

Chap -1 : Basics Of Java

What is Java

Java is a **programming language** and a **platform**.

Java is a high level, robust, secured and object-oriented programming language.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own runtime environment (JRE) and API, it is called platform.

Java Example

Let's have a quick look at java programming example. A detailed description of hello Java example is given in next page.

```
1. class Simple{  
2.     public static void main(String args[]){  
3.         System.out.println("Hello Java");  
4.     }  
5. }
```

History of Java

Java history is interesting to know. The history of java starts from Green Team. Java team members (also known as **Green Team**), initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc.

For the green team members, it was an advance concept at that time. But, it was suited for internet programming. Later, Java technology as incorporated by Netscape.



James Gosling

Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc. There are given the major points that describes the history of java.

- 1) **James Gosling, Mike Sheridan, and Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.
- 2) Originally designed for small, embedded systems in electronic appliances like set-top boxes.

Java Programming by Sharma Sir

3) Firstly, it was called "**Greentalk**" by James Gosling and file extension was .gt.

4) After that, it was called **Oak** and was developed as a part of the Green project.



Why "Oak" name

5) **Why Oak?** Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany, Romania etc.

6) In 1995, Oak was renamed as "**Java**" because it was already a trademark by Oak Technologies.

Why "Java" name

7) **Why had they chosen java name for java language?** The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "Silk", "jolt", "DNA" etc. They wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.

According to James Gosling "Java was one of the top choices along with **Silk**". Since java was so unique, most of the team members preferred java.

8) Java is an island of Indonesia where first coffee was produced (called java coffee).

9) Notice that Java is just a name not an acronym.

10) Originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

11) In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.

12) JDK 1.0 released in (January 23, 1996).

Java Version History

There are many java versions that has been released. Current stable release of Java is Java SE 8.

1. JDK Alpha and Beta (1995)
2. JDK 1.0 (23rd Jan, 1996)
3. JDK 1.1 (19th Feb, 1997)
4. J2SE 1.2 (8th Dec, 1998)
5. J2SE 1.3 (8th May, 2000)
6. J2SE 1.4 (6th Feb, 2002)

Java Programming by Sharma Sir

7. J2SE 5.0 (30th Sep, 2004)
8. Java SE 6 (11th Dec, 2006)
9. Java SE 7 (28th July, 2011)
10. Java SE 8 (18th March, 2014)

Features of Java

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

Simple

According to Sun, Java language is simple because:

syntax is based on C++ (so easier for programmers to learn it after C++).

removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc.

No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

Object-oriented

Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.

Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules.

Basic concepts of OOPs are:

1. Object
2. Class

Java Programming by Sharma Sir

3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Platform Independent

A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

Secured

Java is secured because:

- **No explicit pointer**
- **Java Programs run inside virtual machine sandbox**

Classloader: adds security by separating the package for the classes of the local file system from those that are imported from network sources.

- **Bytecode Verifier:** checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager:** determines what resources a class can access such as reading and writing to the local disk.

These security are provided by java language. Some security can also be provided by application developer through SSL, JAAS, Cryptography etc.

Robust

Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

Java Programming by Sharma Sir

Architecture-neutral

There is no implementation dependent features e.g. size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

Portable

We may carry the java bytecode to any platform.

High-performance

Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

Distributed

We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications etc.

C++ vs Java

There are many differences and similarities between C++ programming language and Java. A list of top differences between C++ and Java are given below:

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
Goto	C++ supports goto statement.	Java doesn't support goto statement.
Multiple inheritance	C++ supports multiple	Java doesn't support multiple

Java Programming by Sharma Sir

	inheritance.	inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. But you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only.	Java uses compiler and interpreter both.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (<code>/** ... */</code>) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator

Java Programming by Sharma Sir

		that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses single inheritance tree always because all classes are the child of Object class in java. Object class is the root of inheritance tree in java.

Simple Program of Java

To create a simple java program, you need to create a class that contains main method. Let's understand the requirement first.

Requirement for Hello Java Example

For executing any java program, you need to

- install the JDK if you don't have installed it, [download the JDK](#) and install it.
- set path of the jdk/bin directory. <http://www.javatpoint.com/how-to-set-path-in-java>
- create the java program
- compile and run the java program

Creating hello java example

Let's create the hello java program:

1. **class** Simple{
2. **public static void** main(String args[]){
3. System.out.println("Hello Java");
4. }
5. }

save this file as Simple.java

To compile: javac Simple.java

To execute: java Simple

Output: Hello Java

Difference between JDK, JRE and JVM

JVM

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

The JVM performs following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JRE

JRE is an acronym for Java Runtime Environment. It is used to provide runtime environment. It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.

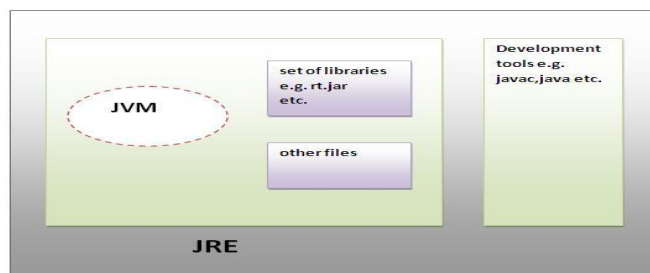
Implementation of JVMs are also actively released by other companies besides Sun Micro Systems.



JRE

JDK

JDK is an acronym for Java Development Kit. It physically exists. It contains JRE + development tools.



JDK

Variables and Data Types in Java

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of data types in java: primitive and non-primitive.

Variable

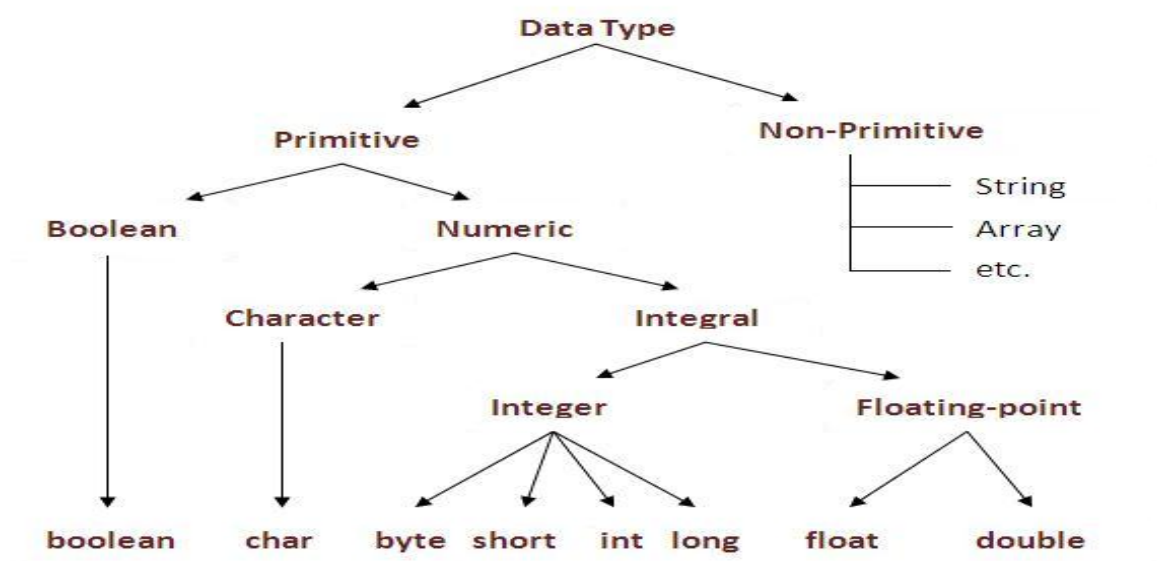
Variable is name of reserved area allocated in memory. In other words, it is a name of memory location. It is a combination of "vary + able" that means its value can be changed.

1. `int data=50;`//Here data is variable

Types of Variable

There are three types of variables in java:

- local variable
- instance variable
- static variable



1) Local Variable

A variable which is declared inside the method is called local variable.

2) Instance Variable

A variable which is declared inside the class but outside the method, is called instance variable . It is not declared as static.

3) Static variable

A variable that is declared as static is called static variable. It cannot be local.

We will have detailed learning of these variables in next chapters.

Example to understand the types of variables in java

1. **class** A{
2. **int** data=50;//instance variable
3. **static int** m=100;//static variable
4. **void** method(){
5. **int** n=90;//local variable
6. }
7. }//end of class

Data Types in Java

Data types represent the different values to be stored in the variable. In java, there are two types of data types:

- Primitive data types
- Non-primitive data types

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Why char uses 2 byte in java and what is \u0000 ?

It is because java uses Unicode system than ASCII code system. The \u0000 is the lowest range of Unicode system. To get detail explanation about Unicode visit next page.

Java Variable Example: Add Two Numbers

1. **class** Simple{
2. **public static void** main(String[] args){
3. **int** a=10;
4. **int** b=10;
5. **int** c=a+b;
6. System.out.println(c);
7. }}

Output:

```
20
```

Java Variable Example: Widening

1. **class** Simple{
2. **public static void** main(String[] args){
3. **int** a=10;
4. **float** f=a;
5. System.out.println(a);
6. System.out.println(f);
7. }}

Output:

```
10
```

```
10.0
```

Java Variable Example: Narrowing (Typecasting)

1. **class** Simple{
2. **public static void** main(String[] args){
3. **float** f=10.5f;
4. //int a=f;//Compile time error
5. **int** a=(**int**)f;
6. System.out.println(f);
7. System.out.println(a);
8. }}

Output:

```
10.5
```

```
10
```

Java Variable Example: Overflow

1. **class** Simple{
2. **public static void** main(String[] args){

```
3. //Overflow
4. int a=130;
5. byte b=(byte)a;
6. System.out.println(a);
7. System.out.println(b);
8. }}
```

Output:

```
130
-126
```

Java Variable Example: Adding Lower Type

```
1. class Simple{
2. public static void main(String[] args){
3. byte a=10;
4. byte b=10;
5. //byte c=a+b;//Compile Time Error: because a+b=20 will be int
6. byte c=(byte)(a+b);
7. System.out.println(c);
8. }}
```

Output:

```
20
```

Operators in java

Operator in java is a symbol that is used to perform operations. For example: +, -, *, / etc.

There are many types of operators in java which are given below:

- Unary Operator,
- Arithmetic Operator,
- shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Java Operator Precedence

Operator Type	Category	Precedence
Unary	postfix	<code>expr++ expr--</code>
	prefix	<code>++expr --expr +expr -expr ~ !</code>
Arithmetic	multiplicative	<code>* / %</code>
	additive	<code>+ -</code>
Shift	shift	<code><< >> >>></code>
Relational	comparison	<code>< > <= >= instanceof</code>
	equality	<code>== !=</code>
Bitwise	bitwise AND	<code>&</code>
	bitwise exclusive OR	<code>^</code>
	bitwise inclusive OR	<code> </code>
Logical	logical AND	<code>&&</code>
	logical OR	<code> </code>
Ternary	ternary	<code>? :</code>
Assignment	assignment	<code>= += -= *= /= %= &= ^= = <<= >>= >>>=</code>

Chap-2 Controls Statements In Java

Java If-else Statement

The Java *if statement* is used to test the condition. It checks boolean condition: *true* or *false*. There are various types of if statement in java.

- if statement
- if-else statement
- if-else-if ladder
- nested if statement

Java IF Statement

The Java if statement tests the condition. It executes the *if block* if condition is true.

Syntax:

1. **if**(condition){
2. //code to be executed
3. }

Java IF-else Statement

The Java if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block* is executed.

Syntax:

1. **if**(condition){
2. //code if condition is true
3. }**else**{
4. //code if condition is false
5. }

Java IF-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

Syntax:

1. **if**(condition1){
2. //code to be executed if condition1 is true
3. }**else if**(condition2){
4. //code to be executed if condition2 is true

```
5. }
6. else if(condition3){
7. //code to be executed if condition3 is true
8. }
9. ...
10.else{
11.//code to be executed if all the conditions are false
12.}
```

Java Switch Statement

The Java *switch statement* executes one statement from multiple conditions. It is like if-else-if ladder statement.

Syntax:

```
1. switch(expression){
2. case value1:
3. //code to be executed;
4. break; //optional
5. case value2:
6. //code to be executed;
7. break; //optional
8. ....
9.
10.default:
11. code to be executed if all cases are not matched;
12.}
```

Java For Loop

The Java *for loop* is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

There are three types of for loop in java.

- Simple For Loop
- For-each or Enhanced For Loop
- Labeled For Loop

Java Simple For Loop

The simple for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

Syntax:

1. **for**(initialization;condition;incr/decr){
2. //code to be executed
3. }

Java While Loop

The Java *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Syntax:

1. **while**(condition){
2. //code to be executed
3. }

Java do-while Loop

The Java *do-while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The Java *do-while loop* is executed at least once because condition is checked after loop body.

Syntax:

1. **do**{
2. //code to be executed
3. }**while**(condition);

Java Break Statement

The Java *break* is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

Syntax:

1. jump-statement;
2. **break**;

Java Continue Statement

The Java *continue statement* is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

Syntax:

1. jump-statement;
2. **continue**;

Java Comments:

The java comments are statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for specific time.

Types of Java Comments

There are 3 types of comments in java.

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment

1) Java Single Line Comment

The single line comment is used to comment only one line.

Syntax:

1. `//This is single line comment`

2) Java Multi Line Comment

The multi line comment is used to comment multiple lines of code.

Syntax:

1. `/*`
2. This
3. is
4. multi line
5. comment
6. `*/`

3) Java Documentation Comment

The documentation comment is used to create documentation API. To create documentation API, you need to use **javadoc tool**.

Syntax:

1. `/**`
2. This
3. is
4. documentation
5. comment
6. `*/`

Chap- 3: Java Object Class

Java OOPs Concepts

Object Oriented Programming is a paradigm that provides many concepts such as **inheritance, data binding, polymorphism** etc.

Simula is considered as the first object-oriented programming language. The programming paradigm where everything is represented as an object, is known as truly object-oriented programming language.

Smalltalk is considered as the first truly object-oriented programming language.

OOPs (Object Oriented Programming System)

Object means a real word entity such as pen, chair, table etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

Collection of objects is called class. It is a logical entity.

Inheritance

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

When **one task is performed by different ways** i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something e.g. cat speaks meow, dog barks woof etc.

Java Programming by Sharma Sir

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.

In java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Advantage of OOPs over Procedure-oriented programming language

- 1)OOPs makes development and maintenance easier where as in Procedure-oriented programming language it is not easy to manage if code grows as project size grows.
- 2)OOPs provides data hiding whereas in Procedure-oriented programming language a global data can be accessed from anywhere.
- 3)OOPs provides ability to simulate real-world event much more effectively. We can provide the solution of real word problem if we are using the Object-Oriented Programming language.

What is difference between object-oriented programming language and object-based programming language?

Object based programming language follows all the features of OOPs except Inheritance. JavaScript and VBScript are examples of object based programming languages.

Class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- **fields**
- **methods**
- **constructors**
- **blocks**
- **nested class and interface**

Syntax to declare a class:

1. **class** <class_name>{

2. field;
3. method;
4. }

Instance variable in Java

A variable which is created inside the class but outside the method, is known as instance variable. Instance variable doesn't get memory at compile time. It gets memory at run time when object(instance) is created. That is why, it is known as instance variable.

Method in Java

In java, a method is like function i.e. used to expose behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

new keyword in Java

The new keyword is used to allocate memory at run time. All objects get memory in Heap memory area.

Object and Class Example: main within class

In this example, we have created a Student class that have two data members id and name. We are creating the object of the Student class by new keyword and printing the objects value.

Here, we are creating main() method inside the class.

File: Student.java

1. **class** Student{
2. **int** id;//field or data member or instance variable
3. String name;
4. **public static void** main(String args[]){
5. Student s1=**new** Student();//creating an object of Student
6. System.out.println(s1.id);//accessing member through reference variable
7. System.out.println(s1.name);
8. }
9. }

Output:

```
0
null
```

Object and Class Example: main outside class

In real time development, we create classes and use it from another class. It is a better approach than previous one. Let's see a simple example, where we are having main() method in another class.

We can have multiple classes in different java files or single java file. If you define multiple classes in a single java source file, it is a good idea to save the file name with the class name which has main() method.

File: TestStudent1.java

```
1. class Student{
2.   int id;
3.   String name;
4. }
5. class TestStudent1{
6.   public static void main(String args[]){
7.     Student s1=new Student();
8.     System.out.println(s1.id);
9.     System.out.println(s1.name);
10.  }
11. }
```

Output:

```
0
null
```

3 Ways to initialize object

There are 3 ways to initialize object in java.

1. By reference variable
2. By method
3. By constructor

1) Object and Class Example: Initialization through reference

Initializing object simply means storing data into object. Let's see a simple example where we are going to initialize object through reference variable.

File: TestStudent2.java

```
1. class Student{
2.   int id;
3.   String name;
4. }
5. class TestStudent2{
```

Java Programming by Sharma Sir

```
6. public static void main(String args[]){
7.   Student s1=new Student();
8.   s1.id=101;
9.   s1.name="Sonoo";
10.  System.out.println(s1.id+" "+s1.name);//printing members with a white space
11. }
12. }
```

Output:

```
101 Sonoo
```

We can also create multiple objects and store information in it through reference variable.

File: TestStudent3.java

```
1. class Student{
2.   int id;
3.   String name;
4. }
5. class TestStudent3{
6.   public static void main(String args[]){
7.     //Creating objects
8.     Student s1=new Student();
9.     Student s2=new Student();
10.    //Initializing objects
11.    s1.id=101;
12.    s1.name="Sonoo";
13.    s2.id=102;
14.    s2.name="Amit";
15.    //Printing data
16.    System.out.println(s1.id+" "+s1.name);
17.    System.out.println(s2.id+" "+s2.name);
18. }
19. }
```

Output:

```
101 Sonoo
102 Amit
```

2) Object and Class Example: Initialization through method

In this example, we are creating the two objects of Student class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

File: TestStudent4.java

```
1. class Student{
2.   int rollno;
3.   String name;
4.   void insertRecord(int r, String n){
5.     rollno=r;
6.     name=n;
7.   }
8.   void displayInformation(){System.out.println(rollno+" "+name);}
9. }
10. class TestStudent4{
11.   public static void main(String args[]){
12.     Student s1=new Student();
13.     Student s2=new Student();
14.     s1.insertRecord(111,"Karan");
15.     s2.insertRecord(222,"Aryan");
16.     s1.displayInformation();
17.     s2.displayInformation();
18.   }
19. }
```

Output:

```
111 Karan
222 Aryan
```

As you can see in the above figure, object gets the memory in heap memory area. The reference variable refers to the object allocated in the heap memory area. Here, s1 and s2 both are reference variables that refer to the objects allocated in memory.

3) Object and Class Example: Initialization through constructor

We will learn about constructors in java later.

Object and Class Example: Employee

Let's see an example where we are maintaining records of employees.

File: TestEmployee.java

```
1. class Employee{
2.     int id;
3.     String name;
4.     float salary;
5.     void insert(int i, String n, float s) {
6.         id=i;
7.         name=n;
8.         salary=s;
9.     }
10.    void display(){System.out.println(id+" "+name+" "+salary);}
11.}
12.public class TestEmployee {
13.public static void main(String[] args) {
14.    Employee e1=new Employee();
15.    Employee e2=new Employee();
16.    Employee e3=new Employee();
17.    e1.insert(101,"ajeet",45000);
18.    e2.insert(102,"irfan",25000);
19.    e3.insert(103,"nakul",55000);
20.    e1.display();
21.    e2.display();
22.    e3.display();
23.}
24.}
```

Output:

```
101 ajeet 45000.0
102 irfan 25000.0
103 nakul 55000.0
```

Object and Class Example: Rectangle

There is given another example that maintains the records of Rectangle class.

File: TestRectangle1.java

```
1. class Rectangle{
2.     int length;
3.     int width;
4.     void insert(int l, int w){
5.     length=l;
```



```
6. width=w;
7. }
8. void calculateArea(){System.out.println(length*width);}
9. }
10. class TestRectangle1{
11. public static void main(String args[]){
12. Rectangle r1=new Rectangle();
13. Rectangle r2=new Rectangle();
14. r1.insert(11,5);
15. r2.insert(3,15);
16. r1.calculateArea();
17. r2.calculateArea();
18. }
19. }
```

Output:

```
55
45
```

What are the different ways to create an object in Java?

Anonymous object

Anonymous simply means nameless. An object which has no reference is known as anonymous object. It can be used at the time of object creation only.

If you have to use an object only once, anonymous object is a good approach. For example:

1. **new** Calculation();//anonymous object

Calling method through reference:

1. Calculation c=**new** Calculation();
2. c.fact(5);

Calling method through anonymous object

1. **new** Calculation().fact(5);

Let's see the full example of anonymous object in java.

1. **class** Calculation{
2. void fact(**int** n){
3. **int** fact=1;
4. **for**(**int** i=1;i<=n;i++){

```
5.    fact=fact*i;
6.    }
7.    System.out.println("factorial is "+fact);
8.    }
9.    public static void main(String args[]){
10.   new Calculation().fact(5);//calling method with anonymous object
11. }
12. }
```

Output:

```
Factorial is 120
```

Creating multiple objects by one type only

We can create multiple objects by one type only as we do in case of primitives.

Initialization of primitive variables:

```
1. int a=10, b=20;
```

Initialization of reference variables:

```
1. Rectangle r1=new Rectangle(), r2=new Rectangle();//creating two objects
```

Let's see the example:

```
1. class Rectangle{
2.   int length;
3.   int width;
4.   void insert(int l,int w){
5.     length=l;
6.     width=w;
7.   }
8.   void calculateArea(){System.out.println(length*width);}
9. }
10. class TestRectangle2{
11.  public static void main(String args[]){
12.   Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects
13.   r1.insert(11,5);
14.   r2.insert(3,15);
15.   r1.calculateArea();
16.   r2.calculateArea();
17. }
18. }
```

Output:

```
55  
45
```

Constructor in Java

In Java, constructor is a block of codes similar to method. It is called when an instance of object is created and memory is allocated for the object.

It is a special type of method which is used to initialize the object.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating java constructor

There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of java constructors

There are two types of constructors in java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Java static keyword

The **static keyword** in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class

1) Java static variable

If you declare any variable as static, it is known static variable.

- The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.
- The static variable gets memory only once in class area at the time of class loading.

Advantage of static variable

It makes your program **memory efficient** (i.e it saves memory).

Understanding problem without static variable

1. **class** Student{
2. **int** rollno;
3. String name;
4. String college="ITS";
5. }

Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created. All student have its unique rollno and name so instance data member is good. Here, college refers to the common property of all objects. If we make it static, this field will get memory only once.

this keyword in java

There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.

Usage of java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

Chap- 4: Inheritance in Java

Concept:

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.

The idea behind inheritance in java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of parent class, and you can add new methods and fields also.

Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Syntax of Java Inheritance

1. **class** Subclass-name **extends** Superclass-name
2. {
3. //methods and fields
4. }

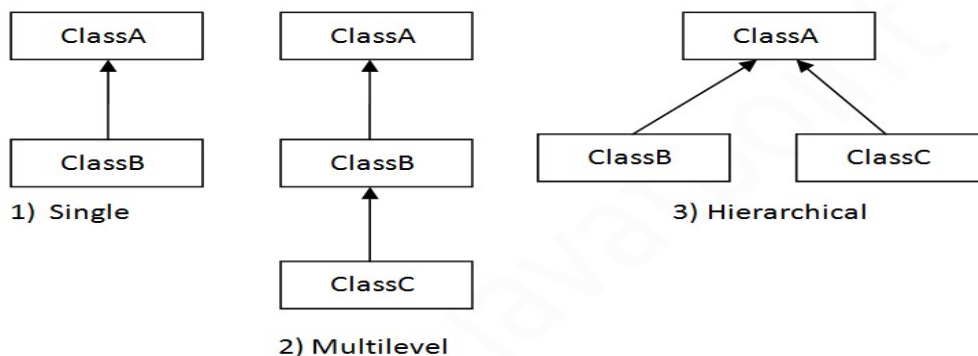
The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called parent or super class and the new class is called child or subclass.

Types of inheritance in java

On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



Note: Multiple and Hybrid Inheritance are not supported by Java.

Single Inheritance Example

File: TestInheritance.java

```
1. class Animal{
2. void eat(){System.out.println("eating...");}
3. }
4. class Dog extends Animal{
5. void bark(){System.out.println("barking...");}
6. }
7. class TestInheritance{
8. public static void main(String args[]){
9. Dog d=new Dog();
10.d.bark();
11.d.eat();
12.}}
```

Output:

```
barking...
eating...
```

Multilevel Inheritance Example

File: TestInheritance2.java

```
1. class Animal{
2. void eat(){System.out.println("eating...");}
3. }
4. class Dog extends Animal{
5. void bark(){System.out.println("barking...");}
6. }
7. class BabyDog extends Dog{
8. void weep(){System.out.println("weeping...");}
9. }
10.class TestInheritance2{
11.public static void main(String args[]){
12.BabyDog d=new BabyDog();
```

```
13.d.weep();
14.d.bark();
15.d.eat();
16.}}
```

Output:

```
weeping...
barking...
eating...
```

Hierarchical Inheritance Example

File: TestInheritance3.java

```
1. class Animal{
2. void eat(){System.out.println("eating...");}
3. }
4. class Dog extends Animal{
5. void bark(){System.out.println("barking...");}
6. }
7. class Cat extends Animal{
8. void meow(){System.out.println("meowing...");}
9. }
10. class TestInheritance3{
11. public static void main(String args[]){
12. Cat c=new Cat();
13. c.meow();
14. c.eat();
15. //c.bark();//C.T.Error
16. }}
```

Output:

```
meowing...
eating...
```

Chap: 5 Polymorphism

Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.

In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. method must have same name as in the parent class
2. method must have same parameter as in the parent class.
3. must be IS-A relationship (inheritance).

super keyword in java

The **super** keyword in java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

Final Keyword In Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example of final variable

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

1. **class** Bike9{
2. **final int** speedlimit=90;//final variable
3. **void** run(){
4. speedlimit=400;
5. }

```
6. public static void main(String args[]){
7.   Bike9 obj=new Bike9();
8.   obj.run();
9. }
10. }//end of class
```

Test it Now

Output:Compile Time Error

2) Java final method

If you make any method as final, you cannot override it.

Example of final method

```
1. class Bike{
2.   final void run(){System.out.println("running");}
3. }
4.
5. class Honda extends Bike{
6.   void run(){System.out.println("running safely with 100kmph");}
7.
8.   public static void main(String args[]){
9.     Honda honda= new Honda();
10.    honda.run();
11.  }
12. }
```

Test it Now

Output:Compile Time Error

3) Java final class

If you make any class as final, you cannot extend it.

Example of final class

```
1. final class Bike{}
2.
3. class Honda1 extends Bike{
4.   void run(){System.out.println("running safely with 100kmph");}
5.
6.   public static void main(String args[]){
7.     Honda1 honda= new Honda1();
```

```
8. honda.run();
9. }
10. }
```

Test it Now

```
Output:Compile Time Error
```

Java instanceof

The **java instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface).

The instanceof in java is also known as *type comparison operator* because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

Simple example of java instanceof

Let's see the simple example of instance operator where it tests the current class.

```
1. class Simple1{
2.     public static void main(String args[]){
3.         Simple1 s=new Simple1();
4.         System.out.println(s instanceof Simple1);//true
5.     }
6. }
```

Test it Now

```
Output:true
```

An object of subclass type is also a type of parent class. For example, if Dog extends Animal then object of Dog can be referred by either Dog or Animal class.

Abstract class in Java

A class that is declared as abstract is known as **abstract class**. It needs to be extended and its method implemented. It cannot be instantiated.

Example abstract class

```
1. abstract class A{}
```

abstract method

A method that is declared as abstract and does not have implementation is known as abstract method.

Example abstract method

1. **abstract void** printStatus();//no body and abstract

Example of abstract class that has abstract method

In this example, Bike the abstract class that contains only one abstract method run. Its implementation is provided by the Honda class.

1. **abstract class** Bike{
2. **abstract void** run();
3. }
4. **class** Honda4 **extends** Bike{
5. **void** run(){System.out.println("running safely..");}
6. **public static void** main(String args[]){
7. Bike obj = **new** Honda4();
8. obj.run();
9. }
10. }

Test it Now

```
running safely..
```

Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

Abstraction lets you focus on what the object does instead of how it does it.

Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Interface in Java

An **interface in java** is a blueprint of a class. It has static constants and abstract methods.

Java Programming by Sharma Sir

The interface in java is **a mechanism to achieve abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.

Java Interface also **represents IS-A relationship**.

It cannot be instantiated just like abstract class.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

Java Interface Example

In this example, Printable interface has only one method, its implementation is provided in the A class.

```
1. interface printable{
2. void print();
3. }
4. class A6 implements printable{
5. public void print(){System.out.println("Hello");}
6.
7. public static void main(String args[]){
8. A6 obj = new A6();
9. obj.print();
10. }
11. }
```

Test it Now

Output:

```
Hello
```

Java Package

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

- 1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- 2) Java package provides access protection.
- 3) Java package removes naming collision.

How to access package from another package?

There are three ways to access the package from outside the package.

1. `import package.*;`
2. `import package.classname;`
3. fully qualified name.

1) Using packagename.*

If you use `package.*` then all the classes and interfaces of this package will be accessible but not subpackages.

The `import` keyword is used to make the classes and interface of another package accessible to the current package.

Example of package that import the packagename.*

1. `//save by A.java`
2. **package** `pack;`
3. **public class** `A{`
4. **public void** `msg(){System.out.println("Hello");}`
5. `}`
1. `//save by B.java`
2. **package** `mypack;`
3. **import** `pack.*;`
4.
5. **class** `B{`
6. **public static void** `main(String args[]){`
7. `A obj = new A();`
8. `obj.msg();`
9. `}`
10. `}`

Output:Hello

2) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
1. //save by A.java
2.
3. package pack;
4. public class A{
5.     public void msg(){System.out.println("Hello");}
6. }
1. //save by B.java
2. package mypack;
3. import pack.A;
4.
5. class B{
6.     public static void main(String args[]){
7.         A obj = new A();
8.         obj.msg();
9.     }
10. }
```

Output:Hello

3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

```
1. //save by A.java
2. package pack;
3. public class A{
4.     public void msg(){System.out.println("Hello");}
```

```
5. }  
1. //save by B.java  
2. package mypack;  
3. class B{  
4.   public static void main(String args[]){  
5.     pack.A obj = new pack.A();//using fully qualified name  
6.     obj.msg();  
7.   }  
8. }
```

Output:Hello

Access Modifiers in java

There are two types of modifiers in java: **access modifiers** and **non-access modifiers**.

The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of java access modifiers:

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

There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc. Here, we will learn access modifiers.

Chap: 6 -Java Array

Normally, array is a collection of similar type of elements that have contiguous memory location.

Java array is an object that contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.

Array in java is index based, first element of the array is stored at 0 index.

Advantage of Java Array

- **Code Optimization:** It makes the code optimized, we can retrieve or sort the data easily.
- **Random access:** We can get any data located at any index position.

Disadvantage of Java Array

- **Size Limit:** We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in java

Syntax to Declare an Array in java

1. `dataType[] arr; (or)`
2. `dataType []arr; (or)`
3. `dataType arr[];`

Instantiation of an Array in java

1. `arrayRefVar=new datatype[size];`

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

1. **int** `a[]={33,3,4,5};`//declaration, instantiation and initialization

Multidimensional array in java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in java

1. `dataType[][] arrayRefVar; (or)`
2. `dataType [][]arrayRefVar; (or)`
3. `dataType arrayRefVar[][]; (or)`
4. `dataType []arrayRefVar[];`

Chap: 7 Java OOPs Misc

Object class in Java

The **Object class** is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, known as upcasting.

Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee, Student etc, we can use Object class reference to refer that object. For example:

1. `Object obj=getObject();` //we don't know what object will be returned from this method

The Object class provides some common behaviors to all the objects such as object can be compared, object can be cloned, object can be notified etc.

Methods of Object class

The Object class provides many methods. They are as follows:

Method	Description
<code>public final Class getClass()</code>	returns the Class class object of this object. The Class class can further be used to get the metadata of this class.
<code>public int hashCode()</code>	returns the hashcode number for this object.
<code>public boolean equals(Object obj)</code>	compares the given object to this object.
<code>protected Object clone() throws CloneNotSupportedException</code>	creates and returns the exact copy (clone) of this object.
<code>public String toString()</code>	returns the string representation of this object.
<code>public final void notify()</code>	wakes up single thread, waiting on this object's monitor.
<code>public final void notifyAll()</code>	wakes up all the threads, waiting on this object's monitor.

Java Programming by Sharma Sir

public final void wait(long timeout)throws InterruptedException	causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait(long timeout,int nanos)throws InterruptedException	causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).
public final void wait()throws InterruptedException	causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).
protected void finalize()throws Throwable	

Wrapper class in Java

Wrapper class in java provides the mechanism *to convert primitive into object and object into primitive*.

Since J2SE 5.0, **autoboxing** and **unboxing** feature converts primitive into object and object into primitive automatically. The automatic conversion of primitive into object is known as autoboxing and vice-versa unboxing.

The eight classes of *java.lang* package are known as wrapper classes in java. The list of eight wrapper classes are given below:

Primitive Type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer

long	Long
float	Float
double	Double

Difference between method overloading and method overriding in java

There are many differences between method overloading and method overriding in java. A list of differences between method overloading and method overriding are given below:

No.	Method Overloading	Method Overriding
1)	Method overloading is used to <i>increase the readability</i> of the program.	Method overriding is used to <i>provide the specific implementation</i> of the method that is already provided by its super class.
2)	Method overloading is performed <i>within class</i> .	Method overriding occurs <i>in two classes</i> that have IS-A (inheritance) relationship.
3)	In case of method overloading, <i>parameter must be different</i> .	In case of method overriding, <i>parameter must be same</i> .
4)	Method overloading is the example of <i>compile time polymorphism</i> .	Method overriding is the example of <i>run time polymorphism</i> .
5)	In java, method overloading can't be performed by changing return type of the method only. <i>Return type can be same or different</i> in method overloading. But you must have to change the parameter.	<i>Return type must be same or covariant</i> in method overriding.

Java Inner Classes

Java inner class or nested class is a class which is declared inside the class or interface.

We use inner classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Additionally, it can access all the members of outer class including private data members and methods.

Syntax of Inner class

1. **class** Java_Outer_class{
2. //code
3. **class** Java_Inner_class{
4. //code
5. }
6. }

Advantage of java inner classes

There are basically three advantages of inner classes in java. They are as follows:

- 1) Nested classes represent a special type of relationship that is **it can access all the members (data members and methods) of outer class** including private.
- 2) Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.
- 3) **Code Optimization:** It requires less code to write.

Chap: 8 Java String

Concept:

In java, string is basically an object that represents sequence of char values. An array of characters works same as java string. For example:

1. **char[]** ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=**new** String(ch);

is same as:

1. String s="javatpoint";

Java String class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

What is String in java

Generally, string is a sequence of characters. But in java, string is an object that represents a sequence of characters. The java.lang.String class is used to create string object.

How to create String object?

There are two ways to create String object:

1. By string literal
2. By new keyword

1) String Literal

Java String literal is created by using double quotes. For Example:

1. String s="welcome";

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

1. String s1="Welcome";
2. String s2="Welcome";//will not create new instance

Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

Java Programming by Sharma Sir

No.	Method	Description
1	<u>char charAt(int index)</u>	returns char value for the particular index
2	<u>int length()</u>	returns string length
3	<u>static String format(String format, Object... args)</u>	returns formatted string
4	<u>static String format(Locale l, String format, Object... args)</u>	returns formatted string with given locale
5	<u>String substring(int beginIndex)</u>	returns substring for given begin index
6	<u>String substring(int beginIndex, int endIndex)</u>	returns substring for given begin index and end index
7	<u>boolean contains(CharSequence s)</u>	returns true or false after matching the sequence of char value
8	<u>static String join(CharSequence delimiter, CharSequence... elements)</u>	returns a joined string
9	<u>static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)</u>	returns a joined string
10	<u>boolean equals(Object another)</u>	checks the equality of string with object
11	<u>boolean isEmpty()</u>	checks if string is empty
12	<u>String concat(String str)</u>	concatinates specified string
13	<u>String replace(char old, char new)</u>	replaces all occurrences of specified char value

Java Programming by Sharma Sir

14	<u>String replace(CharSequence old, CharSequence new)</u>	replaces all occurrences of specified CharSequence
15	<u>static String equalsIgnoreCase(String another)</u>	compares another string. It doesn't check case.
16	<u>String[] split(String regex)</u>	returns splitted string matching regex
17	<u>String[] split(String regex, int limit)</u>	returns splitted string matching regex and limit
18	<u>String intern()</u>	returns interned string
19	<u>int indexOf(int ch)</u>	returns specified char value index
20	<u>int indexOf(int ch, int fromIndex)</u>	returns specified char value index starting with given index
21	<u>int indexOf(String substring)</u>	returns specified substring index
22	<u>int indexOf(String substring, int fromIndex)</u>	returns specified substring index starting with given index
23	<u>String toLowerCase()</u>	returns string in lowercase.
24	<u>String toLowerCase(Locale l)</u>	returns string in lowercase using specified locale.
25	<u>String toUpperCase()</u>	returns string in uppercase.
26	<u>String toUpperCase(Locale l)</u>	returns string in uppercase using specified locale.
27	<u>String trim()</u>	removes beginning and ending spaces of this string.
28	<u>static String valueOf(int value)</u>	converts given type into string. It is overloaded.

Chap: 9 Exception Handling in Java

The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

In this page, we will learn about java exception, its type and the difference between checked and unchecked exceptions.

What is exception

Dictionary Meaning: Exception is an abnormal condition.

In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

What is exception handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IO, SQL, Remote etc.

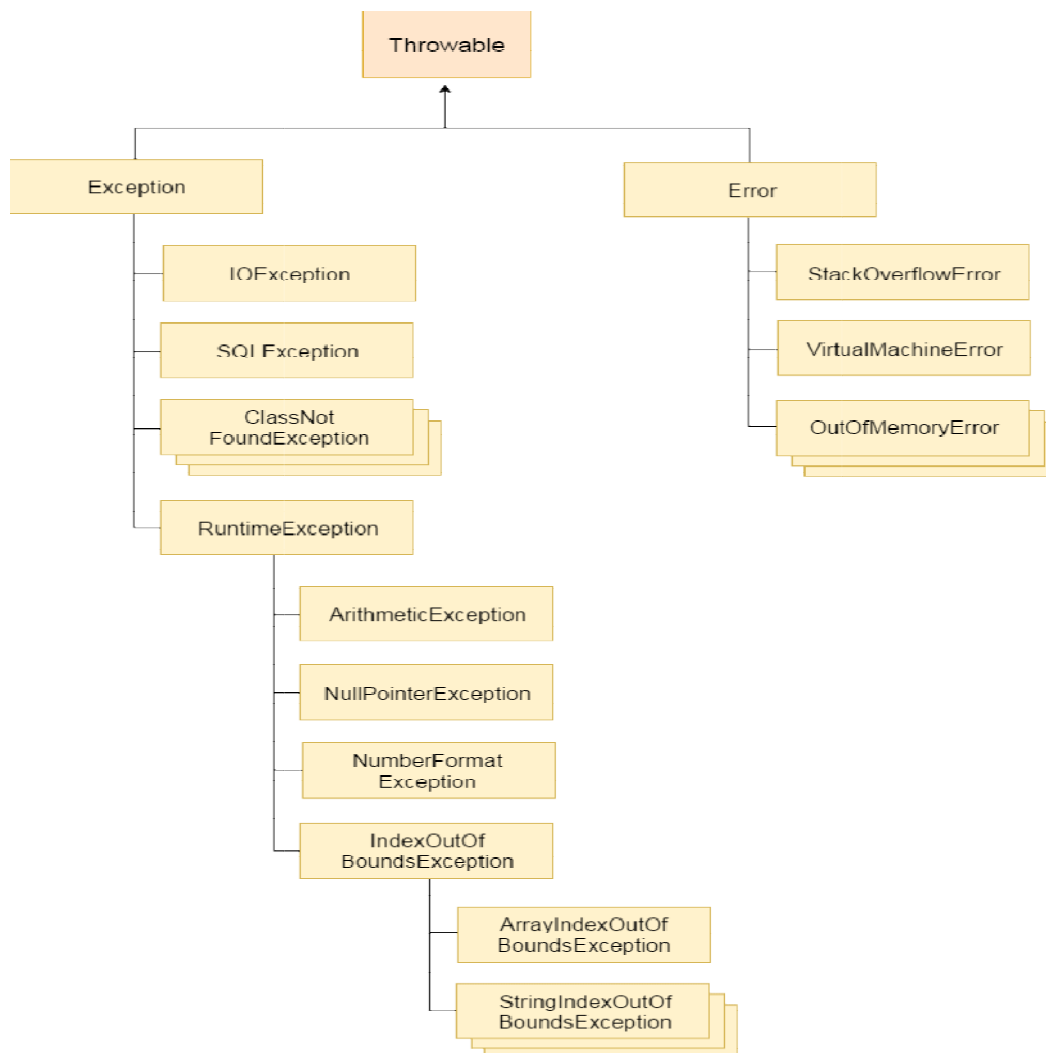
Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

Suppose there is 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java

Hierarchy of Java Exception classes



Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

Difference between checked and unchecked exceptions

1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Common scenarios where exceptions may occur

There are given some scenarios where unchecked exceptions can occur. They are as follows:

1) Scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

1. `int a=50/0;//ArithmeticException`

2) Scenario where NullPointerException occurs

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

1. `String s=null;`
2. `System.out.println(s.length());//NullPointerException`

3) Scenario where NumberFormatException occurs

The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

1. `String s="abc";`
2. `int i=Integer.parseInt(s);//NumberFormatException`

4) Scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

1. `int a[]=new int[5];`
2. `a[10]=50; //ArrayIndexOutOfBoundsException`

Java Exception Handling Keywords

There are 5 keywords used in java exception handling.

1. try
2. catch
3. finally
4. throw
5. throws

Java try block

Java try block is used to enclose the code that might throw an exception. It must be used within the method.

Java try block must be followed by either catch or finally block.

Syntax of java try-catch

1. **try**{
2. //code that may throw exception
3. }**catch**(Exception_class_Name ref){}

Syntax of try-finally block

1. **try**{
2. //code that may throw exception
3. }**finally**{}

Java Multi catch block

If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block.

Let's see a simple example of java multi-catch block.

1. **public class** TestMultipleCatchBlock{
2. **public static void** main(String args[]){
3. **try**{
4. **int** a[]=**new int**[5];
5. a[5]=30/0;
6. }
7. **catch**(ArithmeticException e){System.out.println("task1 is completed");}
8. **catch**(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
9. **catch**(Exception e){System.out.println("common task completed");}
10. }
11. System.out.println("rest of the code...");
12. }
13. }

Java Nested try block

The try block within a try block is known as nested try block in java.

Why use nested try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

Syntax:

```
1. ....
2. try
3. {
4.     statement 1;
5.     statement 2;
6.     try
7.     {
8.         statement 1;
9.         statement 2;
10.    }
11. catch(Exception e)
12. {
13. }
14.}
15.catch(Exception e)
16.{
17.}
18.....
```

Java finally block

Java finally block is a block that is used to *execute important code* such as closing connection, stream etc.

Java finally block is always executed whether exception is handled or not.

Java throw keyword

The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or unchecked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

The syntax of java throw keyword is given below.

```
1. throw exception;
```

Let's see the example of throw IOException.

```
1. throw new IOException("sorry device error);
```

Chap: 10 Java AWT

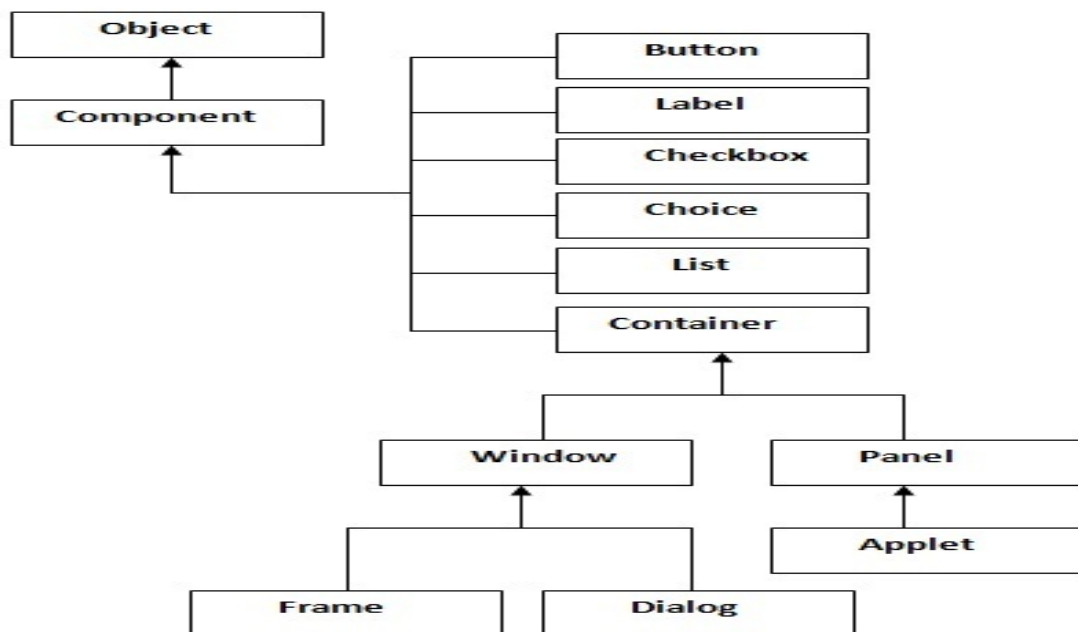
Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Java Programming by Sharma Sir

Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Event and Listener (Java Event Handling)

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

Java Event classes and Listener interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener

Java Programming by Sharma Sir

WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
 - `public void addActionListener(ActionListener a){}`
- **MenuItem**
 - `public void addActionListener(ActionListener a){}`
- **TextField**
 - `public void addActionListener(ActionListener a){}`
 - `public void addTextListener(TextListener a){}`
- **TextArea**
 - `public void addTextListener(TextListener a){}`
- **Checkbox**
 - `public void addItemListener(ItemListener a){}`
- **Choice**
 - `public void addItemListener(ItemListener a){}`
- **List**
 - `public void addActionListener(ActionListener a){}`
 - `public void addItemListener(ItemListener a){}`

Java Event Handling Code

We can put the event handling code into one of the following places:

1. Within class
2. Other class
3. Anonymous class

Chap:11 Java Swing

Java Swing is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

No.	Java AWT	Java Swing
1)	AWT components are platform-dependent .	Java swing components are platform-independent .
2)	AWT components are heavyweight .	Swing components are lightweight .
3)	AWT doesn't support pluggable look and feel .	Swing supports pluggable look and feel .
4)	AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT doesn't follows MVC (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC .

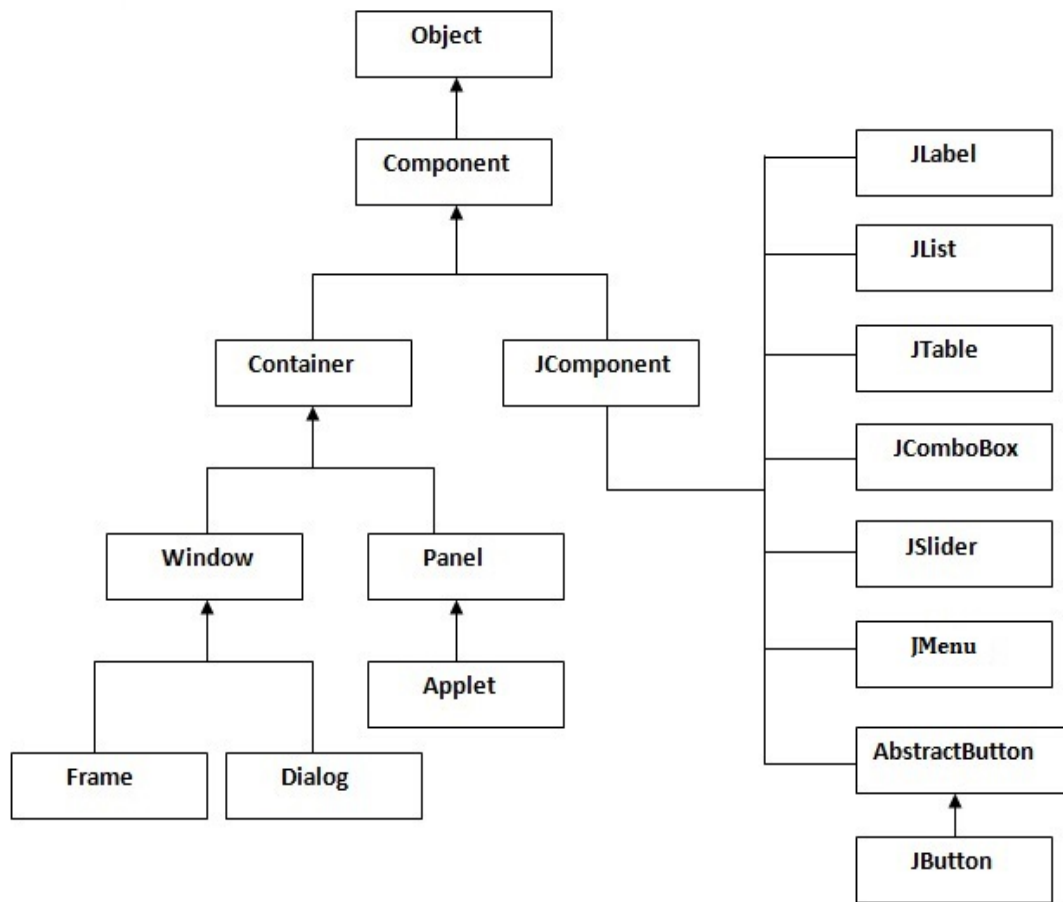
What is JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.

Java Programming by Sharma Sir



Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Chap: 12 LayoutManagers:

Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout

Java BorderLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Java GridLayout

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

Java FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

Fields of FlowLayout class

1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

Java CardLayout

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout class

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

Commonly used methods of CardLayout class

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

Java GridBagLayout

The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.

The components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of constraints object we arrange component's display area on the grid. The GridBagLayout manages each component's minimum and preferred sizes in order to determine component's size.

Fields

Modifier and Type	Field	Description
double[]	columnWeights	It is used to hold the overrides to the column weights.
int[]	columnWidths	It is used to hold the overrides to the column minimum width.
protected Hashtable<Component,GridBagConstraints>	comptable	It is used to maintains the association between a component and its gridbag constraints.
protected GridBagConstraints	defaultConstraints	It is used to hold a gridbag constraints instance containing the default values.
protected GridBagLayoutInfo	layoutInfo	It is used to hold the layout information for the gridbag.
protected static int	MAXGRIDSIZE	No longer in use just for backward compatibility
protected static int	MINSIZE	It is smallest grid that can be laid out by the grid bag layout.
protected static int	PREFERREDSIZE	It is preferred grid size that can be laid out by the grid bag layout.

Java Programming by Sharma Sir

int[]	rowHeights	It is used to hold the overrides to the row minimum heights.
double[]	rowWeights	It is used to hold the overrides to the row weights.

Useful Methods

Modifier and Type	Method	Description
void	addLayoutComponent(Component comp, Object constraints)	It adds specified component to the layout, using the specified constraints object.
void	addLayoutComponent(String name, Component comp)	It has no effect, since this layout manager does not use a per-component string.
protected void	adjustForGravity(GridBagConstraints constraints, Rectangle r)	It adjusts the x, y, width, and height fields to the correct values depending on the constraint geometry and pads.
protected void	AdjustForGravity(GridBagConstraints constraints, Rectangle r)	This method is for backwards compatibility only
protected void	arrangeGrid(Container parent)	Lays out the grid.
protected void	ArrangeGrid(Container parent)	This method is obsolete and supplied for backwards compatibility
GridBagConstraints	getConstraints(Component comp)	It is for getting the constraints for the specified component.
float	getLayoutAlignmentX(Container parent)	It returns the alignment along the x axis.

Java Programming by Sharma Sir

float	getLayoutAlignmentY(Container parent)	It returns the alignment along the y axis.
int[][]	getLayoutDimensions()	It determines column widths and row heights for the layout grid.
protected GridBagLayoutInfo	getLayoutInfo(Container parent, int sizeflag)	This method is obsolete and supplied for backwards compatibility.
protected GridBagLayoutInfo	GetLayoutInfo(Container parent, int sizeflag)	This method is obsolete and supplied for backwards compatibility.
Point	getLayoutOrigin()	It determines the origin of the layout area, in the graphics coordinate space of the target container.
double[][]	getLayoutWeights()	It determines the weights of the layout grid's columns and rows.
protected Dimension	getMinSize(Container parent, GridBagLayoutInfo info)	It figures out the minimum size of the master based on the information from getLayoutInfo.
protected Dimension	GetMinSize(Container parent, GridBagLayoutInfo info)	This method is obsolete and supplied for backwards compatibility only

Chap: 13 Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

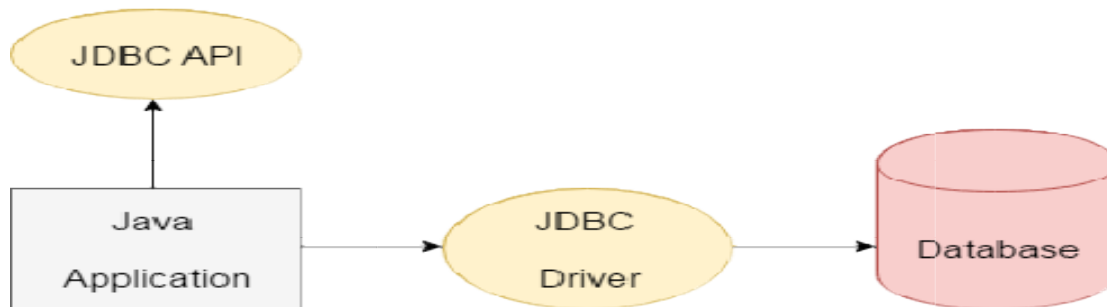
- Plugin is required at client browser to execute applet.

Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Chap:14 Java JDBC

Java JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.



Why use JDBC

Before JDBC, ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

JDBC Driver

1. JDBC Drivers
1. JDBC-ODBC bridge driver
2. Native-API driver
3. Network Protocol driver
4. Thin driver

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

1) JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

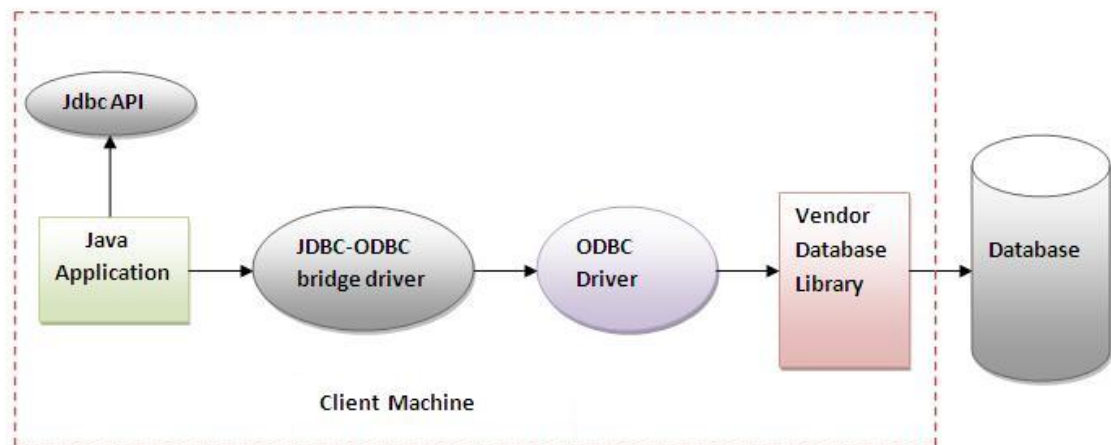


Figure- JDBC-ODBC Bridge Driver

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

2) Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

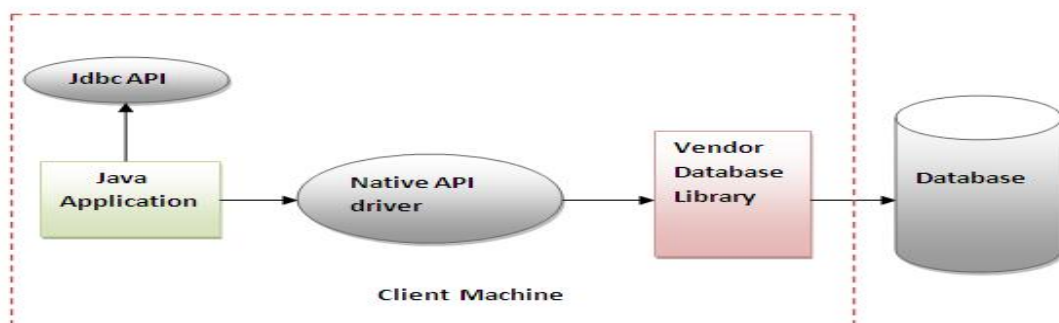


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

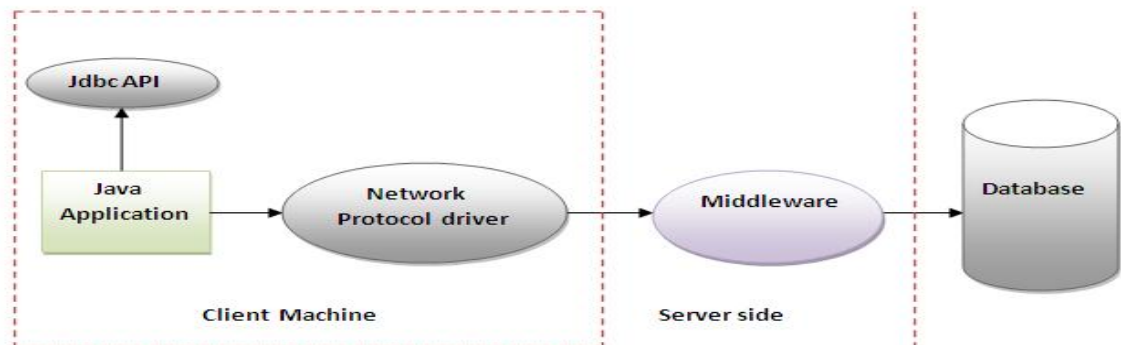


Figure- Network Protocol Driver

Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

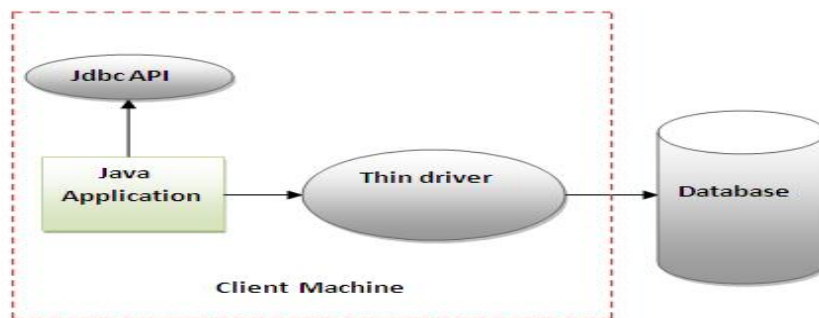


Figure- Thin Driver

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

- Drivers depends on the Database.

5 Steps to connect to the database in java

There are 5 steps to connect any java application with the database in java using JDBC. They are as follows:

1) Register the driver class

The `forName()` method of `Class` class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of `forName()` method

1. **public static void** `forName(String className)`**throws** `ClassNotFoundException`

Example to register the `OracleDriver` class

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

2) Create the connection object

The `getConnection()` method of `DriverManager` class is used to establish connection with the database.

Syntax of `getConnection()` method

- 1) **public static** `Connection getConnection(String url)`**throws** `SQLException`
- 2) **public static** `Connection getConnection(String url,String name,String password)`
3. **throws** `SQLException`

Example to establish connection with the Oracle database

1. `Connection con=DriverManager.getConnection(`
2. `"jdbc:oracle:thin:@localhost:1521:xe","system","password");`

3) Create the Statement object

The `createStatement()` method of `Connection` interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of `createStatement()` method

1. **public** `Statement createStatement()`**throws** `SQLException`

Example to create the statement object

1. `Statement stmt=con.createStatement();`

4) Execute the query

The `executeQuery()` method of `Statement` interface is used to execute queries to the database. This method returns the object of `ResultSet` that can be used to get all the records of a table.

Syntax of `executeQuery()` method

1. **public** `ResultSet executeQuery(String sql)`**throws** `SQLException`

Example to execute query

1. `ResultSet rs=stmt.executeQuery("select * from emp");`
- 2.
3. `while(rs.next()){`
4. `System.out.println(rs.getInt(1)+" "+rs.getString(2));`
5. `}`

5) Close the connection object

By closing connection object statement and `ResultSet` will be closed automatically. The `close()` method of `Connection` interface is used to close the connection.

Syntax of `close()` method

1. **public void** `close()`**throws** `SQLException`

Example to close connection

1. `con.close();`