**INTRODUCTION TO JAVA PROGRAMMING**

**LECTURE NOTES B.TECH II YEAR – II SEM (R18)**

**(2019-2023)**

### DEPARTMENT OF INFORMATION TECHNOLOGY

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **Unit** | **Topic** | **Page no** |
| 1 | I | **OOP Concepts:-** Data abstraction, encapsulation inheritance | 1 |
| 2 | I | Benefits of Inheritance | 2 |
| 3 | I | Polymorphism, classes and objects | 2 |
| 4 | I | Procedural and object oriented programming paradigms | 3 |
| 5 | I | **Java Programming-** History of Java | 4 |
| 6 | I | Comments, Data types, Variables, Constants | 5-9 |
| 7 | I | Scope and Lifetime of variables | 10 |
| 8 | I | Operators, Operator Hierarchy, Expressions | 11-12 |
| 9 | I | Type conversion and casting, Enumerated types | 12-13 |
| 10 | I | Control flow- block scope, conditional statements, loops, break and continue statements | 13-14 |
| 11 | I | Simple java stand alone programs, arrays | 14-18 |
| 12 | I | Console input and output, formatting output | 18-19 |
| 13 | I | Constructors, methods, parameter passing | 19-20 |
| 14 | I | Static fields and methods, access control, this reference, | 21-30 |
| 15 | I | Overloading methods and constructors, recursion, garbage collection, | 30-34 |
| 16 | I | Building strings, exploring string class. | 34-36 |

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **Unit** | **Topic** | **Page no** |
| 17 | II | **Inheritance –** Inheritance hierarchies super and sub classes, Member access rules | 37-40 |
| 18 | II | super keyword, preventing inheritance: final classes and methods, the Object class and its methods. | 40-41 |
| 19 | II | **Polymorphism – dynamic** binding, method overriding, | 41-42 |
| 20 | II | abstract classes and methods**.** | 43 |
| 21 | II | **Interfaces-** Interfaces Vs Abstract classes, defining an interface, implement interfaces | 43-44 |
| 22 | II | Accessing implementations through interface references, extendinginterface. | 45 |
| 23 | II | **Inner classes-** Uses of inner classes, local inner classes | 45-46 |
| 24 | II | Anonymous inner classes, static inner classes, examples. | 46 |
| 25 | II | **Packages-** Defining, creating and accessing a package, | 46-47 |
| 26 | II | Understanding CLASSPATH, importing packages. | 47 |
| 27 | III | **Exception handling-** Dealing with errors, benefits of exception handling | 48 |
| 28 | III | The classification of exceptions- exception hierarchy, checked exceptions and unchecked exceptions | 48-50 |
| 29 | III | Usage of try, catch, throw, throws and finally, | 50-54 |
| 30 | III | Rethrowing exceptions, exception specification, | 54 |
| 31 | III | Built in exceptions, creating own exception sub classes. | 54 |
| 32 | III | **Multithreading –** Differences between multiple processes and multiple threads, thread states | 55-56 |
| 33 | III | Creating threads, interrupting threads, thread priorities, synchronizing threads | 56-59 |
| 34 | III | Inter-thread communication, producer consumer pattern | 59 |
| 35 | III | Exploring java.net and java.text. | 60 |

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **Unit** | **Topic** | **Page no** |
| 36 | IV | **Collection Framework in Java** – Introduction to java collections, Overview of java collection framework, Generics | 62 |
| 37 | IV | Commonly used collection classes- Array List, Vector, Hash table, Stack, Enumeration, Iterator | 63-71 |
| 38 | IV | String Tokenizer, Random, Scanner, Calendar and Properties. | 71-76 |
| 39 | IV | **Files-** Streams**-** Byte streams, Character streams, Text input/output, Binary input/output | 77-82 |
| 40 | IV | Random access file operations, File management using File class. | 83-84 |
| 41 | IV | **Connecting to Database –** JDBC Type 1 to 4 drivers, Connecting to a a database, | 85-88 |
| 42 | IV | Querying a database and processing the results, updating data with JDBC. | 89-94 |
| 43 | V | **GUI Programming with Java-** The AWT class hierarchy,  Introduction to Swing, Swing Vs AWT, Hierarchy for Swing components | 95-100 |
| 44 | V | Containers – Jframe, JApplet, JDialog, JPanel | 100-104 |
| 45 | V | Overview of some Swing components – Jbutton, JLabel, JTextField, JTextArea, simple Swing applications, | 104-108 |
| 46 | V | Layout management – Layout manager types – border, grid and flow | 109-111 |
| 47 | V | **Event Handling-** Events, Event sources, Event classes, Event Listeners, | 111-112 |
| 48 | V | Relationship between Event sources and Listeners, Delegation event model, | 112-113 |
| 49 | V | Handling a button click, Handling Mouse events, Adapter classes. | 114-116 |
| 50 | V | **Applets –** Inheritance hierarchy for applets | 118-119 |
| 51 | V | Differences between applets and applications, Life cycle of an applet, | 120 |
| 52 | V | Passing parameters to applets, applet security issues. | 121 |

# OOP Concepts

Object Oriented Programming is a paradigm that provides many concepts such as

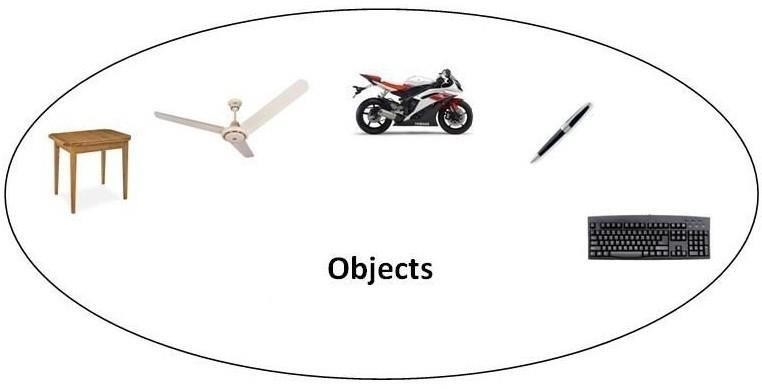
inheritance, data binding, polymorphism etc.

**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 someconcepts:

* 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 meaw, dog barks woof etc.**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.

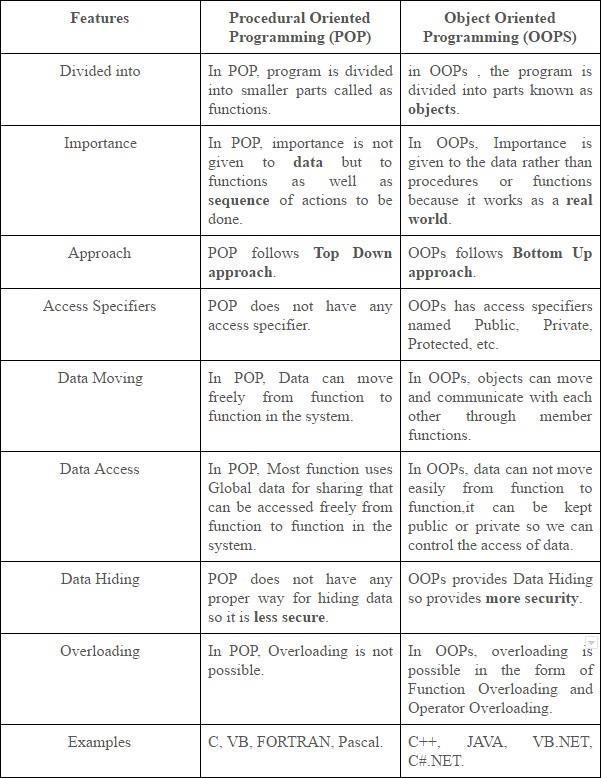
### Encapsulation

**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.

### Benefits of Inheritance

* One of the key benefits of inheritance is to minimize the amount of duplicate code in an application by sharing common code amongst several subclasses. Where equivalent code exists in two related classes, the hierarchy can usually be refactored to move the common code up to a mutual superclass. This also tends to result in a better organization of code and smaller, simpler compilationunits.
  + Inheritance can also make application code more flexible to change because classesthat inherit from a common superclass can be used interchangeably. If the return type of a method issuperclass
  + **Reusability** - facility to use public methods of base class without rewriting thesame.
  + **Extensibility** - extending the base class logic as per business logic of the derivedclass.
  + **Data hiding** - base class can decide to keep some data private so that it cannotbe



altered by the derived class

**Procedural and object oriented programming paradigms**

# Java Programming- History of Java

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, televisionsetc.

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.

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- topboxes.
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.**

### Java Version History

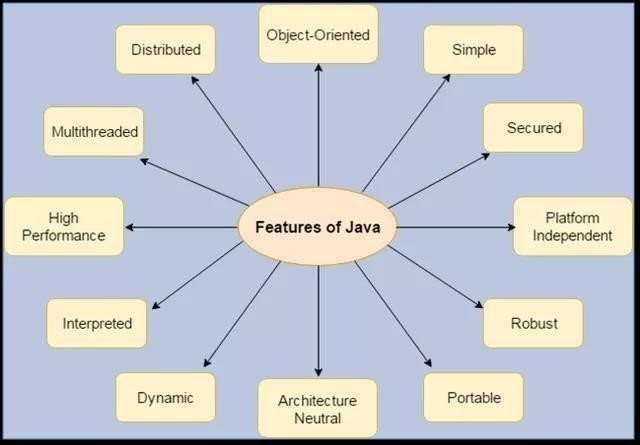
There are many java versions that has been released. Current stable release of Java is Java SE 8.

* 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) 7. J2SE 5.0 (30th Sep,2004)

1. Java SE 6 (11th Dec,2006)
2. 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. Platformindependent
5. Secured
6. Robust
7. Architectureneutral
8. Dynamic
9. Interpreted
10. HighPerformance
11. Multithreaded
12. Distributed

### 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 LineComment
  2. Multi LineComment
  3. DocumentationComment

### Java Single Line Comment

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

Syntax:

1. //This is single line comment

Example:

**public class** CommentExample1 {

**public static void** main(String[] args) { **int** i=10;//Here, i is a variable System.out.println(i);

}

}

Output:

10

### Java Multi Line Comment

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

Syntax:

/\* This is

multi line comment

\*/

Example:

**public class** CommentExample2 {

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

/\* Let's declare and print variable in java.\*/

**int**i=10; System.out.println(i);

} }

Output:

10

### Java Documentation Comment

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

**Syntax:**

/\*\* This is

documentation comment

\*/

**Example:**

/\*\* The Calculator class provides methods to get addition and subtraction of given 2 numbers.\*/

**public class** Calculator {

/\*\* The add() method returns addition of given numbers.\*/

**public static int** add(**int** a, **int** b){**return** a+b;}

/\*\* The sub() method returns subtraction of given numbers.\*/

**public static int** sub(**int** a, **int** b){**return** a-b;}

}

Compile it by javac tool:

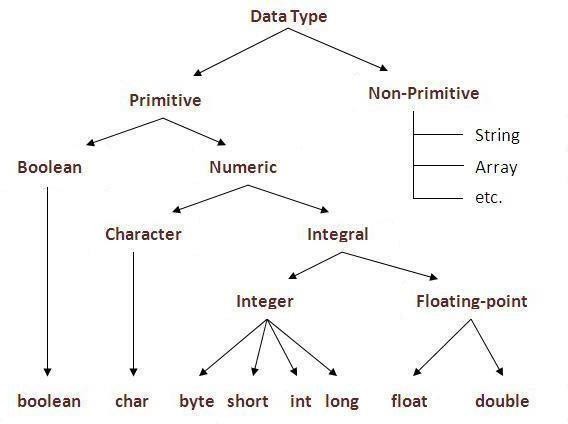
javac Calculator.java

Create Documentation API by javadoc tool:

javadoc Calculator.java

Now, there will be HTML files created for your Calculator class in the current directory. Open the HTML files and see the explanation of Calculator class provided through documentation comment.

### Data Types



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

* Primitive datatypes
* Non-primitive datatypes

|  |  |  |
| --- | --- | --- |
| **DataType DefaultValue 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 |

Java Variable Example: Add Two Numbers

**class**Simple{

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

**int**a=10; **int** b=10; **int** c=a+b;

System.out.println(c);

}}

Output:20

### 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.

### Types of Variable

There are three types of variables in java:

* localvariable
* instancevariable
* staticvariable

1. LocalVariable

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

1. Instance Variable

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

1. Staticvariable

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

**class**A{

**int**data=50;//instance variable **static int** m=100;//static variable **void** method(){

**int**n=90;//local variable

}

}//end of class

**Constants in Java**

A constant is a variable which cannot have its value changed after declaration. It uses the **'final' keyword.**

**Syntax**

modifier**final** dataType variableName = value; *//global constant*

modifier**static final** dataType variableName = value; *//constant within a c*

### Scope and Life Time of Variables

###### The scope of a variable defines the section of the code in which the variable is visible. As a general rule, variables that are defined within a block are not accessible outside that block. The lifetime of a variable refers to how long the variable exists before it isdestroyed. Destroying variables refers to deallocating the memory that was allotted to the variables when declaring it. We have written a few classes till now. You might have observed that not all variables are the same. The ones declared in the body of a method were different from those that were declared in the class itself. There are three types of variables: instance variables, formal parameters or local variables and localvariables.

**Instance variables**

###### Instance variables are those that are defined within a class itself and not in any method or constructor of the class. They are known as instance variables because every instance of the class (object) contains a copy of these variables. The scope of instance variables is determined by the access specifier that is applied to these variables. We have already seen about it earlier. The lifetime of these variables is the same as the lifetime of the object to which it belongs. Object once created do not exist for ever. They are destroyed by the garbage collector of Java when there are no more reference to that object. We shall see about Java's automatic garbage collector later on.

**Argument variables**

###### These are the variables that are defined in the header oaf constructor or a method. The scope of these variables is the method or constructor in which they are defined. The lifetime is limited to the time for which the method keeps executing. Once the method finishes execution, these variables aredestroyed.

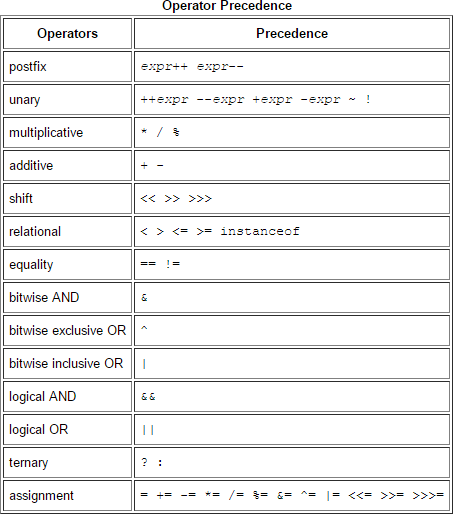
**Local variables**

###### A local variable is the one that is declared within a method or a constructor (not in the header). The scope and lifetime are limited to the methoditself.

One important distinction between these three types of variables is that access specifiers can be applied to instance variables only and not to argument or local variables.

In addition to the local variables defined in a method, we also have variables that are defined in bocks life an if block and an else block. The scope and is the same as that of the block itself.

### Operators in java

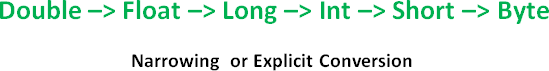


**Operator** in java is a symbol that is used to perform operations. For example: +, -, \*, / etc. There are many types of operators in java which are given below:

* UnaryOperator,
* ArithmeticOperator,
* shiftOperator,
* RelationalOperator,
* BitwiseOperator,
* LogicalOperator,
* Ternary Operatorand
* AssignmentOperator.

### Operators Hierarchy

**Expressions**



Expressions are essential building blocks of any Java program, usually created to produce a new value, although sometimes an expression simply assigns a value to a variable. Expressions are built using values, [variables,](https://www.thoughtco.com/variable-2034325) operators and method calls.

Types of Expressions

While an expression frequently produces a result, it doesn't always. There are three types of expressions in Java:

* + Those that produce a value, i.e. the result of (1 + 1)
  + Those that assign a variable, for example (v =10)
  + Those that have no result but might have a "side effect" because an expression can include a wide range of elements such as method invocations or increment operators that modify the state (i.e. memory) of aprogram.

### Java Type casting and Type conversion

Widening or Automatic Type Conversion

Widening conversion takes place when two data types are automatically converted. This happens when:

* The two data types arecompatible.
* When we assign value of a smaller data type to a bigger datatype.

For Example, in java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean. Also, char and boolean are not compatible with each other.

Narrowing or Explicit Conversion

If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing.

* This is useful for incompatible data types where automatic conversion cannot bedone.
* Here, target-type specifies the desired type to convert the specified valueto.



### Java Enum

**Enum in java** is a data type that contains fixed set of constants.

It can be used for days of the week (SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY and SATURDAY) , directions (NORTH, SOUTH, EAST and WEST)

etc. The java enum constants are static and final implicitly. It is available from JDK 1.5. Java Enums can be thought of as classes that have fixed set of constants.

Simple example of java enum

**class**EnumExample1{

**public enum** Season { WINTER, SPRING, SUMMER, FALL }

**public static void** main(String[] args) { **for** (Season s : Season.values()) System.out.println(s);

}}

**Output:**

WINTER SPRING SUMMER

FALL

Control Flow Statements

The control flow statements in Java allow you to run or skip blocks of code when special conditions are met.

The “if” Statement

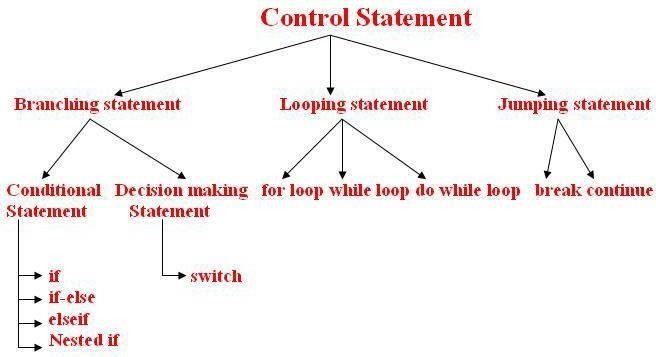
The “if” statement in Java works exactly like in most programming languages. With the help of “if” you can choose to execute a specific block of code when a predefined condition is met. The structure of the “if” statement in Java looks like this:

if(condition) {

**// execute this code**

}

The condition is Boolean. Boolean means it may be true or false. For example you may put a mathematical equation as condition. Look at this full example:



# Creating a Stand-Alone Java Application

1. Write a main method that runs your program. You can write this method anywhere. In this example, I'll write my main method in a class called Main that has no other methods. **For example:**
2. public class Main 3. {

4. public static void main(String[] args) 5. {

6. Game.play();

7. } }

1. Make sure your code is compiled, and that you have tested it thoroughly.
2. If you're using Windows, you will need to set your path to include Java, if you haven't done so already. This is a delicate operation. Open Explorer, and look inside C:\ProgramFiles\Java, and you should see some version of the JDK. Open this folder, and then open the bin folder. Select the complete path from the top of the Explorer window, and press Ctrl-C to copyit.

Next, find the "My Computer" icon (on your Start menu or desktop), right-click it, and select properties. Click on the Advanced tab, and then click on the Environment variables button. Look at the variables listed for all users, and click on the Path variable. Do not delete the contents of this variable! Instead, edit the contents by moving the cursor to the right end, entering a semicolon (;), and pressing Ctrl-V to paste the path you copied earlier. Then go ahead and save your changes. (If you have any Cmd windows open, you will need to close them.)

1. If you're using Windows, go to the Start menu and type "cmd" to run a program that brings up a command prompt window. If you're using a Mac or Linux machine, run the Terminal program to bring up a commandprompt.
2. In Windows, type dir at the command prompt to list the contents of the currentdirectory. On a Mac or Linux machine, type ls to dothis.
3. Now we want to change to the directory/folder that contains your compiled code. Look at the listing of sub-directories within this directory, and identify which one contains yourcode. Type cd followed by the name of that directory, to change to that directory. For example, to change to a directory called Desktop, you would type:

cd Desktop

To change to the parent directory, type:

cd ..

Every time you change to a new directory, list the contents of that directory to see where to go next. Continue listing and changing directories until you reach the directory that contains

your .class files.

1. If you compiled your program using Java 1.6, but plan to run it on a Mac, you'll needto recompile your code from the command line, bytyping:

javac -target 1.5 \*.java

1. Now we'll create a single JAR file containing all of the files needed to run yourprogram.

**Arrays**

Java provides a data structure, the **array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the sametype.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

This tutorial introduces how to declare array variables, create arrays, and process arrays using indexed variables.

Declaring Array Variables:

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

dataType[] arrayRefVar; // preferred way. or

dataType arrayRefVar[]; // works but not preferred way.

**Note:** The style**dataType[] arrayRefVar** is preferred. The style **dataType arrayRefVar[]**

comes from the C/C++ language and was adopted in Java to accommodate C/C++programmers.

Example:

The following code snippets are examples of this syntax:

double[] myList; or doublemyList[];

Creating Arrays:

// preferred way.

// works but not preferred way.

You can create an array by using the new operator with the following syntax:

arrayRefVar= new dataType[arraySize];

The above statement does two things:

* + It creates an array using newdataType[arraySize];
  + It assigns the reference of the newly created array to the variablearrayRefVar.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:

dataType[] arrayRefVar = new dataType[arraySize];

Alternatively you can create arrays as follows:

dataType[] arrayRefVar = {value0, value1, ..., valuek};

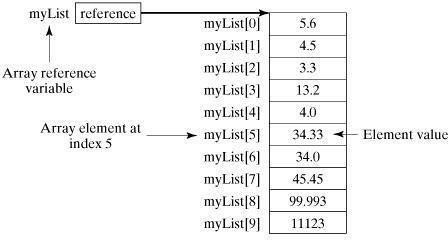
The array elements are accessed through the **index**. Array indices are 0-based; that is, they start from 0 to **arrayRefVar.length-1**.

Example:

Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference tomyList:

double[] myList = new double[10];

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.



Processing Arrays:

When processing array elements, we often use either for loop or for each loop because all of the elements in an array are of the same type and the size of the array is known.

Example:

Here is a complete example of showing how to create, initialize and process arrays:

public class TestArray

{

public static void main(String[] args) { double[] myList = {1.9, 2.9, 3.4, 3.5};

// Print all the array elements for (int i = 0; i <myList.length; i++){

System.out.println(myList[i] + " ");

}

// Summing all elements double total = 0;

for(int i = 0; i <myList.length; i++) { total += myList[i];

}

System.out.println("Total is " + total);

// Finding the largest element double max = myList[0];

for(int i = 1; i <myList.length; i++) { if (myList[i] >max) max = myList[i];

}

System.out.println("Max is " + max);

}

}

This would produce the following result:

1.9

2.9

3.4

3.5

Total is 11.7

Max is 3.5

public class TestArray {

public static void main(String[] args) { double[] myList = {1.9, 2.9, 3.4, 3.5};

// Print all the array elements for(double element: myList) {

System.out.println(element);

}}}

### Java Console Class

The Java Console class is be used to get input from console. It provides methods to read texts and passwords.

If you read password using Console class, it will not be displayed to the user.

The java.io.Console class is attached with system console internally. The Console class is introduced since 1.5.

Let's see a simple example to read text from console.

1. Stringtext=System.console().readLine();
2. System.out.println("Text is:"+text);

**Java ConsoleExample**

**import**java.io.Console; **class**ReadStringTest{

**public static void** main(String args[]){ Console c=System.console(); System.out.println("Enter your name: "); String n=c.readLine(); System.out.println("Welcome"+n); } }

Output

Enter your name: Nakul Jain Welcome Nakul Jain

### Constructors

**Constructor in java** is a *special type of method* that is used to initialize the object.

Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.

There are basically two rules defined for the constructor.

* 1. Constructor name must be same as its classname
  2. Constructor must have no explicit returntype

Types of java constructors

There are two types of constructors:

1. Default constructor (no-argconstructor)
2. Parameterizedconstructor

Java Default Constructor

A constructor that have no parameter is known as default constructor.

Syntax of default constructor:

1. <class\_name>(){}

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

**class**Bike1{

Bike1(){System.out.println("Bike is created");}

**public static void** main(String args[]){ Bike1 b=**new** Bike1();

} }

**Output:**Bike is created

Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

**class**Student4{ **int**id; Stringname;

Student4(**int** i,String n){ id = i;

name = n;

}

**void**display(){System.out.println(id+" "+name);}

**public static void** main(String args[]){ Student4 s1 = **new** Student4(111,"Karan"); Student4 s2 = **new** Student4(222,"Aryan"); s1.display();

s2.display();

} }

Output:

111Karan 222Aryan

**Constructor Overloading in Java**

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example of Constructor Overloading

**class**Student5{ **int**id; String name;

**int**age;

Student5(**int** i,String n){ id = i;

name = n;

}

Student5(**int** i,String n,**int** a){ id = i;

name = n; age=a;

}

**void**display(){System.out.println(id+" "+name+" "+age);}

**public static void** main(String args[]){ Student5 s1 = **new** Student5(111,"Karan"); Student5 s2 = **new** Student5(222,"Aryan",25); s1.display();

s2.display();

} }

Output:

111 Karan 0

222 Aryan 25

**Java CopyConstructor**

There is no copy constructor in java. But, we can copy the values of one object to another like copy constructor inC++.

There are many ways to copy the values of one object into another in java. They are:

oBy constructor

oBy assigning the values of one object into another

oBy clone() method of Object class

In this example, we are going to copy the values of one object into another using java constructor.

**class**Student6{ **int**id; Stringname;

Student6(**int** i,String n){ id = i;

name = n;

}

Student6(Student6 s){ id = s.id;

name =s.name;

}

**void**display(){System.out.println(id+" "+name);}

**public static void** main(String args[]){ Student6 s1 = **new** Student6(111,"Karan"); Student6 s2 = **new** Student6(s1); s1.display();

s2.display();

} }

Output:

111Karan 111Karan

### Java -Methods

A Java method is a collection of statements that are grouped together to perform an operation. When you call the System.out.**println()** method, for example, the system actually executes several statements in order to display a message on the console.

Now you will learn how to create your own methods with or without return values, invoke a method with or without parameters, and apply method abstraction in the program design.

Creating Method

Considering the following example to explain the syntax of a method −

Syntax

public static int methodName(int a, int b) {

// body

}

Here,

* **public static** −modifier
* **int** − returntype
* **methodName** − name of the method
* **a, b** − formalparameters
* **int a, int b** − list ofparameters

Method definition consists of a method header and a method body. The same is shown in the following syntax −

Syntax

modifier returnType nameOfMethod (Parameter List) {

// method body

}

The syntax shown above includes −

* **modifier**− It defines the access type of the method and it is optional touse.
* **returnType**− Method may return avalue.
* **nameOfMethod**− This is the method name. The method signature consists of themethod name and the parameter list.
* **Parameter List** − The list of parameters, it is the type, order, and number of parameters of a method. These are optional, method may contain zeroparameters.
* **method body** − The method body defines what the method does with thestatements.

### Call by Value and Call by Reference in Java

There is only call by value in java, not call by reference. If we call a method passing a value, it is known as call by value. The changes being done in the called method, is not affected in the calling method.

Example of call by value in java

In case of call by value original value is not changed. Let's take a simple example:

**class**Operation{ **int**data=50; **void**change(**int** data){

data=data+100;//changes will be in the local variable only

}

**public static void** main(String args[]){ Operation op=**new** Operation(); System.out.println("before change "+op.data); op.change(500);

System.out.println("after change "+op.data);

}

}

Output:before change 50

after change 50

In Java, parameters are always passed by value. For example, following program prints i = 10, j = 20.

// Test.java class Test {

// swap() doesn't swap i and j

public static void swap(Integer i, Integer j) { Integer temp = new Integer(i);

i = j;

j = temp;

}

public static void main(String[] args) { Integer i = new Integer(10);

Integer j = new Integer(20); swap(i, j);

System.out.println("i = " + i + ", j = " + j);

}

}

### Static Fields and Methods

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 classvariable)
  2. method (also known as classmethod)
  3. block
  4. nestedclass

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 studentsetc.
* The static variable gets memory only once in class area at the time of classloading.

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. Stringname;
4. String college="ITS"; 5.}

Example of static variable

//Program of static variable

**class**Student8{ **int**rollno;

String name;

**static**String college ="ITS"; Student8(**int** r,String n){ rollno =r;

name =n;

}

**void**display (){System.out.println(rollno+" "+name+" "+college);}

**public static void** main(String args[]){ Student8 s1 = **new** Student8(111,"Karan"); Student8 s2 = **new** Student8(222,"Aryan");

s1.display();

s2.display();

} }

**Output**:111 KaranITS

222 AryanITS

### Java static method

If you apply static keyword with any method, it is known as static method.

* + A static method belongs to the class rather than object of aclass.
  + A static method can be invoked without the need for creating an instance of aclass.
  + static method can access static data member and can change the value ofit.

Example of static method

//Program of changing the common property of all objects(static field).

**class**Student9{ **int** rollno; String name;

**static**String college = "ITS"; **static void** change(){ college = "BBDIT";

}

Student9(**int** r, String n){ rollno =r;

name =n;

}

**void**display (){System.out.println(rollno+" "+name+" "+college);}

**public static void** main(String args[]){ Student9.change();

Student9 s1 = **new** Student9 (111,"Karan"); Student9 s2 = **new** Student9 (222,"Aryan"); Student9 s3 = **new** Student9 (333,"Sonoo"); s1.display();

s2.display();

s3.display();

} }

Output:111 Karan BBDIT 222 Aryan BBDIT

333 Sonoo BBDIT

**Java static block**

* + Is used to initialize the static datamember.
  + It is executed before main method at the time of classloading.

Example of static block class A2{

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

} }

**Output:** static block is invoked Hello main

### Access Control

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

private access modifier

The private access modifier is accessible only within class.

Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

**class**A{

**private int** data=40;

**private void** msg(){System.out.println("Hello java");}}

**public class** Simple{

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

System.out.println(obj.data);//Compile Time Error obj.msg();//Compile Time Error

} }

1. default accessmodifier

If you don't use any modifier, it is treated as **default** bydefault. The default modifier is accessible only withinpackage.

Example of default accessmodifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside thepackage.

//save by A.java **package** pack; **class** A{

**void**msg(){System.out.println("Hello");}

}

//save by B.java **package** mypack; **import** pack.\*;

**class**B{

**public static void** main(String args[]){ A obj = **new** A();//Compile Time Error obj.msg();//Compile Time Error }}

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside thepackage.

1. protected accessmodifier

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Example of protected access modifier

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only throughinheritance.

//save by A.java **package** pack; **public class** A{

**protected void** msg(){System.out.println("Hello");} }

//save by B.java **package** mypack; **import** pack.\*; **class** B **extends** A{

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

obj.msg();

} }

Output:Hello

1. public accessmodifier

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

Example of public access modifier

//save by A.java **package** pack; **public class** A{

**public void** msg(){System.out.println("Hello");}}

//save by B.java **package** mypack; **import** pack.\*; **class** B{

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

obj.msg();

} }

Output:Hello

Understanding all java access modifiers

Let's understand the access modifiers by a simple table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **within class** | **within package** | **outsidepackageby subclassonly** | **outside package** |
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

### this keyword in java

Usage of java this keyword

Here is given the 6 usage of java this keyword.

* 1. this can be used to refer current class instancevariable.
  2. this can be used to invoke current class method(implicitly)
  3. this() can be used to invoke current classconstructor.

|  |  |  |
| --- | --- | --- |
| 4. this can be passed as an argument in the method call. | | |
| 5. this can be passed as argument in the constructorcall. | | |
| 6. this can be used to return the current class instance from themethod. | | |
| **class** Student{ | | |
| **int** rollno; | | |
| String name; | | |
| **float** fee; | | |
| Student(**int** rollno,String name,**float** fee){ | | |
| **this**.rollno=rollno; | | |
| **this**.name=name; | | |
| **this**.fee=fee; | | |
| } | | |
| **void** display(){System.out.println(rollno+" "+name+" "+fee);} | | |
| } | | |
| **class** TestThis2{ | | |
| **public static void** main(String args[]){ | | |
| Student s1=**new** Student(111,"ankit",5000f); | | |
| Student s2=**new**Student(112,"sumit",6000f); | | |
| s1.display(); | | |
| s2.display(); | | |
| }} | | |
| Output: | | |
| 111 ankit 5000 | | |
| 112 sumit 6000 | | |
| **Difference between constructor and method in java** | | |
| **Java Constructor** | **Java Method** |  |
| Constructor is used to initialize the state of an object. | Method is used to expose behaviour of anobject. | |
| Constructor must not have return type. | Method must have return type. | |
| Constructor is invoked implicitly. | Method is invoked explicitly. | |
| The java compiler provides a default constructor if you don't have any constructor. | Method is not provided by compiler in any case. | |
| Constructor name must be same as the class name. | Method name may or may not be | |
| JAVA PROGRAMMING |  | Page 30 |

same as class name. There are many differences between constructors and methods. They are given belo

### Constructor Overloading in Java

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

### Example of Constructor Overloading

***class*** *Student5{* **int** id; String name;

**int**age;

Student5(**int** i,String n){ id = i;

name = n;

}

Student5(**int** i,String n,**int** a){ id = i;

name = n; age=a;

}

**void** display(){System.out.println(id+" "+name+" "+age);}

**public static void** main(String args[]){ Student5 s1 = **new** Student5(111,"Karan"); Student5 s2 = **new**Student5(222,"Aryan",25); s1.display();

s2.display();

}

}

**Output:**

111 Karan 0

222 Aryan 25

### Method Overloading in java

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

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

### Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

**class**Adder{

**static int** add(**int** a,**int** b){**return** a+b;}

**static int** add(**int** a,**int** b,**int** c){**return** a+b+c;}

}

**class**TestOverloading1{

**public static void** main(String[] args){ System.out.println(Adder.add(11,11)); System.out.println(Adder.add(11,11,11));

}}

**Output:**

22

33

### Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

### Recursion in Java

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

Java Recursion Example 1: Factorial Number

**public class** RecursionExample3 {

**static int** factorial(**int** n){

**if**(n == 1)

**return** 1;

else

**return**(n \* factorial(n-1));

} }

**public static void** main(String[] args) { System.out.println("Factorial of 5 is: "+factorial(5));

} }

**Output:**

Factorial of 5 is: 120

### Java Garbage Collection

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

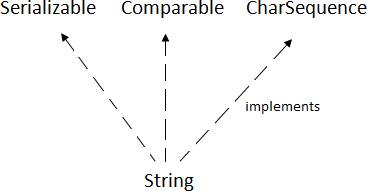
To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

* It makes java **memory efficient** because garbage collector removes theunreferenced objects from heapmemory.
* It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extraefforts.

### gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.



**public static void** gc(){}

Simple Example of garbage collection in java public class TestGarbage1{

**public void** finalize(){System.out.println("object is garbage collected");}

**public static void** main(String args[]){ TestGarbage1 s1=**new** TestGarbage1(); TestGarbage1 s2=**new** TestGarbage1(); s1=**null**;

s2=**null**; System.gc();

} }

object is garbage collected object is garbage collected

# Java String

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

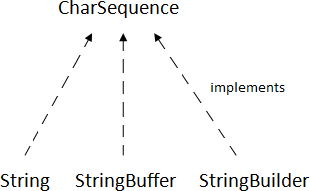
1. **char**[] ch={'j','a','v','a','t','p','o','i','n','t'};

2. String s=**new** String(ch); ssame as:

1. Strings="javatpoint";
2. **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.
3. The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence*interfaces.

CharSequence Interface

The CharSequence interface is used to represent sequence of characters. It is implemented by String, StringBuffer and StringBuilder classes. It means, we can create string in java by using these 3 classes.



The java String is immutable i.e. it cannot be changed. Whenever we change any string, a new instance is created. For mutable string, you can use StringBuffer and StringBuilder classes.

There are two ways to create String object:

1. By stringliteral
2. By newkeyword

#### 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. Strings1="Welcome";
2. String s2="Welcome";//will not create newinstance

By new keyword

1. String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new string object in normal (non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in heap (non pool).

Java String Example

**public class** StringExample{

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

String s1="java";//creating string by java string literal

**char**ch[]={'s','t','r','i','n','g','s'};

String s2=**new** String(ch);//converting char array to string

String s3=**new** String("example");//creating java string by new keyword System.out.println(s1);

System.out.println(s2); System.out.println(s3);

}}

java

strings example

Immutable String in Java

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. Let's try to understand the immutability concept by the example given below:

**class**Testimmutablestring{

**public static void** main(String args[]){ String s="Sachin";

s.concat(" Tendulkar");//concat() method appends the string at the end System.out.println(s);//will print Sachin because strings are immutable objects

}}

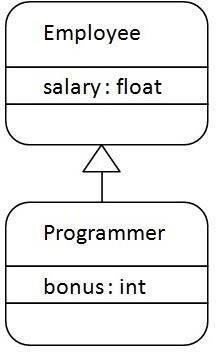
Output:Sachin

***class****Testimmutablestring1{*

***public static void*** *main(String args[]){ String s="Sachin";*

*s=s.concat(" Tendulkar"); System.out.println(s);*

*} }Output:Sachin Tendulkar*



# Inheritance in Java

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

Syntax of Java Inheritance

1. **class**Subclass-name **extends** Superclass-name 2. {

3. //methods and fields4. }

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.

**class**Employee{ **float**salary=40000;

}

**class**Programmer **extends** Employee{

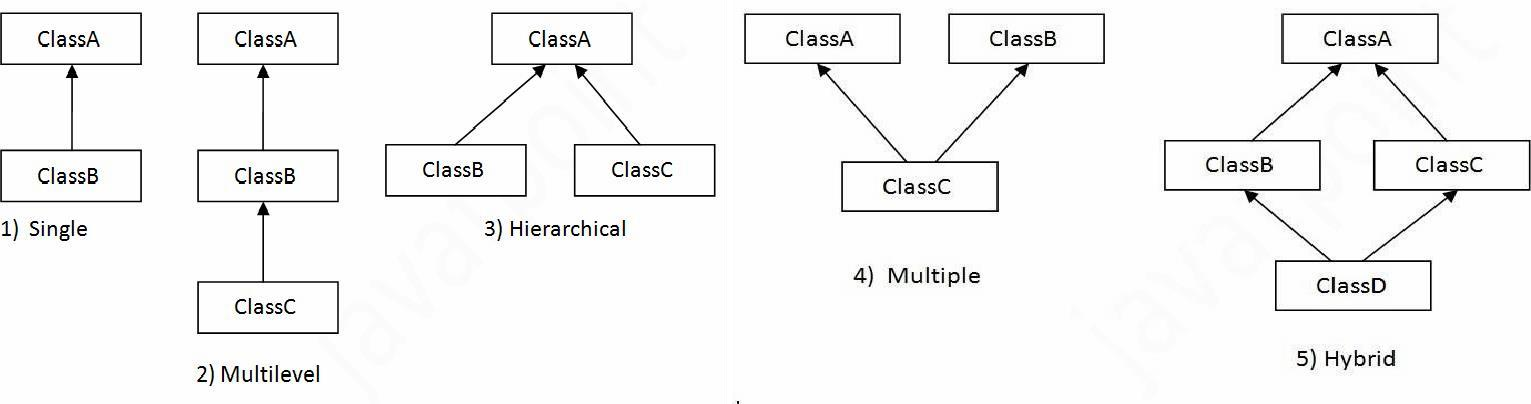
**int**bonus=10000;

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

System.out.println("Programmer salary is:"+p.salary); System.out.println("Bonus of Programmer is:"+p.bonus);

} }Programmer salary is:40000.0

Bonus of programmeris:10000



**Types of inheritance in java**

**Single Inheritance Example**

*File: TestInheritance.java*

**class**Animal{ **void**eat(){System.out.println("eating...");}

}

**class**Dog **extends** Animal{

**void**bark(){System.out.println("barking...");}

}

**class**TestInheritance{

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

d.bark();

d.eat();

}}

Output:

barking... eating...

### Multilevel Inheritance Example

*File: TestInheritance2.java*

**class**Animal{ **void**eat(){System.out.println("eating...");}

}

**class**Dog **extends** Animal{

**void**bark(){System.out.println("barking...");}

}

**class**BabyDog **extends** Dog{

**void**weep(){System.out.println("weeping...");}

}

**class**TestInheritance2{

**public static void** main(String args[]){ BabyDog d=**new** BabyDog(); d.weep();

d.bark();

d.eat();

}}

Output:

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

### Hierarchical Inheritance Example

*File: TestInheritance3.java*

**class**Animal{ **void**eat(){System.out.println("eating...");}

}

**class**Dog **extends** Animal{

**void**bark(){System.out.println("barking...");}

}

**class**Cat **extends** Animal{

**void**meow(){System.out.println("meowing...");}

}

**class**TestInheritance3{

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

c.meow();

c.eat();

//c.bark();//C.T.Error

}}

Output:

meowing... eating...

**Member access and Inheritance**

A subclass includes all of the members of its super class but it cannot access those members of the super class that have been declared as private. Attempt to access a private variable would cause compilation error as it causes access violation. The variables declared as private, is only accessible by other members of its own class. Subclass have no access to it.

### 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 instancevariable.
2. super can be used to invoke immediate parent classmethod.
3. super() can be used to invoke immediate parent classconstructor.

super is used to refer immediate parent class instance variable.

**class**Animal{ Stringcolor="white";

}

**class**Dog **extends** Animal{ Stringcolor="black";

**void**printColor(){ System.out.println(color);//prints color of Dogclass

System.out.println(**super**.color);//prints color of Animal class

}

}

**class**TestSuper1{

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

d.printColor();

}}

Output:

black white

# 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.

### 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, know 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 thismethod

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.

### 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**.

Usage of Java Method Overriding

* Method overriding is used to provide specific implementation of a method that is already provided by its superclass.
* Method overriding is used for runtimepolymorphism

Rules for Java Method Overriding

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

Example of method overriding Class Vehicle{

**void**run(){System.out.println("Vehicle is running");}

}

**class**Bike2 **extends** Vehicle{ **void**run(){System.out.println("Bike is running safely");} **public static void** main(String args[]){

Bike2 obj = **new** Bike2(); obj.run();

}

**Output**:Bike is running safely

1. **class**Bank{ **int**getRateOfInterest(){**return** 0;}

}

**class**SBI **extends** Bank{

**int**getRateOfInterest(){**return** 8;}

}

**class**ICICI **extends** Bank{

**int**getRateOfInterest(){**return** 7;}

}

**class**AXIS **extends** Bank{

**int**getRateOfInterest(){**return** 9;}

}

**class**Test2{

**public static void** main(String args[]){ SBI s=**new** SBI();

ICICI i=**new** ICICI(); AXIS a=**new** AXIS();

System.out.println("SBI Rate of Interest: "+s.getRateOfInterest()); System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest()); System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());

} }

Output:

SBI Rate of Interest: 8

ICICI Rate of Interest: 7 AXIS Rate of Interest: 9

### Abstract class in Java

A class that is declared with abstract keyword is known as abstract class in java. It can have abstract and non-abstract methods (method with body). It needs to be extended and its method implemented. It cannot be instantiated.

Example abstract class

1. **abstract class**A{}

abstractmethod

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

Example of abstract class that has abstract method

**abstract class** Bike{

**abstract void** run();

}

**class**Honda4 **extends** Bike{ **void**run(){System.out.println("running safely..");} **public static void** main(String args[]){

Bike obj = **new** Honda4(); obj.run();

}

1. }

runningsafely..

# Interface in Java

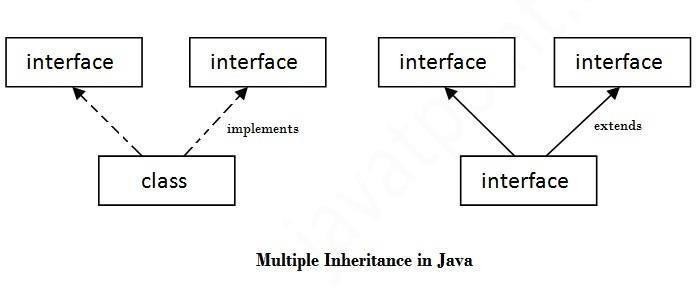
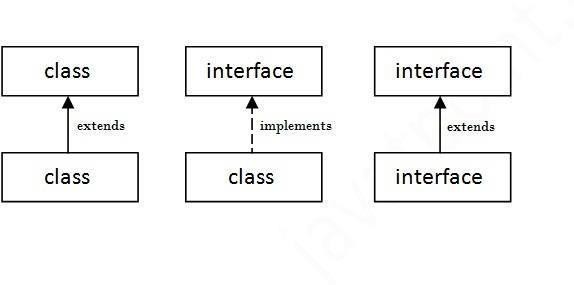
An **interface in java** is a blueprint of a class. It has static constants and abstract methods.

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.

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

* It is used to achieveabstraction.
* By interface, we can support the functionality of multipleinheritance.
* It can be used to achieve loosecoupling.



Internal addition by compiler

Understanding relationship between classes and interfaces

//Interface declaration: by first user

**interface**Drawable{ **void**draw();

}

//Implementation: by second user

**class**Rectangle **implements** Drawable{

**public void** draw(){System.out.println("drawing rectangle");}

}

**class**Circle **implements** Drawable{

**public void** draw(){System.out.println("drawing circle");}

}

//Using interface: by third user

**class**TestInterface1{

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

Drawable d=**new** Circle();//In real scenario, object is provided by method e.g. getDrawable() d.draw();

}}

Output:drawing circle

Multiple inheritance in Java by interface

**interface**Printable{

**void**print();

}

**interface**Showable{ **void**show();

}

**class**A7 **implements** Printable,Showable{

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

A7 obj = **new** A7(); obj.print();

obj.show();

} }

Output:Hello

Welcome

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Abstract class** |  |  | **Interface** |  |  |  |
| 1) Abstract class can **have abstract and non-abstract**methods. | | | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods**also. | | | |
| 2) Abstract class **doesn't support multipleinheritance**. | | | Interface **supports multiple inheritance**. | | | |
| 3) Abstract class **can have final, non- final, static and non-static variables**. | | | Interface has **only static and final variables**. | | | |
| 4) Abstract class **can provide the implementation ofinterface**. | | | Interface **can't provide the implementation of abstract class**. | | | |
| 5) The **abstract keyword** is used to declare abstract class. | | | The**interface** interface. | **keyword** is | used | to declare |
| 6) **Example:**  public abstract  public abstract  } | class void | Shape{ draw(); | **Example:** public void  } | interface |  | Drawable{  draw(); |

# 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.

Syntax of Inner class

1. **class**Java\_Outer\_class{
2. //code
3. **class**Java\_Inner\_class{
4. //code

5. } }

**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** includingprivate.
  2. Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one placeonly.
  3. **Code Optimization**: It requires less code to write.

Difference between nested class and inner class in Java

Inner class is a part of nested class. Non-static nested classes are known as inner classes.

Types of Nested classes

There are two types of nested classes non-static and static nested classes.The non-static nested classes are also known as inner classes.

* Non-static nested class (innerclass)
  + 1. Member inner class
    2. Anonymous innerclass
    3. Local inner class
* Static nestedclass

# Java Package

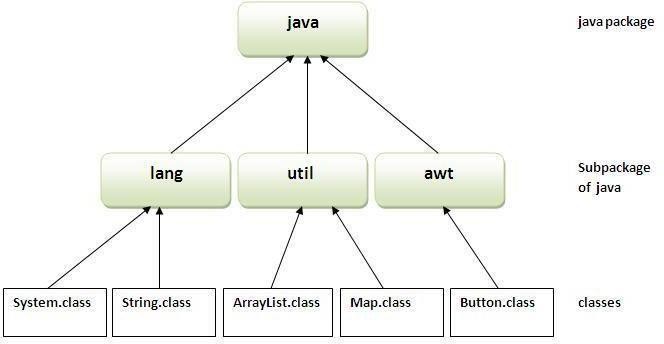
A **java package** is a group of similar types of classes, interfaces and sub-packages. Package in java can be categorized in two form, built-in package and user-defined package. There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.**Advantage of JavaPackage**

1. Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2. Java package provides accessprotection.
3. Java package removes namingcollision.

**package**mypack; **public class** Simple{

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

} }



How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below: javac -d directory javafilename

How to run java package program

**To Compile:** javac -d . Simple.java

**To Run:** java mypack.Simple

Using fully qualified name

Example of package by import fully qualified name

//save by A.java **package** pack; **public class** A{

**public void** msg(){System.out.println("Hello");} }

//save by B.java **package** mypack; **class** B{

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

pack.A obj = **new** pack.A();//using fully qualified name obj.msg();

}

}

Output:Hello

**Exception Handling**

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

### What is exception

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

### 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.

### Types of Exception

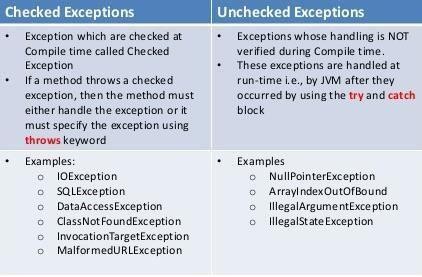
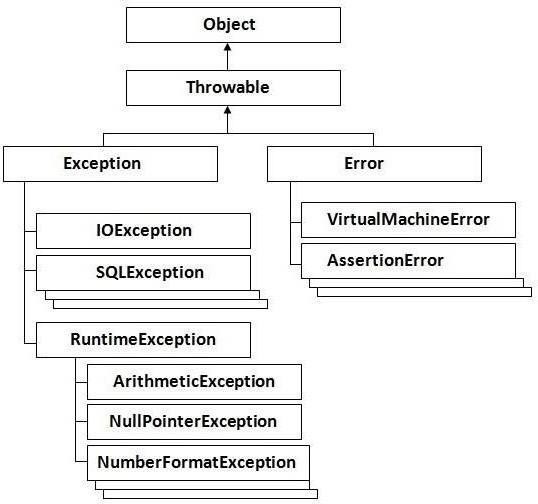
There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

* 1. CheckedException
  2. UncheckedException
  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 atcompile-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 checkedatruntime.**
3. **Error: Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionErroretc.**

### Hierarchy of Java Exception classes



**Checked and UnChecked Exceptions**

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 throwexception
3. }**catch**(Exception\_class\_Name ref){}Syntax of try-finallyblock
4. **try**{
5. //code that may throwexception
6. }**finally**{}

Java catch block

Java catch block is used to handle the Exception. It must be used after the try block only. You can use multiple catch block with a single try.

Problem without exception handling

Let's try to understand the problem if we don't use try-catch block.

**public class** Testtrycatch1{

**public static void** main(String args[]){ **int** data=50/0;//may throw exception System.out.println("rest of thecode...");

} }

Output:

Exception in thread main java.lang.ArithmeticException:/ byzero

As displayed in the above example, rest of the code is not executed (in such case, rest of the code... statement is not printed).

There can be 100 lines of code after exception. So all the code after exception will not be executed.

Solution by exception handling

Let's see the solution of above problem by java try-catch block.

**public class** Testtrycatch2{

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

**try**{

**int**data=50/0;

}**catch**(ArithmeticException e){System.out.println(e);} System.out.println("rest of the code...");

} }

1. Output:

Exception in thread main java.lang.ArithmeticException:/ by zero rest of the code...

Now, as displayed in the above example, rest of the code is executed i.e. rest of the code... statement is printed.

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(Stringargs[]){
3. **try**{
4. **int**a[]=**new int**[5]; 5. a[5]=30/0;

6. }

1. **catch**(ArithmeticException e){System.out.println("task1 iscompleted");}
2. **catch**(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed"); 9. }

10. **catch**(Exception e){System.out.println("common taskcompleted");

11. }

12. System.out.println("rest of the code..."); 13. } }

Output:task1 completed rest of the code...

Java nested try example

Let's see a simple example of java nested try block.

**class**Excep6{

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

**try**{**t ry**{

System.out.println("going to divide");

**int**b =39/0;

}**catch**(ArithmeticException e){System.out.println(e);}

**try**{

**int**a[]=**new int**[5]; a[5]=4;

}**catch**(ArrayIndexOutOfBoundsException e){System.out.println(e);} System.out.println("other statement);

}**catch**(Exception e){System.out.println("handeled");} System.out.println("normal flow..");

} 1. }

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 finally block follows try or catch block.

Usage of Java finally

Case 1

Let's see the java finally example where **exception doesn't occur**. **class**TestFinallyBlock{

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

**try**{ **int**data=25/5;

System.out.println(data);

}

**catch**(NullPointerException e){System.out.println(e);} **finally**{System.out.println("finally block is always executed");} System.out.println("rest of the code...");

}

}

Output:5

finally block is always executed rest of the code...

Java throw keyword

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

We can throw either checked or uncheked 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;

Java throw keyword example

In this example, we have created the validate method that takes integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

1. **public class** TestThrow1{

**static void** validate(**int** age){

**if**(age<18)

**throw new** ArithmeticException("not valid");

else

System.out.println("welcome to vote");

}

**public static void** main(String args[]){ validate(13);

System.out.println("rest of the code...");

}}

Output:

Exception in thread main java.lang.ArithmeticException:notvalid

Java throws keyword

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Syntax of java throws

1. return\_type method\_name() **throws**exception\_class\_name{
2. //method code3. }

4.

Java throws example

Let's see the example of java throws clause which describes that checked exceptions can be propagated by throws keyword.

**import**java.io.IOException; **class**Testthrows1{ **void**m()**throws** IOException{

**throw new** IOException("device error");//checked exception

}

**void**n()**throws** IOException{ m();

}

**void**p(){ **try**{ n();

}**catch**(Exception e){System.out.println("exception handled");}

}

**public static void** main(String args[]){ Testthrows1 obj=**new** Testthrows1(); obj.p();

System.out.println("normal flow..."); } } Output:

exception handled normal flow...

Java Custom Exception

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

By the help of custom exception, you can have your own exception and message. Let's see a simple example of java custom exception.

**class**InvalidAgeException **extends** Exception{ InvalidAgeException(String s){

**super**(s);

} }

**class**TestCustomException1{

**static void** validate(**int** age)**throws** InvalidAgeException{

**if**(age<18)

**throw new** InvalidAgeException("not valid");

else

System.out.println("welcome to vote");

}

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

**try**{ validate(13);

}**catch**(Exception m){System.out.println("Exception occured: "+m);}

System.out.println("rest of the code...");

}}

Output:Exception occured: InvalidAgeException:not valid rest of the code...