.empList.add(emp);
            }
        }
    }
    public Employee findRecord(int empid) {
        if( this.empList != null ) {
            Employee key = new Employee();
            key.setEmpid(empid);
            if( this.empList.contains(key)) {
                int index = this.empList.indexOf(key);
                Employee emp = this.empList.get(index);
                return emp;
            }
        }
        return null;
    }
}
```

```
public boolean removeRecord(int empid) {
    if( this.empList != null ) {
        Employee key = new Employee();
        key.setEmpid(empid);
        if( this.empList.contains(key)) {
            this.empList.remove(key); //method of Collection I/F
            return true;
        }
    }
    return false;
}

public void printRecords( Comparator<Employee> comparator ) {
    if( this.empList != null ) {
        this.empList.sort(comparator);
        for( Employee emp : this.empList )
            System.out.println(emp.toString());
    }
}
}

package com.sunbeaminfo.cj06.collection.util;
import java.util.Comparator;
import com.sunbeaminfo.cj06.collection.model.Employee;

public class CompareByEmpid implements Comparator<Employee> {
    @Override
    public int compare(Employee e1, Employee e2) {
        return e1.getEmpid() - e2.getEmpid();
    }
}

package com.sunbeaminfo.cj06.collection.util;
import java.util.Comparator;
import com.sunbeaminfo.cj06.collection.model.Employee;
public class CompareByName implements Comparator<Employee> {
    @Override
    public int compare(Employee e1, Employee e2) {
        return e1.getName().compareTo(e2.getName());
    }
}
```

```
package com.sunbeaminfo.cj06.collection.util;
import java.util.Comparator;
import com.sunbeaminfo.cj06.collection.model.Employee;

public class CompareBySalary implements Comparator<Employee> {
    @Override
    public int compare(Employee e1, Employee e2) {
        return (int) (e1.getSalary() - e2.getSalary());
    }
}

package com.sunbeaminfo.cj06.collection.test;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedList;
import java.util.List;
import java.util.Scanner;
import java.util.Vector;

import javax.lang.model.element.Element;

import com.sunbeaminfo.cj06.collection.model.Employee;
import com.sunbeaminfo.cj06.collection.util.CompareByEmpid;
import com.sunbeaminfo.cj06.collection.util.CompareByName;
import com.sunbeaminfo.cj06.collection.util.CompareBySalary;

public class Program {
    private static Scanner sc = new Scanner(System.in);
    public static void acceptRecord( int[] empid ) {
        System.out.print("Enter empid : ");
        empid[ 0 ] = sc.nextInt();
    }
    private static void printRecord(Employee emp) {
        if( emp != null )
            System.out.println(emp.toString());
        else

```

```
        System.out.println("Employee not found");
    }
private static void printRecord(boolean removedStatus) {
    if( removedStatus )
        System.out.println("Employee is removed");
    else
        System.out.println("Employee not found");
}
public static Employee[] getEmployees( ) {
    Employee[] arr = new Employee[ 5 ];
    arr[ 0 ] = new Employee( "Nitin", 13, 50000 );
    arr[ 1 ] = new Employee( "Amit", 10, 45000 );
    arr[ 2 ] = new Employee( "Sarang", 15, 40000 );
    arr[ 3 ] = new Employee( "Yogesh", 14, 25000 );
    arr[ 4 ] = new Employee( "Digvijay", 12, 30000 );
    return arr;
}
public static int menuList( ) {
    System.out.println("0.Exit");
    System.out.println("1.Add Record");
    System.out.println("2.Find Record");
    System.out.println("3.Remove Record");
    System.out.println("4.Print Record");
    System.out.print("Enter choice : ");
    return sc.nextInt();
}
public static int subMenuList( ) {
    System.out.println("0.Exit");
    System.out.println("1.Sort By Name");
    System.out.println("2.Sort By Empid");
    System.out.println("3.Sort By Salary");
    System.out.print("Enter choice : ");
    return sc.nextInt();
}
public static void main(String[] args) {
    int choice;
    int[] empid = new int[ 1 ];
    ListTest test = new ListTest();
    //test.setEmpList(new ArrayList<Employee>());
    //test.setEmpList(new Vector<Employee>());
```

```
test.setEmpList(new LinkedList<Employee>());
while( ( choice = Program.menuList( ) ) != 0 ) {
    switch( choice ) {
        case 1:
            Employee[] employees = Program.getEmployees();
            test.addRecord(employees);
            break;
        case 2:
            Program.acceptRecord(empid);
            Employee emp = test.findRecord( empid[ 0 ] );
            Program.printRecord( emp );
            break;
        case 3:
            Program.acceptRecord(empid);
            boolean removedStatus = test.removeRecord( empid[ 0 ] );
            Program.printRecord(removedStatus);
            break;
        case 4:
            while( ( choice = Program.subMenuList( ) ) != 0 ) {
                Comparator<Employee> comparator = null;
                switch( choice ) {
                    case 1:
                        comparator = new CompareByName();
                        break;
                    case 2:
                        comparator = new CompareByEmpid();
                        break;
                    case 3:
                        comparator = new CompareBySalary();
                        break;
                }
                test.printRecords( comparator );
            }
            break;
    }
}
```

Vector<E>

- It is resizable array.
- It implements List, RandomAccess, Cloneable, Serializable.
- It is List collection.
- It is synchronized collection.
- Default capacity of vector is 10. If vector is full then its capacity gets increased by its existing capacity.
- We can traverse elements of vector using Iterator, ListIterator as well as Enumeration.
- It is introduced in jdk 1.0.
- Note: If we want to manage elements of non final type inside Vector then non final type should override "equals" method.

Synchronized Collections

- 1. Vector
- 2. Stack(Sub class of Vector)
- 3. Hashtable
- 4. Properties(Sub class of Hashtable)

Enumeration<E>

- It is interface declared in java.util package.
- Methods of Enumeration I/F
 - 1. boolean hasMoreElements()
 - 2. E nextElement()
- It is used to traverse collection only in forward direction. During traversing, we can not add, set or remove element from collection.
- It is introduced in jdk 1.0.
- "public Enumeration<E> elements()" is a method of Vector class.

Core Java Notes By ASHOK PATE

The screenshot shows an IDE interface with a code editor and a console window. The code in the editor is as follows:

```
1 package com.sunbeaminfo.cj06.collection.test;
2
3 import java.util.Enumeration;
4 import java.util.Vector;
5
6 public class Program {
7     public static Vector<Integer> getCollection( )
8         Vector<Integer> v = new Vector<>();
9         v.add(10);
10        v.add(20);
11        v.add(30);
12        return v;
13    }
14    public static void main(String[] args) {
15        Vector<Integer> v = Program.getCollection();
16        Enumeration<Integer> e = v.elements();
17        Integer element = null;
18        while( e.hasMoreElements() ) {
19            element = e.nextElement();
20            System.out.println(element);
21        }
22    }
23 }
```

The console window on the right displays the output: 10, 20, and 30, each on a new line. The number 30 is highlighted with a blue selection bar.

The screenshot shows an IDE interface with a code editor and a console window. The code in the editor is as follows:

```
1
2
3 import java.util.Enumeration;
4 import java.util.Iterator;
5 import java.util.Vector;
6
7 public class Program {
8     public static Vector<Integer> getCollection( )
9         Vector<Integer> v = new Vector<>();
10        v.add(10);
11        v.add(20);
12        v.add(30);
13        return v;
14    }
15    public static void main(String[] args) {
16        Vector<Integer> v = Program.getCollection();
17        Iterator<Integer> itr = v.iterator();
18        Integer element = null;
19        while( itr.hasNext() ) {
20            element = itr.next();
21            System.out.println(element);
22        }
23    }
}
```

The console window on the right displays the output: 10, 20, and 30, each on a new line. The number 30 is highlighted with a blue selection bar.

The screenshot shows a Java code editor with a file named "Program.java". The code demonstrates how to use a `Vector` and its `ListIterator`. The `getCollection` method creates a `Vector` and adds elements 10, 20, 30, 40, and 50. The `main` method retrieves this collection and prints its elements using a `ListIterator`. The code uses `itr.listIterator()` to start iteration from the beginning of the list.

```
9  public static Vector<Integer> getCollection( ){
10     Vector<Integer> v = new Vector<>();
11     v.add(10);
12     v.add(20);
13     v.add(30);
14     v.add(40);
15     v.add(50);
16     return v;
17 }
18 public static void main(String[] args) {
19     Vector<Integer> v = Program.getCollection();
20     ListIterator<Integer> itr = v.listIterator();           I
21     Integer element = null;
22     while( itr.hasNext() ) {
23         element = itr.next();
24         System.out.print(element+ " ");
25     }
26     System.out.println();
27     while( itr.hasPrevious() ) {
28         element = itr.previous();
29         System.out.print(element+ " ");
30     }
31 }
```

This screenshot shows the same Java code as above, but with a modification in the `main` method. Instead of starting the iteration at the beginning with `listIterator()`, it starts at the end of the list by using `listIterator(v.size())`. This results in the output being printed in reverse order, from 50 down to 10.

```
9  public static Vector<Integer> getCollection( ){
10     Vector<Integer> v = new Vector<>();
11     v.add(10);
12     v.add(20);
13     v.add(30);
14     v.add(40);
15     v.add(50);
16     return v;
17 }
18 public static void main(String[] args) {
19     Vector<Integer> v = Program.getCollection();
20     ListIterator<Integer> itr = v.listIterator( v.size());   I
21     Integer element = null;
22     while( itr.hasPrevious() ) {
23         element = itr.previous();
24         System.out.print(element+ " ");
25     }
26 }
```

Iterator<E>

- It is a interface declared in `java.util` package.
- It is used to traverse collection only in forward direction. During traversing, we can not add or set element but we can remove element from collection.
- Methods of Iterator
 - 1. `boolean hasNext()`
 - 2. `E next()`
 - 3. `default void remove()`
 - 4. `default void forEachRemaining(Consumer<? Super E> action)`
- It is introduced in jdk 1.2

ListIterator<E>

- It is sub interface of Iterator interface.
- It is used to traverse only List Collection in bidirectional.
- During traversing, we can add, set as well as remove element from collection.
- It is introduced in jdk 1.2
- Methods of ListIterator
 1. `boolean hasNext()`
 2. `E next()`
 3. `boolean hasPrevious()`
 4. `E previous()`
 5. `void add(E e)`
 6. `void set(E e)`
 7. `void remove()`

Types of Iterator

1. Fail Fast Iterator

- During traversing, using collection reference, if we try to modify state of collection and if iterator do not allows us to do the same then such iterator is called "Fail Fast" Iterator. In this case JVM throws `ConcurrentModificationException`.

```
Integer element = null;
Iterator<Integer> itr = v.iterator();
while( itr.hasNext()){
    element = itr.next();
    System.out.print(element+" ");
    if( element == 50 )
        v.add(60); //ConcurrentModificationException
}
```

2. Fail Safe Iterator

- During traversing, if iterator allows us to do changes in underlying collection then such iterator is called fail safe iterator.

```
Integer element = null;
Enumeration<Integer> e = v.elements();
while( e.hasMoreElements()){
    element = e.nextElement();
    System.out.print(element+" ");
    if( element == 50 )
        v.add(60); //OK
}
```

Stack<E>

- It is linear data structure which is used to manage elements in Last In First Out order.
- It is sub class of Vector class.
- It is synchronized collection.
- It is List Collection.
- Methods of Stack class
 1. public boolean empty()
 2. public E push(E item)

3. public E peek()
 4. public E pop()
 5. public int search(Object o)
- * Since it is synchronized collection, it slower in performance.
 - * For high performance we should use ArrayDeque class.

```
Program.java
1 package com.sunbeaminfo.cj06.collection.test;
2
3 import java.util.Stack;
4
5 public class Program {
6     public static void main(String[] args) {
7         Stack<Integer> stk = new Stack<Integer>();
8         stk.push(10);
9         stk.push(20);
10        stk.push(30);
11
12        Integer element = null;
13        while( !stk.empty() ) {
14            element = stk.peek();
15            System.out.println("Removed element is : " +element);
16            stk.pop();
17        }
18    }
19 }
20
```

Removed element is : 30
Removed element is : 20
Removed element is : 10

Implementation of Queue using stack :-

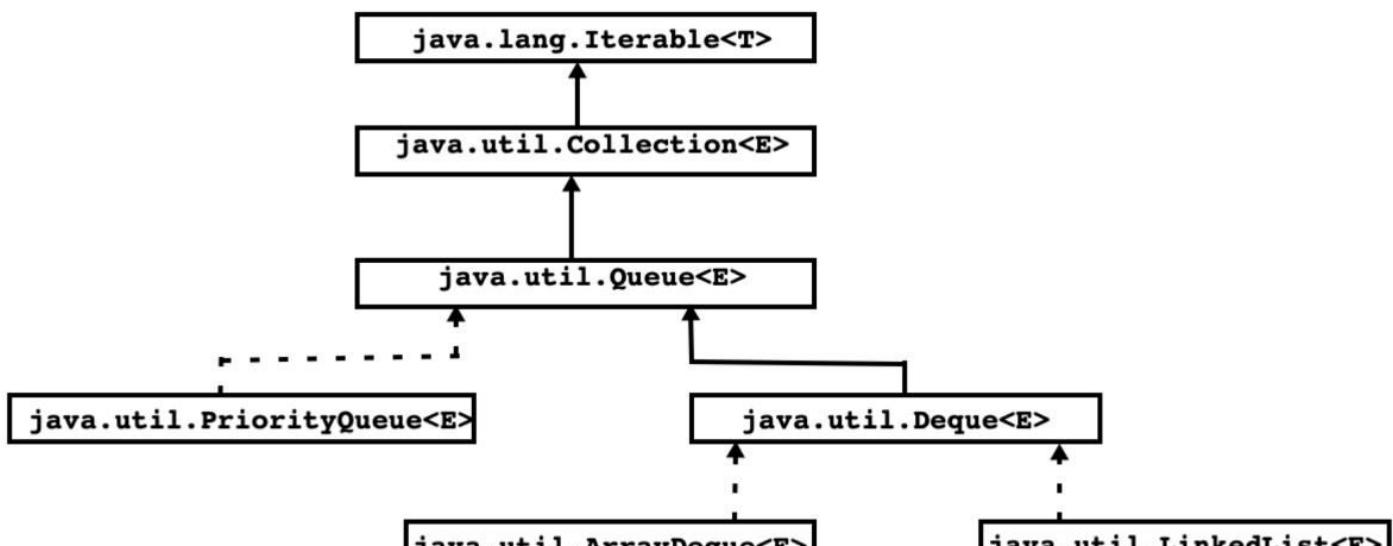
```
Program.java
1 package com.sunbeaminfo.cj06.collection.test;
2
3 import java.util.ArrayDeque;
4 import java.util.Deque;
5 import java.util.Stack;
6
7 public class Program {
8     public static void main(String[] args) {
9         Deque<Integer> stk = new ArrayDeque<Integer>();
10        stk.push(10);
11        stk.push(20);
12        stk.push(30);
13
14        Integer element = null;
15        while( !stk.isEmpty() ) {
16            element = stk.peek();
17            System.out.println("Removed element is : " +element);
18            stk.pop();
19        }
20    }
21 }
```

Removed element is : 10
Removed element is : 20
Removed element is : 30

LinkedList<E>

- It is a List collection.
- It implements List<E>, Deque<E>, Cloneable and Serializable interface.
- Its implementation depends on Doubly linked list.
- It is unsynchronized collection. Using Collections.synchronizedList() method, we can make it synchronized.
 - List list = Collections.synchronizedList(new LinkedList(...));
- It is introduced in jdk 1.2.
- Note : If we want to manage elements of non-final type inside LinkedList then non final type should override "equals" method.
- Instantiation
 - List <integer> list = new LinkedList<>();

Queue Interface Hierarchy:-



Queue Hierarchy

- It is interface declared in **java.util** package.
- It is sub interface of Collection interface.
- It is introduced in jdk 1.5

Summary of Queue methods

	<i>Throws exception</i>	<i>Returns special value</i>
Insert	<code>add(e)</code>	<code>offer(e)</code>
Remove	<code>remove()</code>	<code>poll()</code>
Examine	<code>element()</code>	<code>peek()</code>

```

package com.sunbeaminfo.cj06.collection.test;
import java.util.ArrayDeque;
import java.util.Queue;
public class Program {
    public static void main1(String[] args) {
        Queue<Integer> que = new ArrayDeque<>();
        que.add(10);
        que.add(20);
        que.add(30);

        Integer element = null;
        while( !que.isEmpty( ) ) {
            element = que.element();
            System.out.println(element);
            que.remove();
        }
    }

    public static void main(String[] args) {
        Queue<Integer> que = new ArrayDeque<>();
        que.offer(10); //offer-->add
        que.offer(20); //offer-->add
        que.offer(30); //offer-->add

        Integer element = null;
        while( !que.isEmpty( ) ) {
            element = que.peek(); //element-->peek
            System.out.println(element);
            que.poll(); //remove-->poll
        }
    }
}

```

Deque<E>

- It is usually pronounced "deck".
- It is sub interface of Queue.
- It is introduced in jdk 1.6
- If we want to perform operations from bidirection then we should use Deque interface.

Summary of Deque methods

	First Element (Head)	Last Element (Tail)		
	<i>Throws exception</i>	<i>Special value</i>	<i>Throws exception</i>	<i>Special value</i>
Insert	<code>addFirst(e)</code>	<code>offerFirst(e)</code>	<code>addLast(e)</code>	<code>offerLast(e)</code>
Remove	<code>removeFirst()</code>	<code>pollFirst()</code>	<code>removeLast()</code>	<code>pollLast()</code>
Examine	<code>getFirst()</code>	<code>peekFirst()</code>	<code>getLast()</code>	<code>peekLast()</code>

- Deque Interface is a linear collection that supports element insertion and removal at both ends.
- The class which implements this interface is ArrayDeque.
- It extends the Queue interface.
- Deque is an interface and has two implementations: LinkedList and ArrayDeque.
- Creating a Deque Syntax : Deque dq = new LinkedList(); Deque dq = new ArrayDeque();
- **ArrayDeque** :- ArrayDeque class provides the facility of using deque and resizable-array. It inherits the AbstractCollection class and implements the Deque interface. Syntax : Deque<string> arr = new ArrayDeque();

Priority Queue

- PriorityQueue class provides the functionality of the heap data structure.
- The PriorityQueue class provides the facility of using a queue.
- It does not order the elements in a FIFO manner.
- It is based on Priority Heap.

- The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.

Syntax : PriorityQueue numbers = new PriorityQueue();

```
// it will create PriorityQueue without any arguments  
//head of the queue will be smallest element of the queue  
//elements are removed in ascending order from the queue.
```

```
// package com.sunbeaminfo.cj06.collection.test;
```

```
import java.util.ArrayDeque;  
import java.util.Deque;
```

```
public class Program {
```

```
    public static void main1(String[] args) {  
        Deque<Integer> que = new ArrayDeque<>();  
        que.add(20);  
        que.add(30);  
        que.add(40);  
        que.addFirst(10);  
        que.addLast(50);  
        que.addFirst(5);  
        que.addLast(60);  
        que.removeFirst();  
        que.removeLast();
```

```
        Integer element = null;  
        while( !que.isEmpty()) {  
            element = que.element();  
            System.out.println(element);  
            que.remove();  
        }  
    }
```

```
    public static void main2(String[] args) {  
        Deque<Integer> que = new ArrayDeque<>();
```

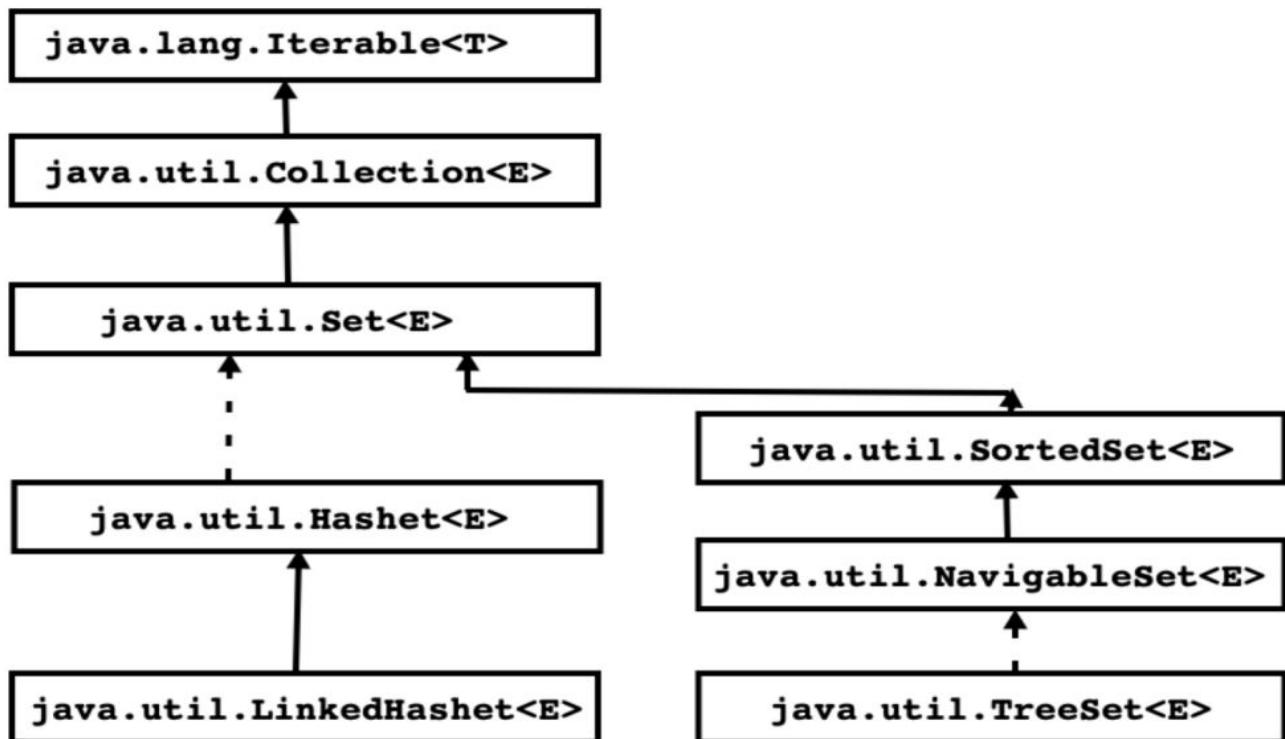
```
que.add(10);
que.add(20);
que.add(30);
que.add(40);
que.add(50);

Integer element = null;
while( !que.isEmpty() ) {
    element = que.getFirst();
    System.out.println(element);
    que.removeFirst();
}
}

public static void main3(String[] args) {
    Deque<Integer> que = new ArrayDeque<>();
    que.add(10);
    que.add(20);
    que.add(30);
    que.add(40);
    que.add(50);

    Integer element = null;
    while( !que.isEmpty() ) {
        element = que.getLast();
        System.out.println(element);
        que.removeLast();
    }
}
}
```

Set Interface Hierarchy :-



- It is sub interface of `java.util.Collection` interface.
- HashSet, LinkedHashSet, TreeSet etc. implements Set interface. It is also called as Set collection.
- Set collections do not contain duplicate elements.
- Set will not maintain any order for elements
- While adding new element it is using Object's equals() and hashCode() method to check , if such element is there or not in set.
- There are three concrete Set implementations that are part of the Collection Framework: HashSet, TreeSet, and LinkedHashSet.
- You use HashSet, which maintains its collection in an unordered manner.
- If this doesn't suit your needs, you can use TreeSet.
- A TreeSet keeps the elements in the collection in sorted order
- While HashSet has an undefined order for its elements, LinkedHashSet supports iterating through its elements in the order they were inserted.
- Understand that the additional features provided by TreeSet and LinkedHashSet add to the runtime costs.

TreeSet<E>

- It is Set collection.
- It can not contain duplicate element as well as null element.
- It is sorted collection.
- Its implementation is based on TreeMap
- It is unsynchronized collection.
- Using "Collections.synchronizedSortedSet()" method we can make it synchronized. → SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));
- It is introduced in jdk 1.2
- Note : If we want to manage elements of non final type inside TreeSet then non final type should implement Comparable interface.
- Instantiation → Set set = new TreeSet<>();

Hashing :-

- Hashing is a searching algorithm which is used to search element in constant time(faster searching).
- In case array, if we know index of element then we can locate it very fast.
- Hashing technique is based on "hashcode".
- Hashcode is not a reference or address of the object rather it is a logical integer number that can be generated by processing state of the object.
- Generating hashcode is a job of hash function/method.
- Generally hashcode is generated using prime number. Hashing.
- If state of object/instance is same then we will get same hashcode.
- Hashcode is required to generate slot.
- If state same. of objects are same then their hashcode and slot will be same.
- By processing state of two different object's , if we get same slot then it is called collision.

- Collision resolution technique:
 - Separate Chaining/ Open Hashing
 - Open Addressing / Close Hashing
- 1. Linear Probing
- 2. Quadratic Probing
- 3. Double Hashing / Rehashing
- Collection(LinkedList/Tree) maintained per slot is called bucket.
- Load Factor = (Count of bucket / Total elements);
- In hashCode based collection, if we want manage elements of non final type then reference type should override equals() and hashCode() method.
- we want to generate hashCode then in sub class.
- hashCode() is non final method of java.lang.Object class.
- Syntax: → public native int hashCode();
- On the basis of state of the object, we should override hashCode() method
- The hashCode method defined by class Object does return distinct integers for distinct objects. This is typically implemented by converting the internal address of the object into an integer.

HashSet<E>

- It Set Collection.
- It can not contain duplicate elements but it can contain null element.
- Its implementation is based on HashTable.
- It is unordered collection.
- It is unsynchronized collection. Using Collections.synchronizedSet() method, we can make it synchronized.
- It is introduced in jdk 1.2
- Note : If we want to manage elements of non final type inside HashSet then non final type should override equals and hashCode() method.
- Instantiation: - Set <Integer> set = new HashSet<>();

LinkedHashSet<E>

- It is sub class of HashSet class.
- Its implementation is based on linked list and Hashtable.
- It is ordered collection.

- It is unsynchronized collection. Using Collections.synchronizedSet() method we can make it synchronized:-

➤ Set s = Collections.synchronizedSet(new LinkedHashSet(...));

- It is introduced in jdk 1.4

- It can not contain duplicate element but it can contain null element.

```
// package com.sunbeaminfo.cj06.collection.test;
import java.util.HashSet;
import java.util.LinkedHashSet;
import java.util.Set;
import java.util.TreeSet;

public class Program {

    public static void main1(String[] args) {
        Set<Integer> set = new TreeSet<>();
        set.add(50);
        set.add(10);
        set.add(40);
        set.add(20);
        set.add(30);

        for (Integer element : set) {
            System.out.println(element);
        }
    }

    public static void main2(String[] args) {
        Set<Integer> set = new TreeSet<>();
        set.add(50);
        set.add(10);
        set.add(40);
        set.add(20);
        set.add(30);
        //Duplicate values not allowed.
        set.add(50);
        set.add(10);
        set.add(40);
        set.add(20);
        set.add(30);
    }
}
```

```
for (Integer element : set) {  
    System.out.println(element);  
}  
}  
  
public static void main3(String[] args) {  
    Set<Integer> set = new TreeSet<>();  
    set.add(50);  
    set.add(10);  
    set.add(40);  
    set.add(20);  
    set.add(30);  
    set.add(null); //NullPointerException  
  
    for (Integer element : set) {  
        System.out.println(element);  
    }  
}  
  
public static void main4(String[] args) {  
    Set<Integer> set = new HashSet<Integer>();  
    set.add(234);  
    set.add(9823);  
    set.add(1863);  
    set.add(754);  
    set.add(50);  
  
    for (Integer element : set) {  
        System.out.println(element);  
    }  
    //output: 754 50 1863 234 9823  
}  
  
public static void main5(String[] args) {  
    Set<Integer> set = new HashSet<Integer>();  
    set.add(234);  
    set.add(9823);  
    set.add(1863);  
    set.add(754);  
    set.add(50);  
  
    set.add(234);
```

```
        set.add(9823);
        set.add(1863);
        set.add(754);
        set.add(50);

        for (Integer element : set) {
            System.out.println(element);
        }
        //output:- 754 50 1863 234 9823
    }

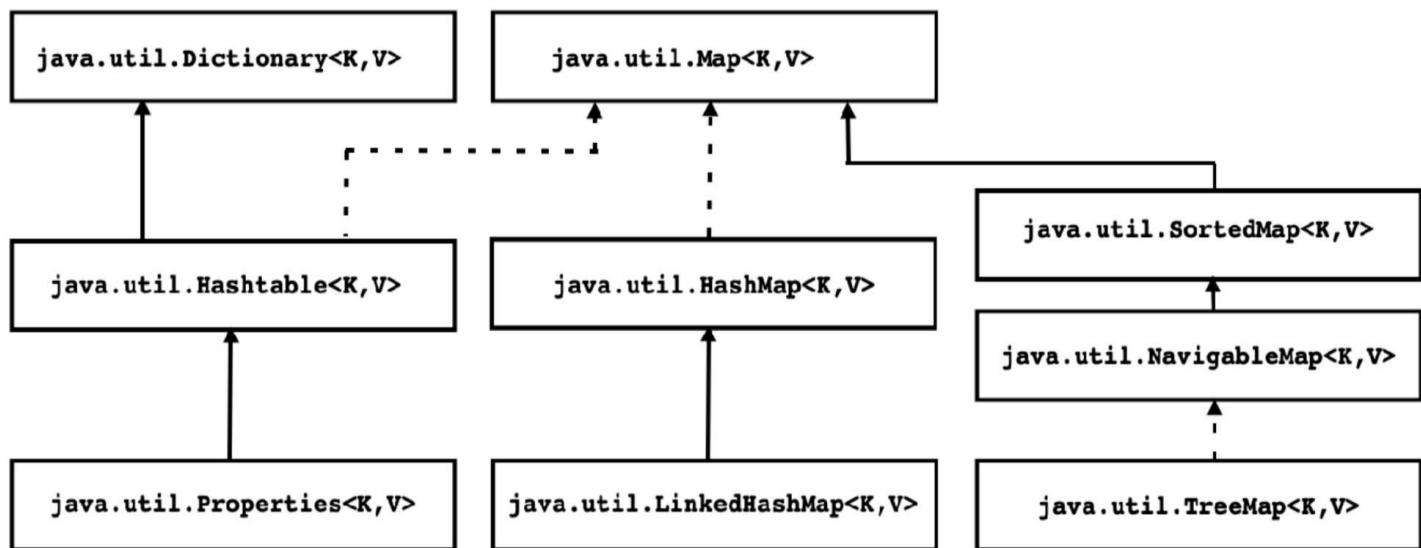
    public static void main6(String[] args) {
        Set<Integer> set = new HashSet<Integer>();
        set.add(234);
        set.add(9823);
        set.add(1863);
        set.add(754);
        set.add(50);
        set.add(null);

        for (Integer element : set) {
            System.out.println(element);
        }
        //output:- null 754 50 1863 234 9823
    }

    public static void main(String[] args) {
        Set<Integer> set = new LinkedHashSet<Integer>();
        set.add(234);
        set.add(9823);
        set.add(1863);
        set.add(754);
        set.add(50);

        for (Integer element : set) {
            System.out.println(element);
        }
        //Output:-234 9823 1863 754 50
    }
}
```

Map Interface Hierarchy :-



Dictionary<K,V>

- It is abstract class declared in `java.util` package.
- It is super class of `Hashtable`.
- It is used to store data in key/value pair format.
- It is not a part of collection framework
- It is introduced in jdk 1.0
- Methods:
 1. `public abstract boolean isEmpty()`
 2. `public abstract V put(K key, V value)`
 3. `public abstract int size()`
 4. `public abstract V get(Object key)`
 5. `public abstract V remove(Object key)`
 6. `public abstract Enumeration<K> keys()`
 7. `public abstract Enumeration<V> elements()`
- Implementation of `Dictionary` is Obsolete.

Map<K,V>

- It is part of collection framework but it doesn't extend Collection interface.
- This interface takes the place of the Dictionary class, which was a abstract class rather than an interface. totally
- HashMap, Hashtable, TreeMap etc are Map collection's.
- Map collection stores data in key/value pair format.
- In map we can not insert duplicate keys but we can insert
- It is introduced in jdk 1.2
- Map.Entry<K,V> is nested interface of Map<K,V>.
- Following are abstract methods of Map.Entry interface. duplicate values.
 1. K getKey()
 2. V getValue()
 3. V setValue(V value)

Map<K,V>

- Abstract Methods of Map<K,V>
 1. boolean isEmpty()
 2. V put(K key, V value)
 3. void putAll(Map keySet())
 - 4.int Size()
 - 5.boolean containsKey(Object Key)
 - 6.boolean containsValue(Object value)
 - 7.V.get(Object Key)
 - 8.V remove(Object Key)
 - 9.Void clear()
 - 10.Set<K> KeySet()
 - 11.Collection values()
 - 12.Set<Map.Entry> entrySet() V> m)
- An instance, whose type implements Map.Entry interface is called enrty instance.

Hashtable<K,V>

- It is Map collection which extends Dictionary class.
- It can not contain duplicate keys but it can contain duplicate values.
- In Hashtable, Key and value can not be null.
- It is synchronized collection.
- It is introduced in jdk 1.0
- In Hashtable, if we want to use instance non final type as key should override equals and hashCode method.

HashMap<K,V>

- It is map collection
- Its implementation is based on Hashtable.
- It can not contain duplicate keys but it can contain duplicate values.
- In HashMap, key and value can be null.
- It is unsynchronized collection. Using Collections.synchronizedMap() method, we can make it synchronized. :-

➤ Map m = Collections.synchronizedMap(new HashMap(...));

- It is introduced in jdk 1.2.
- Instantiation –
 - Map map = new HashMap<>();
- Note : In HashMap, if we want to use element of non final type as a key then it should override equals() and hashCode() method.

LinkedHashMap<K,V>

- It is sub class of HashMap class
- Its implementation is based on LinkedList and Hashtable.
- It is Map collection hence it can not contain duplicate keys but it can contain duplicate values.
- In LinkedHashMap, key and value can be null.
- It is unsynchronized collection. Using Collections.synchronizedMap() method we can make it synchronized:-

➤ Map m = Collections.synchronizedMap(new LinkedHashMap(...));

- LinkedHashMap maintains order of entries according to the key.

- Instantiation:-

➤ Map map = new LinkedHashMap<>();

- It is introduced in jdk 1.4

TreeMap<K,V>

- It is map collection.
- It can not contain duplicate keys but it can contain duplicate values.
- In TreeMap, key must not be null but value can be null.
- Implementation of TreeMap is based on Red-Black Tree.
- It maintains entries in sorted form according to the key.
- It is unsynchronized collection. Using Collections.synchronizedSortedMap() method, we can make it synchronized. →
 - SortedMap m = Collections.synchronizedSortedMap(new TreeMap(...));
- Instantiation: →
 - Map map = new TreeMap<>();
- It is introduced in jdk 1.2
- Note : In TreeMap, if we want to use element of non final type as a key then it should implement Comparable interface

Programs:-

```
package com.sunbeaminfo.cj06.collection.test;

import java.util.Collection;
import java.util.Hashtable;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;

public class Program {
    public static Map<Integer, String> getMap( ){
```

```
Map<Integer, String> map = new Hashtable<>();
map.put(3005, "DAC");
map.put(2291, "DMC");
map.put(1234, "DESD");
map.put(3579, "DBDA");
map.put(4505, "PREDAC");
return map;
}
private static void printKeys(Map<Integer, String> map) {
    Set<Integer> keys = map.keySet();
    for (Integer key : keys) {
        System.out.println(key);
    }
}
private static void printValues(Map<Integer, String> map) {
    Collection<String> values = map.values();
    for (String value : values) {
        System.out.println(value);
    }
}
private static void printEntries(Map<Integer, String> map) {
    Set<Entry<Integer, String>> entries = map.entrySet();
    for (Entry<Integer, String> entry : entries) {
        System.out.println(entry.getKey()+" "+entry.getValue());
    }
}
private static void findEntry(Map<Integer, String> map, int id) {
    Integer key = new Integer(id);
    if( map.containsKey(key)) {
        String value = map.get(key);
        System.out.println(key+" "+value);
    }else
        System.out.println(key+" not found");
}
private static void removeEntry(Map<Integer, String> map, int id)
{
    Integer key = new Integer(id);
    if( map.containsKey(key)) {
        String value = map.remove(key);
        System.out.println(key+" "+value+" is removed");
    }
}
```

```
    }else
        System.out.println(key+" not found");

    }
public static void main(String[] args) {
    Map<Integer, String> map = Program.getMap();
    //Program.printKeys( map );
    //Program.printValues( map );
    //Program.printEntries( map );
    //Program.findEntry( map, 1234 );
    Program.removeEntry( map, 1234 );
}
}
```

freelance_Project available to buy contact on 8007592194

SR.NO	Project NAME	Technology
1	E-Learning HUB	React+Springboot+MySQL
2	PG MATES	React+Springboot+MySQL
3	Tour and Travel	React+Springboot+MySQL
4	Marriage Hall booking	React+Springboot+MySQL
5	Bus ticket booking Mini Project	React+Springboot+MySQL
6	Quizz App /Exam Portal Mini Project	Springboot,MySQL,JSP,Html
7	Event Management System	React+Springboot+MySQL
8	Hotel Mangement System	React+Springboot+MySQL
9	Agriculture Web Project	React+Springboot+MySQL
10	AirLine Reservation System	React+Springboot+MySQL
11	E-Commerce Web Project	React+Springboot+MySQL
12	Sport Ground Booking	React+Springboot+MySQL
13	CharityDonation web project	React+Springboot+MySQL
14	Hospital Management Project	React+Springboot+MySQL
15	Online voting System Mini project	Springboot,MySQL,JSP,Html
16	E-Commerce shop mini project	Springboot,MySQL,JSP,Html
17	Job Portal web project	React+Springboot+MySQL
18	Insurance policy Portal	React+Springboot+MySQL
19	Transpotation Services portal	React+Springboot+MySQL
20	E-RTO Driving licence portal	React+Springboot+MySQL
21	doctor Appointment Portal	React+Springboot+MySQL
22	Online food delivery Project	React+Springboot+MySQL
23	Municipal Corporation Management	React+Springboot+MySQL
24	E-College Portal Project	React+Springboot+MySQL
25	Gym Management	React+Springboot+MySQL
X 26	Bike Booking System Portal	React+Springboot+MySQL
27	Food Waste Management Portal	React+Springboot+MySQL
28	Online Pizza delivery Portal	React+Springboot+MySQL
29	Fruite Delivery portal	React+Springboot+MySQL
30	HomeRental Booking Project	React+Springboot+MySQL
31	FarmerMarketplace	React+Springboot+MySQL