Industrial Java Programming By Kiran

Dedicated to

My Father and

My Lovely Daughter Amullya

Fifth Edition: 2018-19

Printed by :

Cover Designing

Mr. Darshit Rabadiya

Index

About This Book

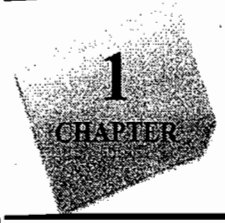
This book is an attempt to teach you all about the Java language in the simplest way possible and how to use it to create business applications. By the time you finish reading this book, you'll know enough about Java to do just about anything.

Who Should Read This Book

This book is intended for people with at least some basic programming background, which includes people with years of programming experience or people with only a small amount of experience. If you understand what variables, loops, and functions are, you'll be just fine reading this book. The sorts of people who might want to read this book include you , If

- You want to move on to the next level in Web page design.
- You have some basic knowledge in manual testing and want to move on selenium automation testing but you've heard Java is easy to learn, really powerful, and very cool.
- You've programmed C and C++ for many years, then you heard about Java, which is becoming really popular and you're wondering what the fuss is all about
- You've been told that Java is really good for Web-based applications, and are curious about how good it is for creating other more general applications.

What if you know programming, but you don't know object-oriented programming? Fear not. This book assumes no background knowledge in object-oriented design. However, If you do, then the first couple of days will be relatively easy for you.



Basics

Why is Java platform independent? Why not C &C++?

- When you run a C native code, it will be executed immediately. Whereas in the case of Java, when you run the byte code, the byte code will first be converted into native code and then be executed
- When we say that java is platform independent, it means that the byte code is independent of any platforms.
- When you create a .exe, it captures a machine code, where as when we create a byte code it does not capture machine code
- Java programming language was developed by the Sun Microsystems and was launched in 1995. The language has had many versions, the latest being Java SE 8, and still finds relevance in the industry

Process of java program execution

Human readable code with java syntax(**.java file**) --> (javac compiler) --> Byte Code(**.class file**) --> (java interpreter) --> native code

In the above given flow diagram, javac and java comes in to our system when we install jdk.

The components involved in writing program up to getting the output are:

- java.exe set up file
- Javac compiler, which checks syntax
- Java interpreter, which understands only byte code and converts it to native code
- Human readable code with proper java syntax, i.e. the Java File
- Byte code - after the compiler compiles the Java File, i.e. Class file
- Native code - We see output same in background is native code out of all these, only the byte code is platform independent

What is a Java compiler?

The Java compiler is a programme which is implemented in C and C++ with the name javac.exe. It compiles the text file into a platform independent java file.

The compiler is responsible for the following tasks:

- Checking syntactical mistakes or Syntax Error
- Converting source code into byte code with the help of JVM

- Adding additional code to your code if required. This can be observed by using a de- compiler, which is called cavaj

What is a Java interpreter?

Java interpreter is a programme which is implemented in C and C++ and goes by the name java.exe. It basically reads the byte code and executes it.

This java interpreter is responsible for the following tasks:

- Converting the byte code to native code line by line
- Executing the native code

What is JIT compiler?(JUST-IN-TIME)

The full form of JIT is Just In Time. It is a programme written in C and C++ with the name Java.exe, and is provided from Java 2 version. The JIT compiler's task is the same as Java interpreter's. It converts the entire byte code into native code once and then executes the same.

What is JVM? (JAVA VIRTUALMACHINE)

JVM, or Java Virtual Machine, is a specification provided by Sun whose implementation provides the environment to run our Java Applications.

JVM implementations are called JRE (JAVA RUNTIME ENVIRONMENT) [The next section will give you further details about JVM]

What is JDK (JAVA DEVELOPMENT KIT) and JRE?

The **Java Development Kit** (JDK) is a software development environment used for developing Java applications and applets. It includes the Java Runtime Environment (JRE), an interpreter/ loader (java), a compiler (javac), an archive (jar), a documentation generator (java doc) and other tools needed in Java development. People new to Java may be confused about whether to use the JRE or the JDK. To run Java applications and applets, simply download the JRE. However, to develop Java applications and applets as well as run them, the JDK is useful.

Java developers are initially presented with two JDK tools, java and javac. Both can be run from the command prompt. Java source files are simple text files saved with a .java extension. After writing and saving the Java source code, the javac compiler is invoked to create a .class file. Once the .class file is created, the 'java' command can be used to run the java program.

For developers who wish to work in an integrated development environment (IDE), a JDK bundled with Net beans can be downloaded from the Oracle website. Such IDEs speed up the development process by introducing point-and-click and drag-and- drop features for creating an application.

There are different JDKs for various platforms. The supporting platforms include Windows, Linux and Solaris. Mac users need a different software development kit, which includes adaptations of some tools found in the JDK.

Remember following statements :

- A 'C' program is platform dependent
- A Java(.class file) program is platform independent
- The Java compiler is platform dependent
- Java interpreter is platform dependent
- JVM is platform dependent
- JRE is platform dependent

Steps to install Java:

Click on the installer called Java.exe setup Select radio button I accept ……..

1) Install
2) Finish
3) First Simple Program in Java

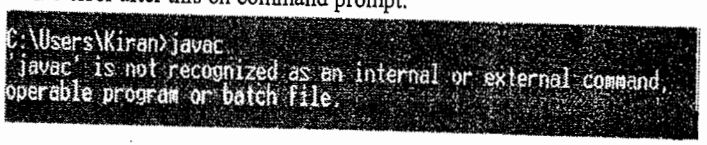
Step 1. Open notepad write as shown below:

```
class Hello {
   public static void main(String[] args) {
      System.out.println("Java By Kiran");
      System.out.println("visit www.javabykiran.com");
   }
}
```

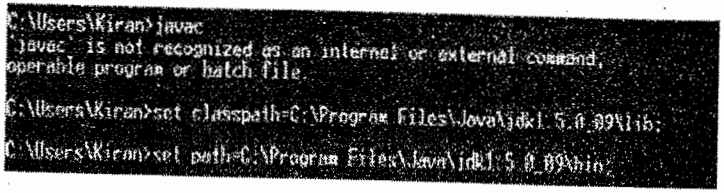
Save with the name Hello.java

javac Hello. Java // Compile File

We may get some error after this on command prompt.



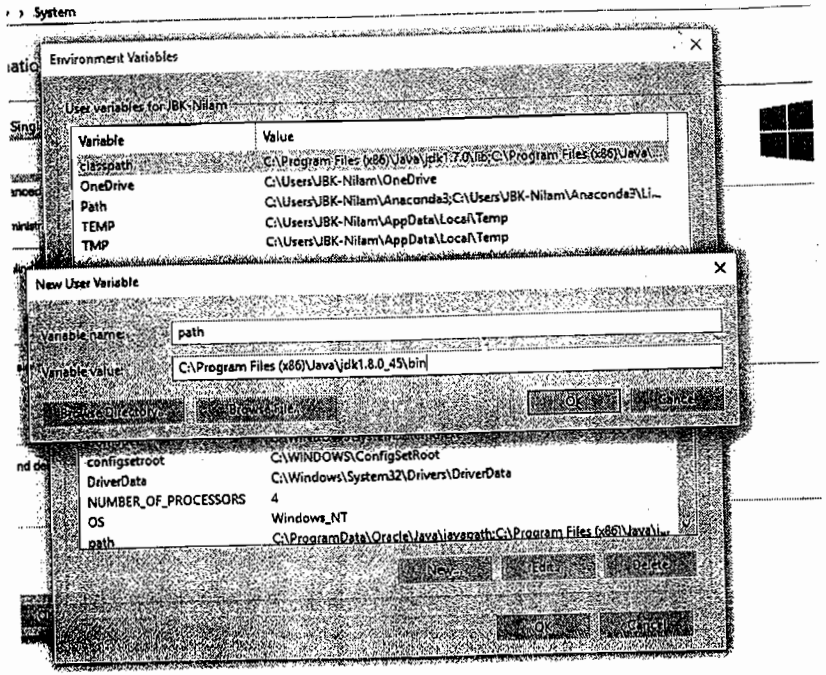
This is because we have not set the path for java and DOS is unable to understand what javac is. We will now configure our system to recognize javac by setting a path like shown below. When you set the path on the command line (see below), it is temporary.

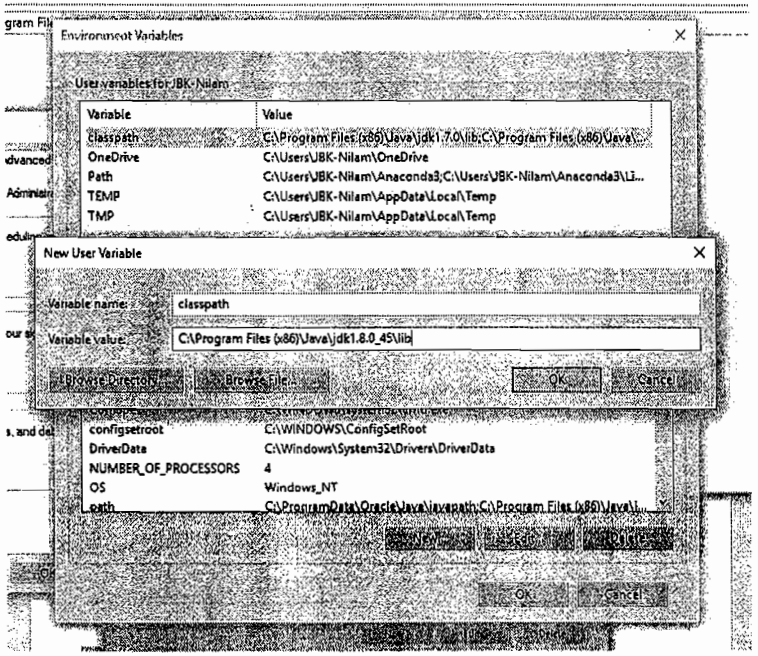


You can set the path permanently in the system environment variables like this:

- Right click on My computer, Click on Properties
- Click on Advanced System Settings
- Click on Advanced tab
- Click on Environment Variables

Add two variables (path and classpath) as shown in the diagram below:





Java Language:

- Character set
- Keywords
- User defined words
- Data types
- Variable
- Constants
- Literals
- Operators
- Control statements
- Arrays

Character set

- Digits(0-9)
- Alphabets (A-Z, a-z)

Keywords

Keywords are predefined code which has a specific meaning and that meaning cannot be changed. They are total 53 in number and cannot be used elsewhere.

For example, these keywords may not be used as variables, methods, classes, etc.

List of Keywords

Data Type (8)	Access Modifiers	Control Statement
byte short int long float double char Boolean	private protected public default	Else switch case default for while Do Continue Break goto [reserved]

Object & Classes (7)	Modifiers (8)	Exceptions
class interface extends implements this super new	static abstract synchronized volatile native transient strictfp final	try catch finally throw throws

Objects & Classes (7)
Modifies (8)
Exceptions (5)

Miscellaneous	Packages (2)	
return	package	JAVA 2 -> 49 - > 52
const(*)	import	Keywords
instance of		JAVA 2 -> 50 -> 53
void		
assert		

User defined words

- They will be used as names to variables, methods, classes, etc. When you are defining user defined words, you need to remember the followingpoints/rules:

- User defined words can contain all the digits, all the alphabets and only two special symbols, which are, '_'(underscore) and '$'(dollar)
- The first character must be an alphabet, underscore or dollar
- Keywords can be used as user defined words

Which of the following are valid identifiers or user defined words?

(1) abc 123 → Invalid (because of the space between characters)
(2) 123abc → Invalid (because it starts with a number)
(3) -123$987 → Invalid (because it starts with a dash '-')
(4) Int → Valid
(5) hello → Valid
(6) a → Valid

Data types

There are two categories of data types:

- Primitive data type
- User defined data type

Primitive data type

Primitive data type is a predefined type of data, and is supported by java. They are eight in number and are listed below:

Data Type	Size (byte)	Default Value	Range
byte	1	0	Minimum value is-128(2^7) Maximum value is 127 (inclusive)(2^7-1)
short	2	0	Minimum value is -32,768(-2^15) Maximum value is 32,767 (inclusive)(2^15-1)
int	4	0	Minimum value is -2,147,483,648(-2^31) Maximum value is 2,147,483,647(2^31-1)
long	8	0L	Minimum value is - 9,223,372,036,854,775,808(-2^63) Maximum value is 9,223,372,036,854,775,807(2^63-1)
float	4	0.0f	Example : float f1 = 234.5f
double	8	0.0d	Example : double d1 = 123.4
char	2	Blank	Minimum value is 'u0000'(or 0) Maximum value is 'uffff'(or 65,535 inclusive)
boolean	1	false	Example : boolean one = true Example : boolean one = false

User defined data type

User defined data type is a data type which is obtained from an already present data type.

Listed below are the types of user defined types :

- Class type
- Interface type
- Two more types called enum and annotation
- Class objects, and various type of array variables come under reference variables
- Default value of any reference variable is null
- A reference variable can be used to refer to any object of the declared or compatible type

  Example: Animal animal = new Animal("giraffe");

Variable

A variable in java is a section of memory that contains or may contain a data value. Therefore, it can be said that is a name allotted to the location of memory.

Syntax :

```
Data type  var_name=value;
e.g.   int a;
       int b = 20;
```

- These two variables a and b are called as primitive variables because they are declared with primitive data type. Here, we can say that b is type of int whose value is 20

  Hello h =new Hello();
- The above given variable, h called reference variables because they are declared with user defined data type. Here we can say h is the type of Hello with Hello(); as the new value

Difference between primitive and reference variable:

Primitive variable	Reference variable
Primitive data type is called primitive variable	Variable declared with user defined data type is called reference variable
Memory allocation for Primitive variables will depend on the primitive Data type used	Always allocate eight bytes of memory for reference variable
The default value for the primitive Variables will depend on the primitive data type used	Null will always be assigned
Primitive variables contain value as address or literal	Reference variable contains the addresses of an object

Constants (will be covered in 'Final' chapter of this book)

However, for your basic understanding, constants may be defined as variables whose values cannot be changed.

- Constants are also called final variables
- Value assigned for the final variable can not be modified

Syntax:

final Data type var_name = value;

For ex. final int a = 0;

//a = a + 1; //Showing error, it is not possible to do any operation on a final variable like this.

Literals

Literals are a value which you can assign to the variable or a constant.

There are five types of literals:

- **Integer literal**
  - It can be assigned to one of the integer data types; there are three types of integer literals
  - Decimal integer literals: Base 10 [0-9]
  - Octal integer literals (starts with 0) [0-7]
  - Hexadecimal integer literals (0-9, A-F)

  Examples (imp. Interview questions)
  - int a=123;
  - int a=0123; (starting with zero)
  - int a=0x123A; (starting with zero & x)
- **Floating point literal**
  - It can be assigned to floating point variables declared with float or double datatypes There are two types of representation:
    - Decimal pointer presentation
    - Exponential representation
  - double d16 = 99.99;
  - double d12 = 99.99e-5; // 99.99e-4
  - double d15 = 9.9E3; // (9.9*10*10*10)
- **Character literal**
  - It is a single character which is enclosed between single quotes ' '

    Example : 'a' '+' ' '

    ' ' // invalid (because of space)
  - A character literal can be assigned to a character type variable

    Example : char ch1='a';

- Each character which is enclosed in single quotation marks will have integer equivalent's value as per ASCII character set as shown in the example below
  - 'a' = 97, 'b' = 98, ---------- ,'y' = 121, 'z' =122
  - 'A' = 65, 'B' = 66, ---------- , 'Y' = 89, 'Z' = 90
  - '0' = 48, '1' = 49, ---------- , '8' = 56, '9' = 57
- int a = 'd'; // ok If we print, it will print 100 as per the ASCII code (ASCII - American Standard Code for Information Interchange)
- Java uses the character set UNICODE (Universal Code)
- UNICODE character set takes two bytes of memory for each character and supports multi languages
- ASCII character set takes only one byte character and supports only English language

➤ Boolean literal

- There are two boolean literals:
  - True
  - False
- Boolean literals can be assigned to the variables which are declared with boolean data type
  Example : boolean b=true;

➤ String literal

- It is a collection of one or more characters enclosed between double quotation marks.
- String literals can be assigned to reference variable of type String

Example

```
class Lab1 {
      String s1 = "hello guys";
      String s2 = "123 guys as d";
      String s3 = "     ";
  public static void main(String as[]) {
      int a = 10;
      int $b = 20;
      // int 1c=30;
      // int ab=40;
      // int 1=10;
      System.out.println(-a);
      System.out.println($b);
  }
}
```


---Output---

```
-10
20
```

Lab2.Java

```
package com.javabykiran.basics;
 class Lab2 {
      byte b; // variable & constants demo
      short s; // 0
      int i; // 0
      long l; // 0
      float f; //0.0
      double d; // 0.0
      char ch; // for 1 character's blankspace
      boolean b1; //  false
    void show1() {
      System.out.println("variables & constants demo");
      System.out.println(b);
      System.out.println(d);
      System.out.println(s);
      System.out.println(ch);
      System.out.println(i);
      System.out.println(b1);
      System.out.println(l);
      System.out.println(f1);
      System.out.println(f);
      // System.out.println (x);x cannot resolved to a variable
      // x=x+1
    }
    void show2() {
      System.out.println("integer literal demo");
      int a = 12;
      int b = 10;
      int c = 0x12a;// integer literal demo
      Byte d = 012;
      System.out.println(a);//12
      System.out.println(b); // 10
      System.out.println(c); // 298
      System.out.println(d);// 10
    }
```

```
void show3() {
     System.out.println("floating point literal demo");
     float f1 = 99.97f;
     double d1 = 9.9e-9;
     double d2 = 9.9E+9;
     System.out.println(f1);// 99.97
     System.out.println(d1); // 9.9E-9
     System.out.println(d2); //9.9E9
 }
void show4() {
  System.out.println("character   literal demo");
  char ch1 = 'A';
  int x1 = 'A';
    // char ch2 = ''; //error invalid character constant
   //int x2 = '' ;//error
   char ch3 = 'I';
   int x3 = 'I';
  System.out.println(ch1); // A
  System.out.println(x1); //65
  // System.out.println (ch2); //error +
  // System.out.println (x2); //error +
  System.out.println(ch3); // I
  System.out.println(x3); // 73
}
void show5() {
  System.out.println("Boolean literal demo");
  boolean bb1 = false;
  boolean bb2 = true;
  System.out.println(bb1); // false
  System.out.println(bb2); //true
}
void show6() {
  System.out.println("string literal demo");
  String ste1 = "";
  String ste2 = "123 abc+";
  String ste3 = "1";
  System.out.println("ste1 " + ste1); // ste1- - "blank"
  System.out.println(ste1.length());
  System.out.println("ste2 = " + ste2); // ste2=123 abc +
  System.out.println(ste2.length());//9
  System.out.println("ste3="   +ste3); //ste = 1
  System.out.println(ste3.length()); // 1
```



```
    }
  }
package com.javabykiran.basics;
 class Lab2Test {
    public static void main(String[] args){
        Lab2 h = new Lab2();
        h.show1();
        h.show2();
        h.show3();
        h.show4();
        h.show5();
        h.show6();
    }
}
```

---Output---

```
variables & constants demo 0
0.0
0
Blank
 0
false
0
10.9
0.0
integer literal demo
12
10
298
10
floating point literal demo
99.97
9.9E-9
9.9E9
character literal demo
A
65
I
73
Boolean literal demo
false
true
string literal demo
ste1
0
```

```
ste2 = 123
abc + 9
ste3= 1
1
```

Operators

Operators are special symbols that perform operations.

Listed below are the types of operators:

- Arithmetic Operator (+, -, *, /,%)
- Relational Operator(>,>=,<,<=,)
- Logical Operator (&&, ||,\)
- Assign Operator (=, +=, -=,*=, /=,&=)
- Increment / Decrement Operator(++,--)
- Ternary Operator (?:)
- Bitwise Operator (&, \,^,)
- Equality Operator (==, != ,)
- Unary Operator (\ , ++,-- ,~)
- Shift Operator (>>,<<)
- Instance of Operator (instanceof)
- New Operator (new)

Arithmetic Operator (+, -, *, /,%)

- Arithmetic operation from arithmetic expression (a+b, a/b)
- The result of arithmetic expression is integer literal or floating point literal

Example

```
package com.javabykiran.basics;
  public class Lab3Test {
     public static void main(String[] args) {
         byte b1 = 10;
         byte b2 = b1 + 2; // error type mismatch
         System.out.println(b2);
         float f1 = 9.9f;
         float f2 = f1 + 10.0; // error type mismatch
         System.out.println(f1);
         System.out.println(f2);
     } }
```

Modifying

byte b2=b1+2 to byte b2=(byte)(b1+2); // type casting

float f2=f1+10.0 to float f2=(float)(f1+10.0);


```
package com.javabykiran.basics;
 public class Lab3Test {
    public static void main(String[ ] args) {
        byte b1 = 10;
        //byte b2 = b1 + 2;
        byte b2=(byte)(b1+2);
        System.out.println(b2);
        float f1 =9.9f;
        //float f2 = f1 + 10.0;
        float f2 = (float)(f1 + 10.0); //  type casting
        System.out.println(f1);
        System.out.println(f2);
    }
}
```

Remember the chart drawn below :

Type byte	Operator +	Type byte	Result Type Byte
byte	+	short	Byte
byte	+	int	Int
byte	+	long	Long
byte	+	float	Float
byte	+	double	Double
short	+	short	Short
short	+	int	Int
short	+	long	Long
short	+	float	Float
short	+	double	Double
int	+	int	Int
int	+	long	long

Example

```
package com.javabykiran.basics;
  public class Lab4Test {
     public static void main(String[] args) {
          int a=0/10;
          System.out.println(a); // 0
           // int b=10/0;
           // java.lang.ArithmeticExcepton:/by zero
           // system.out.println(b);
           double dl = 0.0/10.;
```

```
        System.out.println(dl);//0.0
        double d2= 10.0/00;
        System.out.println(d2);//Infinity
        int x=10%3;
        System.out.println(x);//1
        double d3=10.0/10.0;
        System.out.println(d3);//1.0
        int x1=1u;
        System.out.println(x1);//10
        double d=10.0%1.0;
        System.out.println(d);//0.0
    }
}
```

---Output---

```
0
0.0
Infinity 1
1.0
10
0.0
```

Relational operator

It is a form of relational expression whose result is the 'boolean' value a < b.

Logical operator

A Logical operator forms logical expression, which is a combination of one or more relation expressions

- (a>b) &&(a>c)
- (a>b) | |(a>c)

The result of logical expression is based on Boolean values, which in turn is based on the followi table:

A	B	A&&B [and]	A||B [OR]
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

Assignment operators

Syntax :

destination_var = source_var;

Example: int x=10+9.9; // error - Type mismatch: cannot convert from double to int
System.out.println (10+9.9); //ok
byte b=10; // ok
int x=b; //ok

- When you are using assignment operator, the source type must be the same as destination type
- When the source and destination are not same, then you need to convert the source type to required destination type
- The process of converting the source type to required destination type is called type casting

There are two types of casting:

- **Implicit casting**
- **Explicit casting**

- **Implicit casting :** When the destination is bigger than the source, then conversion happens automatically, which is called implicit casting. Another term for it is widening
- **Explicit casting :** When the destination is smaller than the sources, then you have to do explicit casting or narrowing

Syntax :

destination_var=(destination type)source_var;

Example:

```
int x=10+9.9;  // error-Type mismatch: cannot convert from double to int
int x1=(int)10+9.9;  // error-Type mismatch: cannot convert from double to int
int x2=(int) (10+9.9);  // ok
int x3=10+(int)9.9;  // ok
```

ncrement/Decrement operator

It is a type of arithmetic operator. It allowes you to add and subtract 1 from variables.

Example:

```
package com.javabykiran.basics;
 class Lab5 {
    public static void main(String[] args) {
       int x = 10;
       System.out.println(x);  //o/p 10
       int a = ++x;// prefix increment
       int b = --x;// prefix decrementt
       int c = x++;// postfix increment
```

```
        int d = x--;// postfix decrement
        int e = ++x;// prefix increment
        int f = ++x;// prefix increment
        int g = x++;//
        System.out.println(a);// 11    11    11
        System.out.println(b);// 10    10    10
        System.out.println(c);// 10    11    10
        System.out.println(d);// 11    11    11
        System.out.println(e);// 11    12    11
        System.out.println(f);// 12    12    12
        System.out.println(g);// 12    13    12
    }
}
```

---Output---

```
10
11
10
10
11
11
12
12
```

Ternary Operator

- Ternary operator is used to perform simple conditional checking

  Syntax : (Conditional expression)? s1:s2;

- First, the given expression is evaluated. If the expression is true, then s1 will be executed. If the expression is false, then s2 will be executed.

Question : Write a program to read the two numbers from command line and find the minimum and maximum number.

```
package com.javabykiran.basics;
   public class Lab6 {
       public static void main(String[] as) {
           int a =Integer.parseInt(as[0]);
           int b =Integer.parseInt(as[1]);
           int min=(a<b) ? a:b;
           int max=(a>b) ? a:b;
```



```
           System.out.println("Min :: "+min);
           System.out.println("Max ::"+max);
       }
}
```

---Output---

```
  20
1) javaLab6
   // ArrayIndexOutOfBoundsException:0
2) java Lab6 10 [when we pass single parameter below error will occur]
   // ArrayIndexOutOfBoundException:1
3) java Lab6 10 20 [output would be]
4) Min ::10
   Max :: 20
```

```
package com.javabykiran.basics;
public class Lab7 {
  public static void main(String[] args) {
     int a=10;
     int b=20;
  //int  min=(a+b)? a:b; //error-Type mismatch: cannot
                          convert  // from int to boolean
     int min=(a>b) ? a:b;
     System.out.println(min);
     }
  }
  /*error: incompatible types
  found: int
  required: boolean
  int min= (a+b) ? a:b; */
```

Question: Write a program to take three numbers from the command line and find the minimum and maximum.

```
package com.javabykiran.basics;
   public class Lab8 {
       public static void main(String[] a5) {
           int a =Integer.parseInt(a5[0]);
           int b =Integer.parseInt(a5[1]);
           int c =Integer.parseInt(a5[2]);
           int max=(a>b) ?(a>c) ?a:c: (b>c)? b:c;
```

```
        int min=(a<b)? (a<c) ?a:c: (b<c)? b:c;
        System.out.println("Min: "+min);
        System.out.println("Max "+max);
    }
}
```

---Output---

```
Min: 5
Max 15
```

Bitwise operators

- Bitwise operator acts on individuals bits of given numbers.

1.	Left shift operator	<<
2.	Right shift operator	>>
3.	Bitwise AND	&
4.	Bitwise OR	|
5.	Bitwise XOR	^
6.	Bitwise NOT	~

Examples

```
package com.javabykiran.basics;
  public class Lab10Test {
    public static void main(String[] args) {
        int a = 2; // 0010
        int b = 4; // 0100
        System.out.println("value of a before:"+a);
        System.out.println("value of b before:"+b);
        // bitwise unary complement operator (~)
   System.out.println("value of a after negation: " + ~a); //-3
   System.out.println("value of a after negation: " + ~b); //-5

      // bitwise AND operator &
      System.out.println("Result of a&b   is:"+(a & b)); //0
      // bitwise OR operator |
      System.out.println("Result of a|b is:"+(a| b)); // 6
      // bitwise XOR operator ^
      System.out.println("Result of a^b is:"+(a^ b)); // 6
       int no = 8; // 0000 1000
      System.out.println("Original number:"+no); // 8
```



```
      // left shifting bytes with 1 position
       no = no << 1; // should be 16 i.e. 0001 0000
      // equivalent of multiplication of 2
   System.out.println("value after left shift:" + no);//16
       no = -8;
      //right shifting bytes with sign 1 position
       no=no>>1; //should be 16 i.e.00010000
      // equivalent of division of 2
System.out.println("value after right shift with sign:"+no);//-4
       no = -8;
      //right shifting bytes without sign 1 position
       no=no>>>1; //should be 16 i.e.00010000
      // equivalent of division of 2
System.out.println("value after right shift with sign: " + no);
                                                  // 2147483644
    }
}
```

--Output---

```
38
76
9
5
15
9
16
501
30
2
1
```

ew operator

- A new operator is used to create an object of a given class Example:

  Hello h=new Hello();

ot operator

- A dot operator is used to refer members of a class using class name or object Example:

  Hello h=new Hello();
  h.a;
  h.show();

Operator precedence

Operator	Precedence
Post fix	expr ++ expr - -
Unary	~ ! +!expr -- expr+expr-expr
Multiplicative	* / %
Additive	+ -
Shift	<<>>>>
Relational	<><= >= instance of

Equality

bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	?
assignment	=++-=*=/=%=&= ^=|=<=>=

Note: no sizeof operator in java

```
package com.javabykiran.basics;
  public class Lab11 {
     public static void main(String[] args) {
      double a=19.9;
      //System.out.println(a>>1);//error- operator << cannot
      be applied to double
      //int x = 09; // error - The literal Octal 09 (digit 9)
     of type int is out of range
      // System.out.println(x);// integer number too large :09
      int y = 0101;
      System.out.println(y); // 65 - octal conversion

    }
}
```


---Output---

```
65
```

Control statements

There are three types of control statements:

- Conditional control statement
- Looping control statement
- Unconditional control statement

Conditional control statement:

The conditional control statement controls the order in which a java program is executed.

- If statement
- Switch statement

Syntax :

```
i) if(condition) {
     s1;
   }
ii) if(condition) {
     s1;
   }else {
     s2;
   }

example
    if(1) { //This gives error
      s1; // found int Required boolean
    }
iii) if (condition) {
       s1;
     } else
     if (condition) {
       s2;
     } else {
       s3;
}
```

Question : Write a program to pass three numbers from command lines and find

i) minimum ii) maximum

```
package com.javabykiran.basics;
 public class Lab12 {
    public static void main(String[] args) {
      int a = 5;
      int b = 2; int c = 1;
      if ((a > b) && (a > c)) {
         System.out.println("Min" +a);
      } else if (b > c) {
         System.out.println("Min" +b);
      } else {
          System.out.println("Min" + c);
          }
          if (a < b) {
             if (a < c) {
                System.out.println("Min:" +a);
             } else {
                System.out.println("Min" + c);
             }
          } else {
             if (b < c) { System.out.println("Min" +b);
             } else {
                System.out.println("Min" + c);
             }
       }
    }
 }
```

Practice Questions:

Q. Write a programme to read three numbers from command lines and find the maximum among three.

Q. Write a programme to read a number from the command line and check whether the given number is positive or negative.

Q. Write a programme to read a number from command line and display the corresponding day; the number is between 1 to 7 and if that number is out of range, display message for that, number is out of range.

Switch statement:

```
switch(expr) {
    case No1: s1;
        break;
    case Non: sn;
        break;
  default: s;
}
```

Example :

```
package com.javabykiran.basics;
public class Lab16 {
public static void main(String[] args){
   int a = Integer.parseInt(args[0]);
   switch (a) {
   case 1:
      System.out.println("FRI");
      break;
   case 2:
      System.out.println("MON");
      break;
   case 3:
      System.out.println("TUE");
      break;
   default:
      System.out.println("Invalid No");
      break;
   }
}
}
```

NOTE: The default statement can be written anywhere in the switch statement. But they have a break. In the last statement, it does not have break, but it is also possible.

Types of Looping control statement:

Looping statements execute statements over and over again, in a loop. It continues unt the condition result is true.

The types of looping control statements are:

- For statement
- While statement
- Do while statement

For statement

The for statement lets a code execute in a loop continuously.

Syntax :

```
for (initialization; condition; increment/ decrement) {
     s1; s2;
 }
for(       ;    ;  ) {    // possible empty
   s1; s2;
 }
```

Write a programme to read a number from the command line and display the numbe which divisible by five, from one to t h e given number in both order.

```
package com.javabykiran.basics;
public class Lab17 {
  public static void main(String[] args) {
  int n = 67;
  System.out.println("Forward order");
  for (int i = 1; i <= n; i++){
       if (i % 5 == 0) {
          System.out.println(i);
       }
    }
    System.out.println("Reverse order");
     for (int i = n; i >= 1; i--){
       if (i % 5 == 0) {
          System.out.println(i);
       }
    }
  }
}
```


---Output---

```
Forward order
5
10
15
20
25
30
35
40
45
50
55
60
65
Reverse order
65
60
55
50
45
40
35
30
25
20
15
10
5
```

Question: Write a programme to generate prime numbers from 1 to 100.

```
package com.javabykiran.basics;
/*
Prime Numbers Java Example:
This example shows how to generate prime numbers
between 1 and the given number, using a for
loop.*/
public class GeneratePrimeNumbersExample {
    public static void main(String[] args){
      // define limit
      int limit = 100;
```

```
    System.out.println("Prime numbers between 1 and " +limit);
     // loop through the numbers one by one
    for (int i = 1; i < 100; i++) {
        boolean isPrime = true;
     // check to see if the number is prime
    for (int j = 2; j < i; j++) {
        if (i % j == 0) {
           isPrime = false;
           break;
        }
      }
      // print the number
      if (isPrime)
         System.out.print(i + " ");
    }
  }
}
/*
* Output of Prime Numbers example would be Prime numbers
between 1 and 100 12
*3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83
89 97
*/
```

Question : Write a programme to read a two numbers from command lines and print all t prime numbers between given range.

```
package com.javabykiran.basics;
public class Lab19 {
  static Boolean isPrime(int n) {
  int count = 0;
     for (int i = 2; i <= n / 2; i++) {
     if (n % i == 0) {
           count++;
           break;
        }
     }
     if (count == 0) {
        return true;
     } else {
        return false;
     }
  }
```



```
  public static void main(String[] args) {
  int m = 3;
     int n = 30;
     for (int i = m; i <= n; i++) {
     boolean b = isPrime(i);
     if(b){
        System.out.println(i);
        }
     }
  }
}
```

---Output---

```
3
5
7
11
13
17
19
23
29
```

While statement

The while statement executes a single line continuously until the condition is met.

Syntax:

```
initialization ;
while(condition) {
  s1;
  s2;
Increment/decrement
}
```

Example :

```
package com.javabykiran.basics;
 public class Lab22 {
   public static void main(String[] as) {
       int n = Integer.parseInt(as[0]);
       //int n = 10;
       System.out.println("Forward order");
        int i = 1;
        while (i <= n) {
            if (i % 5 == 0) {
              System.out.println(i);
            }
           i++;
         }
        System.out.println("Reverse order");
           i = n;
        while (i >= 1) {
           if (i % 5 == 0) {
             System.out.println(i);
           }
           i--;
        }
    }
}
```

---Output---

```
Forward order
   5
   10
Reverse order
   10
   5
```

Write a programme to read the number from the command line and display the sum individual digits of the given number.

```
package com.javabykiran.basics;
public class Lab33 {
   public static void main(String[] args){
        int n = 123;
      // int n=Integer.parseInt(args [0]);
```



```
      int sum = 0;
  while (n != 0) {
     int r = n % 10;
     sum += r;
     n = n / 10;
   }
   System.out.println("Sum = " + sum);
 }
}
```

---Output---

```
Sum = 6
```

Do while statement

The do while statement executes a particular chunk of the code until a Boolean expression is true.

```
Initialization;
do {
  //processing s2;
  Increment/decrement
} while ( condition );
```

```
package com.javabykiran.basics;
 public class Lab20 {
     public static void main(String[] args) {
       int n = 30;
       System.out.println("while");
          int i = 1;
          while (i <= n)
             System.out.println("ok" + i);
           if (i% 5 == 0) {
               System.out.println(i);
          }
          i++;
            System.out.println("do_While");
          i = 1;
          do {
            if ((i % 5) == 0) {
```

```
            System.out.println(i);
            }
             i++;
                System.out.println("ok" +i);
        } while (i <= n);
    }
}
```

Enumerate the difference between While and Do While statement

While	Do While
In the case of While, the first condition will be verified and then the statement will be executed, Inside the while block	In the case of Do-While first statement, insid do block is executed and the condition is be verified
If the condition is false the first time, then the block of statement will be executed zero times	If the condition is false the first Time, then the block of statement will be executed once

Types of unconditional control statements:

- Break
- Continue
- Goto

Break

Break statement transfers the control unconditionally to the end of the block

Continue

The Continue statement transfers the control unconditionally to the beginning of the block

Example

```
package com.javabykiran.basics;
public class Lab23 {
   public static void main(String[] args) {
       int n=12;
         for(int i=1; i<n; i++) {
           if(i %2==0){
              System.out.println(i+" Next is continue");
           continue;
          }
```



```
        else if(i==9) {
            System.out.println("i is 9");
            break;
         }
            System.out.println("Ok "+i);
        }
    System.out.println("Done");
  }
}
```

---Output---

```
Ok 1
2 Next is continue
Ok 3
4 Next is continue
Ok 5
6 Next is continue
Ok 7
8 Next is continue
i is 9
Done
```

```
package com.javabykiran.basics;
public class Lab24 {

  public static void main(String[] args){
     /*
      * if(1) { } while(1) { -- }

      */
     int i = 1;
     /*
       *for(i<=3)
           sop(i);
              i++;          }          /* while(true) sop("ok");}
      */
     // System.out.println("Done");
  }
}
```

```
package com.javabykiran.basics;
public class Lab25 {
   public static void main(String[] args) {
      for (int i = 1; i <= 2; i++){
          JBK: for (int j = 1; j <= 2; j++) {
         for (int k = 1; k <= 2; k++){
                if (k <= 3)
                System.out.println(k + "\t");
                else
                     break JBK;
          } // end of k for
          System.out.println("JBK");
       } // end of j for
       System.out.println("*");
    } // end of i for
    System.out.println("Done");
  }
}
```

---Output---

```
1
2
JBK 1
2
JBK
* 1
2
JBK 1
2
JBK
* Done
```


Questions:

1. What is java and javac?
2. What is API?
3. Why does Java take two bytes of memory for store character?
4. Which syntax is followed by java for naming conversion?
5. Why is naming conversion used?
6. What is classloader?
7. State the difference between conditional and looping statement.
8. What is the difference between Path and ClassPath?
9. What is main aim of JIT Compiler?
10. How does Java have high performance?
11. Why do boolean data types take zero byte memory?
12. Why Java is simple and easy?

◆◆◆

Communication Between Two Classes

In java we can call the members of one class from another class by creating an object.
It is useful when we need to use common code in every class again and again.
This is called communication between two classes and can be done in more than one way. Yo can also say that these objects can talk to each other. One object sends a message to which th other object replies, or returns a value. They call each other's methods

Why do we need communication between classes?

Let's say we have two developers, 'Kiran' and 'Sachin'. Their manager asked Kiran t write code for a sum of two numbers, and sachin to write a code for avg.

Both of them started to write the code in their own ways. Kiran wrote as below :

```
package com.javabykiran.CommBetClasses;
   /**
    * @author Kiran
   */
   public class SumLogic {
        int sum(int a, int b) {
           return a + b;
        }
}
```

Sachin wrote as below:

```
package com.javabykiran.CommBetClasses;
   /**
    * @author Sachin
   */
public class AvgLogic {
     int avg(int a, int b) {
          int sum = a + b;
          return sum / 2;
      }
}
```

Now, when they submit their code to the manager, he will say that Kiran wrote the code prope but not Sachin. Why that is so?
The reason is that Sachin wrote an additional logic again even though Kiran had written it earli

Hence, it was a waste of time since no reusability feature is used. We can't write logic twice. And since it had already been written by Kiran, Sachin should have used the former's logic instead of repeating it.

Now Sachin rewrites his code as shown below:

```
package com.javabykiran.CommBetClasses;
/**
 * @author Sachin
 */
public class AvgLogic {
   int avg(int a, int b){
      // creation of object
       SumLogic sl = new SumLogic();
       int sum = sl.sum(a, b); // reusability
       return sum / 2;
   }
}
```

In the industry, a single person cannot write the whole logic.

Because it is a mutual effort, we always have to reuse functionalities of each other. I will explain step by step what has happened here.

SumLogic sl = new SumLogic();

Here we have created an object of class SumLogic. It means that we are trying to load SumLogic class by using a keyword called 'new'.
The memory stores all members of class SumLogic [method sum] and sl knows about address of that location.
We can say that sl is eligible to call every thing [members like sum] of SumLogic class. In java, to call members by address we use the dot operator.
Like sl.sum(), we need to take care while doing this and the parameters should match the method, not just the name of method. In this case,int a and int b. See below for reference.

sl.sum (34, 67);

The parameter sequence also matters if we write

sl.sum (34,"java");

it is wrong because the second parameter must be type of 'int', we can't specify String here. It will show the error, "sum is not found in class SumLogic".
sum( ) method of SumLogic class returns int. Now, its class AvgLogic has a choice whether to take that back or not. Even if we write

sl.sum (34, 67); // it's correct.
int xx= sl.sum(34,67);

In the above case,we get the output of the method sum and stored in the xx variable.We can do anything with this variable after that, including addition,subtraction and any arithmetic operation.

```
int xx= s1.sum(34,67);
int y = xx+89;  // possible
```

Now we explore a little more complex scenario.

```
A m1(){
   return new A();
 }
```

In this case, m1 want to return the instance of class A so we must give type of A in return. Consider

```
int a=90;     in this case we can say that [a is variable of type int whose value is 90]
  A a=new A();      [a is type of A which has the value new A()]
   int m1(){
      return 40;  // correct; because we return type of int.
   }
  A m1(){
     return new A();  // This is also correct as we are returning value as a type of A
  }
```

So, calling always takes care of Type and Type may be any primitives or classes. [int, float or A, Student or Employee]

Consider and study the example below :

```
package com.javabykiran.CommBetClasses;
/**
* @author Kiran
*/
public class B {
        void m6() {
           System.out.println("I am in B-m6");
        }
}
```

```
package com.javabykiran.CommBetClasses;
/**
* @author Kiran
*/
public class A {
```



```
      B m1() {
          System.out.println("I am in A- m1");
          // return new B(); // OR single step B b = new B();
          return b; // This is two steps
       }
public static void main(String[] args){
       A a = new A(); // a is eligible to call
                      //everything of A
        / *  SCENARIO 1 */

       a.m1(); // return B but we may not want to
              // take it back

       /* SCENARIO 2 */

       B b=a.m1(); // we received B . Now b is eligible
              // to call everything of B
       b.m6(); // OK, we get an output of m6 of class B
     }
  }
```

---Output---

```
I am in A- m1
I am in A- m1
I am in B-m6
```

Example

```
package com.javabykiran;
   public class A {
        int a = 10;
        String web = "www.javabykiran.com";
      void showWeb(){
          System.out.println(web);
       }
  }
package com.javabykiran;
 public class B {
    public static void main(String[]args){
       A a = new A();
       System.out.println(a.a);
```

```
        a.showWeb();
    }
}
```

---Output---

```
10
www.javabykiran.com
```

What is System.out.println()?

Here we break down the meaning in three points for you:

- System is class predefined by Sun Microsystem (now Oracle)
- out is the variable declared in System class of type PrintStream which is static
- println() is the method defined in PrintStream class

See the demo below:

```
package com.javabykiran.sop;
public class System {
    static PrintStream out =new PrintStream();
}
package com.javabykiran.sop;
  import java.lang.System;
  public class PrintStream {
       public void println(){
          System.out.println("in a prinln()..");
       }
   }



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

```
package com.javabykiran.sop;
public class Test {
     public static void main(String[] args){
       //We can also do
       System s=new System();
       PrintStream ps=s.out;
```



```
  ps.println();
     /* Also as seen below, which we use daily
        while programming
  Why direct System.out-this is due to static key word  which
will be covered in later chapters */
```

---Output---

```
in a prinln()..
```

3 CHAPTER

Package

What Is a Package?

A package in java is a mechanism to contain classes, sub packages or interfaces. They are used to prevent naming conflicts and for controlling access in addition to make searches and utilisation of classes, enumerations, interfaces and annotations easier. It makes things easier for other programmers as they can easily locate related classes.

Why is a package required?

- When we write a code in a file and place it in some folder, we can have any number of files in it. Now imagine that a code is released to production and some issues occurred that need to be fixed. For this we need to find the java class in that folder where we have lot of files, which may take time to locate and further act on the bugs. If we want the code to be organised and maintained, we must have a proper folder structure, which is possible through the use of packages
- Maintenance - If any developer has newly joined a company, he can easily reach the files needed
- Reusability –A common code can be placed in a common folder so that everybody can check that folder and use it
- The Package concept uses two keywords: package and import

Syntax for Package:

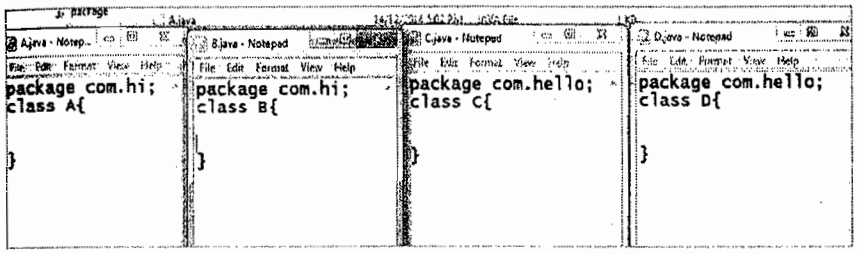
- The syntax for package is:
  - package com.tcs.icici.loan.homeloan.emi.penalty
  - package com.tcs.icici.loan.homeloan.emi.advance
  - package com.tcs.icici.loan.carloan.emi.advance

  In the above example, the folder starts with 'com'. This is generally a company specification. In this case, the root folder is 'com'
- Then we have subfolder 'tcs'.This is the name of the company in which product is developed
- icici is the client name for which we are developing our product.
- loan is project name
- home loan and car loan are loan types, and so on
- Let it be noted that we can give any names, this is just a specification. If we write Package com.hi.hello, it means 'com', 'hi' and 'hello' is the folder structure
- Keep in mind that the root folder should always be same for all classes

Steps to Create, Compile and Execute package

Write classes as below:

- Now, we have multiple classes as shown in above diagram, 4 classes A, B, C and D



- I am now modifying class A as below, so that we can observe how to run a program with packages

```
package com.hi;
class A{
  public static void main(String as[]){
  System.out.println ("javabykiran is good institute !!!!! ");
  }
}
```

- Now we will try to go inside package com then type hi, and try to run it as seen below
- The correct way is as follows: first, go back to before the root folder
  i.e. package will run like this:

Remember

- Package statement can be written at the start of the file only once
- Start with root folder and ends with sub-directory name only, not the file name
- Always compile with –d option. If you are compile from command prompt.
- javac -d .* .java

  Here-
  - -d means directory
  - . [dot] means current directory where you want to place com folder after compilation
  - *.java means all java files will get compiled.

Import

When we want to use one class in another class we must use the import statement. Here, you don't need to refer to the whole package name, but simply refer to classes that are declared in different packages.

After modifying class A from above [added a line C c=new C()]

```
package com.hi;
 public class A{
   public static void main(String args[]){
       System.out.println("javabykiran is a good institute");
       //Try to access class C here even though it's in
       //another folder
       // (class C is in hello folder)
      com.hello.C c=new com.hello.C();
   }
}
```

To give java class file name, always find the root folder first and then go down to the class file. In this case, to write 'C' class name first find class A's root folder that is com, then traverse to required Class file that is C here.

We must write the following:

com. hello.C c=new com.hello.C();

Instead of, this we can use import the code looks like this:

```
package com.hi;
import com.hello.C;
 public class A{
    public static void main(String[] args){
      System.out.println("javabykiran is a good institute");
      //try to access class C here even though it's in another
      //folder
     //(class C is in hello folder)
        C c = new C()
   }
}
```

Here, it's been seen that the readability of code decreases if we write a long path of class in code. So, what Java says is, go for an import statement like given in the example below. It's one and the same thing, but we should remember both ways.

Thus, to maintain readability of code, Java provides another way i.e. import keyword.

The Code will look like this:

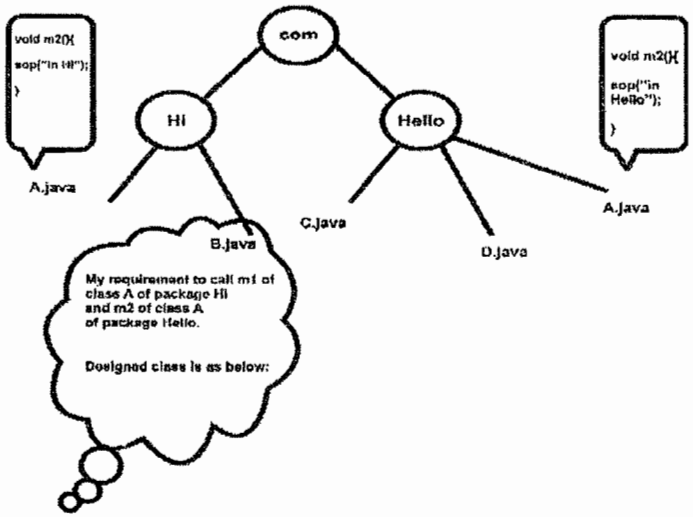
```
package  com.hi;
import com.hello.C;

class A{
public static void main(String args[]){
  System.out.println("javabykiranis a good institute!!!!!");
   //Try to use C class here even though it's in another
   //folder,hello
   // The path is removed now C c=new C();
  }
}
```

When you compile it, it will surely compile. It's same as earlier example using dot, but the readability of the code is increased. If you want to compile it all, then mark all classes as public.

- **You may be asked by interviewer why are you using two ways? And why can't you do everything by a simple import statement. Have you seen any scenario where com.hello.C cc=new com.hello.C () is used?**

Answer:

Take a look at this scenario:



In this case we will try to design class B

```
package com.hi;
import com.hello.A;
public class B { // Line no 1
     void m8() // Line no 2
     {
          A a = new A();  // Line no 3
          a.m1();  // Line no 4
          A a1 = new A();  // Line no 5
          a1.m2(); //  Line no 6
     }
}
```

Will the above code compile?

No, because line number three does not say anything about class A from which package (hi or hello) it is being referred from.

No, because line number five also does not say anything about class A from which package (hi or hello) it is being referred.

No, because line numbers four and six will get confused about which package's class needs to pick here. Therefore, we now have a confused compiler. So, there are many reasons for this.

In this case import is not working, so we will go for a different method, way dot way i.e. using the qualified/full fledge path.

```
package com.hi; // import removed here class
class B{        // Line no 1
void m8() {     // Line no 2
A a=new A();    // Line no 3
   //keep as it is so that it will be from same package 'hi'
a.m1();         // Line no 4
com.hello.A  a1 = new com.hello.A();//Line no 5
//here it directly goes to hello package and call
a1.m2(); // Line no 6
}
}
```

Remember these points while using import statement in compiler perspective:

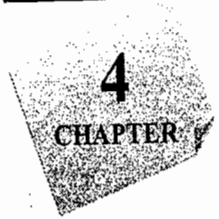
- Import can be written multiple times after the package statement and before the class statement
- It must be start with the root folder name [No subfolder name] and the file name must end with semicolon
- import com.hi.A; -- >correct

import com.hi.*; -- > correct, all files from hi folder imported
import com.hi; -- > wrong, it gives the error that hi file not found in com folder

- **import** does not mean that the memory is allocated, it just gives a path to reach, many factors which later decide whether you can create object or not
- **import com.hi.A;** is always better than **import com.hi.*;**
  In industry, the code reviewer may reject your code if we write
  import com.hi.*;

Questions:

1. What are the advantages of java package?
2. Define packages in Java?
3. Why are packages used?
4. Which package is always imported by default?
5. Can I import the same package/class twice? Will the JVM load the package twice at runtime?
6. Explain the usage of Java packages.
7. What is the base class of all classes?
8. What do you think is the logic behind having a single base class for all classes?
9. Why is most of the Thread functionality specified in Object Class?
10. What is package in Java? Explain in detail.
11. How to create package in Java.
12. How to use package in Java.
13. What are the best practices while using package in Java program?
14. What is a predefined package in Java?
15. What are the things you should know about Package in Java?
16. Does importing a package also import the sub packages as well? Example: Does importing com.bob.* also import com.bob.code.*?



Methods, Variable and Block

A method is something like a subprogram, which when invoked returns a value. I other words, a method is a collection of words or commands that joined to perform function. Once this function or task is completed, it sends a result back to the caller. It can even complete an operation without returning a value, and allows us to use th same code without actually having to type the entire code again and again. Methods ar also known as instance members.

Methods :

Syntax :-

```
[Modifiers] return_type Method_name ( Arguments1, array2---)
{
   ---
}
```

class can contain two types of methods:

- Instance methods
- Static method

Example

```
package com.constructor;
 public class StaticInstance {
   int a= 10;
   static int b=20;   //static variable
      void jbk() {
         System.out.println("Hello JBK");
         System.out.println(a);
         System.out.println(b);
      }
      static void kiran()
      {
         System.out.println("Hello JBK");
         System.out.println(b);
      }
```



```
public static void main(String[] args){
   //by creating object, both variables are printed
         StaticInstance staticInstance=new StaticInstance();
         System.out.println(staticInstance.a);
         System.out.println(staticInstance.b);
   }
}
```

Instance method	Static method
When you define a method inside a class without a static keyword, it is called the instance method or the non-static method	When you define a method with static keyword then it is called the static method
Instance methods must be called by you explicitly. You can call using reference variables which contain the object. For example: (Hello h=new Hello())	Static methods must be called by you explicitly. You can call using the following methods: a) With classname b) With reference variable without initialisation. For example: A a=null;. c) With reference variable which contains object

Variables:

There are two types of variables, local and global. Read the following to get more details about them.

- **Global Variables**
  - Global variables are initialised automatically
  - It is defined outside of method
  - A static keyword can be applied to global variable
  - Its scope is across class and can be used anywhere

Example

```
class A{
      int a;
      A a1;
      A a2 = new A();
 void m1(){
      System.out.println(a) ;     //prints 0
      System.out.println(a1) ;  //prints null as every class is
      // initialised with null
      System.out.println(a2);  //prints address of a2
    }
}
```

- **Local Variables**

- Variables declared inside method or constructor or any block are called local variables
- The scope of the local variable is within the method or constructor or block where it is defined
- Memory will be allocated for the local variable whenever the enclosing block gets executed
- Local variable is not equivalent to instance variable
- Local variables cannot be static
- Local variables cannot be referenced with class name or reference variable
- Local variables will not be initialised by the JVM automatically, like instance and static variable
- If you use local variables without initialising, you get compile time errors like 'variable x might not have been initialised'.
- Local variables can be final, primitive or reference

```
package com.javabykiran.Constructor;
public class AA {
   static int ctr = 0;
   int i = 100;
     {          //This is block
       System.out.println("Before change in local block");
       System.out.println("ctr = " + ctr);
       System.out.println("i = " + i);
       int ctr = 2, i = 200;
       System.out.println("");
       System.out.println("After change in local block");
       System.out.println("ctr = " + ctr);
       System.out.println("i = " + i);
     }
      void display2() {
       System.out.println("In another method");
       System.out.println("ctr = " + ctr);
       System.out.println("i = " + i);
      }
  public static void main(String args[]) {
         AA sObj = new AA();
         System.out.println(" ");
         sObj.display2();
   } }
```

Local Block

- Block defined inside a method or block or constructor is called a local block
- A local block will be executed whenever the enclosing methods or constructor or block is executed
- Local blocks are not equivalent to instance block
- Local blocks cannot be static
- You can write local blocks inside a method or constructor or block, and it can be nested

Example

```
package com.javabykiran.Constructor;
public class Hello {
  static int a = 31;
  static int b;
   static void math(int x) {
       System.out.println("x = " + x);
       System.out.println("a = " + a);
       System.out.println("b = " + b);
   }
   static {
       System.out.println("Static block initialised.");
       b = a * 4;
   }
  public static void main(String args[]) {
       System.out.println("Before static method b = " + b);
       math(42);
   }
}
```

---Output---

```
Static block initialised.
Before static method b = 124
x = 42
a = 31
b = 124
```

◆◆◆

Encapsulation

This is one of most important principles of java. Encapsulation means binding of data into single entity. It is a process of encasing the code and data to form one unit. To do this, you mus make all data members of that class private.

Encapsulation is useful when the programmer or the client wants to hide information details o protect some data. It hides the fields within the class, and makes their access private.

If we look at class, we can see that it can have properties and operations, and in java propertie mean variables and operations mean methods.

```
class A{
// variables a, b, c ...
//methods m1 m2 m3....
}
```

In the above given example, class A is an entity which binds variables and methods. So we can conclude that java supports encapsulation by default.

Why is encapsulation needed?

Let's see, we have a class Student which has a property, age(i.e. variable)

```
public class Student {
  int age;
  }
The clients treat this class as below:
 ----- Client1 -----
Student s1=new Student ();
s1.age= 30;
 ----- Client2 -----
Student s1=new Student ();
s1.age= -45  //negative number
```

Now functionally, can Student's age be **negative? No,** but we have allowed our clients to assig negative values.

He must not allow people to add age as a negative. Here, client 2 is correct but class Student i wrong functionally. So, it is rejected during the code review as per company standards, and th manager will say to person who designed class Student -"Hey, you have not encapsulated you class properly. People are putting invalid values. Please encapsulate properly."

Here is how he changes the code:

```
package com.javabykiran.Encapsulation;
public class Student {
   private int age ;
   public void setAge (int ageStu) {
   if (ageStu> 0) {
           age = ageStu;
      } else {
          age = 0;
       }
      }
}
```

In this case,

Client 2 : Student s1=new Student ();

s1.age= - 45 // as age is private and therefore, not possible

- Not possible as age is private
- Client must go through method setAge(); in this method, the developer will send age not through variable but through the method where we put logic. Then, even if negative value is passed, the assigned value would be Zero, not negative. And that is ok
- Student s1=new Student();
- s1.setAge (-70); //s1.age= - 45 // in RAM, zero will always get stored
- Now we can say that class Student is properly encapsulated
- Encapsulation is very important if we want to design classes properly as per functionality. What we learn here is to always keep global variables private, so that nobody can assign wrong values, or allow others to assign values through public methods.

To know encapsulation completely we must know access specifiers in detail.

Questions:

1. What is Encapsulation?
2. What is the primary benefit of Encapsulation?
3. What is the difference between Encapsulation and Abstraction?
4. What is the difference between Encapsulation and Data Hiding?
5. What are the features of Encapsulation?
6. What is Encapsulation in Java? Explain in detail.
7. What are the examples of Encapsulation in Java?
8. What is the advantage of Encapsulation in Java and OOPS?
9. State the important points about Encapsulation in Java.
10. Write about the design pattern based on Encapsulation in Java.
11. What are the Benefits of Encapsulation?
12. Give example of how to achieve Encapsulation in Java.
13. How can variables of the EncapTestclass be accessed?

6 CHAPTER

Access Modifiers

Java Access Specifiers (also known as Visibility Specifiers) regulate access to classes, fields and methods in Java. They can also be called Access Modifiers. They are usually keywords that determine the accessibility of class, fields and methods in a program. Here the class has control over what data or information is accessed by other classes.

These Specifiers determine whether a field or method in a class can be used or invoked by another method in another class or sub-class. Access Specifiers can also be used to restrict access. Access Specifiers are an integral part of object-oriented programming. In basic terms, they curb access.

There are four Access Specifiers in Java:

- private
- default
- protected
- public

public:

- It can be applied to constructor, global variables, static variables, methods, inner classes or outer classes
- Public members are accessible everywhere, in same class, outside the class, or any package
- Local variables cannot be public as they already have a scope within the method

private:

- A class cannot be private
- Private access specifier can be applied to global variable, method, constructor and inner class only
- Local variables cannot be private but global variables can be private
- Private members can be accessed only within the enclosed brackets
  - The program below will give the error ;
  - Here, we see that the variable is private and not accessible in other classes


```
package com.javabykiram.encapsulation.accessSpecifier;
class A1 {
    private int var = 90;
       void testVar() {
            A1 a = new A1 ();
            System.out.println(a.var); //No Error
       }
}
```

```
class B {
     void testVar() {
            A1 a = new A1 ();
           //System.out.println(a.var);
           //the field a.var is not visible as it is private

      }
}
```

- If we make constructor private, we cannot create an object of that class from other classes
- Question that may be asked - How you will stop others from creating object of your class? The answer would be: By making constructor private.[this will be more clear in constructor chapter]
- In the below given example we may have confusion that
  - private A a1=new A();
  - This statement means a1 address is not visible outside, hence, we are not able to access it from outside
  - Remember, there will be no impact to class A
  - We can't say that class A is private. By this line it's just a variable 'a1' is private

Example

```
package com.javabykiran.Encapsulation.accessSpecifier;
class A {
  int jbk = 90;
}
class VarEx {
      public A a = new A();
      private A a1 = new A();
  }
class TestVarEx {
     public static void main(String[] args) {
        VarEx varex = new VarEx();
```

```
        System.out.println(varex.a.jbk);
       //System.out.println(varex.a1.jbk);
        // wrong, a1 is private
     }
}
```

Question: Why should you know private variable? Where have you used in your project?

Answer: Let's say we have the following requirement:

In a college we want to insert student details and one student insertion means that is:

i. Qualification should be inserted
ii. Personal details should be inserted
iii. Skills should be inserted
iv. Fee details should be inserted

Now we will start writing a simple class where the above specified requirements will be fulfilled

First approach:

```
package com.javabykiran.Encapsulation.accessSpecifier;
   public class Student {
      public void insertStudent() {
             // some code for Qual
               // logic for Qual insertion
            // some code for Personal Detail
               // logic for Personal Detail insertion
           // some code for Skills
              // logic for Skills insertion
          // some code for Fees
             // logic for Fees insertion
      }
   }
```

This approach will **not be accepted** as all code written is at one place. If, in the future, we want to make fees optional, it won't be possible and we will need to change whole code which increases unit testing. The above written code will work properly at the first instance, but if we think for the future, then it will be difficult to manage it.

So, we need to change this code as shown below again:

Second approach:

```
package com.javabykiran.Encapsulation.aceessspecifier;
public class Student {
      public void insertStudent() {
            insertQual();
            insertPD();
            insertSkill();
            insertFees();
      }
       public void insertQual()
      {
            //------
             System.out.println("Qualification");
            //------
      }
      public void insertPD()
      {
            //------
             System.out.println("Personal Details");
            //------
      }
      public void insertSkill()
      {
            //------
             System.out.println("Skill Details");
            //------
      }
      public void insertFees()
      {
            //------
             System.out.println("Fees Details");
            //------
    }
}
```

In this approach, we just tried to bring more readability and modularity in our code. That means we will have less maintenance. This is just code writing in a different way. It is just copy and pasting code at different places. Now, consider that after some years we want to remove skill insertion. We will then need to comment only one line.

```
public void insertStudent() {
    insertQual();
    insertPD();
    // insertSkill();
    insertFees();
     }
```

Still, our code gets rejected as requirement is not completely fulfilled. HOW??

The client can call any methods directly from outside and can insert anything like fees, etc.

But requirement says that all methods should get called so that student will get inserted with all details not just a single or partial detail.

Third approach:

```
package com.javabykiran.Encapsulation.aceessspecifier;
public class Student {
    public void insertStudent()
    {
        insertQual();
        insertPD();
        insertSkill();
        insertFees();
    }
    //all below methods are private
    private void insertQual()
    {
        //------
        System.out.println("Qualification");
        //------
    }
    private void insertPD()
    {
        //------
        System.out.println("Personal Details");
        //------
    }
    private void insertSkill()
    {
        //------
        System.out.println("Skill Details");
        //------
    }
    private void insertFees()
    {
        //------
        System.out.println("Fees Details");
        //------
    }
}
```

This is the most correct approach as all other methods are private and only one method is public which can be accessed from outside.

So, nobody can call any method directly without going through insertStudent method. That's what the requirement says.

Here, it must be noticed how access specifiers play an important role.

If we use them properly, our code will be more managed in the long run.

default:

- When an access specifier is not specified to members of a class, then it is called default
- Default members can be accessible only within same package or folder
- Default can be applied to constructor, global variable, local variable, methods, inner classes, outer classes etc.
- Default is a keyword in java. We may mistakenly think that it is an access specifier, but it actually is for a switch statement.

protected:

- It can be applied to constructor, global variable, methods
- It cannot apply to outer classes, but can be applied to inner classes
- It cannot apply to the local variable
- Protected members are accessible within same package and another package of its subclass only. Inheritance must be there, the caller of protected members is the subclass of protected member's class.

If any method is overridden from super class to sub class, then the access specifier of the overridden method must be protected or public.

To know this completely, we must know inheritance in detail,please go through inheritance chapter

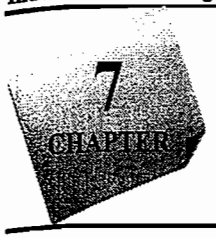
default/No Modifier:

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package sub-class	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclasses	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Questions:

1. Which access specifier can be used with Class?
2. Can we reduce the visibility of the inherited or overridden method?
3. What is the difference between Public, Private, Default and Protected?
4. What will happen if we make the constructor private?
5. Can we instantiate the object of derived class if theparent constructor isprotected?
6. Can we declare an abstract methodprivate?
7. What is Access Specifier?

◆◆◆



Static

Things to remember:

- Static is a keyword used for memory management
- Static means single copy storage for variable or method
- Static keyword can be applied to variables, methods, inner class and blocks
- Static members belongs to class rather than instance of class

Static in java is a keyword that indicates that the variables or functions are shared between all the instances of a particular class since it belongs to the type, not the object. It is used when the programmer wants to share the same variable or method of a class.

Static Variable:

- The variable preceded by 'static' keyword is 'static variable'

```
static int a=10;  //variable
    static void m1() {   // method
    }
```

- Static variable is used to refer common property of all objects of class
  - **How to access static variable?**

    There are two ways to access static variable:

    i) Static variable can be accessed by Class name

    A. a ; [A is class name]

    Where A is the class name and 'a' is a static variable declared in that class

    ii) Static variable can be accessed by object

    I have a class name called 'Sample'. Now, we can create the object of the Sample class

```
Sample h=new Sample ();
System.out.println(h.a);    //'a' is static variable inside 'sample' class
```

  - **How can I access static variable in two ways?**

See in the following program:

```
package com.javabykiran.Static;
/*
 * @author Java By Kiran
 */
public class Staticvar {
  static int i=10;
  public static void main(String[] args) {
     Staticvar s =new Staticvar();
     System.out.println(s.i); //Not Recommended
     System.out.println(Staticvar.i); //Recommended
  }
}
```

---Output---

```
10
10
```

- In the above program, we printed the value of variable by using object name and by using class name
- Static variable gets loaded into the memory at the time of class loading
- So, we can access static variable by reference variables as well.
- In the above program, if we create only the reference of class like Staticvar s1=null;
  System.out.println(s1.i); // possible
  System.out.println(Staticvar.i); //possible

The above example compiles and executes successfully because the static variable get loaded into the memory at the time of class loading

- Static variable and method doesn't belong to Object/Instance of the class since static variables are shared across all the instances of Object

Example 1:

```
package com.jbk;
 /*
 * @author Java By Kiran
 */
public class StaticVar_Demo{
      int a =10;
      static int b =10;
   public static void main(String[]args){
      StaticVar_Demo st= new StaticVar_Demo();
```



```
      System.out.println(st.a);
      System.out.println(st.b);
      StaticVar_Demo st1 = new StaticVar_Demo();
      int x = st1.a++;
      System.out.println(x);
      int y = st1.b++;
      System.out.println(y);
      StaticVar_Demo st2 = new StaticVar_Demo();
      int p = st2.a++;
      System.out.println(p);
      int q = st2.b++;
      System.out.println(q);
      StaticVar_Demo st3 = new StaticVar_Demo();
      int c = st3.a++;
      System.out.println(c);
      int d = st3.b++;
      System.out.println(d);
  }
}
```

---Output---

```
10
10
10
10
10
11
10
12
```

Example 2:

```
package com.javabykiran.Static;
/*
 * @author Java By Kiran
 */
public class Staticvar {
      static int i=10;
      int b=20;
    void m1(){
      System.out.println(i);
```

```
    }
    static void m2(){
       System.out.println(i);
    }
 public static void main(String[] args) {
      Staticvar s =new Staticvar();
       s.m1();
       Staticvar s1=new Staticvar();
       s1.m2();
 }
}
```

---Output---

```
10
10
```

- Local variables cannot be declared as static else the compiler displays a modifier error [a compile time]
- We cannot call non-static members from static members because static variables g stored into memory before object creation and non-static member get stored into memor after object creation
- So, when we access a non-static member through a static member, it leads to a compi time error as they are not present in the memory.

```
package com.javabykiran.Static;
  /*
  * @author Java By Kiran
  */
public class Staticvar {
      static int i = 10;  // static variable
       int b = 20;         // non static variable
    void display() {       // non static method
       static int a = 50;  //here a compile time error shows
       System.out.println(i);
    }
    static void show() {       // static method
       System.out.println(i); // OK
       System.out.println(b);
   //if I want access non static variable in static method,
  //compiler throw error
    }
```



```
public static void main(String[] args) {
    Staticvar s1 = new Staticvar();

    s1.show();
    s1.display();
  }
}
```

Static method:

- If you apply static keyword with any method, it will be treated as static method
- Static method belongs to a class rather than object of a class
- Static method can be invoked without creating instance of the class
- Static method can access static data member and can change its value
- Static methods also load into memory before object creation

```
package com.javabykiran.Static;
/*
 * @author Java By Kiran
 */
public class Staticvar {
      static int i=10;
      int b=20;
    void display(){
        System.out.println(i);
    }
    static void show(){  //static method
       System.out.println(i);
       System.out.println(b);//if we want access non static
       //variable in static method, compiler throw error
    }
 public static void main(String[] args){
    Staticvar s1=new Staticvar();
    s1.show();
    s1.display();
 }
}
```

- Static method can be accessed by nullable reference like

```
Staticvar s1=null;
s1.show();
```

- By using non-static member we can access static members, but by using static member we cannot access non-static members

Example

```
package com.javabykiran.Static;
/*
 * @author Java By Kiran
 */
public class StaticMethod {
    static int i=10; //static variable
    int b=20;        //global variable
  void display() {
    System.out.println("This is display method");
  }
  static void show() { //static method
    System.out.println("Hello,This is JAVA BY KIRAN classes");
  }
  public static void main(String[] args) {
    StaticMethod s =new StaticMethod();
      s.display();
      s.show();
      StaticMethod s1 = null;
      s1.show();
  }
}
```

---Output---

```
This is display method
Hello, This is JAVA BY KIRAN classes
Hello, This is JAVA BY KIRAN classes
```

Static Block:

- Java's static block is the group of statements that gets executed when the class is loaded into memory by Java ClassLoader.
- It is used to initialize static variables of the class. Mostly it's used to create static resources when the class is loaded
- We can't access non-static variables in static block
- We can have multiple static blocks in a class, although it doesn't make much sense
- Static block code is executed only once when the class isloaded into memory

- Static block always get executed first because they get stored into the memory at the time of class loading or before object creation

```
package Static;
public class StaticBolckLab2 {
    StaticBolckLab2(){
      System.out.println("This is constructor");
    }
    {
      System.out.println("This is Non static Block");
    }
  static{
     System.out.println("This is static Block");
   }
 public static void main(String[] args) {
    StaticBolckLab2 s= new StaticBolckLab2();
    StaticBolckLab2 s1=new StaticBolckLab2();
 }
```

---Output---

```
This is static Block
This is Non staticBlock
This is constructor
This is Non static Block
This is constructor
```

In the above example, we can prove that static members have one copy storage

- A static block cannot access non-static variables and methods

```
package com.javabykiran.Static;
/*
 * @author Java By Kiran
 */
public class StaticBlock {
    static int i=10;  //static variable
    int b=20;     //global variable
     void display(){
        System.out.println("This is display method");
     }
     static void show(){  //static method
        System.out.println("JAVA BY KIRAN classes Pune");
        //System.out.println(b);
```

```
        //if we want access variable 'b',
        //non-static method is needed

      //if there is a variable in the static method, JVM
       //throws a compile time error
      }
      {
         display();
      }
      static{
         display();   //this is a compile time error,
                      //we cannot call non-static meth-
         System.out.println(i);
         System.out.println(" "+b);  //this compile time
                      //error shows we cannot access non-static
                      //variable in static block

      }
  public static void main(String[] args) {
        StaticMethod s =new StaticMethod();
        s.display();
        s.show();
        StaticMethod s1 = null;
        s1.show();
    }
  }
```

Static Inner Class:

- Outer class cannot be declared as static
- But inner classes can be used as static

```
package com.javabykiran.Static;
/*
 * @author Java By Kiran
 */
public class StaticInnerclass {
    static class A {  //start of static inner class
    static int a=10;
    int b=20;
   void display(){
       System.out.println("This is local method");
    }
    static void show(){
        int c=63;
        System.out.println("This is static method");
```



```
    }
  }
public static void main(String[] args){
       A a =new A();
       System.out.println(a.b);
       System.out.println(a.a);
       a.display();
       a.show();
    } }
```

---Output---

```
20
10
This is local method
This is static method
```

- Inner class can access their local variables and methods

Example

```
package com.javabykiran.Static;
/*
 * @author Java By Kiran
 */
class StaticClassDemo {
   static class JBKway{  //start of inner class
     static int a=10;
     int b=20;
     void display(){
         System.out.println("This is local method");
     }
     static void show(){
          int c=63;
          System.out.println("This is static method");
          System.out.println("C="+c);
     }
 }
 public static void main(String[]args) {
      JBKway a1 =null;
      //System.out.println(a1.b);//runtime error
      System.out.println(a1.a);
     // a1.display(); //runtime error
       a1.show();
     System.out.println(a1.a);
```

```
    }
}
```

---Output---

```
10
This is static method
C=63
```

Note:

- Local variable cannot be static
- Constructor cannot be static
- Outer class cannot be static

Questions:

1. What is static in java?
2. Why do we use static keyword in java?
3. What is static variable in java?
4. What is static method in java? Give examples.
5. What is static block in java?
6. What is the need of staticblock?
7. Why is the main method static in java?
8. Can we overload static methods in java?
9. Why main method in static?
10. Can we override static methods in java?
11. Can we write static public void main (String[]args)?
12. Can we Overload or Override static methods in java?

◆◆◆

Final

- Final keyword can be applied to variable, method and class. final member cannot be changed
- A variable that is declared as final and not initialized is called a blank final variable. A blank final variable forces the constructors to initialize it
- Java classes declared as final cannot be extended. Restricting inheritance
- Final parameters - values of the parameters cannot be changed after initialization.
- Final method – can not be overridden.

A visible advantage of declaring a java variable as static final is that the compiled java class results in faster performance.

Final is a 'non-access modifier' which can only be applied to a variable, method or class. It is basically unchangeable, as in , its value once assigned cannot be changed. The final keyword may be used in various different scenarios.

Example: Final variable can not be changed

```
package com.final;
 public class {
   final FinalEx
   final int a=10;
   final int i=0;
      for(i=0;i<5;i++) { //compile time error
       //final variable's value can't be change
       System.out.println("value of i="+i);
      }
public static void main(String[] args) {
    FinalEx finalEx=new FinalEx();
    finalEx.JBK();
  }
}
```

Example: Final method cannot be overridden.

```
Package com.javabykiran.Final;
Public class Final {
   FinalEx {
                                    constructor of
```

```
    }
   final int a = 100;
    final void show() {
      System.out.println (a);
   }
   void welcome() {
   System.out.println("Welcome to java by kiran,Pune");
   }
}
```

```
public class FinalEx1Test extends FinalEx1{
 void show(){
   //compile time error because method cannot override
         System.out.println("This is method of FinalEx1Test");
      }
public static void main(String[] args) {
   FinalEx1 finalEx1=new FinalEx1();
   finalEx1.show();
  }
}
```

Example :The final class cannot be extended

```
package FinalEx;
 final class FinalClass {
    void KiranShow(){
       System.out.println("BY Kiran's way final class cannot
                                          Be Inherited");
    }
}
```

```
package FinalEx;
  public class FinalClassTest extends FinalClass{
    //Here compile time error is shown because the final class
   //cannot be extended
    public static void main(String[] args) {
         FinalClass finalclass=new FinalClass();
         finalclass.KiranShow();
  }
}
```

Note:

- Constructor cannot be final
- Block cannot be final

Some more examples:

```
package com.javabykiran.Final;
public class A {
   final B b; //ERROR; as it is not initialised
  }

class B {

 }
```

```
package com.javabykiran.Final;
 public class A {
   final B b = new B();
   void m1() {
       b = new B(); // ERROR as we are trying to change the
                    //address
   }
 }

class B {

 }
```

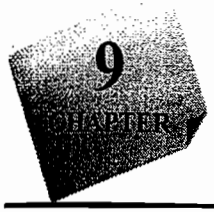
```
package com.javabykiran.Final;
public class A {
  final B b; // NO ERROR as we have done the initialisation in
  //constructor
  final int x;
  A(){
     b =new B();
     x=28;
  }
}

class B {

 }
```

```
package com.javabykiran.Final;
public class A {
    final int a = 90;
    int b = 90;
}
class B {
 void m1() {
    A a = new A();
    a.a=98;  //ERROR -even from another class we cannot change
    a.b=999; //NO ERROR-variable is not final,therefore we
             //can change
  }
}
```

Questions:

1. What is the use of final keyword in java?
2. What is the blank final field?
3. Can we change the state of an object to which a final reference variable is pointing towards?
4. What is the main difference between abstract methods and final methods?
5. What is the use of final class?
6. Where all can we initialise a final non-static global variable if it is not initialized at the time of declaration?
7. What is final class, final method and final variable?
8. Where all we can initialize a final static global variable if it is not initialized at the time of declaration?
9. Can we use non-final local variables inside a local inner class?
10. Can we declare constructors as final?

◆◆◆



Constructor

- Constructor is a special method whose name is same as that of class name
- Constructor should not have any return type, not even void
- Constructor will be invoked by the JVM automatically whenever you create the object

The constructor is a block or code which similar to a method that is called when the instance of an object is created. It is useful for memory management. It provides data for the objects. In other words, it constructs values, which is why it is known as a constructor.

```
package com.javabykiran.Constructor;
class Student {
     int sid;
     String sname;
     String email;
     long phone;
     void Student() { //return type void and constructor does
                      //not have any return type
       // It considered as a method as it has return type
       System.out.println("method-student()");
     }
     Student() {
        System.out.println("zero parameter const");
     }
     Student(int id, String sn, String em){
        System.out.println("3 parameter const");
        sid = id;
        sname = sn;
        email = em;
     }
    Student(int id, String sn, String em, long ph) {
        System.out.println("4 parameter const");
        sid = id;
        sname = sn;
        email = em;
        phone = ph;
  }
```

```
String show() {
      System.out.println(sid);
      System.out.println(sname);
      System.out.println(email);
      System.out.println(phone);
      return"JBK";
   }
}
```

```
package com.javabykiran.Constructor;
public class Lab5 {
   public static void main(String[] args){
      Student s1=new Student();
      s1.show();    // method student
  Student s2 = new Student(102, "kd,kiran@jbk.com",s1.show());
  Student s3 = newStudent(103, "kiran,Kiran@jbk.com",
                          s2.show(),8888809416L);

      s3.show();
      s1.Student();  //calling the normal method,not the
                     //constructor

   }
```

---Output---

```
zero parameter const
0
null
null
0
0
null
null
0
3 parameter const
102
kd,kiran@jbk.com
JBK
0
```



```
4 parameter const
103
kiran,Kiran@jbk.com
JBK
8888809416
method-student()
```

- Constructor will be used to initialize instance variable of the class with a **different set** of values, but it's not necessary to initialise it
- When you are not writing any constructor inside the class then the default constructor will inserted by the JVM automatically,[at compile time].[If the class is public then the constructor will automatically be public, if it is default, then it will have default access specifier.]
- When you write any constructor, then default constructor will not be inserted by the JVM
- You cannot specify the return type for the constructor, but if you do specify, then it will be treated as a normal method
- You can call the constructor with following ways:
  - A a=new A();
  - new A();
  - super(); // this point is covered in later chapters
  - this(); // this point is covered in later chapters
  - class.forName("com.jbk.A").newInstance(); // this point is covered in later chapters
- Constructor can be overloaded i.e. you can write other constructors by changing the arguments
- You cannot write two constructors with the same number of parameters and same kind of types
- Constructors cannot be overridden
- Any access specifiers can be applied to constructors, if private is applied to constructor, then we cannot create an object outside of class
- If somebody says, if they want to create class, noone should instantiate it. Then you can say that the constructor must be private
- Mostly private constructors are used in singleton design pattern
- What is the use of constructors? You can say that if some code needs to be executed at object creation, then we can write that code in constructor. Generally. It's for initialization of global variable

Business use case :

Let's say there is a use case, client want to send sms and email so we can design class as below (without constructor).

```
package com.javabykiran.Constructor;
class Greeting_Message {
   String greeting_msg = "Hello";
   String user_Name ="Guest";

    String getEmailTxt(String emailMsg) {
       String completeMsg = greeting_msg +" " + user_Name +
       " " + emailMsg;
       System.out.println(completeMsg);
       return completeMsg;
    }
    String getSmsTxt(String smsMsg) {
        String completeMsg = greeting_msg + " " + user_Name
        + " "  +smsMsg;
        return completeMsg;
    }
}
```

Now we can observe here client need to send greeting message everytime in each method So we need to reduce efforts of a client by using constructor.

See the below industrial example :

```
package com.javabykiran.Constructor;
class Greeting_Message {
   String greeting_msg = "Hello";
   String user_Name ="Guest";
     Greeting_Message() {
        // this constructor is just for default clients
     }
    Greeting_Message(String gMsg, String uName) {
        greeting_msg = gMsg;
        user_Name = uName;
     }
    String getEmailTxt(String emailMsg) {
       String completeMsg = greeting_msg +" " + user_Name +
       " " + emailMsg;
       System.out.println(completeMsg);
       return completeMsg;
    }
```



```
    String getSmsTxt(String smsMsg) {
        String completeMsg = greeting_msg + " " + user_Name +
        " "  +smsMsg;
        return completeMsg;
    }
}
```

```
package com.javabykiran.Constructor;
public class Greeting_Message_Test{
    public static void main(String[] args){
      String emailMsg = "EMAIL :This is test email by
       javabykiran";
      String smsMsg = "SMS: This is the best book for java
       learners";
      System.out.println("****** ICICI*******");
      // wants all default data
  Greeting_Message greeting_Message = new Greeting_Message();
     greeting_Message.getEmailTxt(emailMsg);
     greeting_Message.getSmsTxt(smsMsg);
     System.out.println("****** BOA*******");
     // Different users and messages

Greeting_Message greeting_MessageBOA = new Greeting_Message
("Hello", "KiranBOA");
     greeting_MessageBOA.getEmailTxt(emailMsg);
     greeting_MessageBOA.getSmsTxt(smsMsg);
     System.out.println("****** HDFC*******");
  Greeting_Message greeting_MessageHDFC =new Greeting_Message
   ("Hi","KiranHDFC");
     greeting_MessageHDFC.getEmailTxt(emailMsg);
     greeting_MessageHDFC.getSmsTxt(smsMsg);
  }
}
```

---Output---

```
****** ICICI *******
Hello Guest EMAIL : This is test email by javabykiran
****** BOA *******
Hello KiranBOA EMAIL : This is test email by javabykiran
****** HDFC *******
Hi KiranHDFC EMAIL : This is test email by javabykiran
```

Here we initialize variables through constructors, we are not forcing to initialise, and therefor we specified the default constructor as well. Different clients can now act differently, just consid the main method to be different for different clients.

Questions:

1. Is constructor definition mandatory in class?
2. Define constructor.
3. Can we define a method with same name of class?
4. What are the rules in defining a constructor?
5. Why is return type not allowed for constructors?
6. How can compiler and JVM differentiate between constructor and method invocations both have the same classname?
7. Why is the compiler given constructor called as default constructor?
8. What is the default accessibility modifier of default constructor?
9. When must the developer provide constructor explicitly?
10. When does the compiler provide default constructor?
11. If a class has an explicit constructor, will it have a default constructor?
12. Can we call sub class constructor from a super class constructor?
13. What is the use of private constructor?
14. Can we use this() and super() in a method?
15. Does constructor create the object?
16. What are the common uses of "this" keyword in java?

◆◆◆

Inheritance

Inheritance is a process of writing a new class by using existing class functionality. 'The process by which one class acquires the properties(instance variables) and functionalities of another class is called inheritance'. This happens when one class adopts the non-private properties and methods of one class. These classes will then be referred to as superclass and subclass.

- The existing class is called super class or base class or parent class
- New class is called sub class or derived class or child class
- Use of inheritance is code re-usability
- Most importantly, we use inheritance if we want to increase features of class

For ex.: Let's say that a student has attributes like age, location, and this class has existed for a long time. After some time, we need to add more attribute like pan card to student.

What options do we have?

Option 1 : We can modify class student
Option 2 : We need to extend student class and add an attribute in that new subclass

- The first option is a violation of principles of java: 'Java classes are not open for modifications', so we should not modify an existing one. This will increase unit testing which will have to be done again for the existing class
- The second option is, without disturbing existing class, we add variable in another class by making subclass. So, the unit testing needs to be done for the child and not parent

```
package com.javabykiran.Inheritance;
 /*
  * @author Kiran
  */
public class A {
     int age;
     int loc;
}
```

```
package com.javabykiran.Inheritance;
/*
 * @author Kiran
 */
public class B extends A {
    int pancard;
}
```

Note: Super class members inherits to subclass provided they are eligible by access specifiers and if they are not already present in subclass.

```
package com.javabykiran.Inheritance;
/*
 * @author Kiran
 */
public class C {
     private int a = 20;
     public int b = 30;
     public int c = 40;
}
```

```
package com.javabykiran.Inheritance;
/*
 * @author Kiran
*/
public class D extends C {
     public int c = 10;
}
```

```
package com.javabykiran.Inheritance;
/*
 * @author Kiran
 */
public class CDTest {
  public static void main(String[] args){
      D d=new D();
  //System.out.println(d.a);
      //Error:as it not accessible;itis private,'a'is not
      //coming to D
    // b is of C class
   System.out.println(d.b);
```



```
   // c is of D class and the answer will be 10, not 40
    System.out.println(d.c);
  }
}
```

---Output---

```
30
10
```

Here are some rules we need to remember:

- We cannot assign super class to subclass
- We cannot extend final class
- We cannot extend class which has only private constructor, but if we have private constructor as well as public constructor then we can extend super to sub class. And in that case only public constructor will work
- One class can extend single class
- Java doesn't support multiple inheritance in case of classes, but it supports it in case of interfaces
- Class cannot extend itself //read super ,this chapter

If we assign subclass reference to super class reference, it is called 'dynamic dispatch'.

Consider the scenarios given below:

```
package com.javabykiran.Inheritance.scenarios;
/**
 * @author Java By Kiran
                                */
public class A {
    int a = 10;
    int b = 20;
    void m1() {
      System.out.println("I am in A-m1");
    }
    void m2() {
      System.out.println("I am in A-m2");
    }
}
```

```
package com.javabykiran.Inheritance.scenarios;
/**
 * @author Java By Kiran
 */
public class B extends A{
    int b = 30;
    int c = 40;
   void m2() {
     System.out.println("I am in B-m2");
   }
   void m3() {
     System.out.println("I am in B-m3");
   }
}
```

Only change this given class for all scenarios given below:

// Scenario I

```
package com.javabykiran.Inheritance.scenarios;
/**
 * @author Java By Kiran
 */
public class Scenarios {
  public static void main(String[] args){
     // Scenario 1
      A a =new A();  // Super class object a is eligible to
                     //call only A
      System.out.println(a.a);
      System.out.println(a.b);
      //System.out.println(a.c);//Error; c does not exist in A
      a.m1();
      a.m2();
      //a.m3(); //Error m3 does not exist in class A
   }
}
```

---Output---

```
10
20
I am in A-m1
I am in A-m2
```


// Scenario 2

Now change the main method with different scenarios:

```
B b = new B();
System.out.println(b.a); // a of A
System.out.println(b.b); // b of B
System.out.println(b.c); // c of B
b.m1();// m1 of A
b.m2();// m2 of B
b.m3(); // m3 of B
```

---Output---

```
10
30
40
I am in A-m1
I am in B-m2
I am in B-m3
```

This example assigns sub class to super class.
This is allowed in java. i.e. A a = new B( ) in Scenario 3

// Scenario 3

```
1) A a =new B();
2) System.out.println(a.a);// a of A
3) System.out.println(a.b);// b of A
4) //System.out.println(a.c);//Error c of A does not exist in A
5) a.m1();// m1 of A
6) a.m2();// m2 of B
7) //a.m3(); //Error m3 of A does not exist in A
```

---Output---

```
10
20
I am in A-m1
I am in B-m2
```

Remember these Rules: In this case a is eligible to call

- **At compile time,** everything of A class will be called other than that everything shows error at compile time. This means that in this case line no.(4) and line no.(7) will have problems.
- **At run time,** all variables and methods of class A will get called, but all those methods which are overridden by class B will get executed. Or exactly similar methods of A into B will be called (Observe Output).

// Scenario 4

Try changing main method like below:

```
A a=new A();
B b=new B();
a=b; //This is equivalent to A a=new B(); refer 3rd Scenario
```

```
1) System.out.println(a.a);// a of A
2) System.out.println(a.b);// b of A
3) //System.out.println(a.c); //Error c of A not exist
4) a.m1(); //m1 of A
5) a.m2(); //m2 of B
6) //a.m3();//Error m3 of A does not exist
```

// Scenario 5

```
B b=new A();    // syntax error
//Try calling everything by using b
```

// Scenario 6

```
A a=new B();
B b=new B();
b=(B)a; // looks like super class assigned to subclass but
        //it translates internally to
B b=new B(); //It's equivalent to 2nd scenario
```

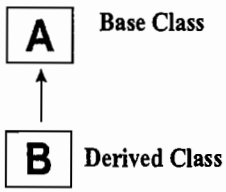
Following are the various types of inheritances:

- Simple/Single Inheritance
- Multi-level Inheritance
- Hierarchical Inheritance
- Multiple Inheritance
- Hybrid Inheritance

Simple/Single Inheritance:

- In this, there will be only one super class and one sub class
- Every class has a super class as Object, and the package for Object class is java.lang.Object.

Diagram:

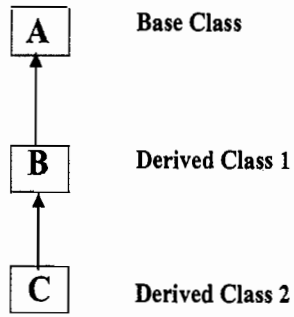


Single Inheritance example program in Java

```
public class A {
    public void methodA(){
         System.out.println("Base class method");
    }
}
public class B extends A {
   public void methodB() {
         System.out.println("Child class method");
    }
public static void main(String args[]){
      B obj = new B();
      obj.methodA();  //calling super class method
      obj.methodB();  //calling local method of B
   }
}
```

Multi-level Inheritance:

- **When a class extends a class, which in turn extends another class, is called multi-level inheritance**

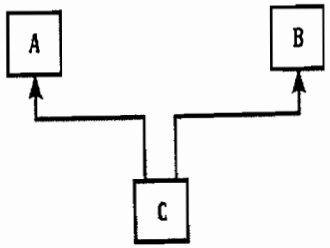



```
public class X {
    public void methodX() {
      System.out.println("Class X method");
    }
 }
public class Y extends X {
    public void methodY() {
       System.out.println("Class Y method");
    }
}
```

```
public class Z extends Y {
     public void methodZ() {
        System.out.println("Class Z method");
     }
public static void main(String args[]) {
      Z obj = new Z();
      obj.methodX(); //calling X grand parent class method
      obj.methodY(); //calling Y parent class method
      obj.methodZ(); //calling Z class local method
  }
}
```

Multiple Inheritance:

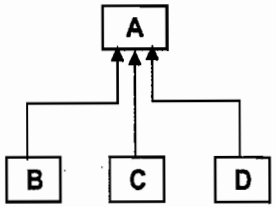
- One class has many super classes
- This is not allowed in java
- One needs to understand why this is not allowed, as you may be asked this question in an interview.



- Note 1 : Multiple Inheritances are very rarely used in software projects. Using multiple inheritances often leads to problems in the hierarchy. This results in unwanted complexity when further extending the class
- Note 2 : Most of the new object oriented languages like Small Talk, Java, and C# do not support multiple inheritances. Multiple Inheritances is supported in C++
- From the diagram above, if class A has m1 and class B also has m1 and they will have different implementations
- As per inheritance concepts both methods should inherited to class C
- If somebody creates object of class C, which m1 will get called? This is ambiguity of methods and therefore not allowed in java
- Class C extends A, B like this syntax is not allowed in java

Hierarchical Inheritance:

In such kind of inheritance one class is inherited by many sub classes. In the example below, class B, C and D inherits the same class.
Class A is parent class (or base class) of B, C and D (or derived classes).



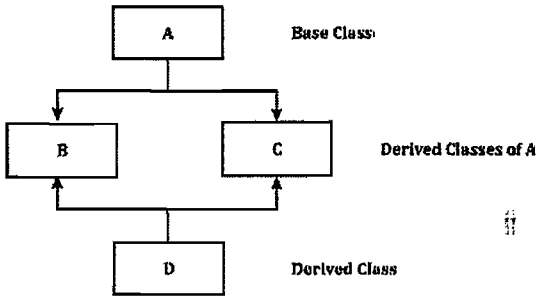

```
public class A {
      public void methodA(){
          System.out.println("method of Class A");
      }
  }
public class B extends A {
      public void methodB() {
           System.out.println("method of Class B");
      }
  }
public class C extends A {
     public void methodC() {
          System.out.println("method of Class C");
      }
 }
public class D extends A {
      public void methodD() {
         System.out.println("method of Class D");
      }
  }
public class MyClass {
      public void methodB() {
          System.out.println("method of Class B");
      }
```

```
    public static void main(String args[]) {
        B obj1 = new B();
        C obj2 = new C();
        D obj3 = new D();
        obj1.methodA();
        obj2.methodA();
        obj3.methodA();
    }
}
```

---Output---

```
method of Class A
method of Class A
method of Class A
```

Hybrid Inheritance



- Not allowed in Java
- In simple terms, you can say that Hybrid inheritance is a combination of Single and Multiple inheritances. A typical flow diagram would look like above. A hybrid inheritance can not be achieved in the java in a same way as multiple inheritances can be!! Using interface it is allowed. Yes you heard it right. By using interfaces you can have multiple as well as hybrid inheritance in Java

Example:

```
class A extends A{

}
/*This is not ok because it is like cyclic inheritance,A
inherits itself.*/

class A {

}
```



```
class B {

}

class C extends A,B {

}

/* Not ok, because multiple inheritance is not allowed.
Here C has two parent classes.*/
```

Questions:

1. What is inheritance?
2. Why do we need to use inheritance?
3. Which of these keywords must be used to inherit a class?
   a) super
   b) this
   c) extent
   d) extends
4. What are the types of inheritances?
5. What is the syntax of inheritance?
6. What is the output of this program?

```
class A {
    int i;
        void display() {
            System.out.println(i);
        }
}

class B extends A {
    int j;
        void display() {
            System.out.println(j);
        }
}

class inheritance_demo {
  public static void main(Stringargs[]) {
        B obj = new B();
        obj.i=1;
        obj.j=2;
```

```
        obj.display();
    }
}
```

a) 1
b) 2
c) Compilation Error

7. How inheritance can be implemented in java?
8. What is multilevel inheritance?
9. What is multiple inheritance? Why doesn't java support multiple inheritance?
10. How do you restrict a member of a class from inheriting to its sub classes?
11. Which of these keywords is used to refer to member of base class from a sub class?
    a) upper
    b) super
    c) this
    d) None of the above
12. How do you implement multiple inheritance in java?
13. What is the output of this program?

```
class A {
   int i;
}

class B extends A {
       int j;
     void display() {
         super.i = j + 1;
         System.out.println(j + " " + i);
     }
}

class inheritance {
   public static void main(String args[]) {
          B obj = new B();
          obj.i=1;
          obj.j=2;
          obj.display();
     }
}
```

a) 22
b) 33

c) 23
d) 32

14. Can a class extend itself?
15. What happens if super class and sub class have the same field name?
16. Does java support multiple inheritance?
17. Are constructors inherited? Can a subclass call the parent's class constructor? When?
18. What is the output of this program?

```
class A {
   public int i;
   private int j;
}
class B extends A {
        void display() {
          super.j = super.i + 1;
          System.out.println(super.i + " " + super.j);
        }
}
class inheritance {
   public static void main(String args[]) {
        B obj = new B();
        obj.i=1;
        obj.j=2;
        obj.display();
   }
}
```

a) 22
b) 33
c) Runtime Error
d) Compilation Error

19. Why doesn't java support multiple inheritance?
20. You know that all classes in java are inherited from java.lang. Object class are interfaces also inherited from Object class?
21. What is the output of this program?

```
class A {
     public int i;
     public int j;
     A() {
        i = 1;
        j = 2;
     }
}
```

```
class B extends A {
    int a;
       B() {
        super();
       }
}
class superuse {
      public static void main(String args[]) {
          B obj = new B();
          System.out.println(obj.i +" "+obj.j);
       }
}
```

a) 12
b) 21
c) Runtime Error
d) Compilation Error

22. Can we reduce the visibility of the inherited or overridden method?
23. A class member declared protected becomes member of subclass of which type?
    a) public member
    b) private member
    c) protected member
    d) static member
24. Which of the following is tightly bound? Inheritance or Composition?
25. What is the difference Between this() and super()?
26. What is the output of this program?

```
class A {
     public int i;
     protected int j;
}
class B extends A {
        int j;
      void display() {
           super.j = 3;
           System.out.println(i + " " + j);
      }
}
class Output {
  public static void main(String args[]) {
         B obj = new B();
         obj.i=1;
```

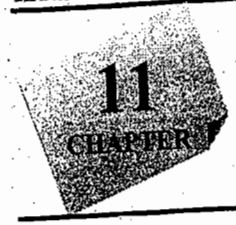


```
         obj.j=2;
         obj.display();
    }
}
```

a) 12
b) 21
c) 13
d) 31

27. What will happen if class implements two interface having common method?
28. Does a class inherit the constructor of its super class?
29. What are the points to consider in terms of access modifier when we are overriding any method?
30. What do you mean by inheritance?
31. What are the types of inheritance?
32. What is covariant return type?
33. How does the compiler handle exceptions in overriding?
34. Can a class extend more than one class or does java support multiple inheritance? If not, why?
35. Can a class extend itself?
36. Which of the these is the correct way of inheriting class A by class B?
    a) class B + class A{}
    b) class B inherits class A {}
    c) class B extends A{}
    d) class B extends class A{}
37. Are constructors and initialisers also inherited to sub classes?
38. How do you implement multiple inheritance in java?
39. What happens if both, super class and sub class, have a field with the same name?
40. Are static members inherited to sub classes?
41. Which of the following statements are incorrect?
    a) public members of class can be accessed by any code in the program
    b) private members of class can only be accessed by other members of the class
    c) private members of class can be inherited by a sub class, and become protected members in subclass
    d) protected members of a class can be inherited by a sub class, and become private members of the subclass

◆◆◆



Super This

Super in java is a keyword which is a reference variable. It is used to refer to t immediate superclass object or instance variable. The word super came into usa because of the concept of inheritance. The keyword super is placed inside a subclass call a method from a superclass.

Why do we use the super and this keyword?

Let's say we have a super class as Person and Employee as a sub class as shown below:

```
package com.javabykiran.Super_This;
  public class Person {
    int age = 45;
   }
```

```
package com.javabykiran.Super_This;
 public class Employee extends Person{
   int age = 29;
       void printStudent() {
          int age = 25;
   //Here are two ways we have to call global age variable
   //of Person class
          // 1
          Person person = new Person();
          System.out.println(person.age); //45
          // 2
          System.out.println(super.age); // 45
          //Local variable
          System.out.println(age); //25
          //Global variable of same class
          System.out.println(this.age); //29

    }
 }
```

Now, if we see Employee class's printStudent() method, there are two ways to call age variable of Person class.

First way is by creating object of Person class. But this is not an affordable option as just for sake of one variable age here we are loading complete class Person, which wastes memory unnecessarily.

Second way is to just call age by using super keyword which will not waste memory and we can achieve our objectives as well

Other two options are to print local variable age or print global variable of same class.

Note : We use the super keyword to bypass class variables and method's local variable with same name while calling and we need not to create objects.

It's all about memory management.

Super

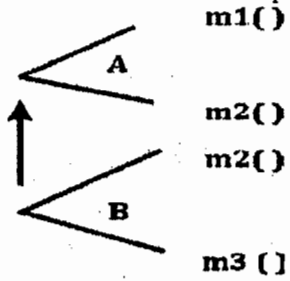
Remember the points given below:

- We must have super class subclass relationship in our program.
- Super can be applied to variable, constructors, and method.
- Super keyword always represents super class object.
- It's generally used to bypass global variable with the same name

Syntax:

```
super.a;      //for Variable
super.m1();   //for Method
super();      // for Constructor
```

- To call super class variable or method we have two options



mple:

```
A a=new A();     //1st way
  a.m2();

super.m2();      //2nd way
```

1st way : In this case we bring all members of capital A in to RAM, out of that we are just calling m2(),so unnecessary memory will get allocated for another members

- **2nd way :** The other option super.m2()
  - ➤ There is no super.super in java
  - ➤ In any of our class's constructors if we don't write any call to constructor of parent class by using super like super(), then JVM puts super call to constructor in the first line of every constructor wherever you did not add your own call to constructor.

Example 1:

```
public class A {
    public A(){    // Default Constructor put by JVM at Run
                   //Time..
      super();          //invisible present here
  }
}
```

Example 2:

```
package com.constructor;
public class A {
  public A(){
     super(10); //Error -Super class has constructor
    //without parameter, in this case it's a Object class
  }
}
```

- ➤ Super call to constructor must be at first line of all constructors.
- ➤ We strictly cannot call super () in methods in any line; this rule applies only to super() not to super.a or super.m1().

```
package com.constructor;
public class A {
    public A(){
      System.out.println("Java by kiran");
      super(); // Error - super () must be at first line
               // of constructor only
    }
    void m1(){
      super(); // Error in method super()not allowed
    }
    A (int x){
      super();   // Correct- Allowed in constructor
      super(10); // Error as super() is call at second line
    }
}
```


ecursion by using super call to constructor will result in compile time error in java.

e compile time error is shown below :

```
ckage com.javabykiran.Super_This;
blic class A extends A{  //class can not extends itself
  //super();  // go on  calling  the same -- recursion
```

e: In this example, recursion between super call to constructor will be there because we are extending same class itself. Consider invisible super() inside constrcuctor;

serve the example below to see how super keyword navigates the flow of our program.

```
ackage com.constructor;
ublic class A {
   A(){
     //super(); //By default super is here
     System.out.println("A");
   }
  A(int x) {
     //super(); //By default super is here
      System.out.println(x);
  }
  public void m6() {
    System.out.println("I am in A m6()");
  }
}//end of class A
```

```
package com.constructor;
public class B extends A{
    B() {
      //super(); //JVM put super() by default atruntime
    }
    public void m6(){
      System.out.println("I am in B m6()");
    }
}//end of class B
```

```
package com.constructor;
 public class C extends A {
     C(){
        //super(); // JVM put super() by default at runtime
        System.out.println("C");
      }
  } // end of class C
```

```
package com.constructor;
public class Client      {
  public static void main(String[] args) {
     C c= new C();
     c.m6();
   }
}
```

---Output---

```
A
C
I am in A m6()
```

This

This keyword in java is used inside a method or constructor of a class. It is used to refer to a member of a current object within the instance of a method or constructor.

- This represents current class object
- Can be applied to variable, constructor and method

Syntax:

```
this.a;        // variable case
this.m1();     // method case
this(10);      // To call parameterize constructor of current class object
```

- In case of variable, same class's global variable will get called
- Call to constructor by using this 'this()' must be a first line of constructor only. This means that we cannot add 'this()' anywhere other than the first line of constructor
- inside method 'this()' is not allowed, that is call to constructor not allowed by using this.
- JVM never puts automatically this() keyword like super()
- If we wrote call to constructor explicitly by using 'this()' the 'super()' call to constructor will not be put by JVM

- Recursion will be there as we call the same class constructor using this() in same constructor, which is not allowed in java

Example 1:

```
package com.This;
 public class A {
     int a=10;
 }

package com.This;
 public class B extends A {
     int a=20;
 }

package  com.This;
 public class C extends A{
     int a=30;
 }

package com.This;
 public class D extends C {
    int a=40;
   void m1(){
      int a=50;
      System.out.println("This is javabykiran class");
      System.out.println(a);       //prints 50
      System.out.println(this.a); //class variable value 40
      System.out.println(super.a); //JVM print 30 Immediate
      //superclass
     B b =new B();
      System.out.println(b.a); //no other way as super.super
      //is not allowed
     A a1=new A();
      System.out.println(a1.a);  //only way to run A class
     //variable
  }
 public static void main(String[] args) {
    D b=new D();
    b.m1();
 }
 //end D
```

---Output---

```
This is java by kiran class 50
40
30
20
10
```

Note: To understand this example in more detail

- Go on commenting every variable start from m1 of Class D then run the program
- Comment global variable of class D then run D.java
- Comment C class's global variable then again run D..java and observe output

Example 2:

```
package com.This;
class X {
     public X() {
          this(20); //recursion
      }
     public X(int i){
          this();//recursion invocation error
      }
}
```

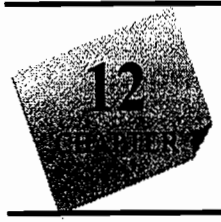
Summary:

This	Super
• 'this' is a reference variable which contains current class objects • 'this' can be used in two ways: • a) To invoke current class variable and methods ▪ this.a; ▪ this.m1(); • b)To invoke current class constructors Ex: this(); this (10,20);	• super is reference variable which contains immediate super class objects • super also can be used in two ways: • a) To invoke immediate super class variable and methods. ▪ super.n; ▪ super.m1(); • b) to invoke immediate super class constructor Ex : super(); super(10);
• This reference variable is instance reference variable and cannot be accessed from static context	• super reference variable is instance reference variable and cannot be accessed from static context
• Call to this() must be at the first line of constructor only	• Call to super() must be at the first line of c[illegible] only

Questions:

1. What is the difference between Super and This keyword?
2. Why do we use This keyword?
3. When do we need super keyword?
4. What is 'super' used for?
5. In a case there are no instance variables, what does the default constructor initialise?

◆◆◆



Polymorphism

- Polymorphism in java, is an object's ability to take on various forms. It enables us to complete one action in many different ways. Polymorphism is useful in carrying out inheritance processes in java.
- An entity which behaves differently in different cases is called as polymorphism
  Example: You are using one button to switch the computer ON and OFF. This is the polymorphic behavior.
- We can see polymorphism in java at compile time and runtime
- Compile time polymorphism is also called method overloading and runtime polymorphism is called method overriding.

Method Overloading

Method overloading is a feature that lets a class have multiple methods with the same name. Here, the method remains the same but the parameters are different. This made simple by varargs (refer jdk 1.5 chapter)

- Method Overloading exists in the same class. There is no need of a superclass and subclass relationship
- Method name must be the same
- Parameters must be different, the count of parameters does not matter
- Parameters must be changed in one of the following three ways:
  - Types of parameters
  - Number of parameters
  - Order of parameters

```
package com.javabykiran.Poly;
public class Arithmatic {
   void sum(int a) {
      System.out.println(a + a);
   }
   void sum(double a) {
      System.out.println(a + a);
    }
   void sum(double a, double b) {
      System.out.println(a + b);
```



```
    }
    int sum(int a, int b) {
    return a + b;
    }
    void sum(int a, double b) {
       System.out.println(a + b);
    }
    void sum(double a, int b) {
       System.out.println(a + b);
    }
}
```

```
package com.javabykiran.Poly;
public class Lab1 {
   public static void main(String[] args) {
       Arithmatic ar = new Arithmatic();
       ar.sum(10);                          //20
       ar.sum(10.5);                        //21.0
       ar.sum(10.5, 20.5);                 // 31.0
       ar.sum(10, 20.9);                   //30.9
       ar.sum(10.5, 20);                  //30.5
       ar.sum(10,10);                     //20
  }
}
```

1. Access specifiers can be anything
2. Return type can be anything
3. Exceptions thrown can be anything
4. Implementation does not matter in this case as the method can have any kind of logic
5. Method overloading is not for changing any business logic but just for increasing the maintainability and readability of a code

Example: When should we use Overloading ? Imagine this class we have created in 2005

```
class A{
public void insertStudent (int age,String loc,int phoneNo){
 }
}
Clients are calling this class as below:
Client 1:
A a =new A();
```

```
a.insertStudent(20,"pune",8888809416);
```

Client 2:

```
A a1 =new A();
a1.insertStudent(20,"nagpur",8888809416);
```

Now in 2010, we have a requirement of capturing the aadhar card of a student.
We have two options:

Option 1

```
class A {
// increasing paramenters in same method
public void insertStudent(int age, String loc,int phoneNo,
int aadharCardNo){
 }
}
```

Option 2

```
class A {
// write another method with a different name
public void AadharCard (int age, String loc, int phone No, int
aadharCardNo){
}
public void  insertStudent(int age, String loc,int phoneNo){
 }    //old existing method
}
```

Here, our requirement is completed in both cases, and we will now consider clients.
In the first option, we will let client know changes in their existing method.
Then the client will say:

- We don't required aadhar card, then why should we change our code?
- Tomorrow, if any parameters are added, will you be asking us to change our code again?
- We need to do a lot of testing for changes, our client may say - "Please do not change methods repeatedly but write other methods so that if we need, we will call separately."
- In the second option we will let the client know about the changes made in new method. They will then say:
- Ok we will accommodate you but what changes you have made in your method?
- Could you explain to us how it will be impacting our code?
- If everything is same and only parameter is increased, why you not gave the same name

of method so that we don't get confused;
so, we will face a lot of issues and queries from clients.

- Tomorrow, if a new developer joins the company, he will not have any idea that both the methods are same except only parameters because the names are very different

Now, we will use third option by using overloading

```
class A{
 // write another method with same name
public void insertStudent(int age,String loc,int phoneNo,
int aadharCardNo){
// logic for aadhar card. Then we have just called existing
//method
    insertStudent(int age, String loc,int phoneNo);
}
public void insertStudent (int age, String loc,int phoneNo){
 }
}
```

How we can communicate to client that: We have overloaded method insertStudent() for aadhar card. Now, our clients will automatically understand what has happened, they will see the impact and they will be sure that there has been no change in logic overall, but for aadhar card only.
Ultimately, the method of overloading achieved maintainability and readability of a code.
In interviews, these questions need to be considered:

- Let's say we have hierarchy A, B, C; where A is superclass, B is the child of A and C is the child of B

Example:

```
public class A {
}

public class B extends A {
}

public class C extends B {
}

public class OverLoadingScenarios {
    void m1(A a) {
        System.out.println("I am in m1 A");
    }
```

```
    void m1(B b) {
       System.out.println("I am in m1  B");
    }
    void m1(C c) {
        System.out.println("I am in m1  C");
    }
```

Testing by client – if we run the below given program, what will be the output?

```
package com.javabykiran.Poly;
public class OverLoadingTest {

   public static void main(String[] args) {
   OverLoadingScenarios loadingScenarios = new
    OverLoadingScenarios();

// Scene I
  A a=new A();
  loadingScenarios.m1(a);

// Scene II
  B b=new B();
  loadingScenarios.m1(b);

// Scene III
  C c=new C();
  loadingScenarios.m1(c);

// Scene IV
   B bc=new C();
   loadingScenarios.m1(bc);

// Scene V
   A ab=new B();
   loadingScenarios.m1(ab);

// Scene VI
    loadingScenarios.m1(null);
  }
}
```


---Output---

```
I am in m1 A
I am in m1 B
I am in m1 C
I am in m1 B
I am in m1 A
I am in m1 C
```

- **Scenario VI**

  In this scenario, null can be accommodated in all m1 method but at the end, only one m1 method will get executed. That is, the more specific m1 will get called among A, B, C and in this case, C is more specific, and has more information out of all three classes.
  So output is -- I am in m1 C

Changes in Scenario

```
public class OverLoadingScenarios {
     void m1(A a) {
        System.out.println("I am in m1 A");
     }
     void m1(B b) {
        System.out.println("I am in m1 B");
     }
     void m1(C c) {
        System.out.println("I am in m1 C");
     }
     void m1(D d) { //D is not related to A,B and C Hierarchy
         System.out.println("I am in m1 D");
     }
}
public static void main(String[] args) {
    OverLoadingScenarios ls =new OverLoadingScenarios();
     // Scene VII
      ls.m1(null);
   }
}
```

- **Scenario VII**

  In this scenario null can be accommodated in all m1, but at the end, only the m1 which is more specific will get called. Among A, B and C, C is more specific and has more information than the other two classes, so m1() of C gets called.

Now D is nowhere related to all these classes. So we can't say which is more specific and therefore, the compiler gets confused and it will show a compile time error.

So, there will be no output but compile time error that is shown is:

"The method m1(null) is ambiguous for the type OverLoadingScenarios"

Consider the program shown below:

```
package com.javabykiran.Poly;
public class X {
    void m1(Object object) {
         System.out.println("m1 with object as a param");
    }
    void m1(String string) {
         System.out.println("m1 with string as a param");
    }
    public static void main(String[] args){
        X x = new X();
        x.m1(null);
        x.m1(new Object());
        x.m1("Kiran Sir");
        x.m1(new X());
        x.m1(new string());
    }
}
```

---Output---

```
m1 with string as a param
m1 with object as a param
m1 with string as a param
m1 with object as a param
m1 with string as a param
```

- Why is method overloading called compile time polymorphism? As it has been identified at compile time, which method to call. Also, if we consider m1 as an entity, then it behaves differently at different times, and called compile time polymorphism.

Method Overriding

The process of implementing superclass method in subclass is called method overriding. As per java principles, classes are closed for modification, so if we want to modify a method in any class, changing the existing method is not a good idea. Instead of that, we should extend that class and override method in child class.


```
public class A {
    public void m1() {
      System.out.println("A-m1");
    }
}
public class B extends A {
    public void m1() {
        System.out.println("B-m1");
    }
    public void m3() {
        System.out.println("B-m3");
    }
}
public class Lab2S {
    public static void main(String args[]){
        B obj =new B ();
        obj.m1 ();
        obj m3 ();
    }
}
```

- In the above program, B is implementing the method m1() with the same signature as super class A i.e m1 () of class B is overriding m1() of class A
- As per object oriented concept, the best practice is that class should not open for modification
- If you want to add new functionality to existing class or if you want to modify the existing functionality of class, then you should not disturb the existing class [in this case class A m1(). You should always write the subclass of the existing class.]
- The Subclass can be written for three reasons:
  - To add new features/properties [if student had properties age and location and in future we get requirement to add address for student, we should go for extending a class and adding a property in subclass as address]
  - To override/change the existing functionality
  - To inherit the existing functionality
- When you are overriding superclass methods in subclass you need to follow certain rules
  - The subclass method name must be the same as superclass method name
  - Subclass method parameters must be the same as superclass method parameters [not even subclass as a parameter is allowed]
  - Subclass method's return type must be the same as superclass method return type [sub type is allowed]

- Subclass method's access modifier must be the same or higher than the superclass method access modifier

superclass	In subclass, we can have access specifier
public	Public
protected	Protected, Public
default	Default, Protected, Public
private	Not possible possible to Override

Dynamic Dispatch :

Dynamic dispatch is the process of assigning subclass object to superclass reference variable.

A a=new B();

B is subclass and A is a superclass.

@Override annotation

➤ @Override is the annotation which is used to make compiler check all the rules of overriding

If this annotation is not written, then the compiler will apply all overriding rules only if

1. the superclass and subclass relation exist
2. method name same
3. the parameters of both methods are same.

But, if we write @Override on subclass method, then the compiler must apply all rules irrespective of the above three points.

Examples:

Without annotation

- In superclass

```
private void m1 () {
        System.out.println("XX-m1");
                }
```

- In subclass

```
private void m1 () {
        System.out.println("XX-m1");
}
// Compile time Ok
// Runtime Ok
```

With annotation:

- In superclass

```
private void m1 () {
        System.out.println("XX-m1");
                }
```

- In subclass

```
@Override
private void m1 () {
        System.out.println("XX-m1");
}
```

Compile time : //Error - private method cannot be overridden

Examples:

Without annotation:

- In superclass

```
public static void m1 () {
    System.out.println("XX-m1");
}
```

- In subclass

```
public static void m1 () {
    System.out.println("YY-m1");
}
```

Compile time

//OK – it is not overriding as static method won't override

//Runtime OK

With annotation:

- In superclass

```
public static void m1 () {
        System.out.println ("XX-m1");
}
```

- In subclass

```
@Override
public static void m1 () {
    System.out.println ("YY-m1");
}
```

Compile time

// Error - static method cannot be overridden

// Runtime Error

- When superclass method throws an exception then the subclass can do one of the following :
  a. Subclass methods can ignore the method level exception
  b. Subclass method can throw the same exception which is thrown by superclass method

c. Subclass method can throw the exception which is the subclass of superclass method's exception
d. Subclass method cannot throw the exception which is the superclass of superclass method's exception
e. Subclass method cannot throw the exception which is non subclass to superclass method exception

Example: If the superclass has

void m1 () and throws B

Then in this subclass we can have assume that A , B, C and D are exception classes as A ' B ' C ' D, A is top class and D is the last child in hierarchy.

- void m1 ()
- void m1 () throws B
- void m1 () throws D
- void m1 () throws A
- void m1 () throws C

Without Annotation

Example:

The assumption is that X is superclass and Y is subclass

- In superclass

```
public void m1 (){
        System.out.println("XX-m1");
            }
```

- In subclass

```
public void m1 () {
        System.out.println("XX-m1");
    }
```

// Compile time Ok
// Runtime Ok

- In superclass

```
public void m1 (){
        System.out.println("XX-m1");
    }
```

- In subclass

```
  void m1 () {
        System.out.println("XX-m1");
   }
```

Compile time //NOT OK - access specifiers must be bigger or same
RunTime //NOT OK

- In superclass


```
public XX m1 (){
        System.out.println("XX-m1");
   }
```

- In subclass

```
public YY m1 () {
        System.out.println("YY-m1");
   }
```

Compile time // OK - If XX is a superclass and YY is a subclass
Runtime // OK

- In superclass

```
public void m1 (XX xx) {
        System.out.println("XX-m1");
    }
```

- In subclass

```
public void m1 (YY yy) {
        System.out.println("YY-m1");
    }
```

// Compile time parameters must be same
Runtime // OK

- In superclass

```
private void m1 () {
        System.out.println("XX-m1");
    }
```

- In subclass

```
private void m1 () {
        System.out.println("YY-m1");
    }
```

Compile time // OK - it's not overriding, it's just an independent method
Runrime // OK - It will run independently

- In superclass

```
 static void m1 () {
        System.out.println("XX-m1");
    }
```

- In subclass

```
static void m1 () {
        System.out.println("YY-m1");
    }
```

Compile time // OK
It's not overriding, it's just an independent method as static method can never be overridden

Runtime // OK

In the industry, people use Overriding to change the behaviour of methods like below:

```
Class A{
    void m1() {
      //MD5 encryption algorithm
    }
}
```

In this class, we have the method m1 which has md5 encryption implemented. Now, when our requirement changes, we need to use the RSA algorithm

Option 1

```
class A{
    void m1(){
        //RSA encryption algorithm
    }
}
```

Option 2

```
class B extends A {
    void m1(){
        //RSA encryption algorithm
    }
}
```

If you observe option 2 above, it is more feasible when we use overriding.
Because if the requirement arises again which says to go for MD5 algorithm, we can just comment

Option 2

```
class B extends A {
/*
    void m1(){
        //RSA encryption algorithm
    }
*/
}
```

Client will be unaffected as they can call.

```
B a=new B();
a.m1();
```

So, if you have method in B it will be called, or if there is not there then superclass metho will be called. So we will have more flexibility, which we should always ensure for futur

maintenance issues.

[In the real sense it is advisable not to disturb clients repeatedly. Do coding in such a way that even after changes are made, our code is easily extensible. Like here, we just need to comment and uncomment for MD5 and RSA according to requirement]

@Override is just for readability. I mean, if the developer has left the company and a new developer joins, then he/she will easily understand that this method is overridden so that he can further modify easily in future. At least he/she will understand that there is some relationship between the two methods.

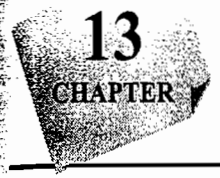
- Can I override private method?
  Answer: No, you cannot override private methods because when superclass method is private it will not be visible to subclass. Whatever the methods you write in subclass will be treated as new method and not an overridden method.
- Can I override a static method?
  Answer: No.
- Can I stop method overriding?
  Answer: Yes, by declaring method as final.
- Can I stop inheritance?
  Answer: Yes, by declaring super class as final.
- Why is method overriding called runtime polymorphism?
  a. We have class A with m1- "A m1" prints
  b. We have subclass B with m1 - "B m1" prints
  c. Client wrote main method and in main,
     i. A a=new B;
     ii. a.m1();
     iii. The above written line is compiled as class A has m1();
     iv. At runtime, we will get output of m1 which, is there at B
     v. Since at compile time we think m1 of A will get called but m1 of B got executed instead. This is called at runtime polymorphism

Questions:

1. What is polymorphism in Java?
2. What is function overloading?
3. State the differences between Overloading and Overriding?
4. What is static member class?
5. What are non-static inner classes?
6. Can a static nested class have access to the enclosing class non- static methods or instance variables?
7. What are the advantages and disadvantages of Inner classes?

8. What are the different types of anonymous classes?
9. If you compile a file containing inner class, how many .class files are created and are all of them accessible in the usual way?
10. How to access the inner class from code within the outer class?
11. How is Polymorphism supported in Java?
12. How to create an inner class instance from outside the outer class instance code?
13. Which modifiers can be applied to the inner class?
14. Can the method local inner class object access method's local variables?
15. Can a method local inner class access the local final variables? Why?
16. Which modifiers can be applied to the method local inner class?
17. Can a local class declared inside a static method have access to the instance members of the outer class?
18. Can a method which is not in the definition of the superclass of an anonymous class be invoked on that anonymous class reference?
19. Can an anonymous class define a method of its own?
20. Can an anonymous class implement an interface and also extend a class at the same time?
21. Where should we use Polymorphism in code?
22. What is Parameteric Polymorphism in Java?

◆◆◆



Abstraction

Abstraction is a process of hiding details about implementation from the user while letting them utilise the services or functions. Therefore, the user will know what an object does, but not how it does it. It is done to simplify the user interface while hiding the complex processes.

- Exposing only required or important things is called abstraction
- Abstraction can be done in two ways :
  - Abstract class
  - Interface
- It is used for maintaining the standards of coding in project

Interface

- One of the java principles says that we must follow the I to C design. This means that every class should be implemented from Interface
- In the industry, architect level people create interfaces and then it is given to the developers for writing classes by implementing interfaces provided
- This is best way to expose our project's API to some other projects. For example, ICICI bank can expose methods or interfaces to various shopping carts
- The interface is the same as ordinary class but remember below points
- We cannot instantiate Interface but we can create reference variables
- The interface doesn't have a constructor
- It can be compiled but cannot run
- Interface can extends interface
- The interface can extend multiple interfaces
- Methods in interface are abstract, remember abstract method means not having a body

  Example: void m1(); //No body Method
- In case of classes, we must define the body of method

```
package com.abstraction;
    public class X {
        void m1(); // compile time error method should have body
    }
```

- In case of classes, if we want the above written lines to compile when we have

method without body, then we must explicitly define that method as abstract in case of classes, but not in case of interface

```
package com.abstraction;
  public class X {
      abstract void m1(); // compile time error
  }
```

- In case of classes, if method is abstract then class must be declared as abstract as well.

  We will learn this in more details in the latter part of this chapter

```
package com.abstraction;
  public abstract class X {
      abstract void m1(); // No error
  }
```

- In case of interface we don't need to do explicit declaration for methods like class but they are, by default, abstract

```
package com.InterFace;
 interface Xi {
    void m1(); // abstract keyword invisibly present
 }
```

- All methods are public in interface. If you want to mention access specifiers, mention them as public, or do not mention any access specifier and JVM will automatically consider it as public, every time

```
package com.InterFace;
  interface xyz {
    void m1(); // public and abstract invisibly put by JVM
  }
package com.InterFace;
 public interface x {
     public abstract void m1(); //OK
     public void m3(); //OK
     abstract void m6();//OK
     void m7();//OK
 }
```

- Variables are public final static in interface and act like constants
- If we want to use methods of interface, we have to implement that interface in any class, and methods of interface must be implemented in a class by using the implements keyword

```
package com.InterFace;
 interface Xi {
     void m1();
 }
package com.InterFace;
 public class  XImpl implements xi{
   //Not compiled as we have not implemented all the methods
}
```



```
package com.InterFace;
  public interface Ai {
      void m1(); //OK
  }
package com.InterFace;
  public interface Bi {
      void m1();          //OK
  }
package com.InterFace;
  public interface Ci extends Ai,Bi {   //OK
      void m1();
  }
```

This is allowed in java

- interface can extend interface1 and interface2
- class extends class implements interface
- class implements interface
- class extends class implements interface1 and interface2

Q. Why do we need Interface?

Answer: To expose the third party API

Q. Why not constructor?

Answer: In interface it will not get called by using super by using hierarchy

Q. As an interface has abstract method, how will you use the abstract method of interface?

Answer: Implement that method in class, or in other words override methods, and then create object of that class and call all methods

Q. Have you ever used others' third party API code?

Answer: Yes, they have their Interface and API and we can use that

Why interface??

Suppose there is a requirement for Flipkart to integrate ICICI bank code into their shopping cart.Their customers want to make payment for products they purchased. Now, to make payment, Flipkart does not have their own bank, so it must take help of others banks.

Let's say ICICI develops code like below:

```
class Transaction {
    void withdrawAmt(int amtToWithdraw) {
        //logic of withdraw
        //ICICI DB connection and updating in their DB
    }
}
```

Flipkart needs this class so they request ICICI bank for the same. The problem with ICICI is that if they give this complete code to flipkart they risk exposing everything of their own database to them as well as their logic, which is a security violation.

Now, what exactly Flipkart needs is the method name and class name. But to compile their class they must get one like below:

```
Transaction t=new Transaction(); //1
t.withdrawAmt(500);  //2
```

If the first line to compile at Flipkart ends, then they must have this class which ICICI cannot give. The best solution here is for ICICI to develop an Interface of Transaction

class as shown below:

```
interface Transactioni {
      void withdrawAmt(int amtToWithdraw) ;
 }
class TransactionImpl implements Transactioni {
     void withdrawAmt(int amtToWithdraw){
         //logic of withdraw
         //ICICI DB connection and updating intheir DB
      }
}
```

- Flipkart will now get the interface butnot class, and they can use it as shown below:
  Transactioni ti=new TransactionImpl(); // the result on the right hand side may be // achieved by webservices or EJB TransactionImpl physically not present at Flipkart ti. withdrawAmt(500);
- In this case, both parties achieved their goal, Flipkart which only needs method name and parameter and they got it

- ICICI only wants to give them the name but not their logic, which they provided. Also it does not matter to ICICI what you purchased, they just want to have amount to be deducted

Abstract class

- Abstract class is just like a normal class but with some extra features which we will see in this chapter
- Abstract class can have constructor
- It can have abstract methods or concrete methods or nothing [empty class] or mix of both methods
- To use abstract method of class, we must extend the abstract class with ordinary class or must implement all abstract methods in that ordinary class
- If we don't want to implement or override that method, make those methods as abstract
- If any methodis abstract in a class then that class must be declared as abstract
- We cannot instantiate abstractclass
- Multiple inheritances are not allowed in abstract class but allowed in interfaces
- Abstract classes are useful if we want others to not call our class directly but though Inheritance
- Abstract static final combinations are not allowed in java

Question: Why does the abstract class have a constructor even though you cannot create object?

Answer: Abstract class can only be extended by class and that class will have its own constructor and that constructor will have super (); which is calling constructor of abstract class. If the abstract class doesn't have constructor then the class extending that abstract class will not get compiled. Therefore, abstract classes have constructors.

Question: Can the abstract class have concrete methods only [concrete method means method withbody]?

Answer: Yes.

Question: If abstract class can have both types of methods [one having body and other without body] then what is use of those methods?

Answer: We can use those methods by (1) extending abstract class or (2) making that method as a static so that we will not need to create an object.

Question: What is difference between abstract class and interface?

Abstract Class	Interface
Constructor Present	No Constructor Present
Multiple inheritance not allowed in case of	Multiple inheritance allowed

classes	
It has abstract and concrete methods To use this class we extend it	Methods are abstract only We implement interfaces in class
After extending abstract class, access specifiers of overridden method must be bigger than or same as that of access specifiers mentioned in method of abstract class	In implementing class all method must be public

Question: How to call m2() which is concrete of abstract class?
Answer: Make it as static and call as Kiran.m2 (); //where Kiran is class name

Question: Can abstract class extend ordinary class?
Answer: Yes.

Somebody may ask you what you will prefer in the case below:
One option

```
Interface A{
        void m1 ();
}
      OR
abstract class A{
        abstract void m1 ();
}
```

1. Suppose you have a class Test and you want to have m1 method in your class. Which opti
   out of the above would you select and why? Justify your answer.
2. In your project where you have used abstract classes and interfaces and why? Could y
   please explain the scenarios?

Questions:

1. What is Abstraction?
2. What is Abstract class?
3. When are Abstract methods used?
4. Can an interface extend another interface in Java?
5. When would we want an interface to extend another interface?
6. What is Marker Interface?
7. Can an inner class be built in an Interface?
8. How to define an Abstract class?
9. How to define an Interface?
10. Why would you use Comparator and Comparable interfaces?
11. What's the difference between an interface and an abstract class? Also discuss the similari

14 CHAPTER

Garbage Collection (GC)

The Garbage Collection feature of the java language is possibly its best feature. Thanks to garbage collection, programmers do not have to worry about memory allocation and dellocation since it automatically frees up memory for further use. It removes unused objects.
It recognises objects that have not been referenced, and seeing as they are not in use, it removes them.

- In C language, we can use malloc() or dealloc() to allocate the memory required. You can also clean that memory using these functions
- In C++ language, we can use new operators to allocate the memory and delete operator to clean the memory which is allocated by new operator
- In java language, we can allocate the memory using new operator. But we don't have any process of cleaning memory in java
- In java, automatic memory cleaning process was designed to reduce the developer's burden, which is called garbage collection
- JVM is responsible to detect unused objects time to time

Following are some of the criteria JVM uses to detect unused objects:

1. When we assign null to the reference variable, then object referred by that reference variable is not referred or unused and is eligible for garbage collection Example:

```
Hello h=new Hello ();
h=null;
```

2. When you assign one reference variable to another reference variable, then the object referred by the reference variable which is on the left side of the assignment operator is not referred and is eligible for GC

```
Hello h1=new Hello();
Hello h2=new Hello();
h1=h2;
```

- When the reference variable is out of scope, then the object referred by that reference variable is not referred or unused and is eligible for GC
- JVM uses the above mentioned said technique and identifies the unused object. When JVM detects a memory shortage problem, it invokes GC by handing over list of unused objects
- GC is a thread, which is mainly responsible for cleaning the memory of unused objects

- Sometimes unused objects may be refused by GC for cleaning the memory because they may hold the references to same resources like JDBC database connection, IO stream connection, printer connection, etc
- When you release the resources which are referred by some unused objects, then you can clean the memory of those unused objects without any trouble
- JVM invokes runFinalization() method of System class which is responsible for invoking Finalize() methods on all unused objects
- JVM works as following algorithm.

Steps are as below :

JVM

- Detecting unused objects
- Prepares a list of unused objects
- When memory shortage problem is detected, the JVM will do the following
  - calls runFinalization() method
    System.runFinalization();
    Runtime.runfinalization();
  - calls gc() method
    System.gc();
    Runtime.gc();

Cleans the memory of unused objects
runFinalization() :- invokes the finalize() method on all unused objects finalize() method is of object class you need to override

That is, override finalize () method in your class when you are using any resources in your class and write the resources cleanup code inside the finalize() method.

```
class Hello {
   connection con=null;
    ---
   void m1();
    ---
}
   void m2() {
   connection con = ---------
   }
   public void finalize()
      {   con.close();
   }
class Hi {
    ---
}
```

Example using Finalize( ) method

```
package com.javabykiran.GC;
 class Hi {
     public void finalize() {
      System.out.println("Hai-finalize()");
     }
}
```

```
package com.javabykiran.GC;
class A {
    void show() {
       Hello h1 = new Hello();
       h1.m1();
       System.runFinalization();
       System.gc();
       Hello h2 = new Hello();
       h2.m1();
       System.runFinalization();
       System.gc();
     }
     public void finalize() {
     System.out.println("A-finalize()");
     }
}

package com.javabykiran.GC;
class Hello {
      void m1() {
       System.out.println("m1-begin");
        Hi h1 = new Hi();
        Hi h2 = new Hi();
        Hi h3 = new Hi();
       System.out.println("m2-begin");
       System.runFinalization();
       System.gc();
         Hi h4 = new Hi();
       System.out.println("m2-end");
      }
```



```
   public void finalize() {
      System.out.println("Hello-Finalize()");

    }
}

package com.javabykiran.GC;
class lab3 {
    public static void main(String[] args) {
        A obj = new A();
        obj.show();
        System.runFinalization();
        System.gc();
    }
}
```

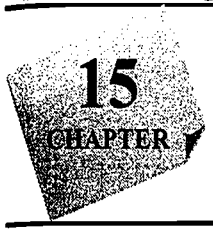
—Output---

```
m1-begin
m2-begin
m2-end
m1-begin
Hai-finalize()
m2-begin
Hai-finalize()
Hai-finalize()
Hai-finalize()
m2-end
Hai-finalize()
Hai-finalize()
Hai-finalize()
Hai-finalize()
Hello-Finalize()
Hello-Finalize()
```

Questions:

1. What is Garbage Collection in Java?
2. Can we force the garbage collector?
3. How can JVM destroy unreferenced objects?
4. Can you guarantee objects destruction?
5. Which part of the memory is involved in Garbage Collection?
6. How can we request JVM to start the garbage collection process?
7. What is the algorithm JVM internally uses for destroying objects?
8. What is the responsibility of Garbage Collector?
9. When does an object become eligible for garbage collection?
10. How many times does the garbage collector call the finalize() method for an object?
11. How to enable or disable the call for the finalize() method of exit of application?
12. What happens if an uncaught exception is thrown during the execution of finalize() method of an object?
13. What are the different ways to call garbage collector?
14. What are the types of Java Garbage Collection?

◆◆◆



Input Output Stream

Stream is nothing but sequence of bytes which makes input and output operations feasible and faster. It is called stream because it facilitates a continuous flow of data. An Input/Output stream is an input source or output destination which represents various types of sources. Java uses the concept of stream to make I/O operation fast. All the classes required for I/O operation are given in java.iopackage

In java, I/O is used to receive input data and produce output by processing the input provided

Example for File class :

```
package com.jbk.fileio;
import java.io.File;
import java.util.Date;
  public class FileInfoExample {
      public static void main(String[] args) {
          File apath = new File("C:/test/abc.txt");
          // Check if exists.
       System.out.println("Path exists? "+ apath.exists());
          if (apath.exists()) {
              // Check if 'apath' is a directory.
       System.out.println("Directory? "+ apath.isDirectory());
             // Check 'apath' is a Hidden path.
       System.out.println("Hidden? "+ apath.isHidden());
       System.out.println("Simple Name: " + apath.getName());
System.out.println("Absolute Path: "+ apath.getAbsolutePath());
            // Check file size (in bytes):
      System.out.println("Length (bytes): " + apath.length());
          // Last modify (in milli second)
           long lastMofifyInMillis = apath.lastModified();
           Date lastModifyDate = new Date(lastMofifyInMillis);
    System.out.println("Last modify date: " + lastModifyDate);
           }

        }
}
```

There are two types of streams, which are Character Stream and Binary stream.

Character Streams

- The Java platform stores character values using Unicode conventions. Character I/O stream automatically translates this internal format to and from the local character set. In Western locales, the local character set is usually an 8-bit superset of ASCII
- For most applications, I/O with character stream is no more complicated than I/O with binary streams. Input and output done with stream classes automatically translates to and from the local character set
- A program that uses character stream in place of byte streams automatically adapts to the local character set and is ready for internationalisation - all without extra effort by the programmer
- All character stream classes are descended from Reader and Writer. As with byte streams, there are character stream classes that specialise in file I/O: FileReader and FileWriter
- Classes under this hierarchy are used for reading and writing characters from file, which are text files

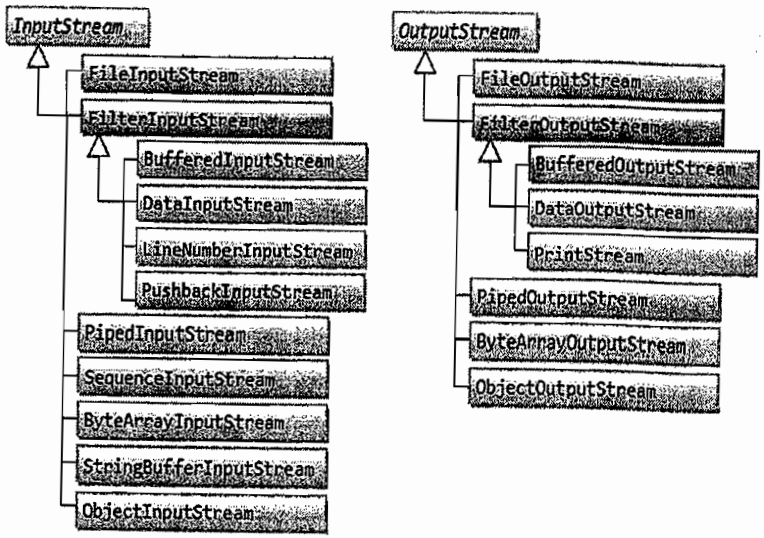
Example:

```
package com.javabykiran.IO;
 Import java.io.Filewriter;
 public class CopyCharacters {
    public static void main(String[] args) throws Exception {
       FileReader reader = null;
       Filewriter writer = null;
 try {
     reader = new FileReader("jbk.txt");
     writer = new FileWriter("coralsoft.txt");
     int c;
     while ((c=reader.read()) !=-1){
          writer.writer(c);
     }
 }finally{
     if (reader !=null){
         reader.close();
     }
     if (writer !=null){
          writer.close();
     }
   }
 }
```


- Now, we have a lot of classes in this hierarchy of reader and writer. Every class has some more features. In the above example we copy characters by characters. For that we use BufferedReader and PrintWriter classes as shown below:

Byte Streams

- Programs use byte streams to perform input and output of 8-bytes. All byte stream classes are descended from InputStream and OutputStream classes
- There are many byte stream classes. To demonstrate how byte streams work, we'll focus on the file I/O byte streams, File Input Stream and File Output Stream. Other kind of byte streams are used much in the same way; the difference is mainly in the way they are constructed
- These classes are used for binary data. We can use these classes for text files as well
- Character streams are wrapper for binary streams. This means that the character stream uses the binary stream internally
- Java uses OutputStream and InputStream for Writing data to destination and Reading data from source, respectively
- OutputStream and InputStream are abstract classes. So we can't use them directly
  - **Output Stream class:** OutputStream class is an abstract class. It is the superclass of all classes representing an output stream of bytes. Like FileOutputStream, ObjectOutputStream, etc.
  - **Input Stream class:** InputStream class is an abstract class. It is the superclass of all classes representing an input stream of bytes, like File Input Stream, Object Input Stream, etc.



Reading Data From an InputStream

- The abstract superclass InputStream declares an abstract method read() to read one data-byte from the input source
- The read() method :
  - Returns the input byte read as an int in the range of 0 to 255, or
  - Returns -1 if "end of stream" condition is detected, or
  - Throws an IOException if it encounters an I/O error
- **The read** () method returns an int instead of a byte, because it uses -1 to indicate end-of-stream
- **The read** () method blocks until a byte is available, an I/O error occurs, or the "end-of-stream" is detected. The term "block" means that the method (and the program) will be suspended. The program will resume only when the methodreturns

Writing to an Output Stream

- Similar to the InputStream, the abstract superclass OutputStream declares an abstract method write() to write a data-byte to the output sink
- *write() :* Takes an int as the input parameter
- The least-significant byte of the int argument is written out; the upper three bytes are discarded
- It throws an IOException if an I/O error occurs (for example, if output stream has been closed)
- InputStream and OutputStream are abstract classes that cannot be instantiated. You need to choose an appropriate concrete subclass to establish a connection to a physical device. For example, you can instantiate a FileInputStream or FileOutputStream to establish a stream to a physical disk file
- We can use FileInputStream and FileOutputStream for File related operations

Example:

How to Read File? [keep file in the d drive notes folder- write as a javabykiran.com in file]

```
package com;
 import java.io.FileInputStream;
 import java.io.IOException;
 public class IOLab1 {
    public static void main(String[] args){
    try {
          //windows file path - D:\notes\abc.txt //not allowed
          //invalid escape sequence
      String path = "D:\\notes\\abc.txt";
         //or path = "D:/notes/abc.txt";
```



```
      FileInputStream fis = new FileInputStream(path);
         int i=0;
         while((i=fis.read())!=-1){
            char c =(char) i;
            System.out.print(c);
         }
         fis.close();
         } catch (IOException e)
           {
           e.printStackTrace();
         }
     }
}
```

—Output---

```
www.javabykiran.com
```

Example:

How to Write to File?

```
package com;
import java.io.FileOutputStream;
import java.io.IOException;
public class IOLab2 {
public static void main(String[] args) {
   try {
        //windows file path - D:\notes\abc.txt // not allowed
        //Envalid escape sequence
        String path = "D:\\notes\\abc.txt";
        //or path = "D:/notes/abc.txt";
FileOutputStream fos = new FileOutputStream(path);
Strings ="javabykiran Add- Karvenagar,pune";
                byte b[] = s.getBytes();
        fos.write(b);
         System.out.println("successfully written");
         fos.close();
      } catch (IOException e) {
         e.printStackTrace();
    }
   }
}
```

—Output---

```
Successfully written
```

Example:

How to copy data from one file to another?

```
package com;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
 public class IOLab3 {
   public static void main(String[] args) {
    try {
 String srcpath =
 "D:\\WorkSpaces\\Eclipse_IndigoWS\\IO\\src\\com\\IOLab3.java";
       String despath = "D:\\notes\\LabProgram3.txt";
    FileInputStream fis = new FileInputStream(srcpath);
    FileOutputStream fos = new FileOutputStream(despath);
    int i=0;
       while((i=fis.read())!=-1){
    fos.write(i);
      }
      fis.close();
   } catch (IOException e) {
     e.printStackTrace();
    }
   }
}
```

There are some important classes that you would need to remember:

BufferedInputStream and BufferedOutputStream

- A BufferedInputStream adds functionality to another input stream, namely, the ability to buffer the input and to support the mark and reset methods
- When the BufferedInputStream is created, an internal buffer array is created. As bytes from the stream are read or skipped, the internal buffer is refilled as necessary from the contained input stream, many bytes at a time
- The mark operation remembers a point in the input stream and the reset operation causes all the bytes read since the most recent mark operation to be reread before new bytes are taken from the contained input stream
- The BufferedOutput Stream implements a BufferedOutputStream. By setting up such an output stream, an application can write bytes to the underlying output stream without necessarily causing a call to the underlying system for each byte written


```
package com;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
 public class IOLab5 {
   public static void main(String[] args) {
    try{
 FileInputStream fin=new FileInputStream("D:\\notes\\abc.txt");
 BufferedInputStream bufferedinputstream = new
 BufferedInputStream();
      int i=0;
     while((i=bufferedInputStream.read())!=-1){
       System.out.print((char)i);
      }
      }catch (Exception e){
       e.printStackTrace();
     }
  }
}
```

FileWriter and FileReader Classes:

The file reader and file writer read and write information from text files.

- These are the convenience classes for reading and writing character-oriented data
- So, it's good to avoid FileInputStream and FileOutputStream if you have to read and write the textual information
- The constructors of this classes assume that the default character encoding and the default byte-buffer size are appropriate

  For example: copying data from one file to another file using FileReader and FileWriter

```
package com;
import java.io.FileReader;
import java.io.IOException;
public class IOLab6 {
  public static void main(String[] args) {
  try {
  String srcpath = "D:\\jbk\\IOLab6.java";
  String despath = "D:\\notes\\LabProgram3.txt";
  FileReader fr = new FileReader(srcpath);
  FileWriter fw = new FileWriter(despath);
      int i=0;
         while((i=fr.read())!=-1){
             fw.write(i);
             System.out.print((char)i);
         }
     } catch (IOException e) {
         e.printStackTrace();
     }
  }
}
```

Reading Data from Keyboard:

There are many ways to read from the keyboard. By using the following classes we can read the data from keyboard:

- InputStreamReader
- BufferedReader
- Scanner
- Console
- By using InputStreamReader and Buffered Reader –

Example:

```
package com;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
public class IOLab7 {
  public static void main(String[] args) {
  try {
  InputStreamReader inputStreamReader =new
InputStreamReader(System.in);
  BufferedReader br = new BufferedReader(inputStreamReader);
  String name =" ";
```



```
      while(true){
      System.out.println("Enter Website :");
      name = br.readLine(); //Reads a line of text
      System.out.println("You Enter :"+name);
      if(name.equals("www.javabykiran.com")){
            break;
       }
    }
       } catch (IOException e) {
         e.printStackTrace();
       }
     }
}
```

---Output---

```
Enter Website :
www.google.com You
Enter : www.google.com
Enter Website :
www.javabykiran.com
You Enter :www.javabykiran.com
```

Example:

Addition of two numbers by scanner class in 1.5 JDK:

```
package com;
 import java.util.Scanner;
 public class IOLab8 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in)
        System.out.print("Enter 1st Number :");
        int a = scanner.nextInt() ;
        System.out.print("Enter 2st Number :");
        int b = scanner.nextInt() ;
        System.out.println("Addtion is:"+(a+b));
   }
 }
```

---Output---

```
Enter 1st Number :12
Enter 2st Number :20
Addition is :32
```

Example:
By using Console class

```
import java.io.Console;
class IOLab9 {
public static void main(String args[]){
Console c = System.console();
    System.out.print("Enter the Name:");
    String name = c.readLine();
    System.out.print("Hello "+name);
  }
}
```

---Output---

```
Enter the Name : java
Hello java
```

SequenceInputStream class:

- This class can be used to read data from multiple streams
- A SequenceInputStream represents the logical concatenation of other inputstreams
- It starts out with an ordered collection of input streams and reads from the first one until end of file is reached, where on it reads from the second one, and so on, until the end of file is reached on the last of the contained input streams

Example:
Reading two files using SequenceInputStream –

```
package com;
import java.io.FileInputStream;
import java.io.SequenceInputStream;
public class IOLab4 {
   public static void main(String[] args) {
   try{
     String path1 = "D:\\notes\\abc.txt";
```



```
      String path2 = "D:\\notes\\xyz.txt";
       FileInputStream fin1 = new FileInputStream(path1);
       FileInputStream fin2 = new FileInputStream(path2);
SequenceInputStream sis = new SequenceInputStream(fin1,fin2);
      int i =0;
      while((i=sis.read())!=-1){
       System.out.print((char)i);
    }
   }catch (Exception e)
   {
   e.printStackTrace();
  }
   }
}
```

---Output---

```
www.javabykiran.comCall us to join : 8888809416
```

◆◆◆

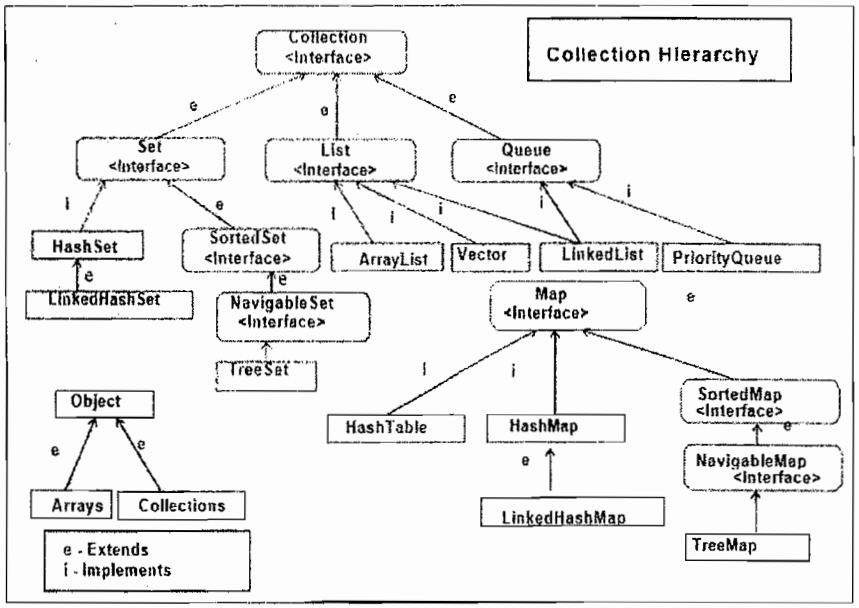
16 CHAPTER

Collection Framework

In this chapter, we will first consider how the collection framework behaves in the 4 version of Java. Once we understand this, we will learn the Collection framework in Java version 1.5 in the next chapter, where we will study features like generics, autoboxing, for each loop etc.

The collection framework in java is made up of classes and interfaces that stores and processes data in an efficient manner.

- The Sun Microsystems has introduced collection framework in Java 2.0
- The hierarchy of the whole collection framework is given below :



List:

- List is an interface that is available in the java.util package
- List is used to store a collection of elements and allows duplicates
- The term 'List is ordered' means that the order is retained in which we add elements, and will get the same sequence while retrieving elements

- List interface has three concrete subclasses:
  - ArrayList
  - LinkedList
  - Vector

Remember the following three important methods of List :

- Boolean add(Object o) ; //Append the specified elements to the end of the list
  - Return type is Boolean
  - Input type is Object
- Object get(int i); //Return the elements as Object of the specified position in the list
  - Return type is Object
  - Input type is int [index of Arraylist]
- int size(); // Return number of elements in the list
  - Return type is int [no of elements exist.]
  - Input type is nothing

ArrayList

The ArrayList extends the AbstractList and implements the List interface. It is usually slower than the other arrays, but is useful where you need to manipulate a lot in programs.

- Uses Dynamic array for storing the elements
- Duplicate elements are allowed
- Maintains insertion order
- Methods are not synchronised
- Random access as it works on index basis
- Manipulation is slow because lot of shifting needs to be done. This means that if the ArrayList has 1000 elements and we remove the 50th element, then the 51st element tries to acquire that 50th position and likewise all elements. So, moving every element consumes a lot of time

Vector

The vector class 'implements a growable array of objects. Similar to an array, it contains components that can be accessed using an integer index'. The vector may expand or contract as per the requirements.

- All methods are synchronised
- It is slower than an Arraylist

- Vector is thread-safe as it has synchronised methods
- It has capacity concept. The default capacity is 10
- It doubles when it reaches its threshold
- Size methods returns the number of elements present
- Capacity method returns the capacity which doubles after reaching the defaul capacity, which is 10
- Vector can be iterated by simple for loop, Iterator, ListIterator and Enumeration

Example :

```
package com.javabykiran.collection;
import java.util.ArrayList;
import java.util.Iterator;
/*
* @author Java By Kiran
*/
public class ArrayListTest {
     public static void main(String[] args) {
       ArrayList al = new ArrayList();
       al.add("aaa");
       al.add("bbb");
       al.add("ccc");
       // see return type of add below
       System.out.println(al.add("ddd"));
       al.size();
       System.out.println(al.size);
       //to check if Arraylist is empty
       al.isEmpty();
     System.out.println("iteration of Arraylist by for loop");
       for (int i = 0; i < al.size(); i++) {
            System.out.println(al.get(i));
        }
     System.out.println("iteration of Arraylist by Iterator");
       Iterator itr = al.iterator();
            while (itr.hasNext()) {
                Object o = itr.next(); //this is removed in
                          //jdk 1.5 and after by autoboxing
                String s = (String) o;
                System.out.println(s);
            }
```



```
System.out.println("iteration of Arraylist by List Iterator");
     ListIterator ltr = al.listIterator();
          while (ltr.hasNext()) {
             Object o = ltr.next();
             String s = (String) o;
             System.out.println(s);
             //Object op = ltr.previous();
             /*String prevStr = (String) op;
      System.out.println(prevStr);*/
     }
   }
}
```

—Output---

```
True
3
iteration of Arraylist by loop
aaa
bbb
ccc
ddd
iteration of Arraylist by Iterator
aaa
bbb
ccc
ddd
iteration of Arraylist by ListIterator
aaa
bbb
ccc
ddd
```

Example for vector :

```
package com.javabykiran.Collection;
import java.util.Vector;
/*
 * @author Java By Kiran
 */
public class VectorTest {
     public static void main(String[] args) {
        Vector v = new Vector();
```

```
        System.out.println(v.size());
        System.out.println(v.capacity());
        //we can change its default
        behaviour Vector v1 = new Vector(2);
        System.out.println(v1.size());
        System.out.println(v1.capacity());
        //now try adding elements
        v1.add("apple");
        v1.add("mango");
        v1.add("grapes");
      System.out.println(v1.size());
    // observe capacity System.out.println(v1.capacity());
    //Iterate by using Enumeration Enumeration
        enumeration=v1.elements();
    while (enumeration.hasMoreElements())  {
      String element =(String)enumeration.nextElement();
        System.out.println(element);
    }
  }
}
```

---Output---

```
0
10
0
2
3
4
apple
mango
grapes
```

- Every List interface implementing class has two constructors :
  - Default constructor
  - Constructor which takes java.util.collection

Example of ArrayList

```
ArrayList al=new ArrayList ();//with default constructor
Arraylist al1=newArrayList (al);//constructor with ArrayList
```



```
ArrayList al2=newArrayList(v);//constructor with vector
ArrayList al3=new ArrayList (c);//with java.util.collection
```

Example of Vector

```
Vector v=new Vector (); //with default
constructor v.add();//to add elements
Vector v1=new Vector (al); //with
java.util.collection Vector v2=new Vector(v); // with vector
```

- **Difference between Iterator and Enumeration:**

Iterator	Enumeration
Iterator can be used to access the elements of six subclasses of collection interface: 1. ArrayList 2. Vector 3. LinkedList 4. LinkedHashSet 5. Treeset and HashSet	Enumeration can be used for accessing elements of vector only
Using the Iterator methods hasNext() and next(), you can access the elements of collection	Using enumeration method has More Elements() and nextElements(), you can access the elements of vector one by one

Using the Iterator you can remove the elements of collection. Example :
Iterator it=al.iterator();
while(it.hasNext()) {
Sring str=it.next().toString();
if(str.equals("jbk")) {
it.remove();
}
}
	Enumeration is read only.
	i.e. you can just the read the data from vector but you cannot remove the element from vector using Enumeration

- **Difference between Iterator and ListIterator:**
  1. ListIterator is the sub interface of Iterator and therefore has more features in ListIterator than Iterator
  2. ListIterator can access elements in forward and reverse directions
  3. Adding elements by using add method
  4. Replacement of existing elements with [illegible] by using the [illegible] method
  5. Access index of element

Iterator	ListIterator
Using Iterator we can access the elements of collection only in forward direction using the hasNext() & next() methods	Using ListIterator we can access the elements in the forward direction using hasNext() and next() methods and in reverse direction using hasPrevious()and previous() methods.
You cannot add elements to collection	You can add the elements collection
You cannot replace the existing elements with new elements.	You can replace the existing elements with a new element by using void set(E e)
Using Iterator you cannot access the indexes of elements of collection	Using ListIterator you can access indexes of elements of collection using nextIndex() and previousIndex() methods
Using Iterator you can remove the elements of a collection.	Using ListIterator you can also remove the elements of collection

- **Difference between Array and ArrayList**

Array	ArrayList
The size of an Array is fixed, i.e. when you create an array with same size you cannot change dynamically	The size of an ArrayList is grow-able, i.e. once the ArrayList is created you can add any number of elements without restriction
When you create an small Array you can't add more elements and when you create a large sized array, the memory will be wasted when we don't put that	You won't get these problems with an ArrayList
When you have an array with a size 100 and has 10 elements only, then you need to repeat the loop 100 times to access 10 elements which is an unnecessary and time consuming process	When you access elements of ArrayList using iterator you won't face this time wastage problem

Array stores only similar types of elements.
You can store primitive and objects in Array
e.g: int a[]=new int[5];
Students[] stu=new Student[5]
	ArrayList store different types of elements
	You can store only objects in an ArrayList
	Student s1= new Student();
	e.g. al.add(s1);
	al.add("sri");

Note: Both Array and ArrayList store the elements internally using indexing representation and these elements can be accessed randomly

- You can access the elements of an Array using index representation
  For example:
  a [1]
- You cannot access the elements of an ArrayList using index representation. For example:
  al [0] is not allowed but al.get(0) is allowed

Comparison between ArrayList ,Vector and LinkedList:

Topic	ArrayList	Vector	LinkedList
Legacy or collection Framework	Collection framework class added in java 2	Legacy class but revised in java 2	Collection framework class added in java 2
Synchronised	Not Synchronised	Synchronised	Not Synchronised
Storage Representation	Index Representation	Index Representation	Node Representation
Accessing Elements	Iterator. List Iterator	Enumeration. Iterator List Iterator	Iterator. List Iterator
Access Type	Random Access	Random Access	--

Set:

- A set is used to store the collection of elements without duplicates
- It is an unordered collection which means that order is not maintained while storing elements and while retrieving them, we may not get the same order that we had put them in
- A set cannot be iterated by using ListIterator but by Iterator
- There are four classes which implement Set interface:
  - HashSet
  - LinkedHashSet
  - TreeSet
  - SortedSet - It uses hash table to store elements. Duplicates are not allowed

Example: Consider the following program of HashSet

```
package com.javabykiran.Collection;
import java.util.HashSet;
import java.util.Iterator;
/*
* @author Java By Kiran
*/
public class HashSetTest {
 public static void main(String[] args) {
     HashSet hs = new HashSet();
      hs.add("aaaa");
      hs.add("bbbb");
// aaaa will not be allowed as the set doesn't allow
duplicates
   hs.add("aaaa");
   hs.add("cccc");
// see size
System.out.println(hs.size());

// we can print values for testing
System.out.println(hs);

// By using iterator
Iterator itr = hs.iterator();
while (itr.hasNext())
   System.out.println(itr.next());
   }
 }
```

---Output---

```
3
[bbbb, aaaa, cccc]
bbbb
aaaa
cccc
```

Comparison between HashSet , LinkedHashSet and TreeSet:

Topic	HashSet	LinkedHashSet	TreeSet
Duplicate	Not Allowed	Not Allowed	Not Allowed
Ordering	Unordered	Maintains insertion order	Maintains sorting order
Null	Allow	Allow	Do not allow
Accessing Elements	Iterator	Iterator	Iterator
Thread Safety	No	No	No

Queue:

- The Queue is an interface of a subtype of the Collection interface
- It represents an ordered list of objects just like a List, but its intended use is slightly different
- A queue is designed to have elements inserted at the end of the queue, and elements removed from the beginning of the queue. Just like a queue in a supermarket or any shop
- Following are the concrete subclasses of the Queue interface:
  - ArrayDeque
  - PriorityQueue
  - PriorityblockingQueue
  - LinkedBlockingQueue
- Each Queue method exists in two forms:
  - One throws an exception if the operation fails
  - The other returns a special value if the operation fails (either null or false, depending on the operation)

Deque:

- It is a sub-interface of Queue interface
- A double-ended-queue is a linear collection of elements that supports the insertion and removal of elements at both end points
- It defines methods to access the elements at both ends of the Deque instance
- Methods for insertion, removal and retrieval of Deque elements are summarised in the following table:

Type of Operation	First Element (Beginning of the Deque instance)	Last Element (End of the deque instance)
Insert	addFirst(e) offerFirst(e)	addLast(e) offerLast(e)
Remove	removeFirst(e) pollFirst(e)	removeLast(e) pollLast(e)
Examine	getFirst(e) peekFirst(e)	getLast(e) peekLast(e)

Map:

- A map is used to store the key-value pair
- It doesn't allow duplicate keys but duplicate values are allowed
- It has the following concrete subclasses:
  - HashMap
  - WeakHashMap
  - HashTable
  - IdentityHashMap

- TreeMap
- LinkedHashMap

HashMap:

- A HashMap is class which implements the Map interface
- It stores values based on key
- It is unordered, which means that the key must be unique
- It may have null key-null value
- For adding elements in HashMap we use the put method
- Return type of put method is Object
- It returns the previous value associated with key or null if there was no mapping for key

Example

```
package com.javabykiran.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
/*
* @author Java By Kiran
*/
public class HashMapTest {
 public static void main(String[] args) {
     // Creation of HashMap
         HashMap hm = new HashMap();
     // Adding elements with key
          hm.put("101", "java");
          hm.put("102", ".Net");
     // Will print null
          Object o = hm.put("103", "C++");
          System.out.println(o);
    // Will print previous value as it is duplicate //value
          Object o1 = hm.put("103", "C");
          System.out.println(o1);

 //1. Retrieving elements from HashMap by using iterator
   System.out.println("======By using Iterator======");
         Set s = hm.keySet(); // set s contains all keys
         Iterator itr = s.iterator();
      while (itr.hasNext()) {
```



```
        String key = (String) itr.next();
        System.out.println("Key :"+key);
        System.out.println("Value :"+ hm.get(key));
      }

    // 2. Retrieving elements from HashMap by using Map.Entry
     System.out.println("====== By using Map.Entry ======");
    // Get a set of the entries
       Set set = hm.entrySet();
    // Get an iterator
       Iterator it = set.iterator();

    // Display elements
       while(it.hasNext()) {
          Map.Entry me = (Map.Entry) it.next();
         System.out.print(me.getKey()+ ": ");
         System.out.println(me.getValue());
       }
   }
}
```

—Output---

```
null
C++
====== By using Iterator ======
Key :103
Value :C
Key :101
Value :java
Key :102
Value :.Net
====== By using Map.Entry ======
103: C
101: java
102: .Net
```

LinkedHashMap:

A LinkedHashMap is a 'hastable and linked list implementation of the map interface with a predictable iteration order.

- It is the same as HashMap except it maintains an insertion order i.e. ordered

Consider the same above programs using LinkedHashMap. Observe the output. Change one line in the above example

In the program, change Instead of

```
HashMap hm=new HashMap ();
Use below
// Creation of LinkedHashMap
LinkedHashMap hm = new LinkedHashMap()
```

---Output---

```
null
C++
====== By using Iterator ======
Key :101
Value :java
Key :102
Value :.Net
Key :103
Value :C
====== By using Map.Entry ======
101: java
102: .Net
103: C
```

TreeMap:

- The TreeMap is a class which implements **NavigableMap** interface which is the sub-interface of **SortedMap**
- It stores values based on key
- It is ordered but in an Ascending manner
- Keys should be unique
- It cannot have null key at run time but can have null values because the interpreter will not understand how to sort null with other values
- Consider the program of storing elements using TreeMap

Example:

```
package com.javabykiran.Collection;
import java.util.Iterator;
import java.util.Set;
import java.util.TreeMap;
/*
* @author Java By Kiran
*/
public class TreeMapTest {
   public static void main(String[] args) {
        // Creation of TreeMap
        TreeMap tm = new TreeMap();
       // adding elements with key
        tm.put("103", "java");
       tm.put("102", ".Net");
    Object o = tm.put("101", "C++");
     // below will print null
      System.out.println (o);
      Object o1 = tm.put("101",
      "C"); System.out.println (o1);
      // retrieving elements from
      TreeMap Set s = tm.keySet();
     // set s contains all keys
     Iterator itr = s.iterator();
       while (itr.hasNext()) {
           String key = (String) itr.next();
         System.out.println("Key :" + key);
         System.out.println("Value :" + tm.get(key));
        }
     // Try putting null value
     // will not throw compile time error
     // but gives runtime error uncomment below
     // tm.put(null, null);
  }
}
```

---Output---

```
null C++
Key :101
Value :C Key :102
```

```
Value :.Net Key :103
Value :java
```

Hashtable:

- Hashtable is a class which implements Map interface and extends Dictionary class
- It stores values based on key
- It is unordered and the key should be unique
- **It cannot have null keys or null values. It gives runtime error if we try to add any null keys or values but will not show an error at compile time**
- It has synchronised methods and slower than hashmap
- Consider the program of storing elements using Hashtable

Example:

```
package com.javabykiran.Collection;
import java.util.Hashtable;
/*
* @author Java By Kiran
*/
public class HashTableTest {
    public static void main(String[] args){
        Hashtable ht = new Hashtable();
        ht.put("ind", "India");
        ht.put("bhu", "Bhutan");
        ht.put("ind", "India");
    //the below will print size of ht by
    //ignoring the duplicate key
 System.out.println("size of hashTable> "+ht.size());  // 2
        //OK - compile time
        //NOT OK AT RUNTIME
        //ht.put(null, "India");
        //OK - compile time
        //NOT OK AT RUNTIME
        //ht.put(null, null);
        //OK - compile time
        //NOT OK AT RUNTIME
        //ht.put("ind", null);
    System.out.println(ht.size()); // 2
   }
 }
```


---Output---

```
size of hashTable > 2
Exception in thread "main" java.lang.NullPointerException
at java.util.Hashtable.put(Unknown Source)
at com.javabykiran.Collection.HashTableTest.main(HashTableTest.java:23)
```

Comparison between HashMap, LinkedHashMap, TreeMap and HashTable:

Topic	HashMap	LinkedHashMap	TreeMap	HashTable
Duplicate Key	Not Allowed	Not Allowed	Not Allowed	Not Allowed
Ordering	Unordered	Maintains insertion order	Maintains in Accessing order	Unordered
Null (Key Value)	Allow	Allow	key Not allowed but value is Iterator	Not Allowed
Accessing Elements	Iterator	Iterator	Iterator	Iterator
Thread Safety	No	No	No	Yes

Properties Class:

It can be used to get properties from xml file or to store data in xml file

- All methods of this class are synchronised, So it is a thread-safe class
- It is since jdk 1.0
- It is the subclass of HashTable
- It stores the key-value pair, but both as string
- It can be used to get property value based on property key
- It can also be used to get properties of system
- It provides easy maintenance

```
package com.javabykiran.Collection;
 import java.io.FileOutputStream;
 import java.util.Iterator;
 import java.util.Properties;
 import java.util.Set;
 /*
 * @author Java By Kiran
 */
 public class PropertiesTest {
     public static void main(String[] args) {
         Properties properties = new Properties();
```

```
        // Same way we can put elements
        properties.put("101", "java");
        properties.put("102", "C");
        properties.put("103", "C++");
        properties.put("104", ".Net");
        // to retrieve the elements.
        Set s = properties.keySet();
        Iterator itr = s.iterator();

    while (itr.hasNext()) {
     System.out.println(properties.get((String)itr.next()));
     }
     // to store in xml file
      try {
FileOutputStream fout = new FileOutputStream("D:\\prop.xml");
     properties.storeToXML(four,"key-value pair");
         } catch (Exception e) {
               e.printStackTrace();
         }
    }
}
```

---Output---

```
.Net
C++
C
Java
```

An XML file is created as shown below in the D drive I have just printed

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPEproperties SYSTEM "http://
java.sun.com/dtd/properties.dtd">
<properties>
<comment>key-value pair</comment>
<entry key="104">.Net</entry>
<entry key="103">C++</entry>
<entry key="102">C</entry>
<entry key="101">java</entry>
</properties>
```


Collections Class:

- All methods of this class are static. Constructor is private
- This class has methods to give extra features to Collection Hierarchy. For example: sort, binary Search, reverse, finding min value, max Value and making list read only.
- Those collection classes which are not synchronised will be synchronised by using methods - synchronizedList (), synchronizedMap ()

Example

```
package com.javabykiran.Collection;
import java.util.ArrayList;
import java.util.Collections;
/*
* @author Java By Kiran
*/
public class CollectionsTest {
   public static void main(String[] args) {
      ArrayList al = new ArrayList();
      al.add("Java");
      al.add("By");
      al.add("Kiran");

    System.out.println("before sorting " + al);
    Collections.sort(al);
    System.out.println("after sorting" + al);
    //To make arraylist object synchronised
     Collections.synchronizedList(al);
   }
}
```

---Output---

```
before sorting [Java, By, Kiran]
after sorting [By, Java, Kiran]
```

Note: To sort custom objects. For example: One student having age and location

We add students in Arraylist and we want to sort these students on the basis of their ages If we use only Collections, it will sort by addresses of the students' object, which we don't want.

This is not possible with the ordinary sort method. For that we need to use **Comparable** or **Comparator** interface, where we need to specify our sorting criteria. Then we can pass that criteria into the sort method.

Comparable Interface:

Comparable interface is primarily used to sort out lists or arrays of custom objects. It contains only one method.

- The class whose object has to be compared has to implement Comparable interface and has to override compareTo () method in the following format
- public int compareTo(Object 0bj)
- Comparable provides only one way of comparison
- You can sort the elements on based on single data member only. For instance it may be rollno, name, age or anything else

Consider the following program of sorting students on the basis of age:

```
package com.javabykiran.Collection;
/*
 * @author Java By Kiran
*/
class Student implements Comparable{
        String name;
        int age;
    public int compareTo(Object obj){
         Student s = (Student)obj;
          if(age==s.age)
            return 0;
         else if(age>s.age)
            return -1;
      }
}
package com.javabykiran.Collection;
import java.util.*;
/*
* @author Java By Kiran
*/
public class CollectionLab9 {
   public static void main(String args[]){
        ArrayList al = new ArrayList ();
        Student s1 = new Student ();
          s1.name = "ABC";
          s1.age = 24;
        Student s2 = new Student ();
          s2.name = "XYZ";
          s2.age = 21;
```



```
        Student s3 = new Student ();
           s3.name = "PQR";
           s3.age = 19;
         al.add(s1);
         al.add(s2);
         al.add(s3);
    Collections.sort(al);
          Iterator itr=al.iterator();
             while(itr.hasNext()){
         Student st= (Student) itr.next ();
         System.out.println(st.name+"--"+st.age);
       }
   }
}
```

—Output---

```
PQR--19
XYZ--21
ABC--24
```

Comparator interface:

Comparator interface is used when you have to order user-defined class objects and contains two methods.

- We need to write separate class by implementing Comparator interface by overriding the following method:
  public int compare(Object o1,Object o2)
- You can write multiple comparators to compare objects in different ways
  To use compare() method of Comparator interface for sorting, use following method:
  Collections.sort(al , new NameComparator());

Example:

```
package com.javabykiran.Collection;
/*
* @author Java By Kiran
*/
class Student {
    String name;
    int age;
}
```

```
package com.javabykiran.Collection;
/*
* @author Java By Kiran
*/
import java.util.*;
public class AgeComparator implements Comparator{
   public int compare(Object o1, Object o2) {
        Student s1=(Student)o1;
        Student s2=(Student)o2;
        if(s1.age==s2.age)
           return 0;
        else if(s1.age>s2.age)
           return 1;
        else return -1;
   }
}
```

```
package com.javabykiran.Collection;
 /*
 * @author Java By Kiran
  */ import java.util.Comparator;
public class NameComparator implements Comparator {
    public int compare(Object o1, Object o2) {
         Student s1=(Student)o1;
         Student s2=(Student)o2;
        return s1.name.compareTo(s2.name);
   }
 }
```

```
package com.javabykiran.Collection;
/*
* @author Java By Kiran
*/
import java.util.*;
public class CollectionLab10 {
    public static void main(String args[]){
      ArrayList al = new ArrayList();
      Student s1 = new Student();
          s1.name = "ABC";
          s1.age = 24;
```



```
      Student s2 = new Student();
         s2.name = "XYZ";
         s2.age = 21;
      Student s3 = new Student();
         s3.name = "PQR";
         s3.age = 19;
       al.add(s1);
       al.add(s2);
       al.add(s3);
   System.out.println("Sorting by Name...");
   Collections.sort(al,new  NameComparator());
         Iterator itr=al.iterator();
          while(itr.hasNext()){
               Student st=(Student)itr.next();
               System.out.println(st.name+"  "+st.age);
          }
   System.out.println("sorting by age...");
     Collections.sort (al,new AgeComparator());
     Iterator itr2=al.iterator();
      while (itr2.hasNext()){
         Student st= (Student)itr2.next();
         System.out.println(st.name);
         System.out.println(st.age);
      }
    }
}
```

---Output---

```
Sorting by Name...
ABC 24
PQR 19
XYZ 21
Sorting by age...
PQR 19
XYZ 21
ABC 24
```

The following table gives the properties of derived classes:

DATA STRUCTURE	INTERFACE	DUPLICATES	METHODS AVAILABLE	DATA STRUCTURE
HASHSET	Set	Unique elements	equals(), hashCode()	Hashtable
LINKEDHASHSET	Set	Unique elements	equals(), hashCode()	Hashtable, doubly-linked list
TREESET	SortedSet	Unique elements	equals(), compareTo()	Balanced Tree
ARRAYLIST	List	Duplicates allowed	equals()	Resizable array
LINKEDLIST	List	Duplicates allowed	equals()	Linked list
VECTOR	List	Duplicates allowed	equals()	Resizable array
HASHMAP	Map	Unique keys	equals() and hashCode()	Hash table
LINKEDHASHMAP	Map	Unique keys	equals() and hashCode()	Hash table and doubly-linked list
HASHTABLE	Map	Unique keys	equals(), hashCode()	Hash table
TREEMAP	SortedMap	Unique keys	equals(), compareTo()	Tree Map

Questions:

1. How can an Array List be synchronised without using Vector?
2. What is the difference between an ArrayList and a vector?
3. Imagine that an Employee class is present and its objects are added in an ArrayList. Now I want the list to be sorted on the basis of the employeeID of Employee class. What are the steps?
4. What is difference between List and a Set?
5. What are the classes implementing the List interface?
6. What is difference between a HashMap and a HashTable?
7. What all classes implement Set interface?
8. What is the difference between Arrays and ArrayList ?
9. When to use ArrayList or LinkedList?
10. What are the advantages of iterating a collection using iterator?
11. Which design pattern does Iterator follow?
12. Why is it preferred to declare:
    List list = new ArrayList(); instead of ArrayList = new ArrayList();
13. Which data structure does HashSet implement?
14. What is a ConcurrentHashMap?
15. What is the difference between iterator access and index access?
16. How to make a List (ArrayList,Vector,LinkedList) read only?
17. How to sort a list in reverse order?

18. Can a null element be added to a Treeset or HashSet?
19. How to sort list of strings - case insensitive?
20. What is WeakHashMap?
21. Which is faster to iterate, LinkedHashSet or LinkedList?
22. Arrange in the order of speed - HashMap, Hash Table, Collections. synchronized Map, concurrent Hashmap
23. What is identity HashMap?
24. What is the difference between List < ?>andList < Object > in Java?
25. What is Bounded and Unbounded wildcards in Generics?
26. Can you pass List< String > to a method which accepts List< Object>?
27. How does HashMap works in Java?
28. What are the advantages of using Generics?
29. What do you understand by iterator fail-fast property?
30. How to avoid ConcurrentModificationException while iterating a collection?
31. What is difference between fail-fast and fail-safe?
32. Can we use any class as Map key?
33. What are the differences between Queue and Stack in Java?
34. How can we make Hashmap synchronised?
35. What are IdentityHashMap and WeakHashMap?
36. What is NavigableMap in Java? What are its benefits over Map?
37. What is the difference between the Iterator and ListIterator?
38. List the differences between Vector and ArrayList.
39. Which two methods do you need to implement for key Object in HashMap?
40. How to make a collection read only?
41. What is BlockingQueue?
42. When do you use ConcurrentHashMap in Java?
43. Which implementation of the List interface provides for the fastest insertion of a new element into the middle of the list?
44. Which collection classes provide random access of its elements?
45. What will happen if we put a key object in a HashMap whichis already there?
46. What will be the problem if you don't override hashcode() method?
47. How to convert a string array to Arraylist?
48. How can you suppress unchecked warning in Java?
49. What is the Properties class?
50. What is Enumeration?
51. How to iterate legacy classes?
52. What is advance feature of JDK 1.5 used in the collection?

◆◆◆

17 CHAPTER

Collection Revisited

This chapter is specially designed to know what all there is in Collection at very advanced level and how it is used in the industry and in business projects. In this chapter you we will explore the following:

- What is introduced in jdk 1.5, 1.6 and 1.7?
- How does the compiler treat Collection framework in advanced versions of Java?

What are the most common interview questions and their solutions. ArrayList

- Size means the number of elements present in ArrayList
- Capacity means the capability to store elements but in Java it is not restricted

When we write

ArrayList al=new ArrayList();

by default, the capacity is 10

This you can see in source code of ArrayList.class.

See the code below from sun microsystem:

```
/*
* Constructs an empty list with an initial capacity of 10
*/
 public ArrayList() {
      this(10);
 }
```

- So, we can specify our own capacity as well
- ArrayList al=new ArrayList(6);
- But remember, this does not mean it only takes six elements

- To add primitives in ArrayList of jdk 1.4 we have add method which asks for the Object

Example:

```
package com.javabykiran.Collection.revisited;
import java.util.ArrayList ;
 /*
 * @author Java By Kiran
 */
 public class ArrayListTest {
```



```
  public static void main(String[] args){
    ArrayList arrayList = new ArrayList();
    //Below is not an error as we put object type
    arrayList.add(String.valueOf(34));
  }
}
```

- Here, in this case, we must change primitive to Wrapper class as given below. For more details read the previous collection chapter

  //Below is not an error as we put Object type arrayList.add (String.valueOf (34));

  OR

  arrayList.add (new Integer (34));

JDK 1.5

- Following are concepts introduced in new version jdk 1.5:
  a. Generics
  b. Auto Boxing
  c. Enhance for loop or for each loop

Generics

- We use generics for type safety, which means only one type of element will get added
- It does not means Sun Microsystem has changed the add method of class ArrayList
- It is same as before and only changes done are in the compiler. In this case, the compiler will do more work on your behalf
- One of compiler tasks, besides compilation, is addition of extra code

Declaring ArrayList in 1.5

- ArrayList <Integer> al = new ArrayList <Integer>();
- This means that the compiler will start giving errors if we try to put any elements other than an integer
- Point to be remembered: If we look at the byte code, these angular brackets are not considered. This is just for the compiler to give errors
- In these <>, classes under Object hierarchy are allowed not primitives
- In both sides, the class type must be same, No superclass or subclass combination is allowed.

  ArrayList <Object> al = new ArrayList <Integer>(); // not allowed

  ArrayList <int> al = new ArrayList <int>(); // not allowed – primitives

In jdk 1.7 right side not needed

Like below :

ArrayList <Integer> al = new ArrayList<>();

Adding elements in jdk 1. 5

1. al.add (33) - after the above declaration, we are able to directly add 33 primitive and there is no need of typecasting. This is one of the advantages of generics
2. Compiler will convert this line on the fly to al.add(new Integer(33))

Retrieving elements in jdk 1.5

1. al.get(index) in this case, in the earlier versions we must follow these three:
   Object obj=al.get (5)
   Integer i = (Integer) obj;
   int val=i.intValue ();
2. int val = al.get (index)- This is also one of the extra features of compiler. Object gets converted into the Integer. If we use the decompiler then byte code we see this in the byte code
3. Another way is by using iterator:

```
Iterator <Integer> itr=al.iterator ();
  while (itr.hasNext()) {
     int element =itr.next(); [ Due to generics we return int directly]
   }
```

foreach loop

The foreach loop in java is used for traversing arrays or collection elements. It is sometimes referred to as the For In loops

- This for loop is introduced in jdk1.5 refer jdk 1.5 chapter for more details
- In this case we need to remember these points below:
  - No increment decrement
  - No size calculation
  - Must use generics for iterating
  - ArrayList<String > al=new ArrayList<String >();

```
for (String element : al){
   System.out.println (element)
}
```

  - This for loop is for compiler. It always gets converted into a simple ordinary for loop
  - ArrayList<Student> al=new ArrayList<Student>();

```
for (Student stu: al) {
   System.out.println (stu.age);
   System.out.println (stu.loc);
}
```

  - This way of iterating element is most commonly used in industrial projects

Autoboxing

- Autoboxing means automatic conversion of objects
- This happens in generics; there we don't need type casting
- Do not think it is related to collection, it's just type
  - Object o =3; // OK in jdk 1.5 because compiler will convert it to Integer
  - Integer i=78; // OK
  - int ii=90; // OK

Business Use

Consider the following requirement:

Question: One student has age and location which needs to be added in the Arraylist, and give to ten students to other department.

```
package com.javabykiran.Collection.revisited;
/*
 * @author Java By Kiran
 */
public class Student {
    int age;
    String loc;
}
```

Answer: First we design student as below:

Example 1: Write class which prepares Students and returns from one of method.

```
package com.javabykiran.Collection.revisited;
import java.util.ArrayList;
/*
* @author Java By Kiran
*/
public class PrepareStudent {
    ArrayList<Student> buildStudents()  {
       Student student1 = new Student();
       student1.age = 28;
       student1.loc = "Pune";
       Student student2 = new Student();
       student2.age = 30;
       student2.loc = "Nagpur";
      // Students added in arraylist
     // remember al knows students' address but not age and loc
 // Here we are placing addresses not actual data in ArrayList
ArrayList<Student> arrayListStu = new ArrayList<Student>();
```

```
        arrayListStu.add(student1);
        arrayListStu.add (student2);
        return arrayListStu;
    }
}
```

Now, write the class which gets students from above class's buildStudents () method and iterate over them to get students age and location.

Example 2:

```
package com.javabykiran.Collection.revisited;
import java.util.ArrayList;
/*
* @author Java By Kiran
*/
public class RetriveStudents {
    void fetchStudent() {
        PrepareStudent prepare = new PrepareStudent ();
        ArrayList<Student> listStu = prepare.buildStudents();
        // enhance for loop -- for each loop
            for (Student student : listStu) {
             System.out.println("students age: " + student.age);
            System.out.println("students location: "+ student.loc);
        }
    }
}
```

Write client class to test our logic.

```
package com.javabykiran.Collection.revisited;
/*
* @author Java By Kiran
*/
public class TestStudentsOperation {
    public static void main(String[] args) {
        // prepare students
        PrepareStudent sp=new PrepareStudent();
        sp.buildStudents();
        //retrieve students
        RetriveStudents retrivestudents = new RetriveStudents();
            retrivestudents.fetchStudent ();
    }
}
```


--Output---

```
ArrayList >>[com.javabykiran.Collection.revisited.Student@182f0db,
com.javabykiran.Collection.revisited.Student@192d342]
students age : 28
students location : Pune
students age : 30
students location : Nagpur
```

Consider requirement:

Question: One student, who has age, location and many phone numbers, needs to be added in ArrayList and given to other department which has ten students.

Example 3:

```
package com.javabykiran.Collection.revisited;
import java.util.ArrayList;
/*
 * @author Java By Kiran
*/
public class Student {
     int age;
     String loc;
    ArrayList<String> alMobNo;
}
```

```
package com.javabykiran.Collection.revisited;
import java.util.ArrayList;
/*
 * @author Java By Kiran
*/
public class StudentPrepare {
     ArrayList<Student> buildStudents(){
        Student student1 = new Student();
        student1.age = 28;
        student1.loc = "Pune";
    //creates list of mobile numbers of student1
  ArrayList<String> alMobNoStu1 = new ArrayList<>();
      alMobNoStu1.add("8888809416");
      alMobNoStu1.add("9552343698");
      student1.alMobNo = alMobNoStu1;
      Student student2 = new Student();
      student2.age = 28;
```

```
    student2.loc = "Pune";
  //creates list of mobilenumbers of student2
  ArrayList<String> alMobNoStu2 = new ArrayList<>();
      alMobNoStu2.add("8234876642");
      alMobNoStu2.add("3455455441");
      student2.alMobNo =  alMobNoStu2;
   // Students added in arraylist
  //remember al knows students' address not their age and loc
  //Here we are placing address, not actual data in ArrayList
    ArrayList<Student> arrayListStu = new ArrayList<>();
       arrayListStu.add(student1);
       arrayListStu.add (student2);
       System.out.println("ArrayList >> "+ arrayListStu);
        return arrayListStu;
    }
}
```

For Fetching Students use the method given below:

Example 4:

```
void fetchStudent() {
  StudentPrepare prepare = new StudentPrepare();
  ArrayList<Student> listStu = prepare.buildStudents();
     //Enhance for loop -- foreach loop
      for(Student student : listStu) {
       System.out.println("students age: " + student.age);
       System.out.println("students location: " + student.loc);
          ArrayList<String> alMobNo  = student.alMobNo;
              for (String mobNo : alMobNo) {
              System.out.println("al mob no" + mobNo);
            } // end 1st For loop
      }// end 2nd for loop
}
```

---Output---

```
ArrayList >> [com.javabykiran.Collection.revisited.Student@182f0db,
com.javabykiran.Collection.revisited.Student@192d342]
students age : 28
students location : Pune
```



```
al mob no 8888809416
al mob no 9552343698
students age : 30
students location : Nagpur
al mob no 8234876642
al mob no 3455455441
```

Consider the following requirement:

Example 5:

Question: One student has age, location and a phone having two types, one landline number and many mobile numbers. It needs to be added in ArrayList and given to other department which has ten students.

First we design Phone as below:

```
package com.javabykiran.Collection.revisited;
import java.util.ArrayList;
/*
 * @author Java By Kiran
 */
public class Phone {
  int landLineNo;
  ArrayList<String> mobNos;
```

First we design Student as below having phone attribute: This also shows one to one relationship.

```
package com.javabykiran.Collection.revisited;
import java.util.ArrayList;
/*
  * @author Java By Kiran
*/
public class Student {
   int age;
   String loc;
   Phone p;
}
```

Try adding students and fetching.

18 CHAPTER

Serialization

Serialization

- Writing objects States into some other sources like Hard Disk, Socket, file etc., is called Serialization.
- Serialization is needed when there is a problem in sending data from one class to another.
  Where the other class is on a different location or Hard Disk. i.e. in Distributed Systems
- The reverse operation of serialisation is called deserialization
- The String Class and all wrapper classes implement serializable interface by default
- Serializable interface is also marker interface which provides the capability of Serialization to your class. So, we should implement a serializable interface if we want to send the state of an object over the network

Consider the following program of Serialization and Deserialization:

Example 1:

```
package com.javabykiran;
import java.io.Serializable;
public class Student implements Serializable {
  String name;
  int age;
  String location;
}
```

```
package com.javabykiran;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
public class SerializeStudent {
  public static void main(String[] args) {
     Student s = new Student();
     s.name = "abc";
     s.age = 25;
     s.location = "pune";
     try{
FileOutputStream fos = new FileOutputStream ("D:\\state.txt");
  ObjectOutputStream oos = new ObjectOutputStream(fos);
```



```
   oos.writeObject(s);
   oos.flush();
   System.out.println("Serialization is done...");
   }catch (Exception e) {
     e.printStackTrace();
    }
   }
}
```

```
package com.javabykiran;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
public class DeserializeStudent {
   public static void main(String[] args){
     try{
 FileInputStream fis = new FileInputStream("D:\\state.txt");
  ObjectInputStream ois = new ObjectInputStream(fis);
   Object o = ois.readObject(); //Read the object
   Student s=(Student)o;//convert to student
      System.out.println(s.name);
      System.out.println(s.age);
      System.out.println(s.location);
       }catch(Exceptione){
         e.printStackTrace();
        }
      }
}
```

---Output---

```
abc
25
pune
```

- The ObjectOutputStream and ObjectInputStream are used to serialize and de-serialize objects respectively.
- If the superclass implements serializable interface, then all its subclasses will be serializable by default
- All static members of class are not serialized because static members are related to class only, not to object
- If we don't want to serialize some fields of class then we use the **transient keyword**. If any member is declared as transient then it won't be serialized

- In case of array or collection, all the objects of array or collection must be serializable; if any object is not serializable then the serialization will fail
- The serialization associated with each serializable class has a version number called Serial Version UID
- It is used during de-serialization to verify that the sender and receiver of a serialized object have loaded classes for that and are compatible with respect to serialization
- If the receiver is loaded with different version of a class that has different serial version UIDs than the corresponding sender's class, then de-serialization will result in an invalid Class Exception
- A Serializable class can declare its own serial version UID explicitly by declaring a field named serial version UID that must be static, final and of type long
- If a superclass variable is made transient, then after de-serialization, it gives default value like zero or null

Consider the above same program in which we don't want to serialize the age of a student:

Example 2:

```
package com.javabykiran;
import java.io.Serializable;
public class Student implements Serializable {
  String name;
  transient int age;
  String location;
}
```

```
package com.javabykiran;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
public class SerializeStudent {
  public static void main(String[] args) {
 Student s = new Student();
    s.name = "abc";
    s.age = 25;//wont be serialized
    s.location = "pune";
    try{
FileOutputStream fos = new FileOutputStream("D:\\state.txt");
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(s);
    oos.flush();
    System.out.println("Serialization is done...");
    }catch (Exception e) {
```



```
    e.printStackTrace();
    }
  }
}
```

```
package com.javabykiran;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
public class DeserializeStudent {
  public static void main(String[] args) {
    try{
   FileInputStream fis = new FileInputStream("D:\\state.txt");
   ObjectInputStream ois = new ObjectInputStream(fis);
   Object o = ois.readObject(); //Read the object
   Student s = (Student)o;//convert to student
   System.out.println(s.name);
   System.out.println(s.age); //wont be deserialize,will
  printdefault value
   System.out.println(s.location);
    }catch(Exception e){
        e.printStackTrace();
    }
  }
}
```

---Output---

```
Abc
0     // Not Serialized as transient
Pune
```

Externalization

Externalization is serialization, except that it is an alternative for it.

- Externalization is nothing but serialization but by implementing Externalizable interface we can persist and restore the object
- To externalize your object, you need to implement Externalizable interface that extends the Serializable interface
- Here we have complete control of what to serialize and what not to serialize
- But with serialization, the identity of all the classes, its superclasses, instance variables and then the contents for these items, is written to the serialization stream
- Externalizable interface is not a marker interface and it provides two methods: writeExternal and readExternal

- How does serialization happen? JVM first checks for the Externalizable interface and if the object supports Externalizable interface, then it serializes the object using write External method. If the object does not support Externalizable but implements Serializable, then the object is saved using ObjectOutputStream
- In case of Serializable, jvm has full control of serializing object, while in case of Externalizable, the application gets the control for persisting objects
- writeExternal(), readExternal() methods provides complete control on format and content of serialization process

Listed here are the differences between Serialization and Externalization:

- In case of serializable, default serialization process is used. While in case of externalization, custom serialization process is used which is implemented by application
- JVM gives a call back to readExternal() and writeExternal of Externalizable interface (application implementation) for restoring and writing objects
- Though Externalizable provides complete control, list also has challenges to serialize super type state and take care of default values in case of transient variable and static variables in Java
- If used correctly, Externalizable interface can improve the performance of the serialization process

Example 3:

```
package com.javabykiran.externalization;
import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import  java.io.ObjectOutput;
public class Person implements Externalizable{
  String name;
  int age;

  @Override
public void writeExternal(ObjectOutput out)throws IOException {
    out.writeObject(name);
    out.writeInt(age);
  }

  @Override
  public void readExternal(ObjectInput in) throws IOException,
       ClassNotFoundException  {
       name = (String)in.readObject();
```



```
       age = in.readInt();
    }
 }
```

Example 4:

```
package com.javabykiran.externalization;
import java.io.FileOutputStream;
import java.io.ObjectOutputStream;
public class SerializeP {
  public static void main(String[] args)  {
      try{
    Person p = new Person();
     p.name = "kiran";
     p.age= 24;
 FileOutputStream fos =new FileOutputStream("D:\\person.txt");
 ObjectOutputStream oos = new ObjectOutputStream(fos);
   oos.writeObject(p);
   System.out.println("successful");
    }catch(Exception e){
        e.printStackTrace();
     }
   }
}
```

```
package com.javabykiran.externalization;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
public class DeserializeP {
  public static void main(String[] args) throws Exception {
   FileInputStream fis = new
   FileInputStream("D:\\person.txt");
   ObjectInputStream ois = new ObjectInputStream(fis);
   Person p =(Person) ois.readObject();
   System.out.println(p.name);
   System.out.println(p.age);
  }
}
```

---Output---

```
Kiran
24
```

Questions :

1. What is Serialization?
2. What is the use of serial version UID?
3. What is the need of Serialization?
4. Other than Serialization, what are the different approaches to make object Serializable?
5. Do we need to implement any method of Serializable interface to make an object serializable?
6. What happens if the object to be serialized includes references to other serializable objects?
7. What happens if an object is serializable but it includes a reference to a non-serializable object?
8. Are the static variables saved as part of serialization?
9. What is a transient variable?
10. What will be the value of the transient variable after de-serialization?
11. Does the order in which the value of the transient variables and the state of the object, using the defaultWriteObject() method are saved during serialization, matter?
12. How can one customise the Serialization process? Or, what is the purpose of implementing the writeObject() and readObject() method?
13. If a class is serializable but its superclass is not, what will be the state of the instance variables inherited from superclass after deserialization?

◆◆◆

Exception

Exceptions are events occurring during a program execution which interrupt the regular flow of data. In java, exceptions are objects which wrap error events that happen inside a method and contain details about the error and its type.

- There are some exceptional cases in java which throw an exception
- In mathematics, 1/0 is infinity, but in java this is an exceptional case. This terminates a program

  In this chapter, we will learn about of 'try', 'catch', 'finally', 'throws' and 'throw' and the use of these five keywords in industry or business.
- All exceptions occur only at runtime while syntax errors occur at compile time

Why do we need to Learn Exception?

Consider the program below. It executes properly and prints the output

```
package com.javabykiran.Exception;
/**
 * @author Java By Kiran
 */
public class ExceptionWithoutError {
    public static void main(String[] args)  {
        System.out.println("111");
        System.out.println("222");
        System.out.println("333");
        System.out.println("444");
  }
}
```

—Output---

```
111
222
333
444
```

➤ Now I add one exceptional code in between, which is shown in bold

```
package com.javabykiran.Exception;
 /**
  * @author Java By Kiran
  */
  public class ExceptionDivideByZero {
       public static void main(String[] args){
           System.out.println("111");
           System.out.println("222");
           int a=1/0;
           System.out.println("333");
           System.out.println("444");
        }
}
```

---Output---

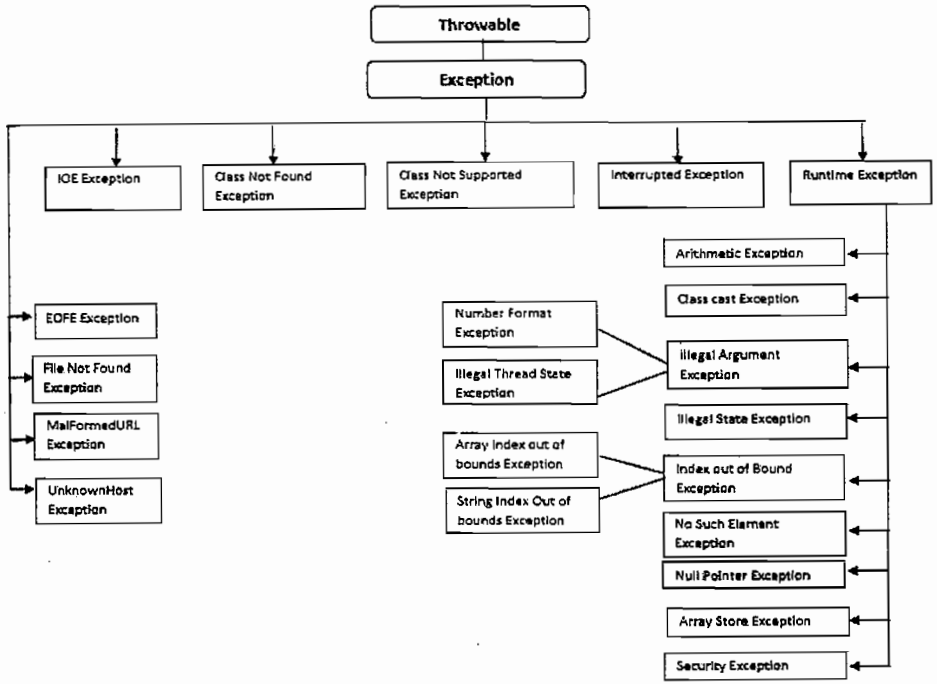
```
111
222
```

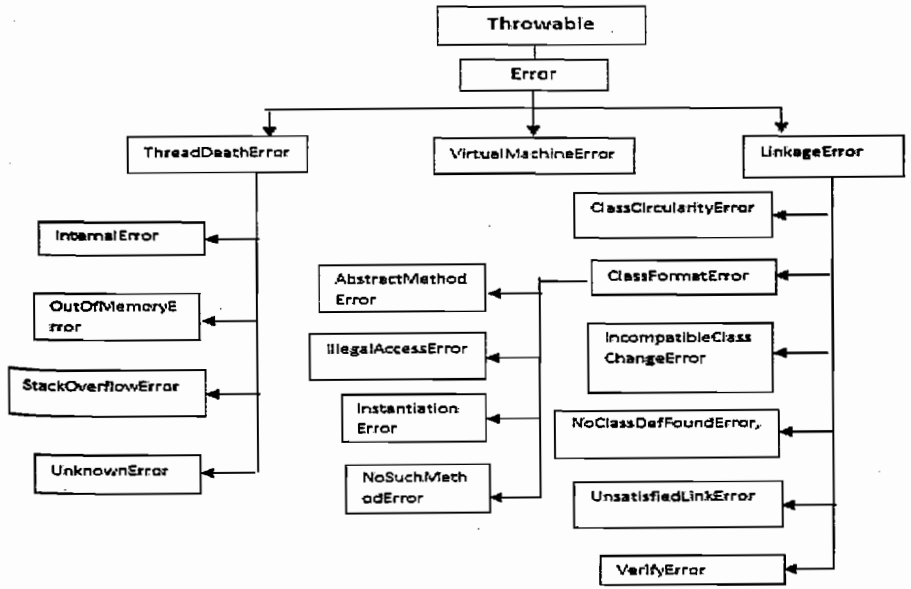
Exception in thread "main" java.lang.ArithmeticException: / by zero at com.javabykiran.Exception.Exception.main(Exception.java:10)

- ➤ In this case we are not able to get an output as expected as the exceptional case is in the middle. After learning this chapter we will be able to run anything and everything by handling such exceptional conditions
- ➤ Here you can see that the client will see this error message and it is not user friendly. It is better to show a more user friendly message
- ➤ We summarise the two uses
  - to make a continuation of a program
  - to avoid java error messages and to provide good, user friendly messages.

We will see more of it later in this chapter

Shown below is the hierarch





Throwable

- In the above given Hierarchy Throwable is a class which is at the top of the exception hierarchy, from which all exception classes are derived
- It is the super class of all Exceptions
- Only objects which are instances of this class can throw exception by using the throw statement
- Only object of this class or one of its subclasses can be the argument type in a catch block
- Problems faced during the program execution be divided into two types:
  - Exceptions
  - Errors
- Both Exception and Errors are java classes which are derived from the Throwable class

Error:

- Error is subclass of throwable class and it terminates the program if there is a problem related to system or resources. It signifies a serious problem in the application and are mostly abnormal conditions
- Error does not occur because of the programmer's mistakes, but when system is not working properly or a resource is not allocated properly
- Memory out of bound exception, stack overflow etc., are examples of Error
- It is common to make errors while developing or typing a program and this may cause the program to produce unexpected results. It is therefore necessary to detect and manage the error conditions in the program so that the program doesn't terminate during the execution
- Sun Microsystems has categorised all exceptions in different categories. For example, for all arithmetic operations are under class called ArithmaticException.java
- For all sql type of exceptions, they created a class called SQLExcption.java, and so on

Internal Flow by JVM

1. Take a look at the above class ExceptionDivideByZero.java [in the beginning of this chapter]
2. When the interpreter reads the line int a=1/0;
3. The control goes to JVM and it searches for a category of exception. In this case it is Arithmetic exception
4. The JVM creates an Object of that class, then it comes back to the same line again with a reference variable of ArithmaticException class
5. It does check if this line is in the try block
6. **If No,** then it goes to the console and prints a message by the getMessage method of ArithmaticException class
7. The program terminates and won't come back to our program again
8. **If Yes,** refers to the program given below with try and catch


```
package com.javabykiran.Exception;
 /**
 @author Java By Kiran
 */
public class ExceptionWithTryCatch {
    public static void main(String[] args) {
        System.out.println("111");
        System.out.println("222");
      try {
         System.out.println("before divide");
          int a = 1 / 0;
        System.out.println("after divide");
          } catch (Exception e) {
        System.out.println("I am in catch block");
            }
        System.out.println("333");
        System.out.println("444");
  } }
```

—Output---

```
111
222
before divide
I am in catch block 333
444
```

9. In this case, it looks for the corresponding catch block and then it sees which class is there in this case (Exception)
10. In this case, the compiler assigns Arithmetic class's object to Exception reference variable
11. Exception e = new ArithmeticException();
12. If the above (11) is correct, then it enters in the catch block
13. If it is not correct, then it considers that the try catch is not written and goes back to console and prints error. Point : 5
14. After executing catch block, the cursor goes to the next line and the execution continues 333 then 444
15. This is because as soon the interpreter see the exception, it goes for catch block, not in between sentences. In our case it is 'after divide'

    Some rules are below:
    - Try should be followed by only catch or only finally block or multiple catch or multiple catch and single finally
    - When exception is not raised in the try block statements then catch block will not

be executed

- You can write multiple try-catch blocks
- When you are writing a multiple catch block, then order or exception must first be subclass and then superclass
- When exception is raised in the try block statements then only one matching catch block will be executed
- You should not write any statement between try block and catch block
- We can write this combination anywhere inside the method. Given below are various combinations that are allowed

```
Not allowed as try must followed by catch or finally
try {
    }
```

Allowed

```
try {
  }
catch(Exception e){
 }
finally{
 }
```

Not Allowed

```
try {
}
catch(Exception e){
}
Finally {
  }catch(Exception e){
 }
```

Allowed

```
try {
 }catch(NullPointerException npe) {
 } catch(ArithmeticException e) {
 } catch(Exception e){
}
```


Not allowed - Reason: The bigger exception cannot be in the first catch because it will accommodate all exceptions and there will be no chance to reach the second catch of NullpointerException

```
try{
}
catch(Exception e) {
}
catch(NullPointerException npe) {
}
```

```
void m1(){
   for ( int i=0;i<100;i++){
    try{
         //Insertion Logic
     }catch(Exception e){
//Student name with error for loop continues
 //again for other students
   }
 }// for loop end
}// m1 ends
```

Business Use

If we have a requirement of inserting 100 students to DB subject to the condition that if there is any issue in insertion for any student, it should not impact the other students.

Now,the requirement is to Add 100 student but if any student fails, other students should not get inserted

```
void m1(){
      try{
          for (int i=0;i<100;i++){
  //Insertion Logic, if   fails should go to   catch block

      }// for loop end
    }catch(Exception e){
  //Student name  with  error; after  this  it will not go in
  //for loop  again

 }
}// m1 ends
```

Finally

The finally block is used when an important part of the code needs to be executed. It is always executed whether or not the exceptions are handled.

- Finally block will always get executed until we shut down JVM. To shut down JVM in java we call System.exit (); or we have infinite loop in try block
- Occurs irrespective of any exception
- Occurs irrespective of any return value in try or catch
- Normally, finally block contains the code to release resources like DB connections, IO streams etc.
- You can also write resource cleanup code inside the finalize() method. But finally block is recommended rather than the finalize method for resource cleanup

Question. What is the difference Between Catch and Finally in java?

Sr.	Catch	Finally
1	Catch block handles the error when it occurs in try block	There is no need of exception thrown by try block
2	Catch block is executed only when the if exception is thrown by try block, otherwise it is not executed	Finally block is always executed whether exception occurs or not
3	We can use multiple catch block for only one try block	Only one finally block is used for one try block
4	We can handle multiple exceptions by using catch blocks	It is not for exception handling

Examples: check compile time error and runtime output in some cases

Question: Will it compile?

```
package com.javabykiran.Exception;
/**
 * @author Java By Kiran
 */
public class FinallyTest {
int m1() {
    try {
       return 10;
    } catch (Exception e) {
      return 20;
    } finally {
       return 30;
```



```
      }
      return 40;
   }
}
```

Answer: It will not compile (Unreachable code).
return 40;

Question: What will this method return?

```
int m1() {
      try {
       return 10;
      } catch (Exception e) {
       return 20;
      } finally {
       return 30;
      }
}
```

Answer: 30

Throw and Throws

Why are they needed?

To delegate exception from one class to another we use throw and throws.

Consider the case below:

```
class A{
   // calls m1 of class B
}
class B{
   // calls m2 of class C
}
class C{
   // calls m3 of class D
}
class D{
   // some logic
}
```

- Consider the above code where A is sending a request to B and
- then to C and then to D

- If the exception occurs at class C m3 method, then what message will A get? Nothing. Program will get terminated at C class
- If we use try and catch at every class and every method then exceptions will get handled but message will not be transferred in the reverse direction, which is from C to B and then to A
- So, to delegate these types of exceptions to caller of the method we use throw and throws from one class to another
- Here, C will delegate its exception to B's m2 and then B's m2 will delegate to A's m1 (); there we will write try and catch and will give an user friendly message
- This scenario is more appropriate when there is remote calling. For example, FLIPKART calls to ICICI bank then ICICI to AIRTEL for sms and to mastercard for authorisation
- In this case, the whole infrastructure is different. This means that there is different ram and different hard disk for every entity. So, if we click a button in FLIPKART and something happens at the ICICI's end, we will just keep waiting. But by using throw, throws we will get a message that is delegated by ICICI

Runtime / Unchecked Exception

RuntimeException is a superclass of exceptions that may be thrown in the regular course of operation of JVM. Runtime exceptions and its subclasses are known as unchecked exceptions.

- This exception occurs at runtime. Actually, even checked exception occurs at runtime
- All exceptions under runtime exception class are called runtime exceptions or unchecked exceptions
- In the industry, it is not expected that the developer leave these exceptions in the program 1/0 operation ... developer should check before this that whether no is zero

```
if( b > 0 ){
  int c=a/b
}
```

- If any method throws runtime exception, then the caller is not restricted to handle that exception.

See the example below:

```
package com.javabykiran.Exception;
public class RuntimeEx {
    void m1() throws NullPointerException {
    }
    void m2(){
        m1(); // no error as m1 throws
        //runtime exception
    }
}
```


Compile time / Checked Exception

Simply put, all exceptions which are not runtime exceptions are compile time exceptions.

They are checked at compile time.

- These exceptions are those which are not under runtime exception class hierarchy
- If thrown from any method, they must be handled by caller of the method otherwise it gives compile time error
- The most confusing thing is that some people say it occurs at compile time. That is wrong; exception always occurs at runtime only

See the example below:

```
package com.javabykiran.Exception;
public class RuntimeEx {
    void m1() throws ClassNotFoundException {
    }
    void m2(){
      m1(); // Error as m1 throws compile time exception
    }
}
```

OR

```
package com.javabykiran.Exception;
public class RuntimeEx {
  void m1() throws ClassNotFoundException {
  }
  void m2() throws ClassNotFoundException{
      m1();
  }
}
```

Throw

The throw keyword declares an exception. It tells the user about the error and helps the user create an exception handling code from the outset.

- Throw keyword is used to throw the exceptions
- Normally JVM is responsible for identifying the exceptional java class, creating its objects and throwing that object
- You can use the throw keyword with this syntax: throw object;

  Example:

  throw new ArithmeticException();

  or

ArithmeticException ae =new ArithmeticException();
throw ae;
throw new StudentNotFoundException():

Throws

- Throws keyword is used to specify the method level exceptions
- If any method throws compile time exception then caller should :
  1. Handle that exception by writing try a block inside the method
  2. Or propagate the exception to caller of the method level exception using throws keyword

```
package com.javabykiran.Exception;
  public class ThrowsTest {
       void m1() throws
ArithmeticException,ArrayIndexOutOfBoundsException {
    try {
        m2();
    } catch (ArrayIndexOutOfBoundsException e) {
    if (1 == 2) {
       throw new ArrayIndexOutOfBoundsException();
    } else {
       throw new ArithmeticException("Don't do");
    }
}
    return null;
}
     void m2()throws ArrayIndexOutOfBoundsException {
           System.out.println("its ok");
   // void m3()throws ArrayIndexOutOfBoundsException {}
     try {
         m4();
     } catch (ArithmeticException e) {
           System.out.println("m3-ok-ok");
           throw new ArrayIndexOutOfBoundsException();
```



```
     }
}
     void m4() throws ArithmeticException {
         m5();
     }
     void m5() throws ArithmeticException {
        int x = 10 / 0;
     }
   }
}
package com.javabykiran.Exception;
  public class ExceptionLab3 {
    public static void  main(String[] args) {
     try {
         ThrowsTest ex = new ThrowsTest();
         ex.m1();
     } catch (Exception e) {
           System.out.println(e);
           System.out.println(e.getMessage());
           e.printStackTrace();
      }
   }
}
```

--Output---

```
its ok
java.lang.ArithmeticException: Don't do
                                    at
com.javabykiran.Exception.ThrowsTest.m1(ThrowsTest.java:12)
                                    at
com.javabykiran.Exception.ExceptionLab3.main(ExceptionLab3.java:7) m3-ok-ok
java.lang.ArithmeticException: Don't do
Don't do
```

Question : What is the difference between Checked Exception and Unchecked Exception keyword?

Sr.	Checked Exception	Unchecked Exception
1	Checked Exceptions are checked at compile time	Unchecked exceptions are not checked at compile time
2	Checked exceptions must be explicitly caught or propagated as described in Basic try-catch-finally Exception Handling	Unchecked Exception doesn't have to be caught or declared thrown
3	Checked exceptions in Java extend the java.lang.Exception class	Unchecked exceptions extend the java.lang.RuntimeException
4	Checked Exception represents a scenario with higher failure rate	Unchecked Exception occurs mostly due to programming mistakes
5	Checked Exception needs to be handled by the caller	Unchecked exceptions are not handled by the caller

Question: What is the difference between Exception and Error?

- Exception represents the problem which can be solved whereas error represents the problem which cannot be solved

  Exceptions can be handled and can contain the program execution where as errors cannot be handled

  Some examples of exceptions are:

  ArithmeticException, IOException, etc.

  Some examples of errors are:

  NoSuchMethodError, ClassDefNotFoundError, OutOfMemoryError

  StackOverflowError, etc.
- All exceptions and errors will always be thrown at runtime only
- Error comes at runtime and syntactical error comes at compile time

Sr. No	Exception	Error
1	Exceptions are related to the application	Error is related to the environment in which the application is running
2	Exceptions may not be fatal in all cases	An Error can't be recovered as it is fatal in nature
3	An Exception is basically divided into two categories,	An error is basically divided into 2 categories

	i.e. Checked and Unchecked Exceptions. A Checked Exception has a special place in the Java programming language and requires a mandatory try catch finally code block to handle it Unchecked Exception is a subclass of Runtime Exception that usually represents programming errors	1) Compile Time 2) Runtime
4	Exceptions can be checked or unchecked. Programmer should handled at the application level	Errors are always unchecked and usually indicate a system error or a problem with a low level resource and should be handled at the system level, if possible
5		Handling an Error is not a good idea because recovery from an Error is usually not possible

Questions:

1. What is an Exception?
2. How are the exceptions handled in java?
3. Exceptions are defined in which javapackage?
4. Explain the Exception hierarchy in java.
5. What is Runtime Exception or Unchecked Exception?
6. What is Checked Exception?
7. What is the difference between Error and Exception?
8. What is the use of throws keyword?
9. What is throw keyword?
10. What is difference between ClassNotFoundException and NoClassDefFoundError?
11. What are the possible combinations to write try, catch finally block?
12. When should you make a custom checked Exception or custom Unchecked Exception?
13. How to create custom Exception?
14. What is StackOverflowError?
15. Why did the designers decide to force a method to specify all uncaught checked exceptions that can be thrown within its scope?
16. Once the control switches to the catch block does it return to the try block to execute the balance code?

17. Why is the clean up code, like release of resources, put in try-catch-finally block and why?
18. Does the order of the catch blocks matter if the Exceptions caught by them are not subtype or supertype of each other?
19. Is it valid to have a try block without catch or finally?
20. How do you get the descriptive information about an Exception that occurs during the program execution?
21. Why is it not considered a good practice to write a single catch all handler to catch all the exceptions?
22. Can a catch block throw the exception caught by itself?
23. What is exception matching?
24. What happens if the handlers for the most specific exceptions are placed above the more general exceptions handler?
25. What happens if a method does not throw a Checked Exception directly but calls a method that does? What does 'Ducking' the exception mean?
26. Is an empty catch block legal?

◆◆◆

Annotations

Annotations

Java annotations are essentially tags used to provide Meta data for your Java code. These are extra details that can be utilised by the java compiler or JVM. Being Meta data, the annotations do not directly affect the execution of your code, although some types of annotations can actually be used for that purpose. Java annotations were added to Java from Java 5.

Java Annotation Purposes

Java annotations are typically used for the following purposes:

- Compiler instructions
- Build-time instructions
- Runtime instructions

Java has three built-in annotations that you can use to give instructions to the Java compiler. These annotations are explained in more detail later in this chapter.

Java annotations can be used at build-time, when you build your software project. The building process includes generating source code, compiling the source, generating XML files (e.g. deployment descriptors), packaging the compiled code and files into a JAR file, etc. Building the software is typically done by an automatic build tool like Apache Ant or Apache Maven. Build tools may scan your Java code for specific annotations and generate source code or other files based on these annotations. Normally, Java annotations are not present in your Java code after compilation. It is possible, however, to define your own annotations that are available at runtime. These annotations can then be accessed via Java Reflection, and used to give instructions to your program, or some third party API.

Annotation Basics

An annotation in its shortest form looks like this:

@SetValueJbk

The @ character signals to the compiler that this is an annotation. The name that follows the @character is the name of the annotation.

@SetValueJbk(value = "Kiran JBK") String name;

The annotation in this example contains a single element named name, with the value set to Kiran JBK. Here, the elements are enclosed inside the parentheses after the annotation name. Annotations without elements do not need parentheses.

Annotation Placement

You can put Java annotations above classes, interfaces, methods, method parameters, fields and local variables.

Example of Annotations:

1) SetValueJbk.java

```
package com.javabykiran.Annotations;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
/**
* @authorKiran
* This annotation is used for
* setting values to variables in a  class
*/
@Retentation(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public@interface SetValueJbk {
   public String value ();
}
```

2) Student.java

```
package com.javabykiran.Annotations;
 public class Student{
     @SetValueJbk(value = "30")
      int age;
     @SetValueJbk(value = "Pune")
      String location;
     @SetValueJbk(value = "Kiran JBK")
      String name;
     @SetValueJbk(value = " Java By Kiran")
      String Owner_of;
      }
```

3) AnnotTest.java

```
package com.javabykiran.Annotations;
 import java.lang.annotation.Annotation;
 import java.lang.reflect.Field;
 public classAnnotTest{
```



```
public static void main(String[]  args){
     /*
     * we will get parameters for student class like values
     for age  and location. Then we can use this class
     anywhere
     * for example,to insert into DB
     */
     Class aclass = Student.class;
     Field field[] = aclass.getDeclaredFields();
  for (Field field : fields) {
       system.out.println("field name >>"+ field.getname());
     Annotation annotation[] = field.getDeclaredAnnotations();
      for(Annotation annotation: annotations){
           SetValueJbk annotRef = (SetValueJbk)annotation;
           system.out.println("name :" +annotRef.value());
           }
        }
      }
    }
```

If there is NO annotation, then we need to set and get values and dynamic usage is not possible. The same thing is internally implemented in Hibernate, which we do by using annotations.

Java's Built-in Annotations

Java comes with three annotations which are used to give the Java compiler instructions. These annotations are:

- @Deprecated
- @Override
- @SuppressWarnings

Each of these annotations is explained in the following sections:

Deprecated

The @Deprecated annotation is used to mark a class, method or field as deprecated, meaning it should no longer be used. If your code uses deprecated classes, methods or fields, the compiler will give you a warning.

Here is an example:

```
@Deprecated
public class MyComponent {
}
```

The use of the @Deprecated annotation above the class declaration marks the class as deprecated.

You can also use the @Deprecated annotation like the above method and field declarations to

mark the method or field as deprecated. When you use the @Deprecated annotation, it is a good idea to also use the corresponding @deprecated JavaDoc symbol, and explain why the class, method or field is deprecated, and what the programmer should use instead.

Override

The @Override annotation is used above methods that override methods in a superclass. If the method does not match a method in the superclass, the compiler will give you an error. It is one of the reasons why it is considered to be a good practice while coding in java.

The @Override annotation is not necessary in order to override a method in a superclass. It is a good idea to still use it though. In case someone changed the name of the overridden method in the superclass, your subclass method would no longer override it. Without the @Override annotation you would not find out what happened. With the @Override annotation, the compiler will tell you that the method in the subclass is not overriding any method in the superclass.

Here is an example use of the @Override annotation:

```
public class A{
    public void m1() {
    System.out.println("A-m1");
    }
}
public class B extends A{
    @Override
    public void m1() {
    System.out.println("B-m1");
    }
}
```

In case the method m1() in A changes signature so that the same method in the subclass no longer overrides it, the compiler will generate an error.

Suppress Warnings

The @Suppress Warnings annotation makes the compiler suppress warnings for a given method or the annotated element. For instance, if a method calls a deprecated method, or makes an sinsecure type cast, the compiler may generate a warning. You can suppress these warnings by annotating the method containing the code with the
@SuppressWarnings annotation. For example:

```
@SuppressWarnings
    public void methodWithWarning() {
}
```


Questions:

1. What is Annotation?
2. What are few of the Annotations predefined by Java?
3. Annotations were introduced with which version of Java?
4. Give an example of Annotations?
5. Name a few meta-annotations?
6. What are meta Annotations?
7. Which annotations are used in Hibernate?
8. What entries do we make in the hibernate config file if we are not using hbm files but Annotations?
9. What are the annotations used in Junit with Junit4?
10. What are the different ID generating strategies using @GeneratedValueannotation?
11. How to create a Junit to make sure that the tested method throws an exception?
12. How should we ignore or avoid executing set of tests?
13. How can we test methods individually which are not visible or declared private?
14. Which of the following annotation is used to avoid executing Junits?
    a. @explicit
    b. @ignore
    c. @avoid
    d. @NoTest
15. What is the @FunctionalInterfaceannotation?
16. What is the package name for Annotation class?
17. Which is the Parent Class of Annotation class?
18. How annotations are retrieved or read?

◆◆◆

21 CHAPTER

Reflection

Reflection in java is an API that examines or modifies the behavior of classes, methods, fields and interfaces during runtime. It is called reflection because of its ability to examine other code, or itself, within the same system. This can be done even when a particular class is inaccessible.

Why do we need Reflection?

Let's say we have a requirement as shown below. Observe the class given:

```
package com.javabykiran.reflect;
 /*
  * @author Java By Kiran
  */
 public class Employee {
  public void m1() {
   System.out.println("I am in  m1");
  }
  public void m2() {
   System.out.println("I am in  m2");
  }
  public void m3() {
   System.out.println("I am in  m3");
  }
  public void m4() {
   System.out.println("I am in  m4");
  }
 }
```

Now we have been asked to call all methods of this class as per client requirement.

a. Client 1 says I need order of calling methods like:
   i. m1,m2,m4,m3
b. Client 2 says I need order of calling methods like:
   i. m4,m2,m4
c. Client 3 says I need order of calling methods like:
   i. m1,m2,m3,m4

If you observe order of methods of every client, you'll find that they'are all different. To fulfill the above given requirement, we will follow the steps given below :

```
package com.javabykiran.reflect;
    /*
 * @author Java By Kiran
 */
public class EmployeeTest {
 public static void main(String[] args)  {
      // Client 1 - requirement
      System.out.println("Client 1 Requirement");
      Employee employee=new Employee();
      employee.m1();
      employee.m2();
      employee.m4();
      employee.m3();
      // Client 2 - requirement
      System.out.println("Client 2 Requirement");
      Employee employee1=new Employee();
      employee1.m4();
      employee1.m2();
      employee1.m4();
      // Client 3 - requirement
      System.out.println("Client 3 Requirement");
      Employee employee2=new Employee();
      employee2.m1();
      employee2.m2();
      employee2.m3();
      employee2.m4();
     // Like above for all clients...
      }
   }
```

---Output---

```
Client 1 Requirement
I am in m1
I am in m2
I am in m4
I am in m3
Client 2 Requirement
```

```
I am in m4
I am in m2
I am in m4
Client 3 Requirement
I am in m1
I am in m2
I am in m3
I am in m4
```

Let's say there are many clients and their requirements will be different. We will go on writing code as shown above EmployeeTest.java

This is not dynamic. Tomorrow, any addition of client causes our code to change. This is what we don't want to do. Hence, this coding practice is not good.

To solve this requirement we must use reflection. We will be able to solve the above given requirement at a later stage of this chapter once we understand reflection completely.

Next Requirement

We have an excel sheet below:

Roll No	Student	Student Phone No	Remark
1733	Kiran	8888809416	Java by kiran
1738	Jayshree	9552343698	www.javabykiran.com

Now we want to insert data from the excel sheet to table Student as shown below:

Columns | Data | Information | Indexes | Constraints | Triggers | Script

Field *	Type *	Collation	Null *	Key *	Default	Extra *
sname	varchar(20)	latin1_swedish_ci	YES		{null}	
sphone	varchar(20)	latin1_swedish_ci	YES		{null}	
remark	varchar(20)	latin1_swedish_ci	YES		{null}	
rollno	varchar(20)	latin1_swedish_ci	YES		{null}	

To use core java logic for insertion easily with less optimised code, you must use reflection.

Important points to remember about Reflection:

1. Java Reflection makes it possible to inspect classes, interfaces, fields and methods at runtime, without knowing the names of the classes, methods, etc.
2. We can create objects dynamically by using reflection
3. We can create an object of a class having private constructors as well, from outside a class.

Hence, we can instantiate class from outside. This is not possible by simply creating an object by using new operator

This chapter will explain the basics of Java Reflection including how to work with arrays, annotations, generics and dynamic proxies, and how to do dynamic class loading and reloading. It will also show you how to do more specific tasks, like reading all getter methods of a class, or accessing private fields and methods of a class.

This chapter will also clear up some of the confusion out there about what Generics information is available at runtime. Some people claim that all Generics information is lost at runtime. This is not true.

4. Package in which all reflection classes are existing :
   java.lang.reflect
5. **For reflection we must know class [Class] first.**
   Class is class name in java and exists in java.lang package.
   Instances of the class Class represent classes and interfaces in a running Java application.
   In reflection we will deal with preexisting classes like Method, Field, and Constructor and so on.

Let's say we have simple class as shown below:

```
package com.javabykiran.reflect;
public class B {
      B() {
         System.out.println ("B Zero parameter const");
       }
      B(int x) {
        System.out.println ("B One parameter const");
      }
     int a=90;

     void m1() {
        System.out.println ("B -m1");
     }

     int m2() {
        System.out.println ("B - m2");
       return 10;
  }
}
```

To use reflection we must first get object of Class class.

The example below shows two ways.

```
package com.javabykiran.reflect;
public class A {
  public static void main(String[] args)   {
     //First scenario Class
     c = B.class;
     //Second scenario
     B b=new B();
     Class cl = b.getClass();
  }
}
```

In this example we will get all the information about class **B** by using c and cl reference variables by using methods of Class class.

a. Above given is class B (every information means -- > two constructors, two methods and one variable)
b. Here we can see the complete class B but sometimes you only know class name not members of that class. Then we must go for reflection
c. To complete this we use the below methods of class Class [In the second step we have added some reflection code in existing class A]

i. To Fetch All methods use the class below:

```
package com.javabykiran.reflect;
import java.lang.reflect.Method;
public class A {
  public static void main(String[] args) {
     // First scenario
     Class c = B.class;
     // Second scenario
     B b = new B();
     Class cl = b.getClass();
// GET ALL METHODS OF CLASS B
// return all public methods of B and inherited methods
     Method[] methodList = cl.getMethods();
// for each loop you can use simple for loop as //well
    for (Method method : methodList) {
      System.out.println("method name >> " + method.getName());
     }
```



```
// below code returns all public/private/
protected/default methods of B only
Method[] methodListAll = cl.getDeclaredMethods();
// for each loop you can also use simple for loop
   for(Method method : methodListAll) {
     System.out.println("declared method name >> " +
                          method.getName());
    }
  }
}
```

—Output---

```
B Zero parameter const
method name >> m1          // if declare with public keyword
method name >> hashCode
method name >> getClass
method name >> wait
method name >> wait
method name >> wait
method name >> equals
method name >> notify
method name >> notifyAll
method name >> toString
declared method name >> m1
declared method name >> m2
```

ii. To Fetch All Variables in class B

```
Add the lines below n class A
// GET ALL VARIABLES OF CLASS B
Field [] fields=cl.getDeclaredFields  ();
for(Field field:fields){
    System.out.println("field   name >> "+field.getName());
     }
```

iii. Getting and Setting Field Values

Once you have obtained a Field reference you can get and set its values using the Field.get() and Field.set()methods, like this: // to execute this add variables in class B

```
Class aClass = B.class;
 Field field = aClass.getField("a");//we are getting one variable
    B bb = new B();
   field.set(bb, 45)
```

The bb is an instance of the class that owns the field. In the above example an instance of B is used.

If the field is a static field (public static ...) pass null as parameter to the get and set methods, instead of the bb parameter passed as above.

iv. Call private methods and variables

To access a private method you will need to call the Class.getDeclaredMethod(String name, Class[] parameterTypes) or Class.getDeclaredMethods() method.

The methods Class.getMethod(String name, Class[] parameterTypes) and Class.getMethods() methods only return public methods, so they won't work.

Here is a simple example of a class with a private method, and below that is the code to access that method via Java reflection.

```
package com.javabykiran.reflect;
import java.lang.reflect.Method;
public class PrivateObject {
  private String privateString = null;
  public PrivateObject(String privateString)
{
    this.privateString = privateString;
  }
  private String getPrivateString() {
    return this.privateString;
  }
  public static void  main(String[] args) throws Exception {
    PrivateObject  privateObject  =  new  PrivateObject("The
    Private Value");
    Method privateMethod =
    PrivateObject.class.getDeclaredMethod(
      "getPrivateString", null);
  privateMethod.setAccessible(true);
 String val=(String)privateMethod.invoke(privateObject, null);
 System.out.println("returnValue = " + val);
   }
}
```

This code example will print out the text "return Value = The Private Value", which is the valu returned by the method getPrivateString () when invoked on the PrivateObject instance created a the beginning of the code sample.

Notice the use of the method PrivateObject.class.getDeclaredMethod ("privateString"). Thi method helps to call private method.

It only returns methods declared in that particular class, not methods declared in an

superclasses.

Observe the line in bold too. By calling Method.setAcessible (true), you turn off the access checks for this particular Method instance for reflection calls only. Now you can access it even if it is private, protected or package scope, even if the caller is not a part of those scopes. You can't access the method using normal code. The compiler won't allow it.

Dynamic Class Loading

Dynamic class loading is a very helpful feature in java that lets us load java code which we are unaware of before the starting of a program. It is invoked at runtime.

Use Class.forName in this case.

```
package com.javabykiran.reflect;
/*
 * @author Java By Kiran
 */
public class DynaLoad {

  Object anyClassObject(String clsName)throws Exception {
     Object object = Class.forName(clsName).newInstance();
     return object;
  }
  public static void main(String[]args)throws Exception {
     DynaLoad dynaLoad=new DynaLoad();
     /* This will give A class object we will pass class name
     as a String*/
A a=(A)dynaLoad.anyClassObject("com.jbk.A");
     // now do operations with a
/*This will give A class object we will pass class name as a
String*/

  Employee employee=(Employee)dynaLoad.anyClassObject
  ("com.jbk.Employee");
     // now do operations with employee
  }
}
```

In this example we see that the single method we used to get a class object. If we do not use this way we need to hard code like:

A a =new A();// can't be changed later // we must know class name

Employee employee =new Employee ();// can't be changed later // we must know class name.

If somebody asks why we use class.forname, the answer will be dynamic class object creation.

◆◆◆

22 CHAPTER

Arrays

Arrays are collection of elements of a similar type

An array is a data structure in Java which stores a sequential collection of elements of a fixed size and having the same data type. Only a fixed set of elements can be stored here.

Arrays are objects in java

Arrays is a class in java package: java.util.Arrays We can define arrays in two ways:

- Static initialization :

  int x =10;

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

  double []d = {1.2, 2.3, 3.4 };
- Dynamic initialization :

  int a[] = new int[4];

  OR

  int a[] = null;

  a = new int[4];

  a[0] = 11;

  a[1] = 22;

  a[2] = 201 + 1 * 4; // = 201+4 =205

  a[3] = 201 + 2 * 4; // = 209

Example 1:

```
package com.javabykiran.basics;
public class Lab26 {
public static void main(String[] args){
      // int b[5]=null;
      // int d[]=new int[];
      int d2[] = null;
      System.out.println(d2);
      d2 = new int[0];
      System.out.println(d2); //array is there, but empty
     // d2[0]=123; //error in java.lang ArrayIndexOutOfBounds
     //Exception: 0
//d2 = new int[-1];//error java.lang.NegativeArraySizeException
      d2 = new int[9];
```



```
    int a[] = null;
  System.out.println(a);
 // a ={10,20,30}; // error (must be at initializer)
  int x[] = {10, 20, 30};
  System.out.println(x);
   int n = x.length;
  for (int i = 0; i < n; i++) {
     System.out.println(x[i]);
  }
  System.out.println(a);
   a = new int[5];
  System.out.println(a);
  for (int i = 0; i < a.length; i++) {
     System.out.println(a[i] + " ");
  }
  System.out.println(" ");
      a[0] = 11;
      a[2] = 222;
      a[4] = 444;
      // a[7]=777;
  for (int i = 0; i < a.length; i++) {
     System.out.println(a[i] + " ");
  }
 }
}
```

---Output---

```
null
[I@18d107f
null
[I@360be0
10
20
30
null
[I@45a877
0
0
0
0
```

```
222
0
444
Done
```

Example 2:

```
package com.javabykiran.Arrays;
public class Lab27 {
   public static void main(String[] args) {
      int a1[] = new int[5];
      int n = 5;
      int a2[] = new int[n];
       // int a3[]=new int[5.5];  //error possible
       // loss of precision found:
       // double required:
      int byte b = 5;
      int a4[] = new int[b];
      int a5[] = new int[(2 + 3) * 2];
      // int a6[]=new int[(2+3.5)*2];
     // found:double required:int
    }
}
```

```
Boolean a2[]=new Boolean[5];
a2[0]=true;
a2[1]=true;
```

```
String x[]=new String[5];
x[0]="java";
x[2]="by";
x[4]="kiran";
```

Two dimensional array: It is also called as arrays of array

```
int a[][]=new int[3] [4];
(Number of arrays required) Three arrays  required
int x[][]={{1,2,3},{4,5},{6,7,8,9},{1,2} }
int s[][]=new int[4][]; //OK
int xx[][]=new int[][]; //error size in first bracket is must
```



```
int s[][] = new int[4][];
s[0] = new int[3];
s[0][0] = 1;
s[0][1] = 2;
s[0][2] = 3;
s[1] = new int[3];
s[1][0] = 1;
s[1][1] = 2;
s[1][2] = 3;
s[2] = new int[3];
s[2][0] = 1;
s[2][1] = 2;
s[2][2] = 3;
s[3] = new int[3];
s[3][0] = 1;
s[3][1] = 2;
s[3][2] = 3;
/* s[4] = new int[3]; //will not correct at runtime
s[4][0] = 1;
s[4][1] = 2;
s[4][2] = 3;*/
```

Example 3: Program to sort array

```
package com.javabykiran.Arrays;
import java.util.Arrays;
 public class Lab30 {
    public static void main(String[] args) {
       int a[] = { 4, 7, 2, 1, 3, 8, 6, 10, 9 };
        System.out.println("Before sorting");
       for (int i = 0; i < a.length; i++) {
           System.out.println(a[i] + " ");
       }
       // System.out.println(a[i]+" ");
      // System.out.println(" ");
     Arrays.sort(a); // merge sorting
     System.out.println("after sorting");
       for (int i = 0; i < a.length; i++) {
          System.out.println(a[i] + " ");// 1 2 -----10
```

```
        }
    System.out.println("");
    System.out.println(Arrays.binarySearch(a, 7));
    System.out.println(Arrays.binarySearch(a, 12));
    System.out.println(" ");
    int x[][] = {{3,1,2},{5,6,7},{9,8,7,9},{2,4,3}};
    System.out.println("Before sorting");
     for (int i = 0; i < x.length; i++)      {
        for (int j = 0; j < x.length; j++) {
            System.out.println(x[i][j] + " ");
         }
          System.out.println(" ");
     }
     for (int i = 0; i < x.length; i++) {
        Arrays.sort(x[i]);
     }
    System.out.println("After sorting");
    for (int i = 0; i < x.length; i++) {
        for (int j = 0; j < x[i].length; j++) {
           System.out.println(x[i][j] + " ");
         }
         System.out.println(" ");
     }
    int s5[] = new int[6];
    s5[0] = 10;
    s5[4] = 44;
    for (int i = 0; i < s5.length; i++) {
        System.out.println(s5[i] + " ");
    }

   Arrays.fill(s5, 99);
      for (int i = 0;i < s5.length; i++) {
          System.out.println(s5[i] + " ");
       }
  }
}
```


1) Program to find max number

```
package com.javabykiran.Arrays;
public class Lab13 {
  public static void main(String[] args) {
    int a =  Integer.parseInt(args[0]);
    int b =  Integer.parseInt(args[1]);
    int c = Integer.parseInt(args[2]);

    if ((a > b) && (a > c)) {
       System.out.println("max" + a);
    } else if (b > c) {
       System.out.println("max" + b);
    } else {
       System.out.println("max" + c);
    }
  }
}
```

2) Program to find odd and even number

```
package com.javabykiran.Arrays;
public class Lab14 {
  public static void main(String[] args) {
     int n = Integer.parseInt(args[0]);
     if (n % 2 == 0) {
       System.out.println("Even number");
      } else {
       System.out.println("Odd Number");
     }
   }
 }
```

3) Program for switch statement demo

```
package com.javabykiran.Arrays;
public class Lab15 {
   public static void main(String[] args) {
   int a = Integer.parseInt(args[0]);
   switch (a) {
      case 1:
         System.out.println("1jun10");
         break;
       case 2:
         System.out.println("19may87");
         break;
   default:
      System.out.println("The number is out of range");
}
```

Questions:

1. What is ArrayStoreException in java? When you will get this exception?
2. What are the drawbacks of the arrays in java?
3. What is an anonymous array? Give examples?
4. Can you pass a negative number as an array size?
5. Can you change the size of the array once you define it? OR Can you insert or delete the elements after creating an array?
6. What is the difference between int[] a and int a[ ]?
7. "int a[] = new int[3]{1, 2, 3}" – Is this a legal way of defining arrays in java?
8. What are the differences between Array and ArrayList in java?
9. There are two array objects of int type. One contains 100 elements and another one contains ten elements. Can you assign an array of 100 elements to an array of ten elements?
10. How do you check the equality of two arrays in java? OR How do you compare two arrays in java?
11. What are the different ways of copying an array into another array?
12. What are jagged arrays in java?
13. What is ArrayIndexOutOfBoundsException in java? When does it occur?
14. How do you sort the array elements?
15. How do you find the intersection of two arrays in java?
16. How do you find duplicate elements in an array?
17. What are the different ways of declaring multidimensional arrays in java?

18. While creating the multidimensional arrays, can you specify an array dimension after an empty dimension?
19. How do you search an array for a specific element?
20. What value do array elements get, if they are not initialised?
21. What are the different ways to iterate over an array in java?
22. How do you find second largest element in an array of integers?
23. How do you find all pairs of elements in an array whose sum is equal to a given number?
24. How do you separate zeros from non-zeros in an integer array?
25. How do you find a continuous sub array whose sum is equal to a given number?

◆◆◆

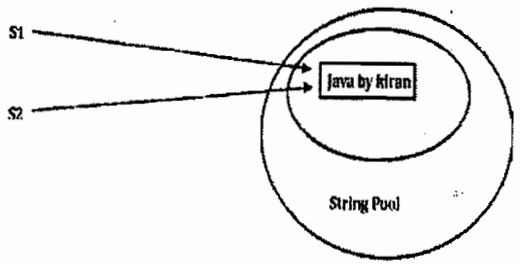
23 CHAPTER

String StringBuffer StringBuilder

- Generally, String is a sequence of characters. But in java, String is an object that represents a sequence of characters. String class is used to create string object
- **String** can be defined as a 'sequence of characters that is treated as a single data item and initialized by null' like String x = null;
- String class is an immutable class which is present inside java.lang package. Immutable means it cannot be changed.
- Now, the question arises as to why the String class is immutable. Memory in Java is divided into three parts, Heap, Stack and String Pool. String is so important in java that it has its own dedicated memory location. What String pool does it that whenever a string gets repeated, it does not allocate new memory for that string. Instead, it uses the same string by making a reference to it, thereby saving memory.
- We can create String object in two ways:
  1. String s1 = "javabykiran";
  2. String s2 = new String("javabykiran");
- Internally, String class uses the String Pool concept. In String pool, we do not have duplicate strings.

  For example:

  String s1 = "javabykiran";
  String s2 = "javabykiran";//will not create new instance.



In the above example, only one instance will be created. First, JVM will not find any string with the value "javabykiran" in string constant pool, so it will create a new instance. After that, for s2, it will find the string with the value "javabykiran" in the pool. This will not create a new instance but will return the reference to the same instance of s1.

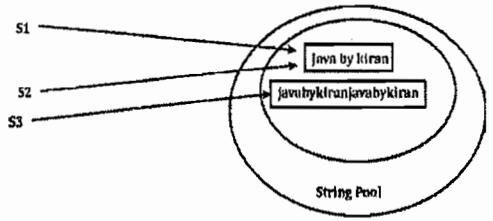
- In java, there are two equals () methods at two places. One in Object class and another in String class.
- The equals () method of object will check reference of objects i.e. addresses. Whereas equals method of String will check contents of string with character sequence

Why is String called immutable?

- In java, string objects are immutable. Immutable simply means unmodifiable or unchangeable. Once string object is created, its data or state can't be changed but a new string object is created.

  Example: Consider the above example of s1 and s2. Now, String s3 = s1 + s2; //s3 will be "javabykiranjavabykiran".
- In this case, s1 and s2 will not change and still exist after concatenation

  ('+' sign is overloaded in java, if it gets digits it adds two numbers and if strings then it concatenates).



What are the advantages of String Pool?

The advantages of a String Pool are stated below:

- Resource management
- Not a heap memory
- Memory efficient
- Reusability
- Duplicate objects are not allowed

Example:

```
String st1="jbk";
String st2="pune";
String st3="kiran";
      st1=st2;
      st3=st1;
 System.out.println(st1);
 System.out.println(st2);
 System.out.println(st3);
```

When you create a String object without a new operator, the following task happens-

- String literal will be taken and verified in string constant pool
- If literal is not available in pool then a new object will be created in the pool and that

---Output---

```
false
false
false
false
true
true
false
false
false
```

The "==" operator in Java is used to compare two objects. It checks to see if the objects refer to the same place in memory. In other words, it checks to see if the two object names are basically references to the same memory location.

Program -Lab2

```
package com;
public class StringLab2 {
    public static void main(String[] args) {
        String s1 = "Hello"; // String literal
        String s2 = "Hello"; // String literal
        String s3 = s1; // same reference
        String s4 = new String("Hello"); // String object
        String s5 = new String("Hello"); // String object
System.out.println(s1 == s1); // true,same pointer
System.out.println(s1 == s2); // true,s1 and s2 share
                               //storage in pool
System.out.println(s1 == s3); //true, s3 is assigned same
                              //pointer as s1
System.out.println(s1.equals(s3)); // true, same contents
System.out.println(s1 == s4); //false, different pointers
     s1.equals(s4); // true, same contents
System.out.println(s4 == s5); // false, different pointers in
                              //heap
System.out.println(s4.equals(s5)); // true, same contents
   }
}
```


address will be assigned to reference variable.

- If string literal is available in the pool then the available object address will be assigned to reference variable.

Program - Lab1

```
package com.javabykiran;
class A {
   String s;
     public A(String s){
      this.s = s;
     }
}
```

```
public class StringLab1 {
  public static void main(String[] args){
     A a = new A("kiran");
     A a1 = new A("jbk");
     A a2 = new A("kiran");
     String x = "javabykiran";
     String y = "jbk";
     String z = new String("javabykiran");
//guess true or false without seeing output
      System.out.println(x==z);
      System.out.println(y==z);
      System.out.println(x.equals(y));
      System.out.println(x==y);
      System.out.println(x.equals(z));
      System.out.println(a==a);
      System.out.println(a==a1);
      System.out.println(a1==a2);
      System.out.println(a.equals(a2));
      System.out.println(a1.equals(y));
   }
}
```

---Output---

```
true
true
true
true
false
false
true
```

StringBuffer

- In java, StringBuffer is a class present in java.lang package. It provides the same functionality as String class but there are also some additional features in StringBuffer class.
- It is slower than String as it has synchronised methods, so it is a Thread-safe class
- The String class is immutable, i.e objects are unmodifiable, whereas StringBuffer class provides mutable sequence of characters. A stringBuffer is like a String, but can be modified.
- The principal operations on a StringBuffer are append() and insert() methods, which are overloaded so as to accept data of any type. The append method always adds these characters at the end of the buffer; the insert method adds the characters at a specified point. For example, if "s" refers to a String Buffer object whose current contents are "java", then the method call s.append("kiran") would cause the String Buffer to contain "javakiran", whereas z.insert(4, "by") would alter the String Buffer to contain "javabykiran".
- StringBuffer class has capacity concept. It always carries 16 extra reserve spaces for characters

  Since the release of JDK 5, this class has been supplemented with an equivalent class designed for use by a multiple thread, StringBuilder. The StringBuilder class should generally be used in preference to this one, as it supports all of the same operations but it is faster as it has non-synchronised methods.

Lab Examples:

```
package com.javabykiran;
  //use of append method
 public class StringBufferLab1 {
     public static void main(String[] args) {
         StringBuffer sb = new StringBuffer("java");
      sb.append("bykiran"); //now original String is changed
      System.out.println(sb); //will print javabykiran
    }
 }
```



```
package com.javabykiran;
  //use of append method
 public class StringBuilderLab1 {
     public static void main(String[] args) {
         StringBuilder sb = new StringBuilder("java");
      sb.append("bykiran"); //now original String is changed
      System.out.println(sb); //will print javabykiran
    }
}
```

```
//using insert method
package com.javabykiran;
   public class StringBufferLab2 {
      public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("java");
      sb.insert(4,"bykiran");//now original String is changed
      System.out.println(sb); //will print javabykiran
   }
 }
```

```
//default capacity
package com.javabykiran;
 public class StringBufferLab3 {
    public static void main(String[] args) {
           StringBuffer sb = new StringBuffer();
   System.out.println(sb.capacity()); //default capacity is 16
   sb.append("java");
   System.out.println(sb.capacity());// again will be 16
   sb.append("is my best lang");//exceeds 16
   System.out.println(sb.capacity());//so now (16*2+2)
   }
}
```

---Output---

```
16
16
34
```

Question: What are the differences between String and String Buffer?

String	StringBuffer

1. It is an immutable class, i.e. its objects cannot be changed.
Example:
package com;
public class AA {
public static void main(String[] args){
String s = new String("java");
s.concat("bykiran");
System.out.println(s);
}
}
Output : Java
	1. It is a mutable class, i.e its objects can be changed only once.
	Example:
	package com;
	public class AA {
	public static void main(String[] args) {
	StringBuffer sb =newStringBuffer("java");
	sb.append("bykiran");
	System.out.println(sb);
	}
	}
	Output : Javabykiran

2. String object can be created in two ways i.Using new operator ii.Without using new operator	2. StringBuffer object can be created in only one way i. Using new operator
3. String object will use String constant pool	3. In StringBuffer object there is no concept of String constant pool

String vs StringBuffer

When to use String, StringBuffer and StringBuilder?

The appropriate class should be used according to the given requirements. Study the poin given below:

- String class should be used when the content is fixed and will not change frequently
- StringBuffer should be used when the content is not fixed and frequently changes, b thread safety is required
- StringBuilder should be used when the content is not fixed and frequently changing a thread safety is not required
- String is immutable, whereas StringBuffer and StringBuilder are mutable.

Questions

1. Reverse a String without using String API?
2. How can we split a String in java?
3. What is String Pool?
4. Sort the String without using String API.

5. Sort the String using String API.
6. Is the Check String a palindrome or not?
7. How can we write a Java Program to split a String using String Split() method?
8. How can we use Java Program to split a String using String Split() method and use each loop to display?
9. How do we write a Java Program to split a String using String Split() method. Before splitting, check whether there is a deliminator is not.
10. How do we write a Java Program to split a String using String Split() method. Before splitting check whether there is a deliminator or not.
11. How to remove a non-Ascii character from a String?
12. How to write a program to check if the given number is a palindrome or not?
13. How do we write a Java Program to remove non-ASCII characters from String?
14. How to write a Java Program to remove multiple spaces in a string?
15. How do we write a Java program to replace or remove a character in a string java?
16. How can we write a Java Program to remove all occurrences of a string?
17. How can we write a Java Program to replace the first occurrence of Specific index character in a String?
18. How should we write a Java Program to remove all numbers from a string?
19. How should we write a Java Program to check whether two strings are anagrams or not?
20. Write a program in Java to check if two strings are anagrams.
21. How to write a Java Program to find the maximum occurring character in a string?
22. How to write a Java Program to count number of vowels in a string?
23. How can we write a Java Program to count the number of occurrences of a character in a String?
24. How should we write a Java Program to count number of words in a String?

◆◆◆

Features and Enhancements in jdk1.5

There are a few features and enhancements that are added in jdk1.5. These are listed below:

- Java Language Features
- Performance Enhancement
- Virtual Machine
- Base Libraries
- User Interface
- Deployment
- Tools Architecture
- OS and Hardware Platforms

Java Language Features were introduced for the ease of programmers and to make it simpler. They include the following:

- Generics
- Enhanced for loop
- Autoboxing/unboxing
- Typesafe Enums
- Varargs
- Static import
- Annotations

Generics

This feature of Java was introduced in 2004 and is used providing compile-time type safety,".

Read on to find out more about it, its uses as well as disadvantages.

- When you take an element out of a Collection, you must cast that element to the type of element that is stored in the collection.
- Besides being inconvenient, this is unsafe. The compiler does not check if your cast is the same as the collection's type, so it can fail at run time.
- Generics provide a way for you to communicate the type of a collection to the compiler so that it can be checked.
- Once the compiler knows the element type of the collection, the compiler can crosscheck whether you have used the collection consistently and can then insert the correct casts on values being taken out of the collection.

- The code using generics is clearer and safer. We have eliminated an unsafe cast and a number of extra parentheses.

When you see <Type>, read as "of type"

For example: ArrayList <String>al = new ArrayList<String>() ;

Read as : al is type of String.

Example of collection without Generics

```
package com.javabykiran;
 import java.util.ArrayList;
 import java.util.Iterator;
public class GenericsLab1 {
   public static void main(String[] args) {
       ArrayList al= new ArrayList(); // without Generics
             al.add("java");
             al.add(new Long("88888809416"));
             Iterator itr = al.iterator();
      while(itr.hasNext()){
        String s =" ";
        Object o =itr.next();
           if(o instanceof String){
                s = (String) o ;
            System.out.println(s);
           }
           if(o instanceof Long){
            System.out.println((Long)o);
           }
      }
   }
}
```

---Output---

```
java
88888809416
```

Example of collection with Generics:

```
package com.javabykiran;
import java.util.ArrayList;
import java.util.Iterator;
public class GenericsLab2 {
   public static void main(String[] args) {
ArrayList<String> al =new ArrayList<String>();//with Generics
```

```
        al.add("java");
        al.add("8888809416");
      Iterator<String> itr = al.iterator();
            while(itr.hasNext()){
                String p= itr.next();
                System.out.println();
             }
      }
}
```

---Output---

```
java
88888809416
```

Note:

1. boolean hasNext() returns true if there are more elements, otherwise it returns as false.
2. Object next() returns the next element. It Throws NoSuchElement Exception if there is no next element present.

Why should we use Generics?

There are a number of reasons why we would use generics.

- Stronger type checks at compile time
  - A Java compiler applies strong type checking to generic code and issues errors if the code violates type safety.
  - Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find
- Elimination of casts

The following code snippet without generics requires casting:

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);
```

When re-written to use generics, the code does not require casting:

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0); // no cast
```


java.util.ArrayList.get(int index)

The method returns the element at the specified position in this list.

public void add(int index, E element);

This adds the element to the specified index in the list.

For each loop or Enhanced For Loop

- The enhanced for loop is a popular feature introduced with the Java SE platform in version 5.0.
- It is also called as for each loop.

Example:

```
package com.javabykiran;
import java.util.ArrayList;
public class ForEachLab3 {
  public static void main(String[] args) {
    ArrayList<String> al = new ArrayList<String>();
      al.add("www.javabykiran.com");
      al.add("8888809416");
     for (String string : al) {   //for each loop
       System.out.println(string);
     }
  }
}
```

---Output---

```
www.javabykiran.com
8888809416
```

Autoboxing

Features and Enhancements Added In Jdk1.5

- The automatic conversion of primitive datatypes into its equivalent wrapper type is called as autoboxing. Its reverse operation is called as Unboxing
- As you know, we can't put an int (or other primitive value) into a collection. Collections can only hold object references, so you have to box primitive values into the appropriate wrapper class (which is an Integer in the case of int). When you take the object out of the collection, you get the Integer that you put in; if you need an int, you must unbox the Integer using the intValue method

Example of Autoboxing and Unboxing:

```
package com.javabykiran;
public class BoxingLab1 {
  public static void main(String[] args) {
      int a = 10;
      Integer i = a;
     //Boxing-- internally :
     Integer i = Integer.valueOf(a);

     System.out.println(i);
     int c = i;    //Unboxing -- :
     int c = i.intValue();
     System.out.println(c);
   }
}
```

TypeSafe Enums:

- An enum is a data type which contains fixed set of constants.
- It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY), or directions (NORTH, SOUTH, EAST and WEST) etc.
- The enum constants are static and final implicitly.
- It is available from JDK 1.5. It can be thought of as a class, i.e, enum can have constructors, methods and variables.
- Enum improves type safety; it can be easily used with a switch statement.

Example:

```
package com.javabykiran;
  enum Day {
          SUNDAY, MONDAY, TUESDAY,WEDNESDAY,THURSDAY,
          FRIDAY,SATURDAY }
public class EnumLab1 {
  public static void main(String[] args) {
       for (Day d : Day.values()) {
            System.out.println(d);
        }
   }
}
```


---Output---

```
SUNDAY
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SATURDAY
```

Note:

- enum have static 'values' method that returns an array containing all of the values of the enum in the order that they are declared in.
- This method is commonly used in combination with the for-each construct to iterate over the values of an enum type.

Varargs

Varargs are Variable Arguments which accepts none or more arguments.

- Varargs can be used for methods that can take zero or more arguments
- **Before varargs was introduced, we had to overload the methods**
- If we are uncertain about the number of arguments we would need, only then should we go for varargs.

Example:

```
package com.javabykiran;
public class VarargsLab {
    void m1(String...s){
        System.out.println("in a m1 method");
          for(String val :s){
            System.out.println(val);
          }
  }
public static void main(String...args) {
  //varargs can be apply to main method
   VarargsLab v = new VarargsLab();
     v.m1();                                  //0-args
     i.m1("www");                            //1-args
     v.m1("www",".","javabykiran");  //3-args
  }
 }
```

---Output---

```
in a m1 method
in a m1 method
www
in a m1 method
www
Javabykiran
```

Example

```
package com.javabykiran;
   public class VarargLab {
      int add (int...i){
      int sum=0;
         for (int add : i ){
            sum= sum+add;
         }
     return sum;
    }
  public static void main(String...args){
      VarargLab vl =new VarargLab();
        int add2=vl.add(1,2);
      System.out.println(add2);
        int add3=vl.add(1,2,3);
      System.out.println(add3);
        int add4=vl.add(1,2,3,4);
      System.out.println(add4);
    }
}
```

---Output---

```
3
6
10
```

Static Import

The static import feature lets fields and methods specified in a class as public static, and is used without naming the class in which the field is defined.

- It is used for accessing static members of any class.
- It is necessary to give the class name from where they came, if we don't use this concept.
- By using static import we can access static members directly, without giving the class name.
- Static imports can be applied only to static members.

Example:

```
package com.javabykiran;
import static java.lang.System.out;
import static java.math.BigDecimal.TEN;
public class StaImpLab {
  public static void main(String[] args){
        out.println("www.javabykiran.com");
         //no need to write System class
        //Because it has already been imported.
        out.println(TEN);
   }
}
```

---Output---

```
www.javabykiran.com
10
```

Annotations

Annotations are a form of metadata which gives information about programs which is not a part of the current program.

- It is a form of metadata.
- It provides data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code they annotate. Java Annotation is a tag that represents the metadata i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.
- There are many built in annotations present and we can also create custom annotations

Annotations are useful in the following ways :

1. **They have information for the compiler -**
   Annotations can be used by the compiler to detect errors or suppress warnings.
2. **They send Compile-time and deployment-time processing -**
   Software tools can process annotation information to generate code, XML files, and so forth.
3. **Runtime processing -**
   Some annotations are available to be examined at runtime.

For example : @Override

It assures that a subclass method is overriding the parent class method.

If it is not so, then the compiler will give an error.

```
package com.javabykiran;
  class A {
     void learn(){
       System.out.println("C");
     }
 }
public class AnnotationLab extends A {
     @Override
     void learn(){
        System.out.println("java by Kiran Sir");
     }
     //error void teach() method does not exist in the
     //parent class.
     /*@Override
      void teach(){
     }*/
  public static void main(String[] args) {
      AnnotationLab lab = new AnnotationLab();
      lab.learn();
   }
}
```

---Output---

```
java by Kiran Sir
```


Features and Enhancements in jdk1.6

Here are some of the features and enhancements in jdk1.6. And we will explore them further in this chapter:

Only important features we need for interview all other features you will understand as you get experience in industry.

- Collections Framework
- Deployment (Java Web Start and Java Plug-in)
- Drag and Drop
- Instrumentation
- Internationalization Support
- I/O Support
- JAR (Java Archive Files) - An annotated list of changes between the 5.0 and 6.0 releases to APIs, the jar command, and the jar/zip implementation
- Java Web Start
- Java DB 10.2 JDBC4 Early Access
- JMX (Java Management Extensions) - A list of JMX API changes between the J2SE 5.0 and Java SE 6 releases
- JPDA (Java Platform Debugger Architecture)
- JVM TI (Java Virtual Machine Tool Interface)
- lang and util Packages
- Monitoring and Management for the Java Platform
- JConsole is Officially Supported in Java SE6
- Networking Features
- Performance
- Reflection
- RMI (Remote Method Invocation)
- Scripting
- Security
- Serialisation of Objects
- Swing
- VM (Java Virtual Machine)

Collection Framework Enhancements

"The collections framework is a unified architecture for representing and manipulating collections, allowing them to be manipulated independently of the details of

their representation. It reduces programming effort while increasing performance.

It allows for interoperability among unrelated APIs, reduces effort in designing and learning new APIs, and fosters software reuse."

These new collection interfaces provided:

- Deque - a double ended queue, supporting element insertion and removal at both ends. It extends the Queue interface
- java.util.Deque interfaces is a subtype of the java.util.Queue interface
- It is also called as deque

Here is a list of topics that will be covered in this chapter:

- Deque Implementations
- Adding and accessing elements
- Removing elements
- Generic Deques

Deque Implementations can be implemented by a doubly linked list or circular array

- Being a Queue subtype, all methods of Queue and Collection interfaces are also available in the Deque interface
- There are two ways to use it:
- java.util.ArrayDeque
- java.util.LinkedList
- Array Deque stores its elements in an array, and as soon as it reaches its length it assigns it to a new array

To create a Deque instance, following the steps given below:

```
Deque dequeA = new LinkedList ();
Deque dequeB = new ArrayDeque ();
```

Adding and Accessing Elements

```
Deque dequeA = new LinkedList();
dequeA.add("element 1");      //add element at tail
dequeA.addFirst("element 2"); //addelement at head
dequeA.addLast ("element 3"); //addelement at tail
Object firstElement = dequeA.element(); //retrieves but does not remove first element
Object firstElement = dequeA.getFirst(); //retrieves the first element of the queue Object
lastElement = dequeA.getLast();              //retrieves the lastelement of the queue
Object firstElement = dequeA.remove(); //to remove an element for a deque
Object firstElement = dequeA.removeFirst(); //remove the first element
Object lastElement = dequeA.removeLast(); //removes the lastelement
```


Removing Elements

```
Object firstElement = dequeA.remove();
Object firstElement = dequeA.removeFirst();
Object lastElement = dequeA.removeLast();
```

GenericsDeque:

The generic deque is used to make sure which type of objects get inserted into deque.

```
Deque<MyObject> deque = new LinkedList<MyObject>();
```

This Deque can now only have MyObject instances inserted into it. You can then access and iterate its elements without casting them.

Here is how it looks:

```
My Object my Object = deque.remove();
  for(MyObject anObject : deque){
        //do someting to anObject...
  }
```

BlockingDeque

A blocking deque supports blocking operations

- It is a Deque with operations that wait for the deque to become non- empty when retrieving an element
- Additionally, it waits for space to become available in the deque when storing an element
- It extends both the Deque and BlockingQueue interfaces (This interface is part of java.util.concurrent)
- It is also thread Safe
- It does not allow null elements in
- It may or may not be capacity-constrained
- A BlockingDeque implementation may be used directly as a FIFO BlockingQueue

NavigableSet

It refers to a navigable set within the collections framework

- It is a SortedSet extended with navigation methods reporting closest matches for given search targets
- A NavigableSet may be accessed and traversed in either ascending or descending order depending on the requirement
- This interface is intended to supersede the SortedSet interface
- It knows all Superinterfaces : Collection<E>, Iterable <E>, Set<E>, Sorted Set<E>
- It knows all Implementing Classes : ConcurrentSkipListSet, TreeSet

NavigableMap

It is a subtype of the SortedMap interface and makes navigation methods more convenient. It also has an added feature to make a submap from an existing map.

- A Navigable map is a SortedMap extended with navigation methods which returns the closest matches for given searchtargets
- A NavigableMap may be accessed and traversed in either ascending or descending key order
- This interface is intended to supersede the SortedMap interface
- Interface NavigableMap<K,V>: K is the type of key maintained by this map and V is the type of mapped values
- It knows all Superinterfaces : Map<K,V>, SortedMap<K,V>
- It knows all SubInterfaces : ConcurrentNavigableMap<K,V>
- It knows all known Implementing Classes: ConcurrentSkipListMap, TreeMap

ConcurrentNavigableMap

The ConcurrentNavigableMap provides support for concurrent access for the submaps. (This interface is a part of java.util.concurrent)

- The ConcurrentMap is also a NavigableMap.
- It knows all Superinterfaces:
  ConcurrentMap<K,V>,Map<K,V>, NavigableMap<K,V>, SortedMap<K,V>
- It knows all Implementing Classes: ConcurrentSkipListMap

The following concrete implementation classes have been added:

ArrayDeque, also known as Array Double Ended Queue or Array Deck, is an efficient resizable-array implementation of the Deque interface

- Concurrent Skip List Set - concurrent scalable skip list implementation of the Navigable Set interface
- Concurrent Skip List Map - concurrent scalable skip list implementation of the ConcurrentNavigableMap interface
- Linked Blocking Deque - concurrent scalable optionally bounded FIFO blocking deque backed by linked nodes
- Abstract Map. Simple Entry - simple mutable implementation of Map.Entry
- Abstract Map.SimpleImmutable Entry - simple immutable implementation of Map.Entry
- These existing classes have been updated to implement new interfaces:
- LinkedList
- TreeSet
- TreeMap
- Collections

Two new methods were added to the Collections utility class:

- new Set From Map (Map) – This creates a general purpose Set implementation from a general purpose Map implementation

There is no IdentityHashSet class, instead, you may use
Set<Object> identityHashSet= Collections.newSetFromMap(new IdentityHashMap<Object, Boolean>());

- asLifoQueue(Deque) - returns a view of a Deque as a Last-in- first-out (LIFO) Queue

The Arrays utility class now has methods copyOf and copyOfRange that can efficiently resize, truncate, or copy subarrays for arrays of all types.

Before:

```
int[] newArray = new int[newLength];
System.arraycopy(oldArray,0, newArray,0,oldArray.length);
After:
int[] newArray = Arrays.copyOf(a, newLength);
```

Deque and ArrayDeque

- Deque is the abbreviation of "Double Ended Queue". It is a collection that allows us to add (or) remove elements at both ends. Deque supports the total size of collection for both fixed and unspecified size limits
- Deque implementation can be used as Stack (Last In First Out) or Queue (First In First Out). For each insertion, retrieval and removal of elements from deque, there are two different methods. One will throw exception if it fails in an operation and the other returns the status or special value for each operation.

Operation	Special value method	Exception throwing method
Insertion at head	offerFirst(e)	addFirst(e)
Removal at head	pollFirst()	removeFirst()
Retrieval at Head	peekFirst()	getFirst()
Insertion at Tail	offerLast(e)	addLast(e)
Removal at Tail	pollLast()	removeLast()
Retrieval at Tail	peekLast()	getLast()

- Implementation of Deque doesn't require preventing the insertion of null, but when we are using special value method, null is returned to indicate that the collection is empty. So, it is recommendable to not allow insertion of null
- ArrayDeque is a class that implements Deque. It has no capacity restrictions. It will perform faster than stack when used as stackand faster than linked list when used as queue. ArrayDeque is not thread safe

The following example explains how to write program using ArrayDeque:

```
package com.javabykiran;
import java.util.ArrayDeque;
import java.util.Iterator;
public class ArrayDequeLabl {
  public static void main(String[] args){

     ArrayDeque<String> ad = new ArrayDeque<String>();
     //Insertion by using various methods ad.add("by");
     ad.addFirst("java");
     ad.addLast("kiran");
     System.out.println(ad);

//Retrievals

System.out.println("Retrieving First Element:"+ad.peekFirst());
System.out.println("Retrieving Last Element :"+ad.peekLast());
//Reverse traversal
System.out.println("Reverse traversal");
     Iterator<String> itr = ad.descendingIterator();
      while(itr.hasNext())
         {
                System.out.println(itr.next());
         }
          //Removals
System.out.println("Removing First Element:"+ ad.pollFirst());
System.out.println("Removing Last Element :"+ ad.pollLast());
     }
  }
```

---Output---

```
[java, by, kiran]
Retrieving First Element :java
Retrieving Last Element :kiran
Reverse traversal
Kiran
By
Java
Removing First Element :java
Removing Last Element :kiran
```


Blocking Deque and Linked Blocking Deque

A blocking deque is basically a deque that waits for a deque to be non-empty while it retrieves an element. It will also wait for an empty space in the deque before storing an element.

A BlockingDeque is similar to Deque and provides additional functionality. When we try to insert an element in a BlockingDeque which is already full, it can wait till the space becomes available to insert an element. We can also specify the time limit for waiting.

There are four BlockingDeque methods:

- Methods throws exception
- Methods returns special value
- Methods that blocks (Waits indefinitely for space to available)
- Methods that times out (Waits for a given time for space to available)

Operation	Special Value	Throws Exception	Block	Times Out
Insertion Head	Add First(e)	Offer First(e)	Put First(e)	Offer First (e, time, unit)
Removal from head	Remove first( )	Poll First( )	Take First( )	Take First (Time, Unit)
Retrieval from head	Get First( )	Peak First( )	NA	NA.
Insertion at tail	Add Last(e)	Offer Last(e)	Put Last(e)	Offer Last (e, Time, Unit)
Removal from tail	RemoveLast( )	Poll Last( )	Take Last( )	Take First (Time, Unit)
Retrieval from tail	Get Last( )	Peak Last( )	NA	NA

A LinkedBlockingDeque is a Collection class, which implements BlockingDeque interface in which we can specify maximum capacity if we want. If we did not specify the capacitythen the maximum capacity will be Integer.MAX_VALUE by default.

Example:

```
package com.javabykiran;
import java.util.concurrent.*;
 class LinkedBlockingDequeLab implements Runnable {
    LinkedBlockingDeque lbd= new LinkedBlockingDeque(1);
    volatile boolean b = true;
   public void run() {
     try {
```

```
/*First thread once enters into the block it modifies
instance variable b to false and prevents second thread to
enter into the block */
  if(b){
     b = false;
   Thread.sleep(5000);//Makes the Thread to sleep for 5
                      // seconds
     System.out.println("Removing"+lbd.peek());
    lbd.poll();//Removing an element from collection
  }
else {
     System.out.println("Waiting ");
/*This method makes to wait till the first thread removes
// an elements*/
     lbd.put('B');
    System.out.println("Inserted "+lbd.peek());
   }
       } catch (Exception e){
  e.printStackTrace();

     }

}
public static void main(String[] args) throws Exception {
LinkedBlockingDequeLab bdeObj = new LinkedBlockingDequeLab();
   bdeObj.lbd.offer('A');
   System.out.println("Inserted"+bdeObj.lbd.peek());
   Thread tMainObj = new
   Thread(bdeObj); tMainObj.start();
   Thread tSubObj = new
   Thread(bdeObj); tSubObj.start();
  }
}
```

---Output---

```
Inserted A
Waiting
Removing A
Inserted B
```


NavigableSet and ConcurrentSkipListSet

"A navigable set is a sorted set that lets you work with its subsets in a variety of ways."

Suppose we have a requirement to:

- Retrieve the element which is immediately greater than or lower than 15 from the given set: [5,10,15,20]
- Retrieve all elements greater than or lower than 10

  With the help of existing methods we need to take few risks to achieve them. But with NavigableSet methods it becomes just a method call.

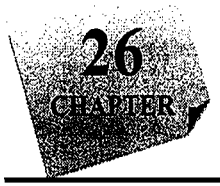
  The NavigableSet method is used to return the closest matches of elements for the given elements in the collection.

  The ConcurrentSkipListSet is one of the classes that implements NavigableSet.
- Navigable set has the following interfaces: Collection<E>, Iterable<E>, Set<E>, SortedSet<E>, All Known implementing classes ConcurrentSkipListSet, TreeSet

What are the salient features of NavigableMap and ConcurrentSkipListMap and how do they differ?

- NaviagableMap is similar to NaviagableSet
- ConcurrentNavigableMap<K,V>
- In NavigableSet, methods are used to return values, but in NaviagableMap, methods used to return the key, value pair
- ConcurrentSkipListMap is the one of the classes which implements NaviagableMap
- It knows all Superinterfaces Map<K,V>, SortedMap<K,V>
- It knows all known implementing Classes ConcurrentSkipListMap, TreeMap

◆◆◆



Features and Enhancements in jdk 1.7

Given below are the areas in which new features are added and enhancement is done:

- Swing
- IO and New IO
- Networking
- Security
- Concurrency Utilities
- Rich Internet Applications (RIA)/Deployment
- Java 2D
- Java XML - JAXP, JAXB, and JAX-WS
- Internationalization
- java.lang Package
- Java Programming Language
- Java Virtual Machine (JVM)
- JDBC
- Java Language Features
- **Binary Literals**
- **Underscore in Numeric literals**
- **String in Switch statements**
- **The try with resources statement**
- **Catching multiple exceptions by single Catch**

Binary Literals

Binary literals is a new addition introduced in Java 7, wherein we can express the integral types (byte,short,int & long) by using Binary number System. To specify a binary literal, add the prefix 0b or 0B to the number.

The following example shows how binary literals work :

```
package com.javabykiran;
public class BinaryLiteLabl {
   public static void main(String[] args) {
       int One = 0b001; //1 in binary form
       int Two = 0b010; //2 in binary form
       System.out.println(One);
       System.out.println(Two);
   }
}
```

---Output---

```
1
2
```


Underscore in Numeric Literals

From the time Java 7 was introduced we can write any number of underscore characters like (_) and which can appear anywhere between the digits. This feature enables you to separate groups of digits in numeric literals,which can improve the readability of your code.

It must be kept in mind though, that underscores will be omitted in operations.

```
package com.javabykiran;
public  class UnderScoreLabl {
   public static void main(String[] args) {
      long creditCardNumber = 1234_5678_9012_3456L;
      long adharCardNumber  = 1232_3400_5432_4564L;
      System.out.println(creditCardNumber);
      System.out.println(adharCardNumber);
   }
}
```

---Output---

```
1234567890123456
1232340054324564
```

String in Switch statements

Before jdk 7, only numbers were allowed in switch Case statement. But now, in jdk 7, we can use string object in switch case Statements.
Switch case strings should be enclosed in "double quotes."

Example:

```
package com.javabykiran;
public class SwitchCaseLab {
   public void getTypeofDay(String dayOfWeekArg) {
     String typeOfDay;
    switch (dayOfWeekArg) {
    case "Monday":
```

```
            typeOfDay = "Start of work week";
               System.out.println(typeOfDay);
            break;
        case "Tuesday":
        case "Wednesday":
        case "Thursday":
             typeOfDay = "Midweek";
               System.out.println(typeOfDay);
             break;
        case "Friday":
             typeOfDay = "End of work week";
                System.out.println(typeOfDay);
             break;
        case "Saturday":
        case "Sunday":
            typeofDay = "Weekend";
                System.out.println(typeOfDay);
            break;
        default:
      throw new IllegalArgumentException("Invalid day of the week:
" + dayOfWeekArg);
          }
      }
     public static void main(String[] args){
        SwitchCaseLab lab = new SwitchCaseLab();
               lab.getTypeofDay("Monday");
               lab.getTypeofDay("Sunday");
         }
}
```

---Output---

```
Start of work week
Weekend
```


The try with resources statement

Question : How to deal with resource management and how has it changed with Java 7?

Managing resources that needed to be explicitly closed was somewhat tedious before Java 7. Any object that implements java.lang.AutoCloseable, can be used as a resource. So, to have the best resource management, you need to:

- You have to write finally block to make sure that all the resources are freed before closing of the program before this concept was introduced
- Try with resources that will make sure that the resource will be freed no matter whether try completes abruptly or successfully
- Try must have catch or must have throws clause to use this concept

Consider the following Example:

```
package com.javabykiran;
import java.io.FileInputStream;
import java.io.IOException;
public class TryWithResLab {
  private static void printFileJava7() throws IOException {
     try(FileInputStream input = new FileInputStream
                                    ("D:\\notes\\abc.txt")) {
           int data = input.read();
           while(data != -1){
              System.out.print((char) data);
              data = input.read();
            }
       }
  }
public static void main(String[] args) throws IOException {
       TryWithResLab.printFileJava7();
   }
}
```

---Output---

```
www.javabykiran.com
```

How can we Catch multiple exceptions by using single Catch:

- Before this concept introduced for handling more than one exception we need to write mutiple catch statements
- Now, in Jdk 7, we can write multiple catch exceptions in single catch statement

Example

A single catch block can handle more than one typeof exception

```
package com.javabykiran;
import java.sql.SQLException;
public class TryWithMultiLab{
   public static void main(String[] args) {
     try{
         //int i =
         10/0; int i=
         10/5;
         throw new SQLException();
      }catch (ArithmeticException | SQLException e) {
          e.printStackTrace();
      }
   }
}
```

---Output---

```
java.sql.SQLException
at com. javabykiran. Try With Multi Lab. main (Try With Multi Lab. java:13)
```


27 CHAPTER Features and Enhancements in jdk 1.8

Before we look into Java Stream API Examples, let's see why it was required. Suppose we want to iterate over a list of integers and find out sum of all the integers greater than 10. All the Java Stream API interfaces and classes are in the java.util.stream package.

Prior to Java 8, the approach for getting 1 to 10 sum, it would be:

```
int addNumbersNojdk8(List<Integer> list) {
     int sum = 0;
     for (int number : list) {
          if (number > 10) {
               sum = sum + number;
          }
     }
     return sum;
}
```

After java 8 it would be:

```
int addNumbersWithjdk8(List<Integer> list) {
       return list.stream().filter(p->p>10).mapToInt(p ->
          p).sum();
  }
```

One more example where we will see use of IntStream. Let's say we want to find a valid string having only alphabets.

```
package com.javabykiran.jdk8.ex;
public class testStringWithAlphabets {
     public static boolean isStringOnlyAlphabet(String str){
          return ((str != null) && (!str.equals("")) &&
        (str.chars().allMatch(Character::isLetter)));
     }
```

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

In above program highlighted allMatch is from Stream package and chars method is added in jdk1.8.

Now we need to understand about functional programing. Java 8 supports functional programing where Stream API's are majorly used.

Stream explanation will be discussed in later part of this chapter.

Functional programming

Program with Functional Programming

```
public double CalWithFunctionalPrograming(double x,double y){
        return x * x + y * y;
}
```

Program without Functional Programming

```
public double CalWithOutFunctionalPrograming(double x,double y)
        double z=x+x;
        double z1=y+y;
        double z2=z+z1;
        return z2;
    }
```

Remember for functional programming

A programming style that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.

F(x)=y or g(f(y(x))) =z

Based on lambda calculus.(Some inputs are transformed to some output without modifying the input)

No iteration, No for loop, no variables

Functional Interface

Before proceeding, one more concept we need to understand is functional interface.

An interface with exactly one abstract method is called Functional Interface.

@FunctionalInterface annotation is added so that we can mark an interface as functional interface.

It is not mandatory to use it, but it's recommended to avoid addition of extra methods accidentally. If the interface is annotated with @FunctionalInterface annotation and we try to have more than one abstract method, it throws compiler error.

Java 8 Collections API has been rewritten and new Stream API is introduced that uses a lot of functional interfaces. Java 8 has defined a lot of functional interfaces in java.util.function Package. Some of the useful java 8 functional interfaces are Consumer , Supplier , Function and Predicate.

java.lang.Runnable is a great example of functional interface with single abstract method run().

```
@FunctionalInterface
public interface MyFirstFunctionalInterface {
    public void firstWork();
}
```

Also, since default methods are not abstract you're free to add default methods to your functional interface as many as you like.

Below program will have compile time errors.

```
package com.javabykiran.FunctionInterfaceEx;
@FunctionalInterface
public interface TestI {
    int add(int a, int b);
    int sub(int a, int b); // Not allowed
}
```

Try to understand below example :

```
package com.javabykiran.FunctionInterfaceEx;
@FunctionalInterface
public interface TestI {
      int doOperation(int a, int b); //only one allowed

      default String multiply() { // have a body
            return null;
      }
      default String divide() { // have a body
            return null;
      }
      @Override
      public String toString();
      // toString can't have body as it is from  object class
}
```

Summarizing below points about functional interface.

- Functional interface have only one abstract method, No need to count object class methods here [toString method in above example]
- Can have any no of default methods [default methods with body]
- Cannot have same name for default methods for those methods which are in object class [in above example toString method we just write without body]
- @FunctionalInterface annotation is added so that we can mark an interface as functional interface which is not mandatory.

After functional interface understanding we will try to understand Lambda expressions. Lambda expression facilitates functional programming, and simplifies the development a lot.

Lambda expressions

Syntax:

Option 1:

Input parameter - > expression on input parameters

Option 2:

Function Interface = Input parameter - > expression on input parameters

Lambda expressions are used primarily to define inline implementation of a functional interface, i.e., an interface with a single abstract method only.

How to use functional interface with lambda expressions

```
package com.javabykiran.FunctionInterfaceEx;
@FunctionalInterface
public interface TestI {
      int doOperation(int a, int b);
}
```

```
package com.javabykiran.FunctionInterfaceEx;
public class TestImpl {
      public static void main(String[] args) {
            TestI testI = (a, b) -> a + b;
      System.out.println(testI.doOperation(334, 556));
            TestI testISub = (a, b) -> a - b;
      System.out.println(testISub.doOperation(334, 556));
  }
}
```

Question: What advantage you think we get from functional interface and lambda expression here?

Answer: We can have different implementation of doOperation method inline here. Anonymous inner class can also do same but lot more code we needed to write here.

I will show you different flavors to call this operation

```
public class TestImpl {
    public static void main(String[] args) {
        // with type declaration
        TestI add = (int a, int b) -> a + b;
        // with out type declaration
        TestI sub = (a, b) -> a - b;
        // with return statement along with curly braces
        TestI mul = (a, b) -> {return a + b;};
    //without return statement and without curly braces
        TestI divide= (a, b) -> a / b;
    }
}
```

default method

Example to use default method

```
public interface TestI {
    default String multiply() {
        return "I am in TestI mulitply";
    }
}
```

```
public class TestImplTest implements TestI {
    public static void main(String[] args) {
        TestImplTest tit = new TestImplTest();
        tit.multiply();
    }
}
```

---Output---

```
I am in TestI mulitply
```



```
// If TestImplTest wants to implements its own behavior then
// we can implement it as below
public class TestImplTest implements TestI {
    String multiply() {
        return "I am in TestImplTest mulitply";
    }
    public static void main(String[] args) {
        TestImplTest tit = new TestImplTest();
        tit.multiply(); // calling default method
    }
}
```

---Output---

```
I am in TestImplTest multiply
```

Static default method

- We can have only static or default or abstract method in interfaces in jdk 8.
- Static or default method will have a body
- Only one keyword can be used at a time. Static or default
- You can define static default methods in interface which will be available to all instances of class which implement this interface.
  - This makes it easier for you to organize helper methods in your libraries; you can keep static methods specific to an interface in the same interface rather than in a separate class.
  - This enables you to define methods out of your class and yet share with all child classes.
  - They provide you a highly desired capability of adding a capability to number of classes without even touching their code. Simply add a default method in interface which they all implement.

Example to use static default method.

```
package com.javabykiran.FunctionInterfaceEx;
public interface TestI {
	static String multiply() {
		return null;
	}
	default String divide() {
		return null;
	}
	@Override
	public String toString();
}
```

```
public class TestImplTest  {
	public static void main(String[] args) {
		TestI.multiply();//  calling static method
	}
}
```

Till now we have covered below concepts

1. Functional programing
2. Functional interface
3. Lambda expression and syntax
4. Default method in interface
5. Static method in interface

Stream API

Majorly we use three methods in this map, filter and collect method

See below example :

```
ArrayList<Employee> alEmpList = new ArrayList<> ();
ArrayList <Integer> alEmp = (ArrayList<Integer>)
   alEmpList.stream().map(age -> age.getAge()).filter(age ->
age > 30).collect(Collectors.toList());
```

Explanation is ,

In this example we want to filter all age greater than 30.Expectation is output should be all numbers greater than 30.

1) **Stream Method** : It is converting array list of Employees into small chunks and making it ready for next processing.
2) **Map method** : It is used to take out some elements from stream of collection. With example we will try to understand.
   a. ArrayList contain Employee and Employee contains age if we use map on Stream of ArrayList then we get collection of age.
   b. ArrayList contain Employee and Employee contains Phone and Phone contain mobile no as integer if we use map on Stream of ArrayList then we get collection of Phone then again use Map on Phone to get collection of mobile nos.

```
alEmpList.stream().map(phone->phone.getPhone())
                  .map(mobileNo->mobileNo.getMobileNo())
                  .filter(mobileNo -> mobileNo > 30)
                  .collect(Collectors.toList());
```

3) **Filter Method :** It is used to filters a data from map here we can use lambda expressions to make it short. We can pass predicates as well here see next section.
4) **Collect Method** : It is used to collection all elements in a list

Below is complete example for all above methods :

```
package com.javabykiran;
  public class Employee {
        int age;
        String loc;
        String name;
        Phone phone;

     public Phone getPhone() {
          return phone;
     }
     public void setPhone(Phone phone) {
          this.phone = phone;
     }
     public Employee(int age, String loc, String name) {
          super();
          this.age = age;
          this.loc = loc;
           this.name = name;
     }
     public int getAge() {
          return age;
     }
     public void setAge(int age) {
          this.age = age;
     }
     public String getLoc() {
          return loc;
     }
     public void setLoc(String loc) {
          this.loc = loc;
     }
     public String getName() {
          return name;
     }
     public void setName(String name) {
          this.name = name;
     }
```



```
     @Override
     public String toString() {
          return "Employee [age=" + age + ", loc=" + loc + ",
 name=" + name + ", phone=" + phone + "]";
     }
 }
```

```
package com.javabykiran;

 public class Phone {
     int mobileNo;

     public int getMobileNo() {
          return mobileNo;
     }
     public void setMobileNo(int mobileNo) {
          this.mobileNo = mobileNo;
     }
     @Override
     public String toString() {
          return "Phone [mobileNo=" + mobileNo + "]";
     }
 }
```

Above classes were just POJO now we will see how we can do different operations with collections

```
package com.javabykiran;
 import java.util.ArrayList;
 import java.util.List;
 import java.util.stream.Collectors;
 public class MapFilterCollectEx {
     public static ArrayList<Employee> createEmployees(){
          ArrayList<Employee> alEmp = new ArrayList<>();
          alEmp.add(new Employee(20, "Pune", "Kiran"));
          alEmp.add(new Employee(46, "Nagpur", "Kiran1"));
          alEmp.add(new Employee(34, "Mumbai", "Kiran2"));
          return alEmp;
     }
 // return all emp of more age than 30
```

```
    public static ArrayList<Employee> allEmpAge() {
          ArrayList<Employee> alEmpList = createEmployees();
          alEmpList = (ArrayList<Employee>)
          alEmpList.stream().filter(finalAge ->
          finalAge.getAge()>30)
                    .collect(Collectors.toList());
          return alEmpList;
    }

    // return all elements
    public static List<Integer> allEmpMobileNo() {
          ArrayList<Employee> alEmpList = createEmployees();
          List<Integer> alEmp = alEmpList.stream().map
          (phone ->phone.getPhone()).map(mobileNo ->
          mobileNo.getMobileNo()).filter(mobileNo ->
          mobileNo > 30).collect(Collectors.toList());
          return alEmp;
    }

    // return all emp of more age than 30
  public static ArrayList<Integer> allEmpAgeGreaterThan30() {
          ArrayList<Employee> alEmpList = createEmployees();
          ArrayList<Integer> alEmp = (ArrayList<Integer>)
        alEmpList.stream().map(age -> age.getAge())
        .filter(age -> age > 30).collect(Collectors.toList());
          return alEmp;
    }
public static void main(String[] args) {
          for (int age : allEmpAgeGreaterThan30()) {
                System.out.println(age);
          }
          for (Employee empAge : allEmpAge()) {
                System.out.println(empAge.getAge());
          }
          for (int empMobNo : allEmpMobileNo()) {
                System.out.println(empMobNo);
          }
    }

}
```


We can directly apply filter method on Stream as well

```
int addNumbersWithjdk8(List<Integer> list) {
          return list.stream()
                    .filter(p->p>10)
                    .mapToInt(p -> p).sum();
     }
 List<Integer> getListGreaterthan10(List<Integer> list){
          return list.stream()
                    .filter(p->p>10)
                    .collect(Collectors.toList());
     }
```

Predicate

How to design predicate :

```
public static Predicate<Employee> getAllOldEmp() {
          return p -> p.getAge() > 30;
     }
public static Predicate<Employee> getAllOldEmp1() {
          return p -> p.getAge() > 45;
     }
```

In this case P can be anything may be x or y or z

P indicates Employee in this case. Automatically it behaves like employee object.

Now to filter employees how I can use above predicate?

```
public static List<Employee> filterEmployees(List<Employee>
 employees, Predicate<Employee> predicate) {

       return employees.stream().filter (getAllOldEmp())
                         .collect (Collectors.toList ());
 }
```

Above method filters employee as per predicates provided.

Some more simple examples of predicate

```
public class PredicateExample1 {

 public static void main(String[] args) {

   //Predicate String
   Predicate<String> predicateString =
   s -> { return s.equals("Hello"); };
   System.out.println(predicateString.test("Hello"));
    System.out.println(predicateString.test("Hello World"));

   //Predicate integer
    Predicate<Integer> predicateInt =i -> { return i > 0; };
   System.out.println(predicateInt.test(5));
    System.out.println(predicateInt.test(-5));
   }

 }
```

---Output---

```
true
false
true
false
```

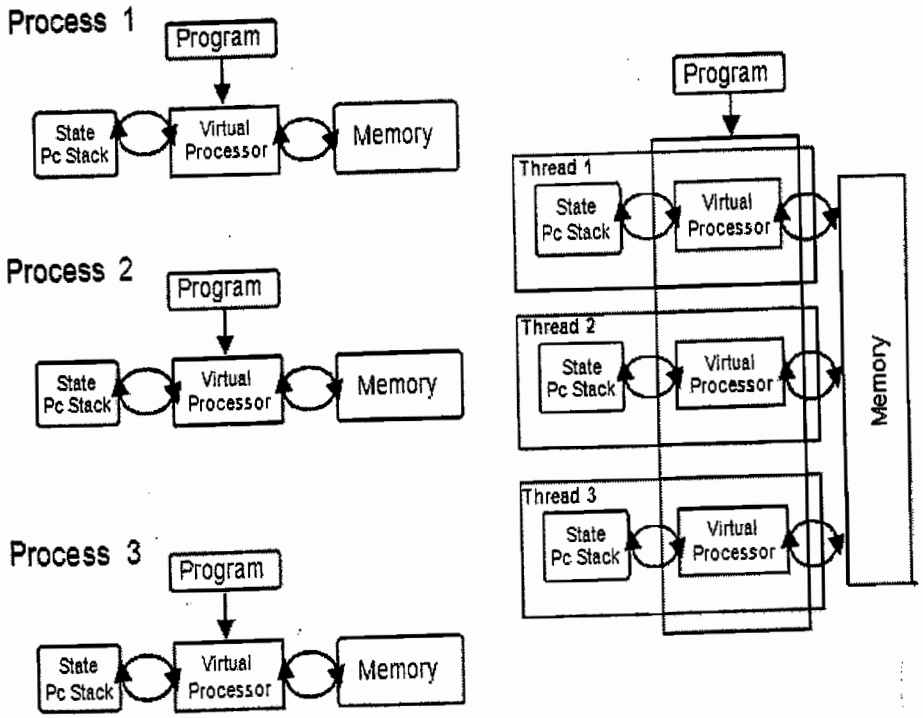

28 CHAPTER

Multithreading

Multitasking is a process of executing one or more tasks concurrently or at a time. Similarly, multithreading is the process of executing one or more threads at the same time or concurrently.

Multitasking can be implanting in two ways:

- Multiprocessing
- Multithreading



Multiprocessing	Multithreading
Executing multiple programs or processes concurrently is called multiprocessing. These programs may be belong to the same programming language or a different one	Executing multiple threads belonging to a single program or different programs which are implemented in same programming language is called multithreading.
Multiple programs running concurrently may occupy different memory spaces.	Different threads running concurrently may occupy a single memory space.
Context switching between processes is expensive	Context switching between threads is easy and not expensive
Data sharing between programs is expensive	Data sharing threads is easy and cheap

```
package com.javabykiran.Thread;
/**
 * @author Java By Kiran
 */
public class Lab1 {
    public static void main(String[] args) {
        System.out.println("main begin");
        Thread t = Thread.currentThread();
        System.out.println(t);
        System.out.println(t.getName());
        System.out.println(t.getPriority());
        t.setName("JAVABYKIRAN");
        t.setPriority(9);
        System.out.println(t);
        System.out.println(t.getName());
        System.out.println(t.getPriority());
        System.out.println("main end");
    }
}
```

---Output---

```
main begin
Thread[main,5,main
] main
5
Thread[JAVABYKIRAN,9,main]
JAVABYKIRAN
9
main end
```


When we try to run any class

We need keep in mind the following: whenever main method get call automatically

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

1) Takes the class name
2) Takes the command line value
3) Creates the String array object
4) If command line value is there then it stores command line value in String array
5) Creates the Thread group with name main
6) Creates the Thread with :
   a. Name-main
   b. Priority-5
   c. Group - main
7) Starts the main thread

Listed below are a few points that must be noted:

- By default, JVM is created the main thread and is started, which is responsible for invoking the main method
- You can write your own threads which are called by user defined threads and are called as child threads
- You can also implement code for required tasks in the child threads
- The JVM always starts the main method and all child threads will also start from the main thread

You can develop user defined threads in two way:

- By extending java.lang.Thread class
- By implementing java.lang.Runnable interface

This is how you can develop Threads by extending java.lang.Thread class:

```
package com.javabykiran.thread;
public class ThreadOne extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            // Displaying the numbers from this thread
            System.out.println("Running First Thread : " + i);
            try {
                Thread.sleep(1000);
            } catch (Exception e) {

            }
        }
    }
}
```

```
package com.javabykiran.thread;

public class ThreadTwo extends Thread {

     public void run() {
          for (int i = 1; i <= 5; i++) {
           System.out.println("Running Second Thread : " +
i);

                try {
                     Thread.sleep(500);
                } catch (Exception e) {

                }
           }
     }
}
```

```
package com.javabykiran.thread;

public class ThreadTest {
    public static void main(String[] args) {
     System.out.println(Thread.currentThread().getName() +
" Thread Begin...");
          ThreadOne firstThread = new ThreadOne();

          ThreadTwo secondThread = new ThreadTwo();

          firstThread.start();

          secondThread.start();
     }

}
```

---Output---

```
main Thread Begin...
Running First Thread : 1
Running Second Thread : 1
Running Second Thread : 2
Running First Thread : 2
Running Second Thread : 3
Running Second Thread : 4
Running First Thread : 3
Running Second Thread : 5
Running First Thread : 4
Running First Thread : 5
```


And this is the way to Develop Threads by implementing java.lang.Runnable interface:

```
package com.javabykiran.Thread;
 /**
  * @author Java By Kiran
 */
 public class Stack {
  int x;
   void push(int a) {
     x = a;
   }
   int pop() {
      return x;
   }
}
```

```
package com.javabykiran.Thread;
/**
 * @author Java By Kiran
 */
public class ThreadsA implements Runnable {
 Stack st = null;
  public ThreadsA(Stack st, String tname) {
  Thread t = new Thread(this, tname);
     this.st = st;
     t.start();
  }
  public void run() {
     for (int i = 1; i <= 5; i++)
         { st.push(i);
        System.out.println(i + " is pushed by A");
        try {
             Thread.sleep(500);
          } catch(Exception e) {
            }
       }
    }
}
```

```
package com.javabykiran.Thread;
/**
* @author Java By Kiran
*/
public class ThreadsB extends Thread {
  Stack st = null;
public ThreadsB(Stack st, String tname) {
  super(tname);
  this.st = st;
  start();
}
public void run() {
  for (int i = 1; i <= 5; i++) {
   int x = st.pop();
      System.out.println(x + " is poped by B");
  Try {
          Thread.sleep(500);
        } catch (Exception e) {
       }
     }
   }
 }
```

```
package com.javabykiran.Thread;
/**
 * @author Java By Kiran
 */
public class Lab3 {
  public static void main(String[] args) {
  Stack st = new Stack();
     ThreadsA a = new ThreadsA(st, "A");
     ThreadsB a1 = new ThreadsB(st, "B");
     for (int i = 1; i <= 5; i++) {
   System.out.println(i+"by"+Thread.currentThread().getName());
        try {
            Thread.sleep(500);
        } catch (Exception e) {
        }
      }
    }
  }
```


Extending thread	Implementing Runnable
Your user defined thread class has to extend java.lang.Thread From your thread class, you can invoke the following threads class constructors: Thread (string) and Thread (Thread Group, string). **Example:** `public class Demo extends Thread { public void run() { } public static void main(String args[]){ Demo d1 = new Demo(); d1.start(); } }`	Your user defined thread class has to implement java.lang.Runnable. When your thread class is implementing Runnable interface, you have to explicitly create the thread with one of the following constructors: Thread(Runnable) Thread(Runnable, string) Thread (Thread Group, Runnable, string) Thread (Thread Group, Runnable) **Example:** `public class Demo implements Runnable { public void run() { } public static void main(String args[]) { Demo d1 = new Demo(); Thread t1 = new Thread(d1) // composition t1.start(); } }`

- When you create and start the thread, it will be placed in a ready to run state, and the thread will just wait for the CPU → start()
- When the CPU scheduler allocates the CPU for them, t thread will be placed in running and be utilised by the CPU → run()
- Depending on the CPU scheduling algorithm, the thread will be moved to ready to run from running state
- When you call the sleep() method on the running thread, the thread will go into a sleep state and stay in that sleep state for a specified amount of time. Once the time is up, the thread will be moved into a ready to run state automatically
- sleep() method will overload with the following version: voidsleep(long)
  - The thread sleeps for specified amount of milliseconds and nano seconds void sleep(long , int)
- When you call the wait() method during the running state, the thread will go into the wait state

- To move the thread to wait, you can call one of the following methods of object class
  - void wait()
  - void wait(long)
  - void wait(long,int)

Thread waiting in wait state there until either notify() or notifyAll() is called

- If notify() or notifyAll() is called before the specified amount of time, the thread will be moved to ready to run state
- If specified amount of time is over without notifying, the thread will be moved to ready to run state automatically
- When you call the notify() method, the thread which has been waiting for a long time in the wait state will be moved into the ready to run state
- When you call the notifyAll() method, all the threads which are waiting will be moved to ready to run state
- When a thread is waiting for an unavailable resource, the thread will be blocked and placed in blocked state. Later, when the requested resource is available, it will be moved to ready to run state
- A blocked state is very dangerous since it may leads to deadlocks
- Consider the following scenario: there are threads running concurrently. Call t1 and t2 and hold the resources called r1 and r2 respectively.

  What would you do? → class A t1
  → class B t2

```
package com.javabykiran.Thread;
public class A extends Thread {
   @Override
   public void run()  {
      synchronized (String.class)
      {
         // here T1 comes wait for
         // somebody to release Integer lock
         // but here T1 is already acquired this lock
         // so not possible
       synchronized (Integer.class) {
             System.out.println("I am in A");
             }
          }
      }
 }
```



```
package com.javabykiran.Thread;
public class B extends Thread {
   @Override
    public void run() {
          synchronized (Integer.class) {
         // here T2 comes wait for
         // somebody to release String lock
         // but here T2 is already acquired this lock
         // so not possible
      synchronized (String.class) {
           System.out.println("I am in B");
       }
      }
    }
 }
```

```
package
com.javabykiran.Thread; public
class DeadLockTest {
  public static void main(String[] args)
     { A a = new A();
     a.setName("A")
     ; a.start();
     B b=new B();
     b.setName("B")
     ; b.start();
   }
 }
```

The thread will be dead in two ways:

i) It will be destroyed automatically whenever its task is completed (run () method execution is completed)

ii) You can destroy the thread before finishing its task by calling the destroy () method

Once it is dead, you cannot get it back to utilise the CPU time.

CPU Scheduling Algorithm

CPU schedulers are responsible for allocating CPU time for various processes or threads running in system. There FCFS or First Come First Served one is the simplest.

List of scheduling algorithms:

- Shortest job first algorithm
- First come first served schedulers
- Time slice or round robin
- Primitive scheduler algorithm
- Non primitive scheduler algorithm

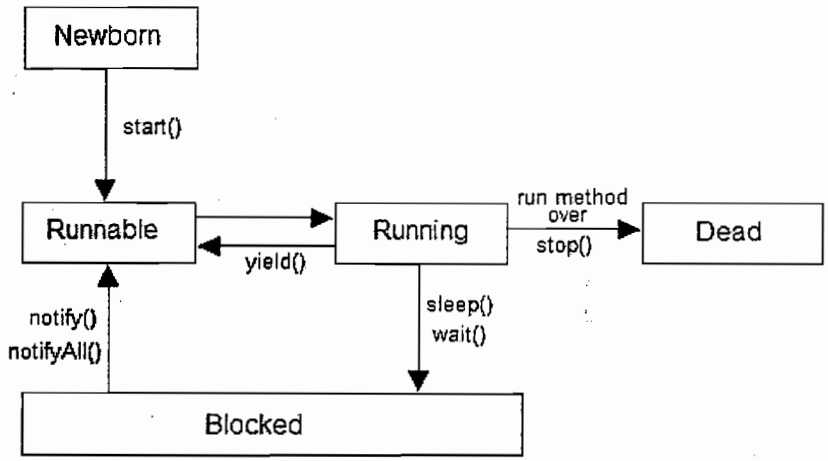
Thread Priority

Priority is a number assigned to the thread which will be used by the CPU scheduler to give more preference to the threads.

Can I overload run() method?

Yes, you can overload. But JVM calls the run() method whose signature is available in runnable interface i.e. public void run() and other overloaded methods should not be called by the JVM.

Thread Life Cycle :



Method	Meaning
getName	Obtain a thread's name
getPriority	Obtain a thread's priority
isAlive	Determine if a thread is still running
join	pause the current thread execution until the specified thread is dead
run	Entry point for the thread
sleep	Suspend a thread for a period of time
start	Start a thread by calling its run method

Questions:

1. What is a Thread?
2. What is the difference between a thread and a process?
3. What are the advantages or usage of threads?
4. What are the two ways of creating threads?
5. What are the different states of a thread's lifecycle?
6. What is use of synchronised keyword?
7. What is the difference between a synchronised keyword being applied to a static method and a non-static method?
8. What is a volatile keyword?
9. What is the difference between yield() and sleep()?
10. What is the difference between wait() and sleep()?
11. What is difference between notify() and notfiyAll()?
12. What happens if a start method is not invoked and the run method is directly invoked?
13. What happens when start() is called?
14. If the code running is a thread which creates a new thread, what will be the initial priority of the newly created thread?
15. When jvm starts up, which thread will be started up first?
16. What are daemon threads?
17. Can the variables or classes be synchronised?
18. How many locks does an object have?
19. Can a class have both synchronised and non-synchronised methods?
20. If a class has a synchronised method and non-synchronised method, can multiple threads execute the non-synchronised method?
21. If a thread goes to sleep, does it hold the lock?
22. Can a thread hold multiple locks at the same time?
23. Can a thread call multiple synchronised methods on the object of which it holds the lock?
24. Can static methods be synchronised?
25. Can two threads call two different static synchronised methods of the same class?
26. Does a static synchronised method block a non-static synchronised method?
27. Once a thread has been started, can it be started again?
28. When does deadlock occur and how can we avoid it?
29. What is a better way of creating a multithreaded application - Extending Thread class or implementing Runnable?
30. Can the start() method of the Thread class be overridden? If yes, should it be overridden?
31. What are the methods of the thread class used to schedule threads?
32. Which thread related methods are available in Object class?
33. Which thread related methods are available in Thread class?

34. List the methods which when called, the thread does not release the locks held?
35. List the methods which when called on the object, the thread releases the locks held on that object?
36. Does each thread have its own thread stack?
37. What is thread starvation?
38. What is threadLocal variable?

◆◆◆

29 CHAPTER

JVM Memory Management

JVM (Java Virtual Machine)

The JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java byte code can be executed. It allows a system to run a program. The three notions of JVM are specification, implementation and instance.

JVMs are available for many hardware and software platforms (i.e.JVM is platform dependent).

We will learn more about JVM in this chapter.

What is JVM?

- A specification where the working of Java Virtual Machine is specified, but the implementation provider is allowed to choose the algorithm independently. Its implementation has been provided by Sun micro system and other companies.
- An implementation is known as JRE (Java Runtime Environment) is used here
- Runtime Instance : Whenever you write a java command on the command prompt to run the java class, an instance of JVM is created

What does JVM do?

The JVM performs following operations:

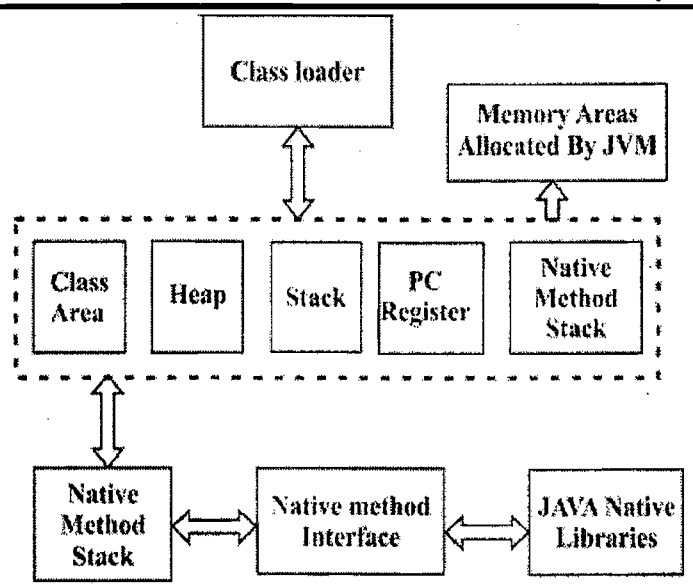
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

Internal Architecture of JVM

Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc. Study the diagram below for more clarity:



- **Class loader :**
  Class loader is a subsystem of JVM that is used to load class files.
- **Class (Method) Area :**
  Class (Method) Area stores per-class structures, such as the runtime constant pool, field and method data, the code for methods.
- **Heap :**
  It is the runtime data area in which objects are allocated.
- **Stack:**
  Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.
  Each thread has a private JVM stack, created at the same time as thread.
  A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.
- **Program Counter Register:**
  It contains the address of the Java virtual machine instruction currently being executed.
- **Native Method Stack:**
  It contains all the native methods used in the application.
- **Execution Engine:**
  It contains :
  - **A virtual processor**
  - **Interpreter :** Which reads byte code stream then executes the instructions

- **Just-In-Time (JIT) compiler:** It is used to improve the performance of the program. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term compiler refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

JVM Memory Management

At JVM start up, JVM occupies the same memory from operating system to run the java programs. Objects in java are stored in what is called as the heap, which when full, activates the garbage collection. During the garbage collection process, only the objects which are no longer in use are cleared. This opens up more memory space.

To understand JVM memory better, you need to know that it is divided into two parts:

i) Stack memory
   Stack memory is used for static memory allocation which stores local variables and function calls. It is a private memory and cannot be viewed by everyone.

ii) Heap memory
   Heap memory stores objects in Java and global.

Whenever you run the program, JVM allocates the memory for variables used in the program and objects created in the program. It is a public memory and can be viewed by anyone.

```
package com.javabykiran.MemMgmt; public
class Hello {
   int a;
   int b;
   static int y = 9;
   {
      System.out.println("Hello-I.B.");
   }
   static {
      System.out.println("Hello-S.B.");
   }
   Hello(int x) {
   System.out.println("Hello-1 arg
   const");
      a = x;
      b = x;
   }
// Hello(int x, int y) {
```

```
// System.out.println("Hello-2 arg const");
// a=x;
// b=y;
// }
Hello(int x, int y) {
   System.out.println("Hello=2 arg
   const"); a = x;
   b = y;
}
void show()
   { int x =
   111; int
   y = 222;
   System.out.println(a);
   System.out.println(b);
   System.out.println(this.y);
   System.out.println(x);
   System.out.println(y);
}
}
```

```
package com.javabykiran.MemMgmt;
public class Lab12 {
  int x = 10;
  static int y = 20;
  {
     System.out.println("Lab12=I.B.");
  }
  static {
     System.out.println("Lab12-S.B.");
  }
 public static void main(String[] args) {
     Hello h1 = new Hello(100);
     h1.show();
     Hello h2 = new Hello(100, 200);
     h2.show();
      } // end main
    } // end class
  {
```



```
   System.out.println("Lab12-I.B.");
  }
  static{
     System.out.println ("Lab12-S.B");
        }
  main(-) {
  Hello h1=new Hello(100) h1.show();
  Hello h2=new Hello(100,200) h2.show();
  }
  Stack of Hello class
  { System.out.println ("Hello-I.B."); }
  static { sop("Hello-S.B."); }
  Hello(int x) {---}
  Hello(int x, int y)
  {----}
  void show() {--}
Stack memory                Heap memory Stack of Lab12 class
```

When JVM is loading class into main memory, the following tasks are performed:

- Default objects will be created, which will be type java.lang.Class
- Stacks will be constructed with all the block members of the class
- Memory will be allocated for static variables of the class
- Static blocks will be executed

When JVM is creating the object, the following tasks will happen:

- Memory will be allocated for the reference variables
- Memory will be allocated for the instance variables
- Instance blocks will be executed
- The appropriate constructor will be called

Newly created object address will be assigned to reference variables.

```
package com.javabykiran.MemMgmt;
class Hi {
   int a = 88;
   static int b = 99;
   Hi() {
         System.out.println(" in const");
   }
   void m1() {
   System.out.println(a);
   System.out.println(b);
   }
```

```
   static void m2() {
           // System.out.println(a); //non static variable
       System.out.println(b);
   }
{
   System.out.println("Hello-I.B.");
  //System.out.println(a); //non-static variable 'a' cannot
                          // be referenced from static context.
   System.out.println(b);
}
static {
   System.out.println("Hello-S.B.");
     //System.out.println(a); //a not allowed as
     static can not acomodate
   // non static System.out.println(b); // b
}
}
package com.javabykiran.MemMgmt;
class Lab13 {
public static void main(String[] args) {
Hi h = new Hi();
h.m1();
h.m2();
    }
 }
```

---Output---

```
A: non-static variable 'a' cannot be referenced from static context
B: Illegal forward reference
*/
```

Call by value and call by reference

```
class Hui {
int x=10;
}
class Hello {
void m1(int p) {
     System.out.println ("m1 Begin");
     System.out.println (p); p=p+10;
     System.out.println (p);
     System.out.println ("m1 end')
}
void m2(Hui hui) {
     System.out.println ("m2 begin");
```



```
     System.out.println (hui.x);
     Hui.x=hui.x+10;
     System.out.println (hui.x);
     System.out.println ("m2 end");
}
}

   class Lab14 {
 public static void main (string as[]) {
 Hello h=new Hello();
     int a=100;
     System.out.println(a);
     h.m1(a);
     System.out.println(a);
     Hui ha=new Hui();
     System.out.println(ha.x);
     h.m2(ha);
     System.out.println(ha.x);
}
}
```

---Output---

```
100
m1 begin 100
110
m1end 100
```

Call by value:

- When you invoke a method by passing primitive as parameter, it is called call by value mechanism
- In case call by value happens, modifications that have happened in the method will not be reflected to the caller of the method

Call by Reference:

- When you invoke a method by passing object as parameters, it is called a call by reflected mechanism
- In case of call by reference, modifications that have happened in the method will be reflected to the caller of the method

Q. Can I overload the main method?

Ans. Yes, you can overloaded it.

Q. Which main method will be called by JVM when a class contains multiple overload main methods?

```
JVM always calls the main method which is
public static void main (String as[])
class Lab15 {
public static void main (string as[]) {
    System.out.println ("main()");
}
public static void main(int x) {
    System.out.println ("main(int x)");
}
public static void main(string as) {
    System.out.println ("main(string as)");
main();
main(10);
main("jbk");
 }
}
```

Q. Can you call the main method inside main()?

Ans. Yes, you can.

You may call not only the main method, but any method inside the same method, which is called exclusive calling. Recursive calling must be terminated at the same point with the same condition otherwise you will get a stack overflow error

```
class Hello
{
int fact(int n)
  { if(n==0) {
  return 1;
  } else
 {
 int x=n*fact(n-1);
   return x; }
  }
}
class Lab16 {
public static void main (string as[]) {
    Hello h=new Hello();
  int a=IntegerparseInt(as[0]);
```



```
   if(a>=0) {
int f=h.fact(a);
}
else {
System.out.println ("plz enter positive number");
}
}
}
```

Q. I have two classes called hello and hui, and both contain main methods. Which main() will be called?

Ans: The main () invocation will depend on the class name which you are passing command line.

For Example: When you write

Java Hello- Main of Hello class will be called Java Hui- Main at Hui class will be called.

When you try to use the class, the JVM loads class into main memory

We can use the class's code in two ways:

- Creating the object for the class
- Accessing static members with the class

When JVM is loading the class into main memory it will perform the following task:

- Memory will be allocated for static variables
- Static block will be executed

When you create an object, the following task will be performed : Hello h=new Hello ();

- Hello h : Memory will be allocated to the reference variables (static ariable)
- New Hello : Memory will be allocated for the instance variables of the class (non-static variables)
- Hello : Instance block will be executed
- () : Appropriate constructor will be called
- = : Newly created object's address will be assigned to the reference variable

Question: Can I declare the constructor as static?

Ans: NO -- > Error Modifier static is not allowed here.

Question: Can I access instance variables with the class name?

Ans: NO. Hello.a will only access static variables.

Error non-static variable cannot be referenced from a static can text.

JVM (Java Virtual Machine) Questions:

1. What does JVM mean? What is JVM?
2. How is your Java program executed inside JVM?
3. What does a JVM consist of?
4. What is Java class file's magic number?
5. What is heap and stack?
6. What is a class loader and what are its responsibilities?
7. How can we profile heap usage?
8. How does JVM perform Garbage Collection?
9. Should one pool objects to help Garbage Collector? Should you call System.gc() periodically?
10. What will you do if JVM exits while printing "OutOfMemoryError" and increasing max heap size doesn't help?
11. If your program is I/O bound or running in native methods, do these activities engage JVM?
12. What is 64 bit Java?
13. What is the difference between JVM and JRE?

JVM Memory management Questions:

1. How many types of memory are used by JVM?
2. Which memory segment loads the java code?
3. When are static variables loaded in memory?
4. Does garbage collection guarantee that a program will not run out of memory?
5. What are the different segments of memory?
6. Describe what happens when an object is created in Java?
7. How many objects will be e created with this code? String s =new String("abc");
8. Describe, in general, how java's garbage collector works.
9. Can I import the same package/class twice? Will the JVM load the package twice at runtime?
10. What is a class loader? What are the different class loaders used by JVM?
11. Is JVM, a compiler or interpreter?
12. Explain java.lang.OutOfMemoryError.
13. What is the difference between loadClass and Class.forName?
14. What kind of memory is used for storing object member variables and function local variables?
15. Why do member variables have default values whereas local variables don't have any default value?
16. What are the various types of Class loaders used by JVM?
17. How are classes loaded by JVM?
18. Name few tools for probing Java Memory Leaks.

19. Which memory areas do instance and static variables use?
20. What is PermGen or Permanent Generation?
21. What is metaspace?
22. Why does Java not use pointers?
23. What are the differences between static and dynamic class loading?
24. Can we call the garbage collector explicitly?
25. How does the substring() method of String class create memory leaks?
26. What are strong, soft, weak and phantom references in Java?
27. How does Java provide high performance?
28. Why is Java considered as a Portable Language?
29. How can you find out whetehr JVM is 32 or 64 bit from Java program?
30. What is the Java (JVM) Memory Model?
31. What is the Young Generation of Java (JVM) Memory Model?
32. What is Old Generation?
33. What are Java heap memory switches?
34. What is Java Stack Memory?
35. What is Method Area?
36. What is Runtime Constant Pool?

◆◆◆

30 CHAPTER

JDBC

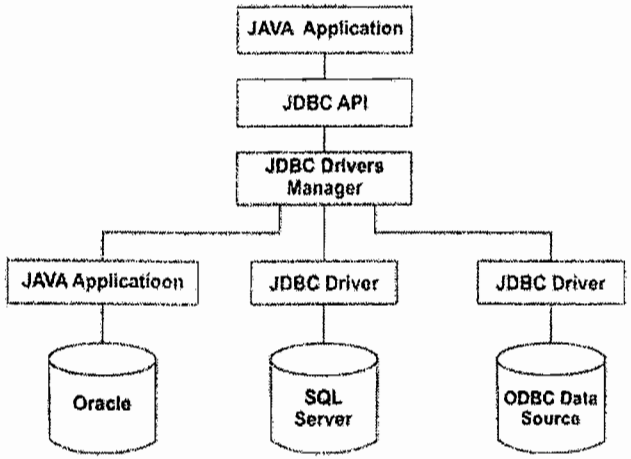
JDBC stands for Java Database Connectivity. It is an API for the Java programming language. It allows the client to access the database and also establishes how the client may do so. It can work on a number of operating systems or platforms, like Windows, Mac, etc. It is basically a connection between the database and the application.

If you want to interact with database using c & c++, you need to use database specific libraries in your application directly. Later, if you want to migrate the database, you need to rewrite the entire application using real database specific libraries. This increases the maintenance of the application.

To avoid this, Microsoft has introduced ODBC Driver ODC (open database connectivity). Given below are some points related to the ODBC Driver.

- With the ODBC Driver, you don't need to use the application directly, because ODBC itself contain various database vendor specific libraries
- Your application now contacts the ODBC Driver instead of using database specific libraries directly. This reduces maintenance issues
- But ODBC has a limitation, i.e., the ODBC setup is available only on the Windows OS and neither has it shown good performance

To avoid these limitations and to provide a uniform method to interact with any database, Sun has provided us with the JDBC API and JDBC Drivers.



Here are the steps to write a JDBC program:

Step 1: Load the Driver class
Step 2: Establish the connection
Step 3: Create the required statement
Step 4: Prepare the Required SQL statement
Step 5: Submit the SQL statement to Database
Step 6: Process the Results
Step 7: Release the Resources

JDBC utility class [we have called this class in all examples of this chapter]

```
package com.javabykiran.JDBC;
import java.sql.*;
public class DBUtil {
   public static Connection getOracleConnection() {
      Connection con = null;
   try {
         // 1.Load the Driver class
        Class.forName
        ("oracle.jdbc.driver.oracle.Driver");
     // 2.Establish the   connection con =
     DriverManager.getConnection(
   "jdbc:oracle:thin:@localhost:1521.XE","system", "jbk");
      } catch (Exception e) {
        e.printStackTrace();
      }
      return con;
   }
public static Connection getMySQLConnection() {
      Connection con = null;
 try {
         // 1. Load the Driver class

Class.forName("com.mysql.jdbc.Driver");
         // 2. Establish the connection
 con =DriverManager.getConnection("jdbc:mySQL://local
host:3306/jbkdb", "root", "root");
      } catch (Exception e) {
         e.printStackTrace();
      }
      return con;
   }
```

```
  public static void  cleanup(Connection con, Statement
     st, ResultSet rs) {
   try {
     // 7.Release the Resources
     if (rs != null)
        rs.close();
     if (st != null)
        st.close();
     if (con != null)
        con.close();
   } catch (Exception e) {
     e.printStackTrace();
   }
}
   public static void cleanup(Connection con, Statement st) {
        try {
           // 7.Release the Resources
           if (st != null)
                st.close();
           if (con != null)
                con.close();
        } catch (Exception e) {
           e.printStackTrace();
        }
     }
   }
```

```
package com.javabykiran.JDBC;
import java.sql.*;
class JDBCLab2 {
public static void main(String as[]) {
   int sid = Integer.parseInt(as[0]);
   String sn = as[1];
   String em = as[2];
   long ph = Long.parseLong(as[3]);
     String ci = as[4];
      double fee = Double.parseDouble(as[5]);
      Connection con = null;
      Statement st = null; ResultSet rs = null;
```



```
     try {
      con= DBUtil.getOracleConnection();
      // 3.create the Required JDBC statement
      st = con.createStatement();
      // 4.prepare the Required SQL statement
      String sql = null;
      System.out.println(sql);
      // 5.submit the sql statement to DB
      int x = st.executeUpdate(sql);
   // 6.process the Results
      if (x == 1) {
           System.out.println("Record is inserted");
      } else {
      System.out.println("sorry,Record is not inserted");
      }

   // 4.prepare the Required SQL statement
      String sql="select  *from b27 students";
  // 5 submit the SQL statement to DB.
      rs = st.executeQuery(sql);
  // 6.process the Results.
     while (rs.next()) {

           sid = rs.getInt(1);
           sn = rs.getString(2);
           em = rs.getString(3);
           ph = rs.getLong(4);
           ci = rs.getString(5);
           fee = rs.getDouble(6);
           System.out.println("");

        }
      } catch (Exception e) {
      e.printStackTrace();
      } finally {
         DBUtil.cleanup(con, st, rs);
      }
   }
}
```

Types of JDBC statement:

There are three types of JDBC statements. They are:

- Statement
- PreparedStatement
- CallableStatement

1) **Statement**

- Statement is an interface available in java.sql package
- The statement object can be created using one of the following methods of connection interface:
  - Statement createStatement();
  - Statement createStatement(int,int);
  - Statement createStatement(int,int,int);
- Once the statement object is created, you can call one of the following methods of statement interface:
  - ResultSet executeQuery(String)
  - int executeUpdate(String)
  - boolean execute(String)

a) The executeQuery()method can be used to submit the selected SQL statement to the SQL Engine.
This method returns the Resultset object which contains the number of records returned by the given selected SQL statement.

b) The executeUpdate() method can be used to submit insert, update, and delete SQL statement to SQL Engine.
This method returns the integer number which represents the number of record affected by the given SQL statement.

c) The execute() method can be used to submit insert, update, delete SQL statement to SQL Engine.
This method returns the Boolean value which represents whether the given operation is insert/update/delete (false) OR Fetch (true).

- Using one statement object, you can submit one or more SQLstatements
- When you submit the SQL statement to SQL Engine using statement object, the SQL statement will be compiled and executed every time

Java ## DB

- 5 seconds to reach to DB
- 5 seconds to compile the sql query
- 5 seconds to execute the sql query
- 5 seconds to load data from DB

Total time taken for executing a single query for all four steps is:
=5ms+5ms+5ms+5ms
=20ms
1000 Queries =1000*20
=20,000

PreparedStatement

- PreparedStatement is an interface available in java.sql package and it extends the Statement interface
- The PreparedStatement object can be created using one of the following methods of connection interface:
  - PreparedStatement (String);
  - PreparedStatement (String, int, int);
  - PreparedStatement(String,int,int,int);
- Once the preparedStatement object is created, you can call one of the following methods of preparedStatement interface:
  - ResultSet executeQuery()
  - int executeUpdate()
  - boolean execute()
- Using one preparedStatement object, you can submit only one type of SQL statement
- When you submit the SQL statement to SQL Engine using preparedStatement object, the SQL statement will be compiled only once the first time, and will be executed every time without compilation

For first query, time taken for executing prepared statement query will be:
1Query = 1+2+3+4
=5ms+5ms+5ms+5ms
=20ms

For the second query onwards, time taken for executing prepared statement query will be:
1Query = 1+3+4
5+5+5=15ms
1000-15000
=5000ms (saving time)

```
package com.javabykiran.JDBC;
import java.sql.*;
class JDBCLab {
 public static void main(String[] as) {
  int sid = Integer.parseInt(as[0]);
   String sn = as[1];
    String em = as[2];
    long ph = Long.parseLong(as[3]);
```

```
    String ci = as[4];
    double fee = Double.parseDouble(as[5]);
    Connection con = null;
    PreparedStatement ps1 = null;
    PreparedStatement ps2 =  null;
    ResultSet rs = null;
    try {
    con = DBUtil.getMySQLConnection();
    ps1   =   con.prepareStatement("insert   into   b27students
  values(?,?,?,?,?,?)");
    ps1.setInt(1, sid);
    ps1.setString(2, sn);
    ps1.setString(3, em);
    ps1.setLong(4, ph);
    ps1.setString(5, ci);
    ps1.setDouble(6, fee);
    int x = ps1.executeUpdate();
    if (x == 1) {
       System.out.println("Record is inserted");
    } else {
       System.out.println("sorry, Record is not inserted");
    }
    ps2 =con.prepareStatement("select* from jbkStudentns");
    rs = ps2.executeQuery();
     while (rs.next()) {
       sid = rs.getInt(1);
       sn = rs.getString(2);
       em = rs.getString(3);
       ph = rs.getLong(4);
       ci = rs.getString(5);
       fee = rs.getDouble(6);
   System.out.println(" " + sid + "!t" + sn + "!t" + em +
   "!t" + ph + "!t" + ci + "!t" +  fee);
   }
   } catch (Exception e) {
     e.printStackTrace();
   } finally {
       DBUtil.cleanup(con, ps1, rs);
       DBUtil.cleanup(con, ps2,  rs);
  }
}
```


- PreparedStatement is also a sub interface of Statement. So, we pass (con,ps1,rs) in the cleanup method of DBUtil class.

  Where
  - con is Connection object
  - rs is a ResultSet
  - ps1 is a Statement

CallableStatement

- The CallableStatement is an interface available in java.sql package and is an extension of the preparedStatement interface
- The CallableStatement object can be created using one of the following methods of connection interface:
  - CallableStatement preparecall(String);
  - CallableStatement preparecall(String,int,int,);
  - CallableStatement preparecall(String,int,int,int,);
- Once callableStatement object is created, you can call one of the following methods of callableStatement interface:
  - ResultSet executeQuery()
  - int executeUpdate()
  - boolean execute()
- CallableStatement is mainly used to execute stored procedures running in the database.

  Using one CallableStatement object. You can submit only one call one stored procedure
- Stored procedure is a set of pre-compiled procedures i.e, when you create the procedure, it will be compiled and stored in the database memory. When you call the procedure it will be executed directly

Example that invokes stored procedure with IN parameters:

```
/*
 Create table hello (
   a number(4),
   b number(4),
   tot number(8)
 );
 Create or replace procedure p1 (a in number, b in number) as
 tot number(8);
 begin
 tot = a+b:
```

```
insert into hello values (a,b,tot);
end;
*/
```

```
package com.javabykiran.JDBC;
import java.sql.CallableStatement;
import java.sql.Connection;
public class JDBCLab5 {
     public static void main(String as[])
     { int a =  Integer.parseInt(as[0]);
     int b = Integer.parseInt(as[1]);
     Connection con = null;
     CallableStatement cs = null;
        try {
           con = DBUtil.getOracleConnection();
           cs =con.prepareCall("{callp1(?,?)}");
           cs.setInt(1, a);
           cs.setInt(2, b);
           cs.executeUpdate();
        } catch (Exception e) {
           e.printStackTrace();
        } finally { DBUtil.cleanup(con, cs);
        }
     }
   }
   /*
```

Example that invokes stored procedure with IN and OUT parameters:

```
*/
create or replace procedure p2(a IN number, b IN number, tot
OUT number, dif OUT number, mul OUT number, div OUT number) as
begin
tot:=a=b;
dif:=a-b;
mul:=a*b;
div:=a/b;
end;
*/
```



```
package com.javabykiran.JDBC;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.Types;

public class JDBCLab6 {
  public static void main(String[] args) {
  int a = Integer.parseInt(args[0]);
  int b = Integer.parseInt(args[1]);
  Connection con = null;
  CallableStatement cs = null;

     try {
        con = DBUtil.getOracleConnection();
        cs = con.prepareCall("{call p2(?,?,?,?,?,?(}");
        cs.setInt(1, a);
        cs.setInt(2, b);
        cs.registerOutParameter(3, Types.INTEGER);
        cs.registerOutParameter(4, 4);
        cs.registerOutParameter(5, Types.INTEGER);
        cs.registerOutParameter(6, 4);
        cs.executeUpdate();

        int c = cs.getInt(3);
        int d = cs.getInt(4);
       int e = cs.getInt(5);
       int f = cs.getInt(6);

     System.out.println(c);
     System.out.println(d);
     System.out.println(e);
     System.out.println(f);
  } catch (Exception e) {
     e.printStackTrace();
  } finally {
     DBUtil.cleanup(con, cs);
  }
 }
 }
```

Operation	Statement	Prepared Statement	Callable Statement
Executing different types of SQL stints	✓	x	x
Executing only one type of SQL stints	✓	✓	x
Executing stored procedure	x	x	✓
Executing pre-compiled Queries	x	✓	✓ (Procedures are pre-compiled)
Faster execution	x	✓	✓

Database Metadata

- It is an interface available in java.sql package, which provides various useful methods to get information about your database, which you are linked to
- You can create database metadata object as follows:
  - DatabaseMetaData dbmd=con.getMetaData();
  - By using MetaData we can get any information about Database

Resultset Metadata

- ResultsetMetaData is an interface available in java.sql package, which gives the information about Resultset object, like number of columns available in Resultset, names of the columns name of the table from where column is fetched etc.
- You can create the ResultsetMetaData object as follows:-
  - ResultsetMetaData rsmd=rs.getMetaData();
  - DatabaseMetaData and ResultsetMetaData are interfaces in java.sql.package
- Oracle has provided subclasses for these interface in oracle.jdbc.driver package called oracleDatabaseMetaData and oracleResultsetMetaData
- MySQL has provided sub classes for these interfaces in com.mysql.jdbc package called DatabaseMetaData and ResultsetMetaData

Question: In java.sql package, there is a majority of interface only. How will instances be created in these interfaces?

```
class DriverManager {
  static Connection getconnection(url,un,pw)
     {
     }
  }
```


- It takes the url and checks whether driver class is loaded for this url
- If not loaded then it gives this error:NosuitableDriver error
- If loaded then
- It creates the object of subclass of connection interfaces related vendor oracleConnection/MySQLconnection class.

Given below are vendor implementations:

```
Class OracleConnection implements java.sql.Connection {
   Statement createStatement ()
   {
      return new oracleStatement ();
   }
}
```

```
Class MySQLConnection implements java.sql.connection
{
      Statement createStatement ()
   {
      return new MySQLstatement();
   }
}
```

Question: I have loaded oracle Driver and MySQL, which Database connection will be established when trying to get the connection?

Answer: Depending on the url you are passing as the parameters to getConnection() method, the corresponding database connection will be established.

Sample code:

```
class.forName("oracle.jdbc.driver.OracleDriver");
class.forName("com.MySQL.jdbc.Driver");
connection con=DM.getConnection(URL);
```

ResultSet

- Resultset is a package which is in package java.sql package
- The Resultset object can be used to store multiple records returned by select statement
- When Resultset record is created initially result set cursor points to before to the first record.

Types of ResultSet

Depending on the ResultSet cursor movement, you can divide the ResultSet into 2 types:

- Forward only Resultsets
- Scrollable Resultsets

Forward only Resultsets

- When Resultset is forward, only then you can move the Resultset cursor, that too only in the forward direction
- You can invoke the following methods on forward only Resultsets:
  - next();
  - getXX(); [XX can be String or Int or according to data type]
  - close();

next() :

Checks whether next Record is available or not.

If it is available then it :

a. moves the pointer to next Record
b. returns true

If it is not available then :

a. moves the pointer to next Record
b. returns false

By default, ResultSets are forward only, you can specify the Resultsets as forward only explicitly as follows:

```
Statement stmt=con.createStatement();
Statement stmt = con.createStatement(ResultSet.TYPE_FORWARD_ONLY,
ResultSet.CONCUR_READ_ONLY);
ResultSet rs=stmt.executeQuery(sql);
```

Now, ResultSet is forward only and read only

1) Scrollable Resulsets

- When Resultset is scrollable, you can move the Resultset cursor both in the forward direction and in the reverse direction a number of many times
- You can invoke the following methods on scrollable ResultSets

next()	isAfterLast()	isBeforeFirst()	isLast()	isFirst()
previous()	absolute()	relative()	afterLast()	beforeFirst()
getxx()	close()	first()	last()	

You can specify the ResultSets as scrollable explicitly as follows:

```
Statement stmt=con.createStatement();
Statement stmt = con.createStatement
  (ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_READ_ONLY);
     ResultSet rs=stmt.executeQuery(sql);
```

Now ResultSet is scrollable and read only.

absolute() :

- It will directly it will goto the that record rs.absolute(4), (which is specified)

relative() :

- It means it will go to that record which is specified depending on the absolute value

Example:

rs.absolute(4); rs.relative(-3) rs.relative(?)

getxx() :-To get me column value

```
Statement stmt = con.createStatement();
   Statement stmt =
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
```

```
ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery(sql);
```

Now, ResultSet is scrollable and updatable.

movetoInsertRow() :

- First set will contain empty row in the Resultset object

Update() :

- The only corresponding row of the ResultSet will be updated
- Make sure that the table contains a primary key, otherwise the updatable record will not be applicable

Batch update set

Below given is a process to execute statements in JDBC:

- load SQL
- compile SQL
- process SQL
- load Data

With Statement

- 1SQL= 5+5+5+5=20ms

With Prepared Statements

- 1SQL=5+0+5+5=15ms

With Batch update

- 1SQL=0+0+5+5=15ms

- If you want to submit multiple queries to the database one by one, a lot of time will get wasted on the requesting response
- Instead of submitting SQL statements one by one, we can submit multiple statements all at once to the database as a batch using Batch update concepts

- To implement the batch update, you can use the following methods of statement interface:
  - void addbatch(sql)
  - int[] executebatch()
- Using Batch updates, we can submit multiple insert, update and delete statements. This means that a select statement will not do

Questions:

1. What is Java JDBC?
2. What are the types of statements in JDBC?
3. Why should we use JDBC?
4. What restrictions are placed on method overriding?
5. What does setAutoCommit do?
6. What are the types of JDBC drivers?
7. Why are Prepared Statements faster?
8. What are the differences between setMaxRows(int) and SetFetchSize(int)?
9. How can I manage special characters when I execute an INSERT query?
10. How can you load the drivers?
11. How do I insert an image file (or other raw data) into a database?
12. Discuss the significance of JDBC.
13. Name the new features added in JDBC 4.0.
14. Is it possible to connect to multiple databases simultaneously? Can one update or extract data from multiple databases using a single statement?
15. What is the benefit of having a JdbcRowSet implementation? Why do we need a JdbcRowSet like wrapper around ResultSet?
16. I have the choice of manipulating database data using a byte[] or a java.sql.Blob. Which gives the best performance?
17. How do Java applications access the database using JDBC?
18. Explain the life cycle of JDBC.
19. Describe how the JDBC application works.
20. Explain in brief about the JDBC Architecture.
21. What are DML and DDL?
22. Explain the basic Steps involved in writing a Java program using JDBC.
23. What does the JDBC Driver interface do?
24. How can a database driver be loaded with JDBC 4.0 / Java 6?
25. What is a statement?
26. What is represented by the connection object?

27. Define PreparedStatement.
28. Differentiate between a Statement and a PreparedStatement.
29. What is the function of setAutoCommit?
30. How do we call a stored procedure from JDBC?
31. Name the types of JDBC Drivers and write a brief explanation of each.
32. How can we move the cursor in a scrollable ResultSet?
33. What is SQL Warning and discuss the procedure of retrieving warnings?
34. How do we load the drivers?
35. What does Class.forName d while loading the drivers?
36. What are the factors that the JDBC driver performance depends on?
37. How can you make a connection?
38. Explain the method of calling a stored procedure from JDBC.
39. How do determine whether a parameter exists in the request object?
40. Exlain the method of calling a stored procedure from JDBC.
41. Explain how to make updates to the Updatable ResultSets.
42. Name the types of JDBC drivers.
43. What are the utilities of the callable statements?
44. Differentiate between TYPE_SCROLL_INSENSITIVE and TYPE_SCROLL_SENSITIVE.
45. How are JDBC Statements used?
46. Name the different types of Statements? How will you use PreparedStatement?
47. How can we retrieve data from the ResultSet?

◆◆◆

Database

Database Concepts

Now that we have learned about Java Database Connectivity (JDBC), it is time to learn about some database concepts which you find useful during interviews or while programming.

- The DBMS is the software which is situated between a human being and the data storage. If you want to manipulate the data available in the storage, then you will have to interact the DBMS with SQL commands
- The SQL (Structure Query Language) is a standard which is followed by all the DB vendors to interact with the data and is provided by ANSI (American National Standard Institute)
- There is some validation by the DB task to implement their database
- Some ANSI style syntaxes are more complicated and make it tricky to code sometimes. To avoid these complications DB vendors provide their own syntax called the Theta style syntax
- All the DBMS store their data in a tabular format with rows and columns
- Row contains all the values related to one entity.
- The column contains similar types of values related to all the entities.
- A Database contains tables, and tables contain rows and columns. One row denotes one record. One column represents the common entity of all the rows

Data Types

This represents which type of information can be stored in the database. There are eight primitive data types in java. They are divided into numeric, textual and Boolean and null primitive data types.

Oracle Database Data Types

There are seven Oracle Database data types. They are as follows:

CHAR

- The CHAR data type stores fixed-length character strings. When you create a table with a CHAR column, you must specify a string length (in bytes or characters) between 1 and 2000 bytes, for the CHAR column width. The default is 1 byte. Oracle then guarantees that:
- When you insert or update a row in the table, the value for the CHAR column has a fixed length
- If you give a shorter value, then the value is blank-padded to the fixed length

- If a value is too large, the Oracle Database returns an error.

Oracle Database compares CHAR values using blank-padded comparison semantics

VARCHAR2

(reserved keyword for future reference, as varchar will be replaced eventually)

- The Varchar2 data type stores variable-length character strings. When you create a table with a VARCHAR2 column, you specify a maximum string length (in bytes or characters) between 1 and 4000 bytes for the VARCHAR2 column. For each row, the Oracle Database stores each value in the column as a variable-length field unless a particular value exceeds the column's maximum length, in which case the Oracle Database returns an error. Using VARCHAR2 and VARCHAR saves a lot of space used by the table

For example, assume that you have declared a column VARCHAR2 with a maximum size of 50 characters.

In a single-byte character set, if only 10 characters are given for the VARCHAR2 column value in a particular row, the column in the row's row piece stores only 10 characters (10 bytes), not 50

- Oracle Database compares VARCHAR2 values using non-padded comparison semantics

Varchar Datatype

The VARCHAR datatype is synonymous with the VARCHAR2 datatype.

To avoid possible changes in behavior, always use the VARCHAR2 datatype to store variable-length character strings.

Date

- The Date datatype stores point-in-time values (dates and times) in a table. The Date datatype stores the year (including the century), the month, the day, the hours, the minutes, and the seconds (after midnight)
- For input and output of dates, the standard Oracle date format is DD-MON-YY, as follows:
- '13-NOV-92'

Blob

- The Blob data type stores unstructured binary data in the database. Blobs can store up to 128 terabytes of binary data
- Blobs participate fully in transactions. Changes made to a Blob value by the DBMS_LOB package, PL/SQL, or the OCI can be committed or rolled back. However, Blob locators cannot span transactions or sessions

Clob

- The Clob and NClob datatypes store up to 128 terabytes of character data in the database. CLOBs store database character set data, and NCLOBs store Unicode national

character set data. Storing varying-width LOB data in a fixed-width Unicode character set internally enables the Oracle Database to provide efficient character-based random access on CLOBs and NCLOBs

- CLOBs and NCLOBs participate fully in transactions. Changes made to a CLOB or NCLOB value by the DBMS_LOB package, PL/SQL, or the OCI can be committed or rolled back. However, CLOB and NCLOB locators cannot span transactions or sessions. You cannot create an object type with NCLOB attributes, but you can specify NCLOB parameters in a method for an object type

- **File**
  - **MysQL Database Data Types**
    - Char,Varchar
    - text, (Unlimited Char use)
    - int
    - long
    - float
    - double
    - date
    - blob
    - clob
    - file

Create Command

In MysQL

Syntax to create a table in MySql is as follow

CREATE TABLE tablename (column name datatype (size)… .....);

Example:

CREATE TABLE authors (aid INT, aname CHAR(20), dod DATE, quali CHAR(20), email CHAR(20), phonenumber VARCHAR(12));

Insert Command

Syntax to insert value:

Insert Into tablename(Col1, Col2……) values (Val1,Val2……);

Example:

- **In Mysql:** Insert into authors values (101, 'Kiran', '2001-10-10 ', 'M.Sc', ' kiran@fle.com',

123456);

- **In Oracle:** Insert into authors values (102, 'vas', '10-oct-2001', 'M.Sc', 'vas@fle.com', 123456);

Authors Table With Data

Aid	Aname	DOB	Quail	Email	Phone
101	Kiran	2001-10-10	M.Sc	kiran@fle.com	123456

- **Rectify It :** Insert into authors values (103, 'sd', 'M.Sc' 12345);
- **Rectified:** Insert Into authors (aid, aname, quali, phonenumber) VALUES (103, 'sd', 'M.Sc', 12345);

Notes:

1. Char & date type values must be enclosed in single quotation marks
2. Whenever you are inserting date type values, you should make sure that the correct date format is used
3. Whenever you are not providing values for all the columns, you must specify the first column name and add that table name
4. Whenever you are not supplying values for a column, a null will be placed in that column

Question: What will happen when you give the following command:

INSERT INTO authors (phonenumber, aname, aid) VALUES (1234, 'sd',104);

Answer : It will execute successfully.

Notes : Using one insert statement inserts only one record.

Update Command

Syntax:

Update table-name set col1 = Val1, Col2 = val2 Where condition;

Example:

Single Update:

UPDATE authors SET email = 'kiran@fle.com', phonenumber = 999 WHERE aid = 101;

Multiple Update:

UPDATE authors SET phonenumber = 999;

Note:

1. Using one update state, you can update zero or more records
2. ROLLBACK - insert, delete and update not for create and drop

Delete Command

Syntax :

DELETE FROM tablename WHERE CONDITIONS;

Example:

1. **Single Deletion:** DELETE FROM authors WHERE aid = 103;
2. **Multiple Deletion:** DELETE FROM authors;

(DELETES ALL DATA IN AUTHOR'S TABLE)

Note:
Using one delete statement, you can delete zero or more records.

Drop Command
Syntax : DROP TABLE tablename;
Example : DROP TABLE authors;
Truncate The Table Syntax : TRUNCATE TABLE tablename;
Example : TRUNCATE TABLE authors; DELETE FROM authors;

1. Both the statements will give you an empty tab at the end
2. The Delete statement deletes all records one by one, whereas the Truncate state does the table simply and creates a new table
3. Delete takes more time than the Truncate state
4. Delete operation can be cancelled i.e. it can be called back, whereas Truncate operations cannot be callback.
5. DELETE deletes the entire table with its structure whereas TRUNCATE deletes only the data that is inside that particular table

Syntax :
Select (or) cal1, col2 ..... tab1, tab2
Where condition Order by col-name Group by col-name Having condition;

Books Table

BookID			Cost	Isbn	Pub	Qty	Edition	Tob

Question 1: Display complete information of all the books.
Question 2: Display the bookID, bookname, cost and the edition of all the books.
Question 3: Display the books published by DLC.
Question 4: Display the books written by Kiran.
Question 5: Display the books written by Kiran in 2010.
Question 6: Display the book whose name you learnt in the beginning.
Question 7: Display the books which cost less than INR 200 and are written by an author whose name ends with Vas.
Question 8: Display the books published between 2005 and 2010
Question 9: Display the books published by "TATA", "ONLYFORJAVA" and "P5".
Question 10: Display the books which cost from INR50 to 300 in ascending order of cost.
Question 11: Display the books published in 2009 in descending order of their costs.

SQL Operations:
The SQL operations are as follows:

- Arithmetic operations (+, -, *, / , mod)
- Relation operations (>, >=, <=, <, <>,=)

- Logical operations (and, or, not)
- In operators
- Between and operator
- Like operator

Answer list for the question given on page 498:

1. Select * frombooks;
2. Select bookID, bookname, cost, edition, from books;
3. Select * from books where pub= 'ONLYFORJAVA';
4. Select * from books where author ='kiran';
5. Select * from books where author = 'kiran' and YOP =2010;
6. Select * from books where book name like 'learn%';
7. Select cost, author from book where cost <200 and author like '%Vas';
8. Select * from books where YOP>=2005 and YOP<=2010; **or**
   Select * from books where YOP between 2005 and 2010;
9. Select * from books where pub='TATA' or pub='ONLYFORJAVA' or pube='P5'; **or**
   Select * from books where pub in ('TATA', 'ONLYFORJAVA', 'P5');
10. Select * from books where cost between 50 and 300 order by cost asc;
11. Select * from books where YOP=200g order by cost describtion;

- Display the books which are not published by ONLYFORJAVA Select *from books where PUB<>'ONLYFORJAVA'

  or

  Where PUB not in 'ONLYFORJAVA';
- Display the books written by the author whose name second character is S. Select *from books where author like '-s%';

SQL Function:

- Math functions
- String functions
- Date functions
- Conversion functions
- Aggregate functions

Conversation Function:

- to – number → Character to number
- to – character→ Number to character or date to character
- to - date → Character to date or date of one format to date or another format

- **Aggregate Functions:**
  - Count · Sum · Average · Max · Min

Question:

1) **Display the number of books available in the book store.**
   Select count (*)
2) **Display different types of total books available.**
   Select count (*) from books;
3) **Display total books available.**
   Select sum (qly) from books;
4) **Display the maximum cost of books.**
   Select max (cost) from books; Select min (cost) from books;

Group By and Having Clause

- **Group By**
  The Group by clause is used to divide the table into multiple groups based on the specified columns.
- **Having**
  The having clause is used to specify the condition placed on the groups.

Students

```
mysql> select * from student;
```

sid	sname	totalfees	feepaid	balance	city
1	Deepak	5000	2000	3000	pune
2	Kumar	6000	1000	5000	mumbai
3	Singh	3000	1000	2000	nagpur

Q. 1) Display the total fee collected.
Select sum (fee paid) from students;

```
mysql> select sum<feepaid> from student
```

Sum<feepaid>
4000

Q. 2) Display the total fee collected from the Madiwala branch.
Select sum (fee paid) from students where branch = 'madiwala'


```
mysql> select * from student ;
```

sid	Sname	totalfees	feepaid	balance	bob	city
1	Deepak	5000	2000	3000	mathikere	pune
2	kumar	6000	1000	5000	madiwala	Mumbai
3	singh	3000	1000	2000	macerthali	Nagpur

4 rows in set <0.00 sec>

```
mysql> select sum<bal> from student where bob= 'madiwala' ;
```

Sum<bal>
6000

1 rows in set <0.04 sec>

Q. 3) Display the branch wise collection in the following formats:

KarveNagar	20000
Madiwala	28000
Macerthali	12000

Select sum (fee paid) from student group by branch;
Gives only o/p sum (fee paid)

```
mysql> select * from student ;
```

sid	Sname	totalfees	feepaid	balance	bob	city
1	Deepak	5000	2000	3000	mathikere	pune
2	kumar	6000	1000	5000	madiwala	Mumbai
3	singh	3000	1000	2000	macerthali	Nagpur
4	java	2000	1000	1000	madiwala	Mumbai
5	By	4000	3000	1000	mathikere	Pune
6	Kiran	5000	3000	2000	macerthali	Nagpur

6 rows in set <0.00 sec>

```
mysql> select sum<feepaid> from student group by bob;
```

Sum<feepaid>
4000
2000
5000

Q. 4) Select branch, sum (fee paid) from student group by branch.

→ Will give values with branch

```
mysql> select * from students;
+-----+--------+----------+---------+------+-----------+--------+
| sid | sname  | totalfee | feepaid | bal  | bob       | city   |
+-----+--------+----------+---------+------+-----------+--------+
|   1 | deepak |     5000 |    2000 | 3000 | mathikere | pune   |
|   2 | kumar  |     6000 |    1000 | 5000 | madiwala  | mumbai |
|   3 | singh  |     3000 |    1000 | 2000 | macerthali| nagpur |
|   4 | java   |     2000 |    1000 | 1000 | madiwala  | mumbai |
|   5 | by     |     4000 |    3000 | 1000 | mathikere | pune   |
|   6 | kiran  |     5000 |    3000 | 2000 | macerthali| nagpur |
+-----+--------+----------+---------+------+-----------+--------+
6 rows in set (0.00 sec)

mysql> select bob, sum(feepaid) from students group by bob;
+-----------+---------------+
| bob       | sum(feepaid)  |
+-----------+---------------+
| macerthali|          4000 |
| madiwala  |          2000 |
| mathikere |          5000 |
+-----------+---------------+
3 rows in set (0.00 sec)
```

Q. 5) Select branch as "Branch Name", sum (fee paid) "Total collection" from student group by branch

```
mysql> select * from students;
+-----+--------+----------+---------+------+-----------+--------+
| sid | sname  | totalfee | feepaid | bal  | bob       | city   |
+-----+--------+----------+---------+------+-----------+--------+
|   1 | deepak |     5000 |    2000 | 3000 | mathikere | pune   |
|   2 | kumar  |     6000 |    1000 | 5000 | madiwala  | mumbai |
|   3 | singh  |     3000 |    1000 | 2000 | macerthali| nagpur |
|   4 | java   |     2000 |    1000 | 1000 | madiwala  | mumbai |
|   5 | by     |     4000 |    3000 | 1000 | mathikere | pune   |
|   6 | kiran  |     5000 |    3000 | 2000 | macerthali| nagpur |
+-----+--------+----------+---------+------+-----------+--------+
6 rows in set (0.00 sec)

mysql> select bob as "branch name", sum(feepaid) from students group by bob;
+-------------+---------------+
| branch name | sum(feepaid)  |
+-------------+---------------+
| macerthali  |          4000 |
| madiwala    |          2000 |
| mathikere   |          5000 |
+-------------+---------------+
3 rows in set (0.00 sec)
```

Q. 6) Display the fee collected from KarveNagar and Madiwala branc

Select sum (Fee paid) from students where branch in ('madiwala', 'KarveNagar');(or)

Select branch as "Branch Name", sum (fee paid) "Total collection" from student group by branch having branch in ('madiwala', 'KarveNagar'); KarveNagar 20000 Madiwala 28000

Note: Where condition will be applied to individual rows of a table whereas the having condition will be applied to the individual groups of a table.


```
mysql> select * from students;
+-----+--------+----------+---------+------+-----------+--------+
| sid | sname  | totalfee | feepaid | bal  | bob       | city   |
+-----+--------+----------+---------+------+-----------+--------+
|   1 | deepak |     5000 |    2000 | 3000 | mathikere | pune   |
|   2 | kumar  |     6000 |    1000 | 5000 | madiwala  | mumbai |
|   3 | singh  |     3000 |    1000 | 2000 | macerthali| nagpur |
|   4 | java   |     2000 |    1000 | 1000 | madiwala  | mumbai |
|   5 | by     |     4000 |    3000 | 1000 | mathikere | pune   |
|   6 | kiran  |     5000 |    3000 | 2000 | macerthali| nagpur |
+-----+--------+----------+---------+------+-----------+--------+
6 rows in set (0.00 sec)

mysql> select sum(feepaid), bob from students group by bob having bob in ('madiw
ala','mathikere');
+---------------+-----------+
| sum(feepaid)  | bob       |
+---------------+-----------+
|          2000 | madiwala  |
|          5000 | mathikere |
+---------------+-----------+
2 rows in set (0.00 sec)
```

Q. 7) Display the city wise fee balances in the following format: City Name Total Balance

Pune	25000
Mysore	22000
Delhi	18000
Pune	15000

Select city as "City Name ", Sum (fee paid) as "Total Balance" from student group by city;

mysql> select * from student ;

sid	Sname	totalfees	feepaid	balance	bob	city
1	Deepak	5000	2000	3000	mathikere	pune
2	kumar	6000	1000	5000	madiwala	Mumbai
3	singh	3000	1000	2000	macerthali	Nagpur
4	java	2000	1000	1000	madiwala	Mumbai
5	By	4000	3000	1000	mathikere	Pune
6	Kiran	5000	3000	2000	macerthali	Nagpur

6 rows in set <0.00 sec>

mysql> select sum<bal>,city from student group by city;

Sum<bal>	city
6000	Mumbai
4000	Nagpur
4000	pune

Sum<feepaid>
4000
2000
5000

3 rows in set <0.00 sec>

Q. 8) Display city as "City Name" sum (fee balance) as "Total Balance " from student group by city.

Having city in ('Pune','Pune');

mysql> select * from student ;

sid	Sname	totalfees	feepaid	balance	bob	city
1	Deepak	5000	2000	3000	mathikere	pune
2	kumar	6000	1000	5000	madiwala	Mumbai
3	singh	3000	1000	2000	macerthali	Nagpur
4	java	2000	1000	1000	madiwala	Mumbai
5	By	4000	3000	1000	mathikere	Pune
6	Kiran	5000	3000	2000	macerthali	Nagpur

6 rows in set <0.00 sec>

mysql> select sum<bal> as 'balance fees' ,city as 'city name' from student group by city having city in<'pune','mumbai'>;

Balance fees	City name
6000	Mumbai
4000	pune

2 rows in set <0.00 sec>

Constraints

Constraints are the rules which can be applied to the data getting inserted in the table or while updating the table. The types of constraints are given below:

- Not null constraint
- Unique constraint
- Primary key constraint
- Foreign key constraint
- Check constraint

Not Null Constraints:

The not null constraint makes sure that a field is never empty or null, but is always assigned a value. It also ensures that null values are not accepted by a column.

- Whenever the programmer has not provided a value for any columns, then null will be automatically inserted by the DBMS
- Null means value is not available
- Null is not equivalent to zero or space or any other thing
- If you do not want another value for any given column and force the user to supply the value then you can apply not null constraints
- **Usage of the Not Null constraint:**
  CREATE TABLE students (sid INT, sname, CHAR (10) NOT NULL , totalfee DOUBLE NOT NULL.....);

Question: What will happen when we try to insert a record into students table without total fee?

Answer: An error will occur as it is not null specified and you have to provide a value.

Unique Constraints:

If you want to provide only unique values for any given column then you can apply the unique constraint.

- **Usage of Unique constraints:**
  Create table students S. ID int not null unique,
  S. Name Char (10) not null, Email char (10) not null unique, Phone long unique,);

Q. 1) What will happen when two students provide the same number? Answer: The second student's record insertion will fail.

Q. 2) Can I have multiple null values for the phone number columns? Answer: Yes

Primary Key Constraints:

Primary Key = Not Null + Unique.

The Primary key constraints checks both not null and unique constraints. It should have a unique value and cannot be null. It provides a unique identification for every row in the table. Therefore, it has properties of both not null and unique constraints.

- **Usages:**
  Create table students (sid INT PRIMARY KEY, sname CHAR(10) NOT NULL, email CHAR (10) NOT NULL UNIQUE, phone LONG NOT NULL UNIQUE,.....);

The table should contain only one primary key.

There are two types of primary key

- Simple primary key
- Composite primary key

1) **Simple Primary Key :**
   When you specify the key for a single column then it is called simple primary key.
   Example: Create table students (sid INT PRIMARY KEY, sname CHAR (10) NOT NULL UNIQUE.....);
2) **Composite Primary Key:**
   When you specify the primary key for a combination of two or more columns, then it is called composite primary key.
   Example: Create table, accounts (Bcode int, acc no int, atype int, bal double not null, primary key (bcode, atype, accno));
   Only for balance
   Select balance of accounts where balance code = 0047 and account types = 01 and account no = 530255
   Full Accounts No: 004701530255

Foreign Key Constraints:

The Foreign key constraint is used to the establish the relationship among the tables.

Customers

Customer ID	Customer Name	Email	Phone

Accounts

Customer ID	Account Type	Account type	Amount	Balance

Transactions

Branch Code	Account Type	Account No	Trans. ID	Trans. Date	Trans. Type	Amount	Total

Usages:

Create table customers (Customer ID, Interest Primary Key, Customer name, character (10) not null .....);

```
mysql> create table customers(customerid int , interest int primary key, custome
rname char(10) not null);
Query OK, 0 rows affected (0.39 sec)
```

Create table accounts (customer int references customers, branch code int, account type int, balance double not null, primary key (branch code, account type, account no));

mysql> create table accounts<custmoridint references customers, branchcodeint, accounttypeint, balance double not null, accountnumberint not null, primary

Query OK, 0 rows affected <0.30>sec

mysql> describe accounts ;

field	Type	Null	Key	defult	extra
Customerid	Int<11<	YES		NULL	
Branchcode	Int<11>	NO	PRI	0	
Accounttype	Int<11>	NO	PRI	0	
Balance	Double	NO		NULL	
Accountnumber	Int<11>	NO	PRI	0	

5 rows in set<0.05 sec>

Create table transactions (branchcode int, accounttype int, account no int, TxID int, Txdate date, Tx type char (2), amount double, foreign key accounts (branch code, account type, account no));

Notes: A Table can contain many foreign keys but only one primary key.

Check Constraints:

The Check constraint checks the specified condition on the specified columns. These conditions have to be met by the column. However, it can be placed only on one particular table, and not all the tables.

- **Usages** :Create table students (student ID int primary key, student name char (10) not null, total fee double check total fee >=1500; fee paid double check fee paid double check fee paid >= 1000, city character (20) check city in ('Banglalore', 'mysore'));
  mysql> create table students <id int primary key, name char<10> not null, total if fee double check<totalfee>=1500>, feepaid double check<feepaid>=1000>, city char<Query OK, 0 rows affected <0.31 sec>
- It will make sure that total fee inserted should be greater than or equal to 1500
- It will make sure that the fee paid should be greater than or equal to 1000.
- The city name can only be Pune or mysore.
- 31 January 2011

ZIPCODE : Substr (Zipcode, 1, 3) 11 '-' substr (zipcode,4)

To- date ('20-Jul-11' 'DD-MONTH-YYYY')

Ltrim - side space

Ltrim - left side space

Diagram -20

Select sum (fee paid) 'total' from student group by branch order by total;

mysql> select * from student ;

id	Sname	totalfees	feepaid	city
1	Deepak	5000	2000	Pune
2	Kumar	6000	1000	Mumbai
3	Singh	3000	1000	Nagpur
4	Java	2000	1000	Mumbai
5	By	4000	3000	Pune
6	Kiran	5000	3000	Nagpur

6 rows in set <0.00 sec>

mysql> select sum<feepaid> as total from student group by city order by total;

total
4000
5000
6000

3 rows in set <0.03 sec>

```
mysql>select sum<feepaid> as total from student group by city order by total desc;
```

total
6000
5000
4000

3 rows in set <0.03 sec>

- Always use Desc and asc after the order by and column name;
- Occupation - create certified professional

Joins

Definitions

If you want to get data from multiple table data type using the single Select statement, you have to use joins. It gathers data from all the tables specified.

Types of Joins:

- Inner joins or equi-joins
- Outer joins
  - Left outer join
  - Right outer join
  - Full outer join
- Selfjoins

Customers

create table customers(
customerid int primary key auto_increment not null, customername varchar(32),
email varchar(32), phone int,
cstatus varchar(32) check(cstatus in('enable','disable')));

Customer Table

	customertid	customername	Email	Phone	Cstatus
▶	1	Ajay	ajay@gmail.com	9890	enable
	2	Vijay	vijay@gmail.com	9891	enable
	3	Ramesh	ramesh@gmail.com	9892	enable
	4	Suresh	suresh@gmail.com	9894	enable
	5	Aditya	aditya@gmail.com	9895	enable
	6	Deepak	deepak@gmail.com	9896	enable
	7	Kiran	kiran@gmail.com	9097	enable
	8	Mahesh	mahesh@gmail.com	9898	enable
	9	Mukesh	mukesh@gmail.com	9899	enable
	10	anand	anand@gmail.com	9900	enable
*	Null	Null	Null	Null	Null

Accounts

create table accounts(customerid int,
accountsid int primary key auto_increment not null,
accountstype varchar(2) check(accountstype in('sa','cu')), balance int, foreign key(customerid) references customers(customerid));

Accounts Table

	customertid	accountsid	accountstype	balance
▶	1	1	sa	10000
	2	2	cu	20000
	3	3	sa	30000
	4	4	cu	40000
	5	5	sa	50000
	6	6	cu	60000
	7	7	sa	70000

Address

create table address(customerid int, street varchar(32),
city varchar(32), country varchar(32),
foreign key(customerid) references customers(customerid));

Address Table

	customertid	street	city	country
▶	1	Lane 10	pune	india
	2	Lane 11	mysore	india
	3	Lane 12	pune	japan
	4	Lane 13	pune	india
	5	Lane 14	mysore	japan
	6	Lane 15	pune	india
	7	Lane 16	mysore	japan

1) Inner joins or equi-joins:

Inner joins gives you making secrets from the joined tables.

Q. 1) Display Customer name, email, account no., balance from customers and accounts tables.

- select customername, email, accountsid, balance from customers, accounts ;
  - This gives you product of both tables, which is invalid. So, when you are joining the tables you must remember to specify the join condition
- select customername, email, accountsid, balance from customers, accounts where customers.customerid = accounts.customerid;
  - This gives you exactly the matchingrecords

Q. 2) Display customer ID, customer name, status, city, country.

➤ select customers.customerid, customername, cstatus, city, country
from customers, address
where customers.customerid = address.customerid;

Q. 3) Display customer name, account no, email, balance of the customers who have saving accounts.

➤ select customername, accountsid, email, balance
from customers, accounts
where customers.customerid = accounts.customerid and accountstype='sa';

Q. 4) Display customer name, status, city, country of the customers who stay in Pune and are active.

➤ select customername, cstatus, city, country
from customers, address
where customers.customerid =address.customerid and city='pune' and cstatus='enable';

Q. 5) Display account no, account type, Balance, city of the customers who have current account and balance between 10000 and 100000 and are staying in Pune or Mysore.

➤ select accountsid, accountstype, balance, city
from accounts, address
where accounts.customerid = address.customerid and accountstype='cu' and balance between 10000 and 1000000 and city in('pune','mysore');

Q. 6) Display customer name, status, account no, balance, city and country of all the customers.

➤ select customername, cstatus, accountsid, balance, city, country
from customers, accounts, address
where customers.customerid = customers.customerid and customers.customerid = address.customerid;

Q. 7) Display customer name, status, account no, balance, city, country of all the customers who are deactivated and have a balance of less than 1000 and are not staying in India.

➤ select customername, cstatus, accountsid, balance, city, country
from customers, accounts, address
where customers.customerid = customers.customerid and customers.customerid = address.customerid
and accountstype='cu' and cstatus='disable' and balance<1000 and country<>'india';

Left Outer Joins :

Left outer joins gives matched rows plus remaining rows on the left hand side table, with null values for the table on the right hand.

Q. 1) Display customer name, email, account no, balance from customers and accounts tables.

➤ select customername, email, accountsid, balance
from customers left join accounts
on customers.customerid = accounts.customerid;

	customername	Email	accountsid	balance
▶	Ajay	ajay@gmail.com	1	10000
	Vijay	vijay@gmail.com	2	20000
	Ramesh	ramesh@gmail.com	3	30000
	Suresh	suresh@gmail.com	4	40000
	Aditya	aditya@gmail.com	5	50000
	Deepak	deepak@gmail.com	6	60000
	Kiran	kiran@gmail.com	7	70000
	Mahesh	mahesh@gmail.com	Null	Null
	Mukesh	mukesh@gmail.com	Null	Null
	anand	anand@gmail.com	Null	Null

New Syntax:

(For left Outer Join)

➤ Select account customer ID, customer name, email, account no, balance from customers customer left outer join account,account no account customer ID = customer ID

New Syntax (For Inner Join)

➤ Select customers.customerid,customername,email, accountsid,balance from customers INNER JOIN accounts on customers.customerid = accounts.customerid;

		customername	Email	accountsid	balance
▶	1	Ajay	ajay@gmail.com	1	10000
	2	Vijay	vijay@gmail.com	2	20000
	3	Ramesh	ramesh@gmail.com	3	30000
	4	Suresh	suresh@gmail.com	4	40000
	5	Aditya	aditya@gmail.com	5	50000
	6	Deepak	deepak@gmail.com	6	60000
	7	Kiran	kiran@gmail.com	7	70000

Right Outer Joins:

➤ Its same as left outer join, but the only difference is that it will give the matching records + remaining rows of right hand side table

➤ Right outer joins gives you matching records plus records remaining in the right side table

Example: select customers.customerid, customername, email, accountsid, balance from customers right outer join accounts on customers.customerid = accounts.customerid;

Full Outer Joins

Definitions:

- In mysql it is the combination of left join union right join, mysql avoids full outer join
- Full outer joins gives you matching record plus records remaining on the left side table plus the right side table

Note:

In mysql, there is no full outer join. It is either left outer join union or right outer join.

```
select *
from customers left join accounts
on customers.customerid = accounts.customerid union select *
from customers right join accounts
on customers.customerid = accounts.customerid;
```

customerid	customername	email	phone	cstatus	customerid	accountsid	accountstype	balance
1	ajay	ajay@gmail.com	9390	enable	1	1	sa	10000
2	vijay	vijay@gmail.com	9891	enable	2	2	cu	20000
3	ramesh	ramesh@gmail.com	9392	enable	3	3	sa	30000
4	suresh	suresh@gmail.com	9894	enable	4	4	cu	40000
5	aditya	aditya@gmail.com	9395	enable	5	5	sa	50000
6	deepak	deepak@gmail.com	9896	disable	6	6	cu	60000
7	kiran	kiran@gmail.com	9897	disable	7	7	sa	70000
8	mahesh	mahesh@gmail.com	9898	disable	8	8	sa	80000
9	mukesh	mukesh@gmail.com	9899	disable	9	9	sa	90000
10	anand	anand@gmail.com	9900	enable	10	10	sa	100000

Self Joins:

Joining in the table itself is called self-join.

Emplovee

employeeid	Email	managerid
101	ajay@gmail.com	110
102	vijay@gmail.com	109
103	ramesh@gmail.com	108
104	suresh@gmail.com	107
105	aditya@gmail.com	106
106	deepak@gmail.com	105
107	kiran@gmail.com	104
108	mahesh@gmail.com	103
109	mukesh@gmail.com	102
110	anand@gmail.com	101
Null	Null	Null


```
select a.employeeid,a.email, m.managerid
from employee a inner join employee m
where a.employeeid = m.managerid;
```

employeeid	Email	managerid
101	ajay@gmail.com	110
102	vijay@gmail.com	109
103	ramesh@gmail.com	108
104	suresh@gmail.com	107
105	aditya@gmail.com	106
106	deepak@gmail.com	105
107	kiran@gmail.com	104
108	mahesh@gmail.com	103
109	mukesh@gmail.com	102
110	anand@gmail.com	101

Sub Queries:

- You can include one query as part of another query, this is called a sub query
- The output of a sub query is used inside the main query
- It is also called inner query

Example:

- First, the sub query is evaluated and then the result of that sub query will be used to evaluate the main query
- Depending on the scenario, the sub query may return one or moreresults

Q. 1) What is the name of the customer whose account no is 8?

- Select customer name from customers where
  CID = (select cid from accounts where acc. No. = 3);

Q. 2) Display the account number and balance of the customers who areactive.

- Select account no, balance of accounts
  Where customer ID in (select C.ID from customers where status = 'enabled');
  (In operator is used for more than one record)

Q. 3) Display the balance of customers who are staying in Pune.

- Select balance of accounts where C. ID in (Select C. ID from address
  Where city = 'pune');

Q. 4) Display the name and status of the customers who are staying in Pune and Mysore.

- Select customer name, status from customer Where C.ID in (select city from address
  Where city in ('pune','mysore','pune');

Q. 5) Display the name and email of the customer who has a saving accounts and balance between 5000 and 10000.

➤ Select customer name, email from customer where C.ID in (Select atype, balance of accounts where account type ='SA' and balance between 5000 & 10000)

How can we create aTable From another Table Using Sub Queries?

➤ **With Data:**

Create table customer 1 as select
from customers;

➤ **With Data:**

Create table customer 1 as Select C.ID, Customer name, status
from customer where Status = 'enabled';

➤ **Without Data:**

Create table customer 3 as Select from customers where 1 =2;
(condition false: empty records Create dummy records here).

➤ **With Data:**

Create table customer 4 as
Select customer name, email, acc.no. balance
from Customers Cust., accounts acc.,
Where customer C.ID = account C.ID;

Q. 1) Display the name and balance of the customers who are staying in Pune.

➤ Select customer name, balance from customer cust, accounts acc where customer C.ID = account C.ID and C.ID in (Select C.ID from address where City = 'pune');

Alter Command

The alter command modifies the table by adding, deleting or modifying columns in a table which already exists.

Create table student (S.ID int, S.name char (10));

1) Adding the columns: Syntax :

alter table tab_name add (col_name type (size), col_name type (size));

Example:

Alter table students add (email char (15)); Alter table students add fee double, After student name; (in MYSQL)

Drop Command

The drop command removes a table from the database.

Syntax:

Alter table tab_name drop column col_name;

Example:

Alter table student drop column fee;

Modify Command

Syntax:

- Alter table tab_name modify Col_name type (size);

Example:

- Alter table students modify C.ID int;
- Alter table students modify student name char (2);

1. **Adding yhe Primary Key: Syntax:**
   Alter table tablename add primary key Column name;
   Example:
   Alter table student add primary key S.ID;
2. **Dropping the primary key:**
   Syntax:
   Alter table tab_name drop primary key;
   Example:
   Alter table student drop primary key;
3. **Adding the constraint:**
   Syntax:
   Alter table tab_name add constraint constraint_name
   ..... your constraint here.....;
   Example:
   Alter table student add constraint ck 1, email, not null;
4. **Dropping the constraint:**
   Syntax:
   Alter table tab_name drop Constraint cons_name;
   Example:
   Alter table student drop constraint CK1;

Index:

Definition

An Index is an ordered list of elements belonging to one column or a combination of two or more columns.

Syntax:

CREATE UNIQUE INDEX index_name ON table_name (column1, column2,...);

There are two types of indexes:

- Simple Index
- Composite Index

➤ **Simple Index:**

When an index is created on a single column of a tablethen it is called simple index.

Example:

- Create index i1 On customers (email);
- Now on disk. a space will be occupied with email in ascending order and a pointer to them. You search will become easiernow

➤ **Composite Index:**

When an index is created with a combination of two or more columns then it is called composite index.

Example:

Create index i2 on accounts (branch code, account type, account no.);

Customer ID	Customer Name	Email	Phone	City
1				
2				
3				
4				
5				

Select from customers where email='....'

This query directly goes through the customers table when there is no index table. But it is time consuming and not efficient.

Index table

- It goes into the index table when the index is created. Indexes mainly help to speed up the result of a query.
- If you do not have any index for assuming columns, then all the elements of that column will be searched for a given element one by one. But it takes more time when data is vast.
- To avoid this delay we can create indexes on that column
- When you create an index then all the elements of that column will be created and will be placed in a separate object
- When you search elements, DBMS will use this order list to find an element using a better search algorithm
- No need to use index directly because the index will be used by DBMS automatically wherever select statement contains the column which has the index
- If aprimary key exists, DBMS will create the index automatically
- If the tables contain a composite primary key, a composite index is created by the DBMS automatically
- A table can have one or more indexes

Dropping Index

Syntax: Drop index index_name;

Example : drop index i1;

Sequence

Sequences are used to generate the values automatically for any given column. They generate only number type value and arenot suitable for character type values

Syntax:

Create sequence seq_name Start with int_value
Min value int_value Max value int_value Increment by int_value Cycle
No cycle;

Example:

Create sequence s1 Start with 101
Increment by 1;

Sequence object has the following variations:

1. Current Value - return the current value
2. Next Value - returns the current value and then increment

Example:

Create sequences S2 Start with 101
Max value 1000
Increment by 1 Cycle;
Create sequence S3 Start with 1000
Min Value 101
Increment by -1 No cycle;

Dropping Sequence

This drops a sequence from the table.

Syntax:

Drop sequence seq_name;

Example:

Drop Sequence S1;

Views

Views are a logical entity which act as a mask on the table

Views will be created based on the table by fetching the data from the table, but the view does not contain the data physically. When view needs the data, it will get it from the table.

Syntax:

Create view view_name Select statement;

Query:

```
create table students(
studentid int primary key auto_increment not null, studentname varchar(32) not null,
email varchar(32) not null, phone int not null,
feepaid int, feebalance int, jobstatus varchar(32));
```

Student

Field	Type	Null	Key	Default	Extra
Studentid	Int(11)	NO	PRI	Null	auto_increment
Studentname	Varcharchar(32)	NO		Null	
Email	Varcharchar(32)	NO		Null	
Phone	Int(11)	NO		Null	
Feepaid	Int(11)	YES		Null	
Feebalance	Int(11)	YES		Null	
Jobstatus	Varcharchar(32)	YES		Null	

Role

admin mg - email, phone

Account mgr - Student ID, Fee Paid, Fee Balance

HR mgr - Student ID, Student name, email, phone, job status

Admin_view

create view admin as

select email

Email	phone
deepak@gmail.com	9390
monu@gmail.com	9891
om@gmail.com	9892

Account_view

create view accountmanager as

select studentid, feepaid, feebalance from students;

Studentid	Feepaid	feebalance
1	2000	3000
2	3000	2000
3	3500	1500

HR_View

select studentid, studentname, email, phone, jobstatus from students;

Studentid	Studentname	Email	Phone	Jobstatus
1	deepak	deepak@gmail.com	9890	On
2	monu	monu@gmail.com	9891	On
3	om	om@gmail.com	9892	Off

Advantages

1. Increase the security levels
2. Decreases redundancy

Types

1. Static views

2. Dynamic views

Static Views

- Static views also called as read only views
- We cannot perform insert, update and delete operations onstatic views

Dynamic Views

- Dynamic views are also called updatable views
- You can perform select, insert, update and delete operations on the updatable views or in the dynamic views

Sr. No	Select state of view	Static View	Dynamic View
1.	Contains primary key	-	.
2.	Does not contain a primary key	.	-
3.	Contains aggregate functions	.	-
4.	Does not contain agg format	-	.
5.	Contains group by	.	-
6.	Does not contain group by	-	.
7.	Contain joins	.	-
8.	Does not contain join	-	.

Example:

create table hello(id int primary key auto_increment not null,a int, b int, c int);

Table hello

id	a	b	c
1	10	20	30
2	40	50	60
3	70	80	90
4	0	80	90
Null	Null	Null	Null

create view v1 as

select sum(a) from hello;

Sum(a)
120

Granting The Permission

Syntax :

Grant select, delete, update, insert, create, drop,.....

On object_name to user_name;
Table name, view name index name, seq. name

Example:

Grant select, update on students to kiran; Grant select on ad_view to kiran, vas;

Revoking Permissions

Syntax :

Revoke select, delete, update, insert, create, drop,.....
On object name from user_name;

Example:

Revoke select, update on students from kiran; Revoke select on ad_view from kiran, vas;

Special Queries

In MySQL :

Create table account (ID int, name char (12), balance double);

In Create:

Create table account (ID number (2), name char (5);
balance number (g,2));
Insert into account values (1,'a', 1000);
Insert into account values (2, 'b', 2000);
Insert into account values (3, 'c', 4000);
Insert into account values (4, 'd', 9000);
Insert into account values (5, 'e', 6000);
Insert into account values (6, 'f', 7000);
Insert into account values (7, 'g', 5000);
Insert into account values (8, '4', 3000);
Insert into account values (9, '4', 4000);

1. **Display nth row(6) (Oracle)**

Select *from (select id, name, bal, row number from accounts where
Row num (8) where rn=6;
Select from account limit 5,1

2. **Display rows from m(2) to n(5) (Oracle)**

Select *from (select balance, row number from account where row number <7)
Where in between 2 & 5;

MySQL

Select *from account limit 1,4

3. **Display nth highest**

Oracle ad MySQL
(Select *from account where 3=(select count (distinct balance) from account
where acc.bal<=bal);

4. **Display Top N rows : (Oracle)**

Select *from (select *from account order by ID) Where row num < 5;
Select *from (select *from account order by ID description) Where row num <5;
Select*from account where row num <5;

MySQL :

Select *from account limit 4; (limit start index, no. of records)

5. **Display every n(3)th row : (Oracle)**

Select *from account where (row ID, 0) in
(Select row ID mod (rownum, 3) from accounts);

Invalid relational operator MySQL:

Select *from account where (ID.0) in
(select ID, mod(ID, 3) from account);

PL / SQL:

PL / SQL is a programming long with SQL which allows you to write sets of statements as one block called as a PL / SQL block

PL / SQL blocks contain the following :

1. Variables
2. Constants
3. Conditional stats
4. Looping stats
5. SQL states
6. All SQL operators & functions etc

Syntax of PL / SQL block:

Declare
Variable declaration Constant declaration

Begin

End;

Q. Write a PL / SQL block for the following requirements:

a) Declare 3 variables with the names a,b and c
b) Assign the same values to a and b
c) Find the sum of a and b starting in variable c
d) Display the sum Set server output on;

Declare

a. Number (4);
b. Number (4);
c. Number (6);

Begin

a: = 10;
b: = 20;
c: = a+b; dbms_output_line (c);

end;

It is used to alter end and then enter this is for execution SQD Set serveroutput on;
You have to write this before PL / SQL block (before declare)

Conditional Statements:

1) if Syntax

```
If (condition) then S1;
S2;
End if;
```

```
If (condition) then or if (condition) then
        S1;                    S1;
else                    else if (condition)
        S2                     S2:
end if                  else
        S3
        End if;
```

Example:

```
declare
    a number (4);
    b number (4);
begin
    a:=10;          b:=20;
    if (a>b) then dbms_output.put_line (a);
    else
    dbms output.put_line (b);
end if;
```

Looping Statements:

- **Simple Loop:**

```
Loop
S1;
S2;
Exit when condition End loop;
```


- **While loop**

```
while (condition) loop
S1;
S2;
End loop;
```

- **For Loop**

Syntax:

```
For value in (Reverse) Start Value end value Loop
----
----
End loop;
```

Q. Display the five divisible from 1 to 100

2)

```
declare
    i number (4); begin
    I:=1;
    loop
if (i mod 5=0) then dbms_output. Put_line (i); end if;
i:=i+1
exit when i=100; end loop;
end;
```

3)

```
declare
    i number (4); begin
    i : = 1;
    while (i<=100) loop
if (i mod 5=0)        then dbms_output.put_line(i); end if;
end loop; end;
```

1)

```
Declare i number (4); begin
    for i in 1....100 loop
dbms_output.put_line(i);
end loop;
end;
```

Stored Procedures:

The stored procedure is PL/SQL block name which takes some parameter and returns some parameters

Syntax :

Create or replace procedure pro_name (var_name(in/out/inout) data type, ...)

```
as
  ------------
  ------------
Begin
  --------------
end;
```

Q. 1) Write a stored procedure with the following requirements:

a. It has to take two parameters, m and n
b. Display the number from n to m

```
Create or replace procedure P1 (m number in number) as
i number (4); begin
for i in reverse m....n loop dbms_output.put_line(i); end loop;
end;
Call P1 (10, 25);
```

(This is used for changing the values without changing the complete block)

Q. 2) Write astored procedure with the following requirements:

Considering the following table:

ID	A	B	C	D
1	10	20	Null	Null
2	20	40	Null	Null
3	0	0	Null	Null
4	-	5	Null	Null

a) The procedure has to take ID as parameter and has to fetch a and b values of the given ID
b) It has to find do addition and sub of a and b
c) Results have to be updated with the hello table
d) If the sum is recorded, then delete therecord

Q. 3) Create or replace procedure P2 (id, number) as

```
a1 number (4);
b1 number (4);
c1 number (4);
d1 number (4); begin
select a,b into a1, b1 from hello where ID = x;
```



```
C1:=a1+b1 D1:=a1-b1;
If (c1=0) then
delete from hello where id=x; else
update hello set c=c1, d=d1 where id=x; end if;
end;
call p2(1);
```

1) Create the table called hello
2) Insert 4 sample records
3) Call P2

Types of parameters:

1. In parameters
2. Out parameters
3. Inout parameters

In Parameters:

In parameters carry the data from the caller of the procedure to the procedure.
Create procedure P1 (a in num, b number) as By default all the parameters are in parameter.

Out Parameter:

Out parameter carries the data from a procedure to the caller.

Example:

Create procedure P1 (a in number, b number, c out number, d out number);

Inout Parameter:

Inout parameter is used to carry the data from caller to procedure and from procedure to caller.
Create procedure P1 (a inout number) as

Advantage of stored procedure:

With Stored Procedure
= 5 ms +0 ms + 5000 ms + 5 ms
= 5010 ms

- When you submit your SQL stat to SQL engine, it will be compiled and executed when different users are submitting same query repeatedly. The query will be compiled every time which is unnecessary
- When one user is submitting a set of queries, then all queries will be compiled every time.
  This will increase the request processing time
- If you write a stored procedure with a set of SQL then you can reduce the request

processing time because stored procedure and SQL stat used in the stored procedure will be compiled only once and will be executed every time directly

Write the stored procedure with the llowing requirements:

- Considering the following student and results table

Students:

Student ID	Student Name	M1	M2	M3	M4	Total	Average	Status
101	A	50	80	70	80	Null	Null	Null

- The procedure has to take student ID as parameter and do the following tasks:
  a. Collects marks m1, m2, m3 and m4
  b. Find the total and average
  c. Find the status depending on the average
  d. Update the results table depending on the status
  e. Update the student table total, average, fails ID

- **Create the table called students**

```
Create table students (
Student ID number (4), Primary key, student name char 912) m1 number (2), m2 number (2), m3 number (2), m4 number (2)
Total number (3),
Average number (3),
Status char (10));
```

- **Insert four sample records**
  - Insert into students (SID, name, m1, m2, m3, m4) values (101, 'A', 90,70,80,90);
  - Insert into students (SID, name, m1, m2, m3, m4) values (102,'B',10,30,20,50),
  - Insert into students (SID, name, m1, m2, m3, m4) values (103, 'c', 199,90, 90,70);
- **Create the table called results**

```
Create table results (passed a number (2), fail number (2));
```

- **Insert one sample record**

```
Insert into results values (0,0);
```

- **Create a replace procedure P3 (ID number) as**

```
mm1 number (2);
mm2 number (2);
mm3 number (2);
mm4 number (2);
total number (2)
average number (2);
```



```
Stat char (10)
X number (2);
Begin
Select m1, m2, m3, m4 into mm1, mm2, mm3, mm4, from students
Where SID = ID;
    Total : = mm1 + mm2 + mm3 + mm4;
    Average : = Total /4;
    If (avg<50) then Stat:="passed";
    End if;
Update students select total =101, average = avg status =stat
Where SID = ID;
    if (stat='passed') then
    select passed into x from results;
    x=x+1;
    update results set passed = x;
    else
    select failed into x from results;
    x=x+1;
    update results set failed = x;
    end if;
    end;
    Call P3 (101);
    Call P3 (102);
    Call P3 (103);
```

Trigger

- Trigger is a PL / SQL block within same name
- Trigger will be called automatically whenever you do insert, update or delete operations on the table

Syntax:

```
Create or replace trigger name
[after / before]
[insert / update / delete]
On table_name
[for each row]
declare
--------
Variable declarations
--------
```

```
Begin
End;
```

Q. Write a trigger with following requirements:

1) Consider the following two tables:

Ill. Students

Student ID	Student Name	Email	Phone
101	A		1111111111

Idl student_backup

Student ID	Student Name	Email	Phone	Operation	Update
101	A		1111	Update	8 feb
102	A		1234	Update	8 feb
102	A		1234	Update	8 feb

Old is an object representing 1 row and it is only for update and delete open

A. New used for insert open

2) Whenever update or delete operation is issued on ONLYFORJAVA student table, then the trigger has to be invoked automatically and has to move the existing records into the ONLYFORJAVA student_backup table

Example:

1. Create ONLYFORJAVA student table and insert for the record and then ONLYFORJAVA student_backup table and insert six records

```
create or replace trigger
before update or delete
on ONLYFORJAVAstudents
declare
ID number (3);
Sn Char (10);
Ph number (7);
Op char (10)
Opd char (10)
begin
id : = :old.sid;
sn : = old.sname;
em : = old.email;
```



```
ph : = old.phone;
if (updating) then op : ='update';
end if;
if (deleting) then op : = 'delete';
end if;
opd ; = sysdate;
insert into ONLYFORJAVA students_backup
Values (ID, Student name, email, phone, OP, OPD);
End;
```

Transactional Statements:

There are three transactional statements:

1. Commitment
2. Rollback
3. Save point

Definitions:

The transaction is the process of performing a set of database operations as one unit

When all the database operations in this unit are successful. then we make the operation permanent by issuing commit

When any one database operation unit fails, then we have to cancel the operation by issuing rollback

```
SQL > S1
SQL > S2
SQL > S3
SQL > Commit; (roll back;)
SQL > Select
SQL > Savepoint SP1;
SQL > S1
SWL > S2
SQL > S3
SQL > Select
SQL > Commit: (Rollback):
SQL > Savepoint SP1; (Rollback SP1);
SQL > Select
SQL > Savepoint SP2; (rollback SP2)
```

Questions:

1. What is DBMS?
2. What is SQL?
3. What are the different types of SQL's statements?
4. What is RDBMS?
5. What is a field in a database?
6. What are the advantages of SQL?

7. What is a Record in a database?
8. What is a Table in a database?

◆◆◆