Course Content

Course Objectives

- You will be able to
  - Develop programs using Forte
  - Write simple programs using primitive data types, control statements, methods, and arrays.
  - Create and use methods
  - Write interesting projects
  - Establish a firm foundation on Java concepts

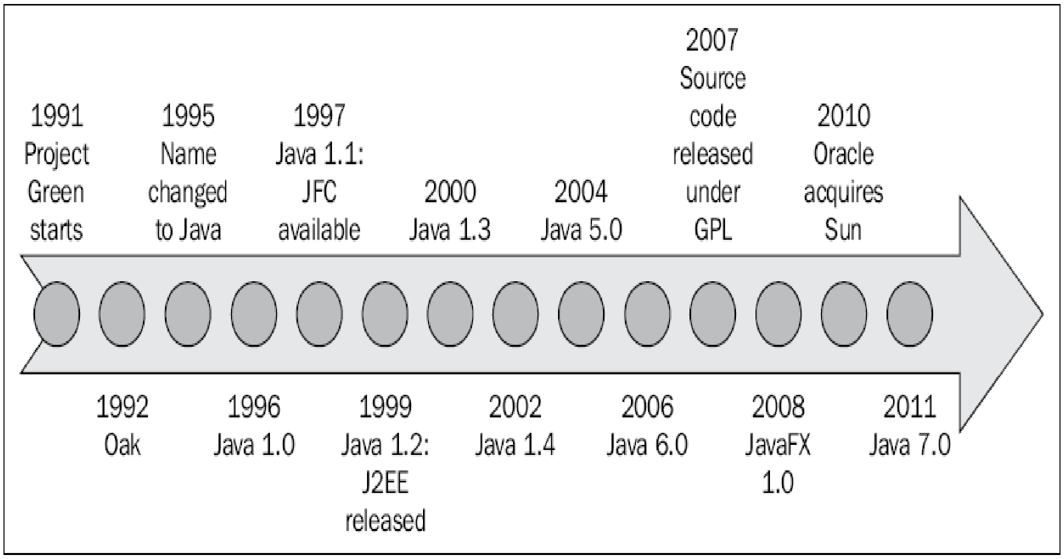
Introduction to Java Programming

- **What is JAVA .?**

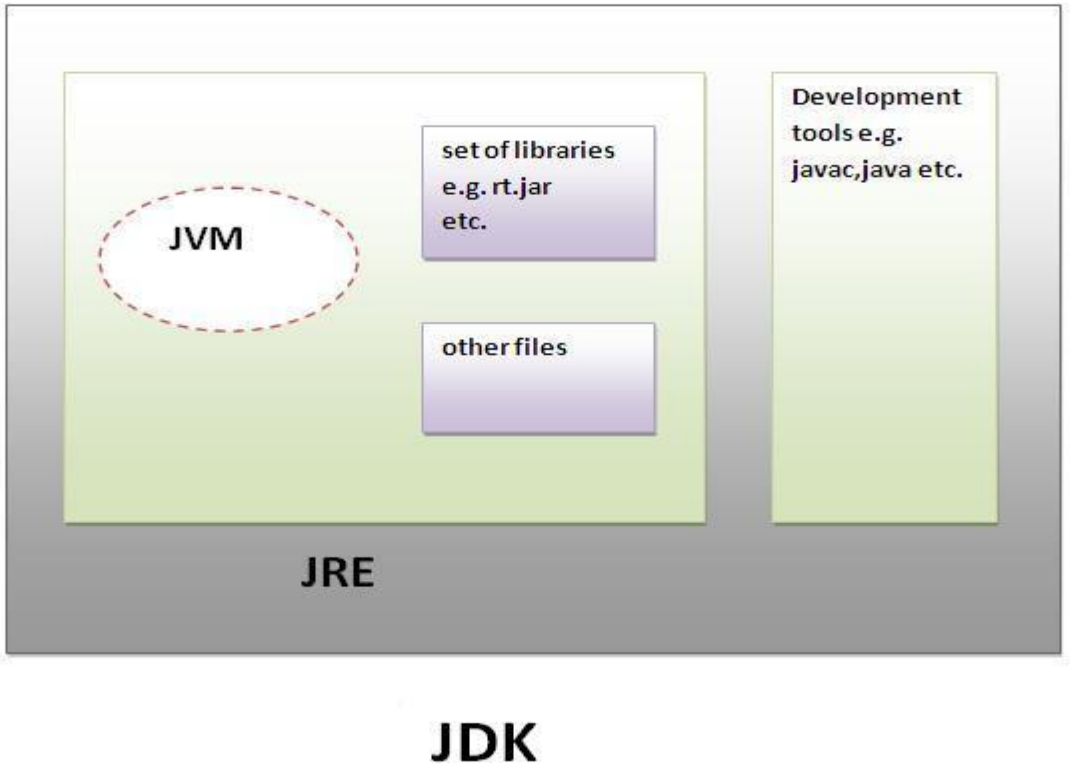
- **Where we use JAVA .?**

- **Java History**

Java is a object oriented programming language developed by Sun Microsystems of USA in 1991.Originally called Oak by **James Gosling** (one of the inventor of the language).



- **Introduction JVM, API, JRE, JDK.**


JVM
set of libraries e.g. rt.jar etc.
other files
Development tools e.g. javac,java etc.
JRE
JDK

- **Three Products of JAVA**
  - **JAVA SE – JAVA Standard Edition.**
    - Stand alone application
    - Applet
  - **JAVA EE – JAVA Enterprise Edition.**
    - eCommerce
    - eBusiness
  - **JAVA ME – JAVA Micro Edition.**
    - Cell Phone
    - PAD's
    - TV set-top box

OOP Concept and Introduction of JAVA

Fundamental of OOPS

- Polymorphism.
- Encapsulation.
- Inheritance.
- Abstract.

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.

- Simple
- Object-Oriented
- Platform independent
- Secured
- Robust
- Architecture neutral
- Portable
- Interpreted
- High Performance
- Multithreaded
- 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

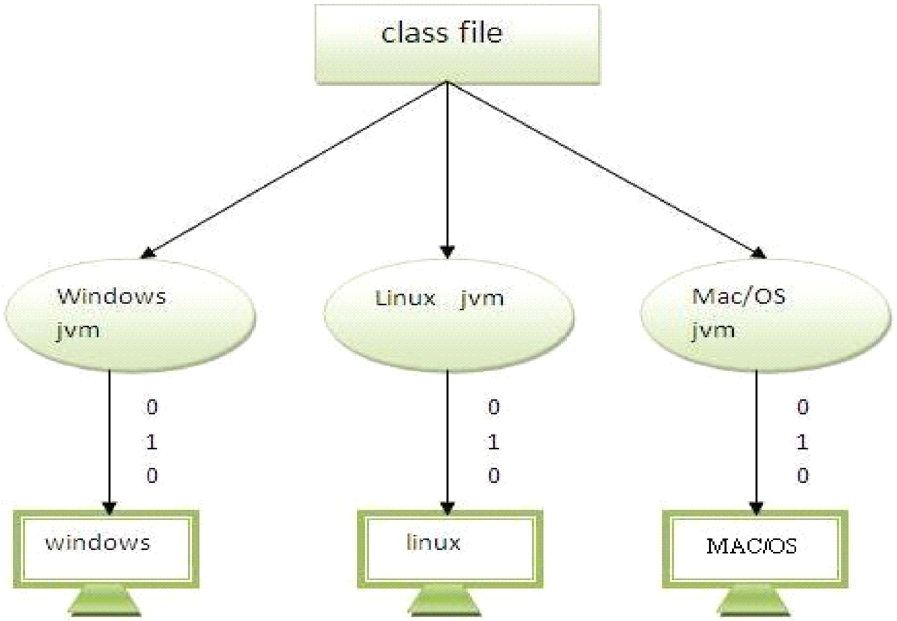
Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour.
Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules.
Basic concepts of OOPs are:
• Object

- Class
- Inheritance
- Polymorphism
- Abstraction
- 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's a software-based platform that runs on top of other hardware-based platforms.It has two components:

- Runtime Environment
- API(Application Programming Interface)

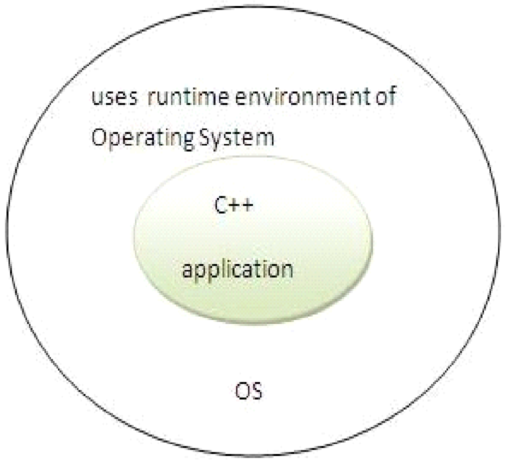


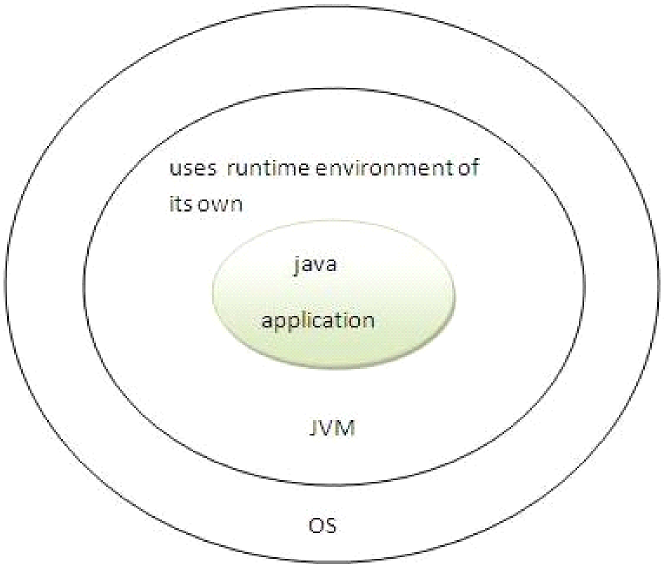
Java code can be run on multiple platforms e.g.Windows,Linux,Sun Solaris,Mac/OS etc. Java code is compiled by the compiler and converted into bytecode.This bytecode is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

Secured

Java is secured because:

- No explicit pointer
- 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

Java has been designed for writing highly reliable or robust software:
· language restrictions (e.g. no pointer arithmetic and real arrays) to make it impossible for applications to smash memory (e.g overwriting memory and corrupting data)
· Java does **automatic garbage collection**, which prevents memory leaks
· extensive compile-time checking so bugs can be found early; this is repeated at runtime for flexibility and to check consistency

Portable

We may carry the java byte code 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.

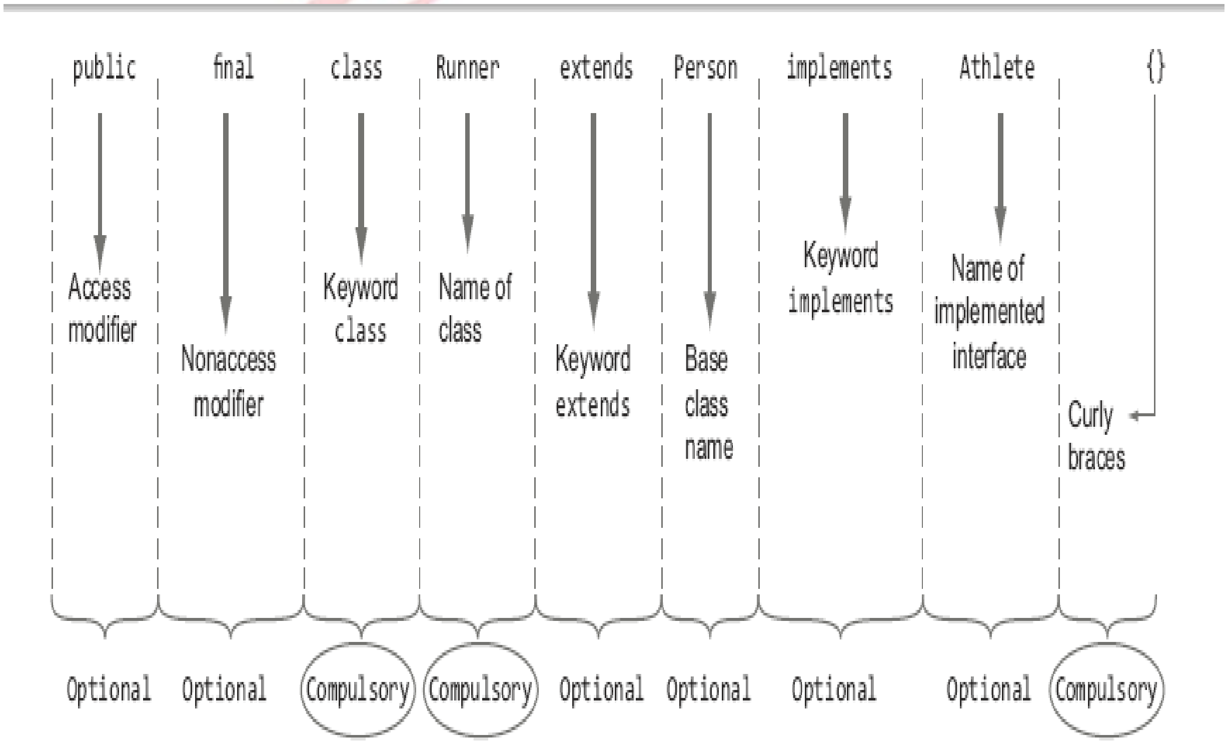
Multi-threaded

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 shares the same memory. Threads are important for multi-media, Web applications etc.

How JVM Work.

Developing First Java Program

Class Components :-



Primitive Data Types , Operations, Expressions, Casting.

Uses of Variable

- Holding unique attribute data for an instance variable.
- Assigning the value of one variable to another.
- Representing values within a mathematical expression.
- Printing value on screen.
- Holding Reference to other objects.

Variable Declaration and initialization

- Syntax for attribute variable declaration and initialization.
  [modifier] data_type identifier [=value]

- Syntax for declaring variable
  Data_type identifier;

- Syntax for initialization variable
  Identifier = value;

- Syntax for declaration and initialization
  Data_type identifier [=value];

Data Types.

Data Type

Primitive data type

Non-Primitive Data type

Numeric	**Float Point**	**Textual**	**Logical**	Array Enum
Byte :- 8bits	float :- 32 bits	char :- 16 bits	boolean	
Short :- 16 bits	* double :- 64 bits			
* int :- 32 bits				
long :- 64 bits				

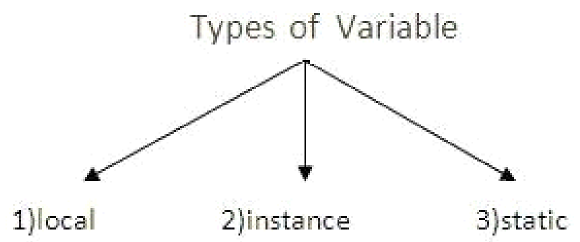
http://unicode-table.com/en/#control-character

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

Types of Variable (Scope Of Variable)

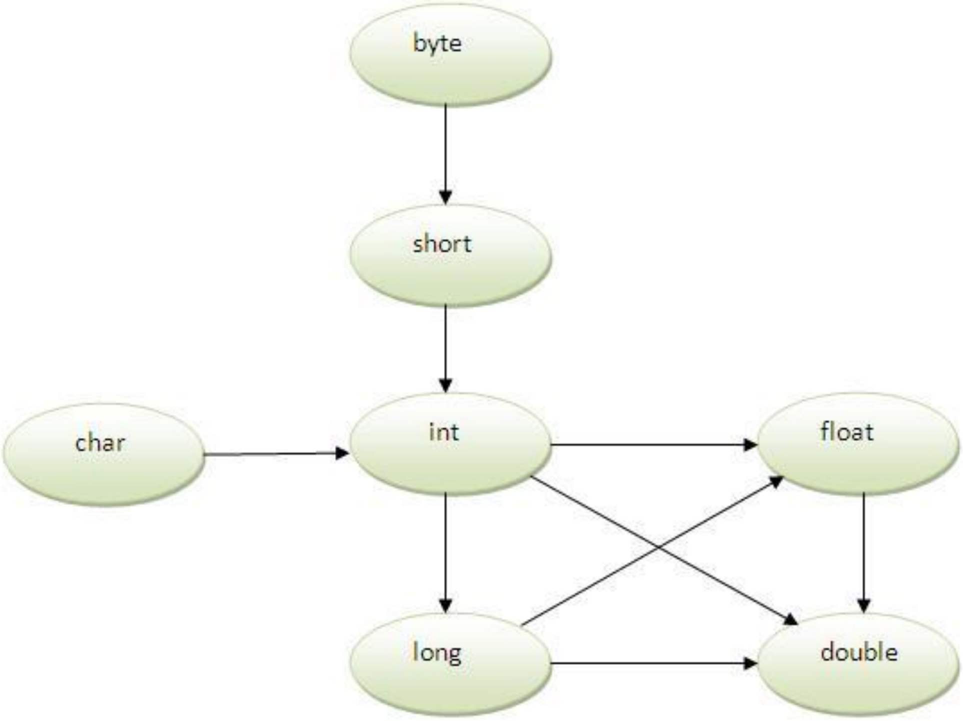
There are three types of variables in java

- local variable.
- instance variable.
- static variable.

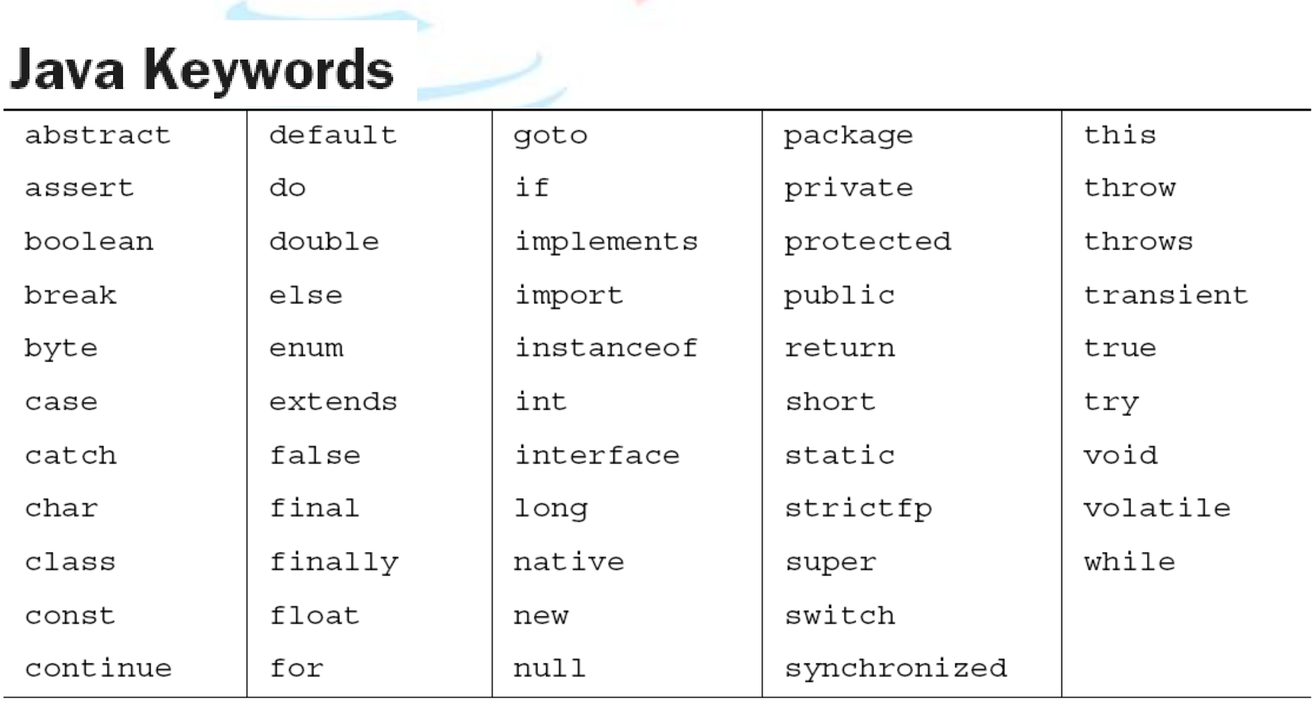


Type Casting.

- Down Casting. (implicit casting)



- Up Casting.(explicit casting)

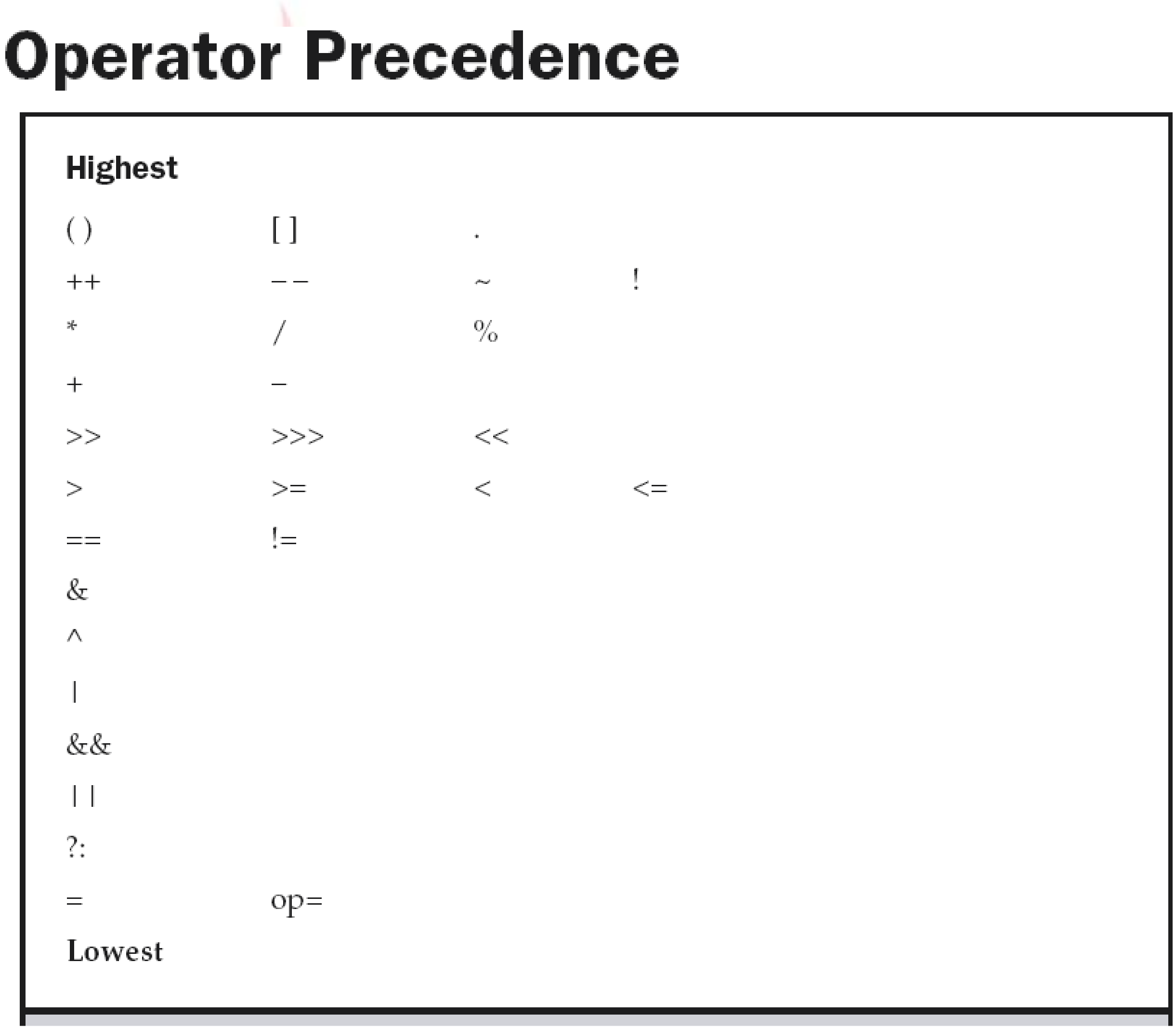


Java Keywords

abstract
assert
boolean
break
byte
case
catch
char
class
const
continue
default
do
double
else
enum
extends
false
final
finally
float
for
goto
if
implements
import
instanceof
int
interface
long
native
new
null
package
private
protected
public
return
short
static
strictfp
super
switch
synchronized
this
throw
throws
transient
true
try
void
volatile
while

Operators and expressions.

- Assignment Operator. =, +=, -=, *=, /=
- Arithmetic Operator. +, -, *, /, %
- Relational Operator. <, <=, >, >=, ==, !=
- Logical Operator. !, & , |
- Increment & Decrement operator. ++, --
- Short Circuit. && , ||
- Right Shift & Left Shift. << (number * 2^bitstoshift) ,>>



Operator Precedence

Highest			
( )	[ ]	.	
++	– –	~	!
*	/	%	
+	–		
>>	>>>	<<	
>	>=	<	<=
==	!=		
&			
^			
|			
&&			
| |			
?:			
=	op=		
Lowest			

Control Flow statement

- Create if and if-else constructs.
- Use a switch statement.
- Create and use while loops.
- Create and use for loops, including the enhanced for loop.
- Create and use do-while loops.
- Use break and continue statements.

The if and if-else constructs

- if
- if-else
- if-else-if-else

Switch Statement

Loops

For loop

Simple for loop
Nested for loop

While loop
Do while loop.
Break
Continue

Objects and Classes

Methods

Identifying Object

Object have its own state and behavior which is define inside class.

- Object has Attribute.
- Object has Operations.
- Object can be physical or conceptual.
- Independent Existence.

Class Structure

- Package statement.
- Import Statement.
- Comment.
- Class Definition.
  [modifier]   class   class_identifier { }
- Variable declaration and Assignments.
- Constructor.
- Methods/ operations.
  [modifier] return_type method_identifier (argument list) { }

```
public class class_Identifier{
        State / field / Data / Variable;
        behavior / methods / Function
}
```

Modeling Class

ClassName
AttributeVariableName [range] ……………
metodName (parameters) …………….

Creating Object

- Declare Object Reference Variable.
  className Identifier;
- Instantiating an Object.
  New className();
- Initializing object reference variable.

Use object reference variable to manipulate data

Use object reference variable to call the method.

Storing object reference in memory.(Stack and Heap memory allocation).

Assigning one reference variable to another reference variable.

Memory Allocation Twist

```
public class Test{
        Test t;
        int val;
        public Test(int val){
                this.val=val;
        }
        public Test(int val,Test t){
                this.val=val;
                this.t=t;
        }
        public static void main(String a[]){
                Test t1=new Test(1);
                Test t2=new Test(2,t1);
                Test t3=new Test(3,t1);
                Test t4=new Test(4,t2);
                t2.t=t3;
                t3.t=t4;
                t1.t=t2.t;
                t2.t=t4.t;
                System.out.println(t1.t.val);
                System.out.println(t2.t.val);
                System.out.println(t3.t.val);
                System.out.println(t4.t.val);
        }
                }
```

Encapsulation

1. Encapsulated Code is more **flexible** and easy to change with new requirements
2. Allows you to control **who can access what**. (!!!)
3. Helps to write **`immutable class`** in Java
4. It allows you to change **one part** of code without affecting other part of code.

Constructor

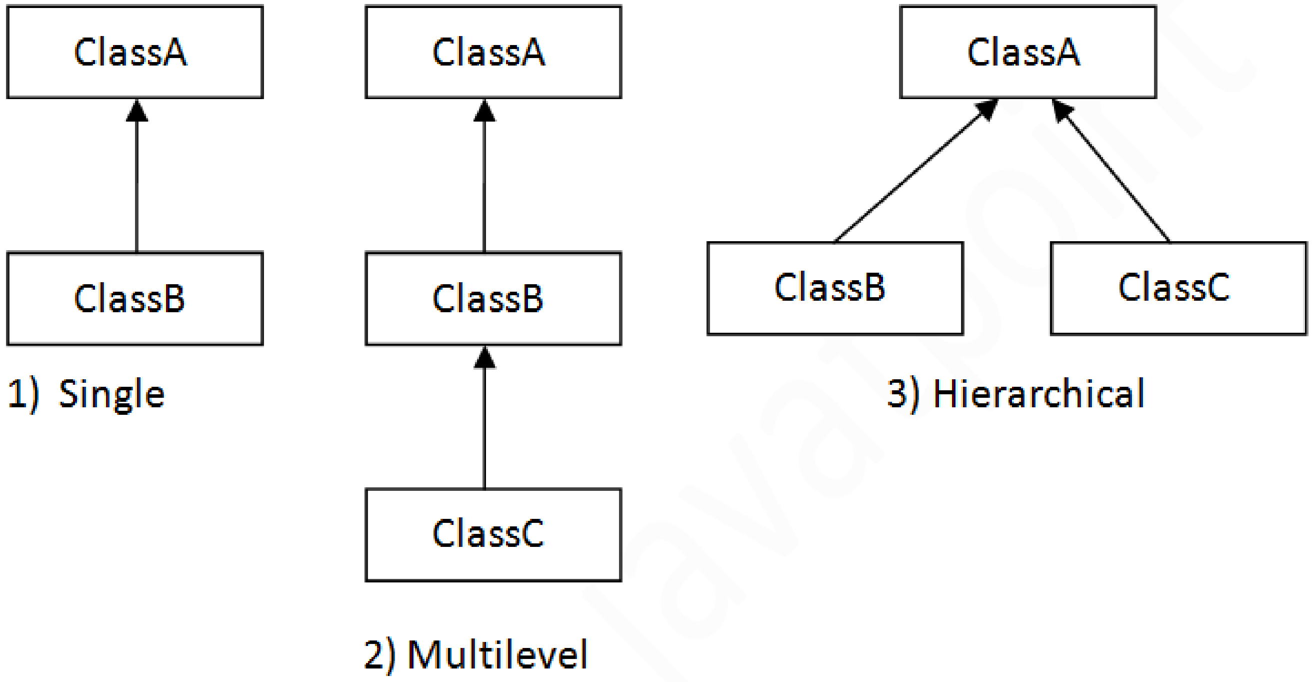
Default Constructor
parameterized constructor.

Inheritance

Types of Inheritance

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

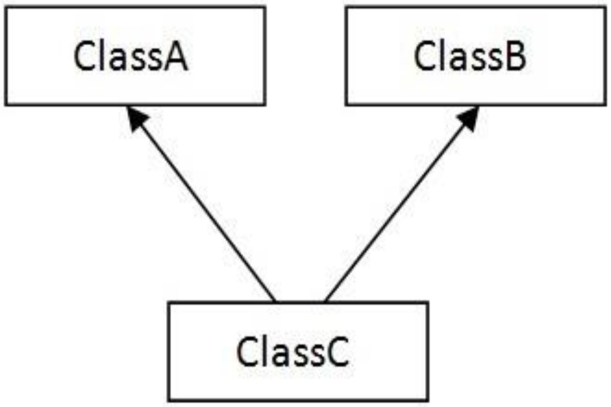
In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



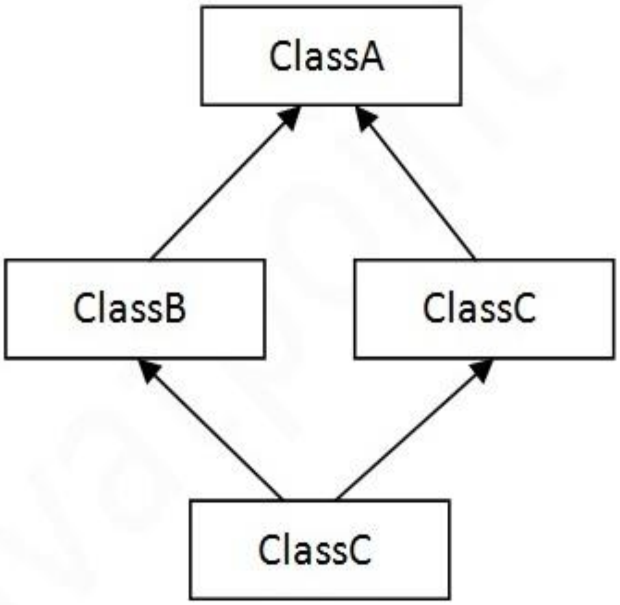
1) Single

2) Multilevel

3) Hierarchical



4) Multiple



5) Hybrid

Access Control

Public , private , protected , default.

Overloading

Overloading Rules :-

- Overloaded methods MUST change the argument list.
- Overloaded methods CAN change the return type.
- Overloaded methods CAN change the access modifier.
- Overloaded methods CAN declare new or broader checked exceptions.

Constructor Overloading.
Method overloading.
Super keyword.
This keyword
Var-args

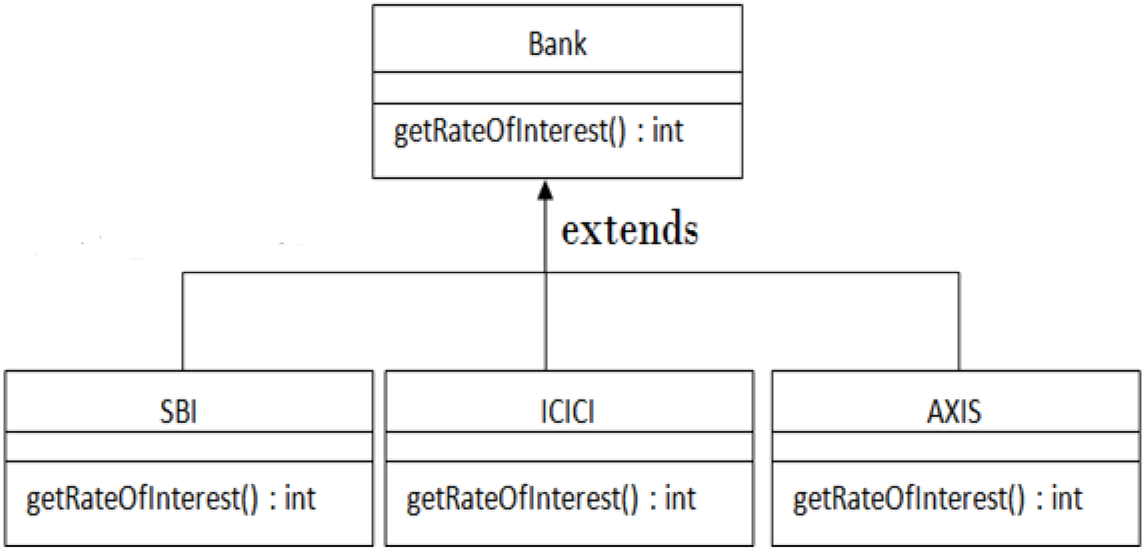
Overriding

Runtime polymorphism / Compile time polymorphism.
Dynamic polymorphism / Static polymorphism.
Late binding / early binding.

Overriding Rules :-

- The argument list must exactly match that of the overridden method.
- The return type must be the same as, or a subtype of, the return type declared in the original overridden method in the superclass.
- The access level can't be more restrictive than the overridden method's.
- The access level CAN be less restrictive than that of the overridden method.
- Instance methods can be overridden only if they are inherited by the subclass. The overriding method must NOT throw checked exceptions that are new or broader than those declared by the overridden method.
- You cannot override a method marked final.
- You cannot override a method marked static.

Example.


Bank
getRateOfInterest() : int
extends
SBI
getRateOfInterest() : int
ICICI
getRateOfInterest() : int
AXIS
getRateOfInterest() : int

Test

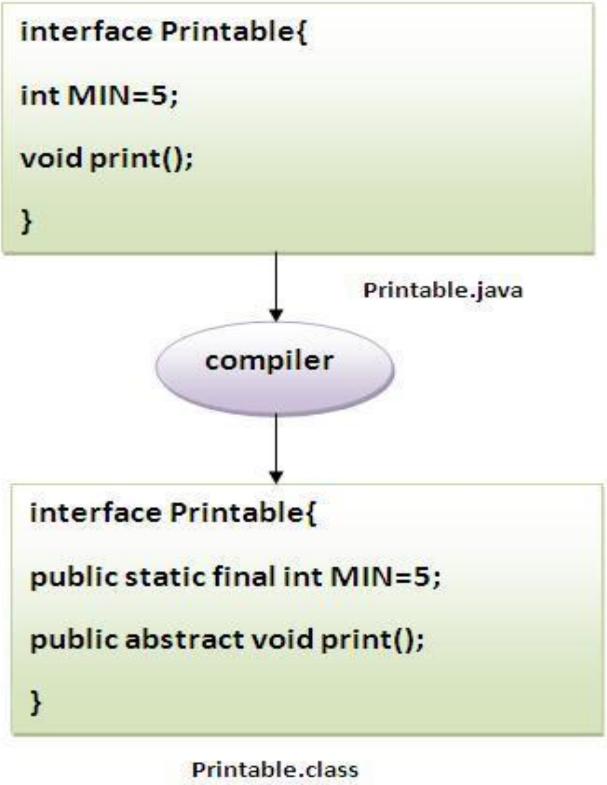
```
int a=043;
System.out.println(“a : ”+a);

int a=0x4b;
System.out.println(“a : ”+a);

char ch=-20;
System.out.println(“Char : ”+ch);

System.out.println(“Add : ”+11+2+3);

float x=12,y=11.1;
System.out.println(“Add : ”+1+x+y);

int a = 10;
long b = 20;
short c = 30;
System.out.println(++a + b++ * c);

char a = 'a'; // line 3
char b = 10; // line 4
char c = '1'; // line 5
int d = 1000; // line 6
System.out.println(++a + b++ * c - d);

int a=10,b=10;
String name= a==b ? a>10 ? "Hi" : “Hello”;
System.out.println("Name : "+name);
```

IS-A , HAS-A relation.

Interface & Abstract class.

Interface

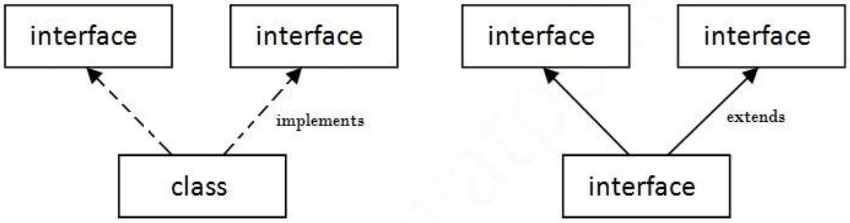
- All interface methods are implicitly `public` and `abstract`.
- All variables defined in an interface must be `public`, `static`, and `final`.



```
interface Printable{

int MIN=5;

void print();

}
```

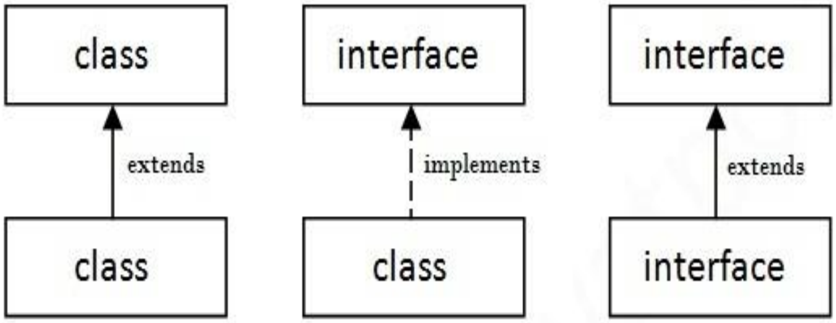
```
interface Printable{

public static final int MIN=5;

public abstract void print();

}
```

- Interface methods must not be `static`.
- Because interface methods are abstract, they cannot be marked `final`.
- Class can implement multiple (multiple inheritance in java).



Multiple Inheritance in Java

- An interface can *extend* one or more other interfaces.
- An interface cannot extend anything but another interface.
- An interface cannot implement another interface or class.
- An interface must be declared with the keyword `interface`.



Abstract Class

- The abstract keyword can be applied to a class or a method but not to a field.
- An abstract class cannot be instantiated. You can, however, create reference variables of an abstract class type. In fact, you can create objects of the classes derived from an abstract class and make the abstract class references refer to the created derived class objects.
- An abstract class can extend another abstract class or can implement an interface.
- An abstract class can be derived from a concrete class! Although the language allows it, it is not a good idea to do so.
- An abstract class need not declare an abstract method, which means it is not necessary for an abstract class to have methods declared as abstract. However, if a class has an abstract method, it should be declared as an abstract class.
- A subclass of an abstract class needs to provide implementation of all the abstract methods; otherwise you need to declare that subclass as an abstract class.
- An abstract class may have methods or fields declared static.

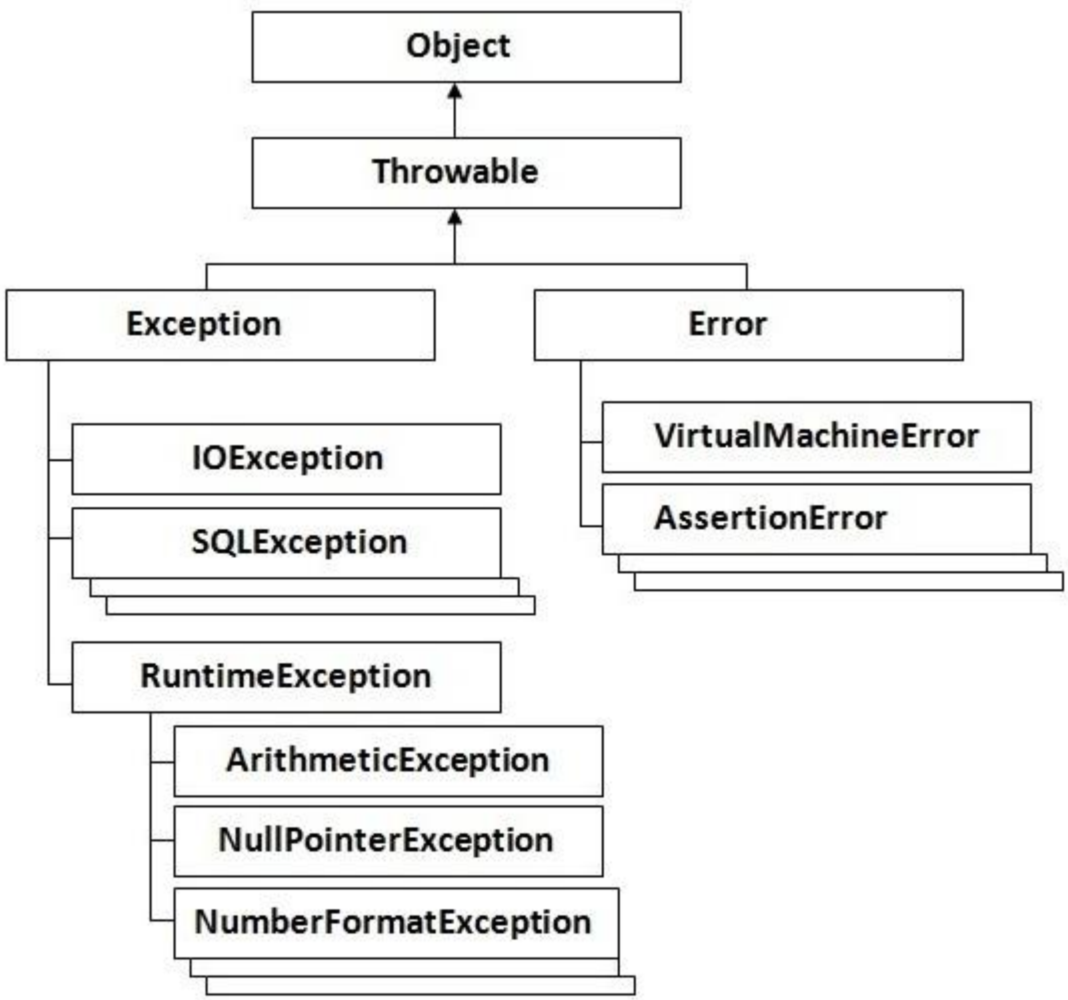
Exception handling

Types of Exception

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

1. Checked Exception
2. Unchecked Exception
3. Error

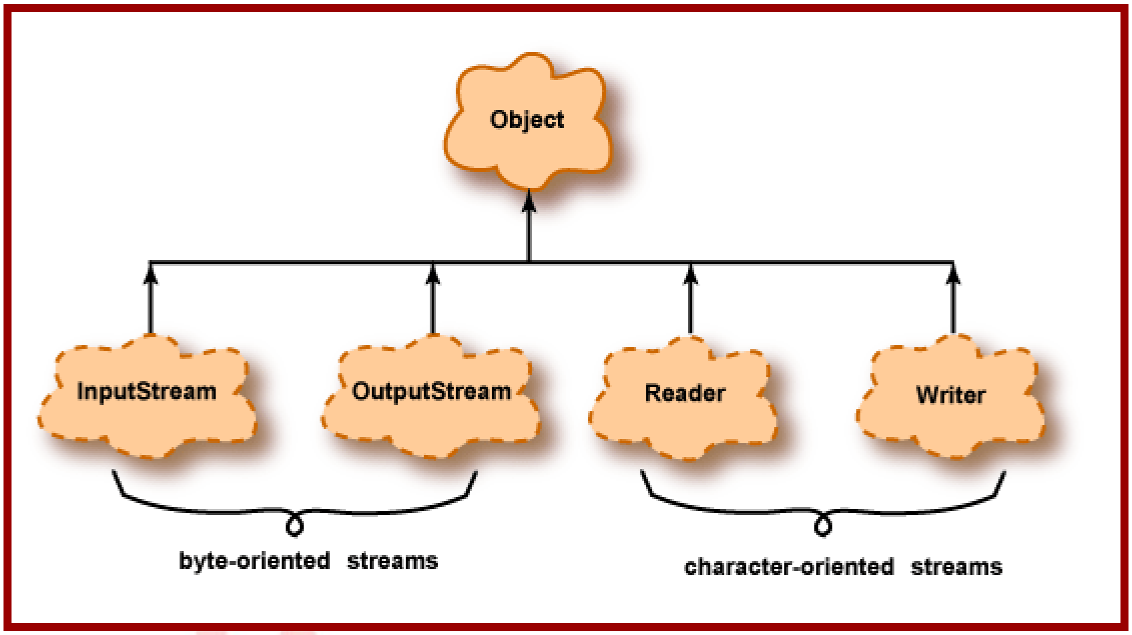
Hierarchy of Exception classes



Five keywords used in Exception handling:

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

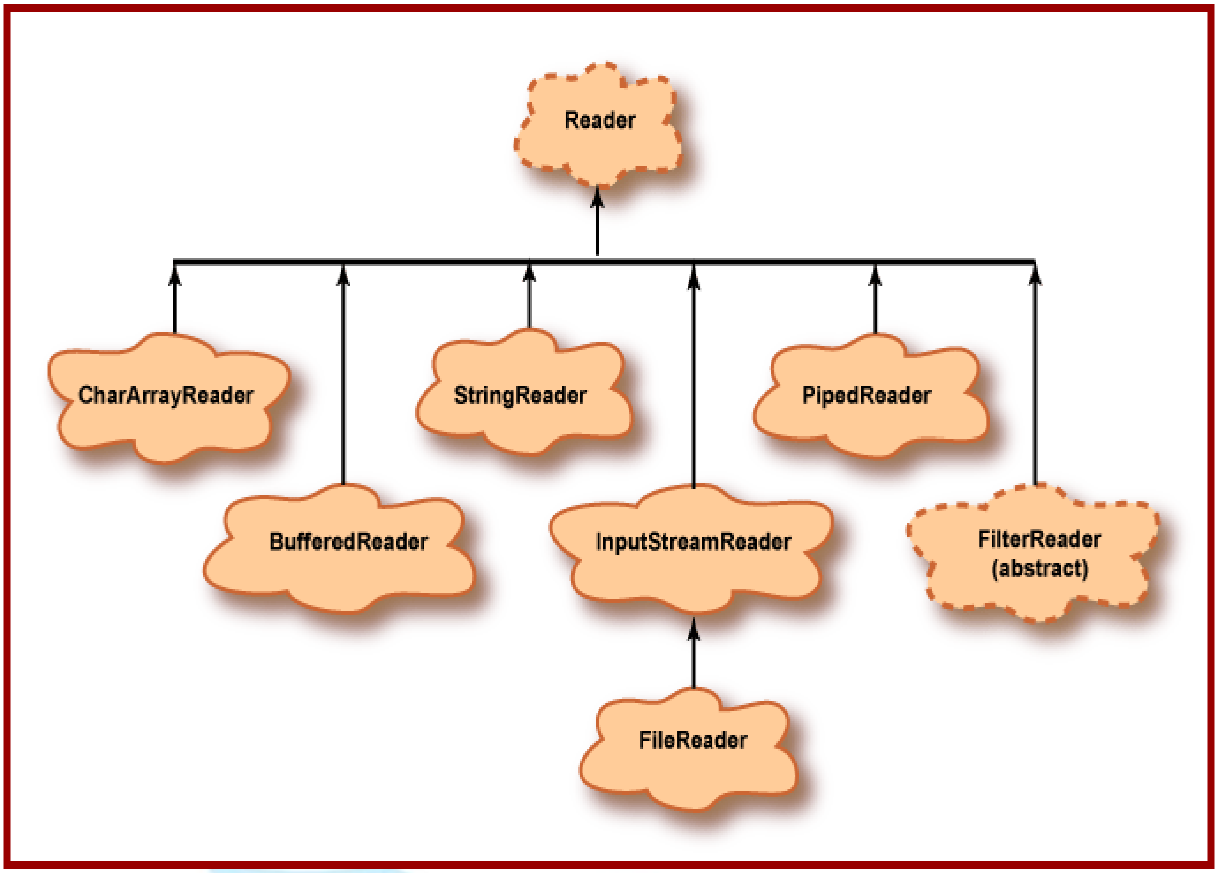
I/O Stream



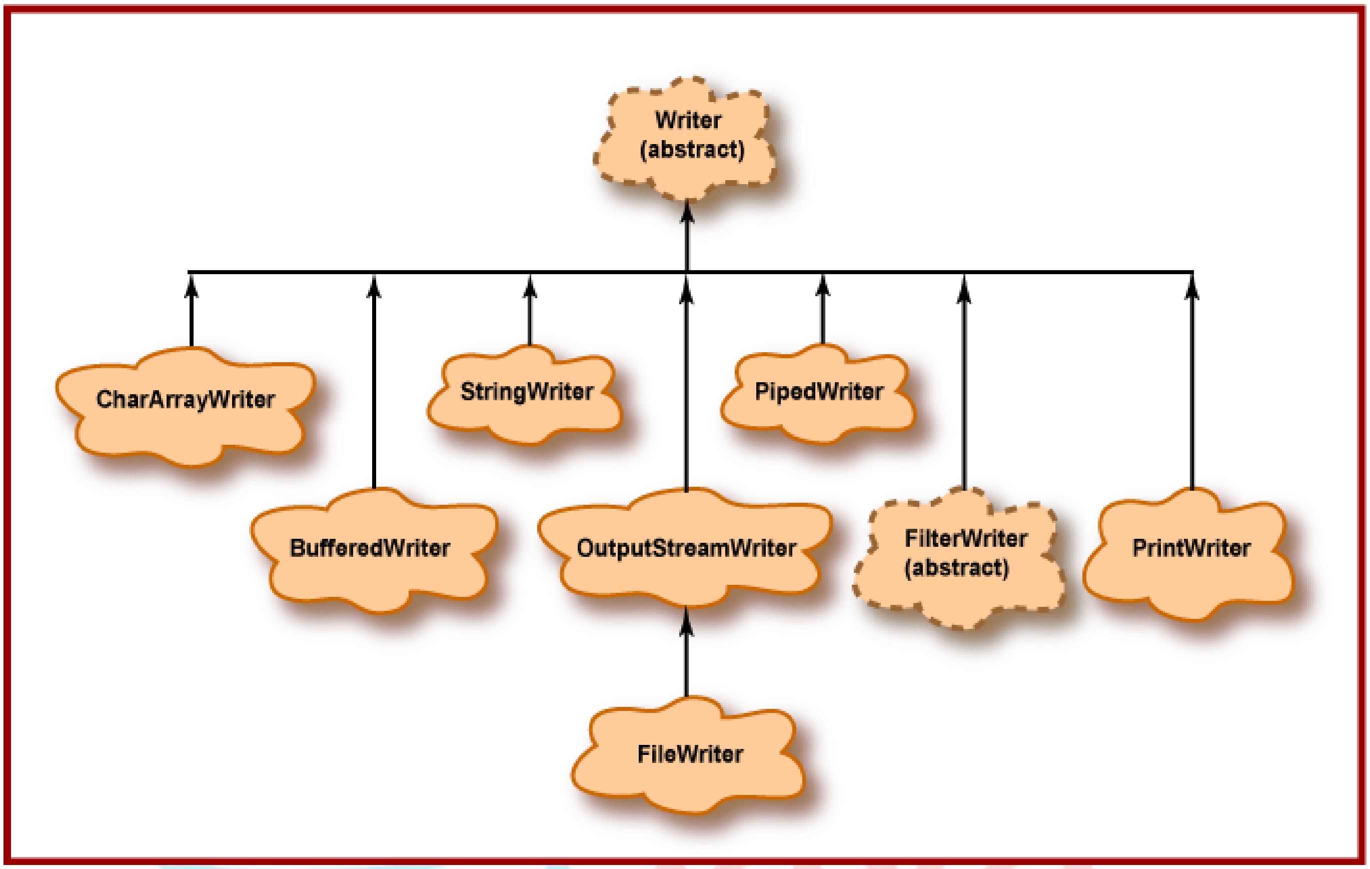
Character streams	Byte streams
Meant for reading or writing to character- or text-based I/O such as text files, text documents, XML, and HTML files.	Meant for reading or writing to binary data I/O such as executable files, image files, and files in low-level file formats such as .zip, .class, .obj, and .exe.
Input and output character streams are called *readers* and *writers*, respectively.	Input and output byte streams are simply called *input streams* and *output streams*, respectively.
The abstract classes of Reader and Writer and their derived classes in the java.io package provide support for character streams.	The abstract classes of InputStream and OutputStreamc and their derived classes in the java.io package provide support for byte streams.

Character Stream

Reader

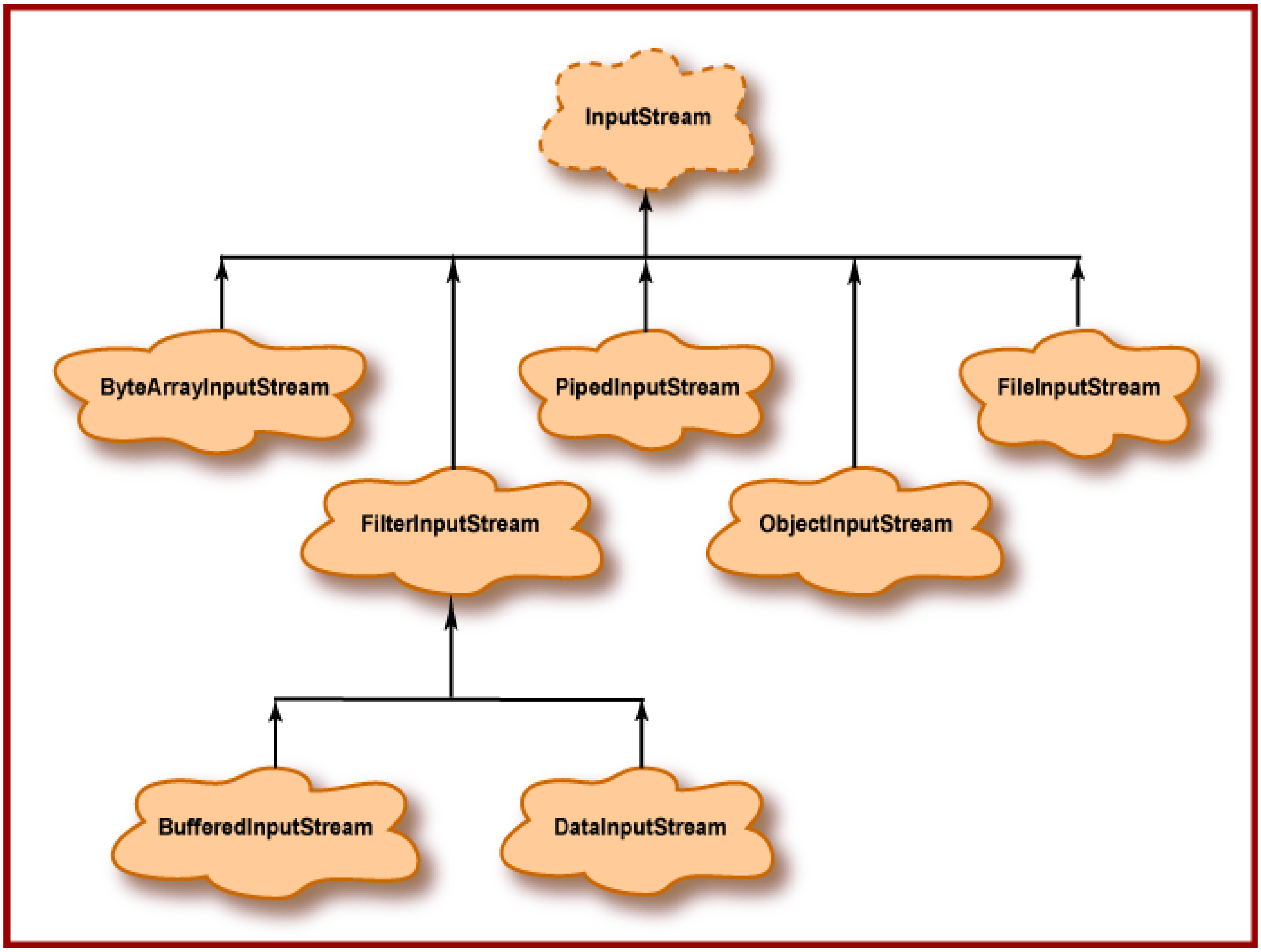


Writer

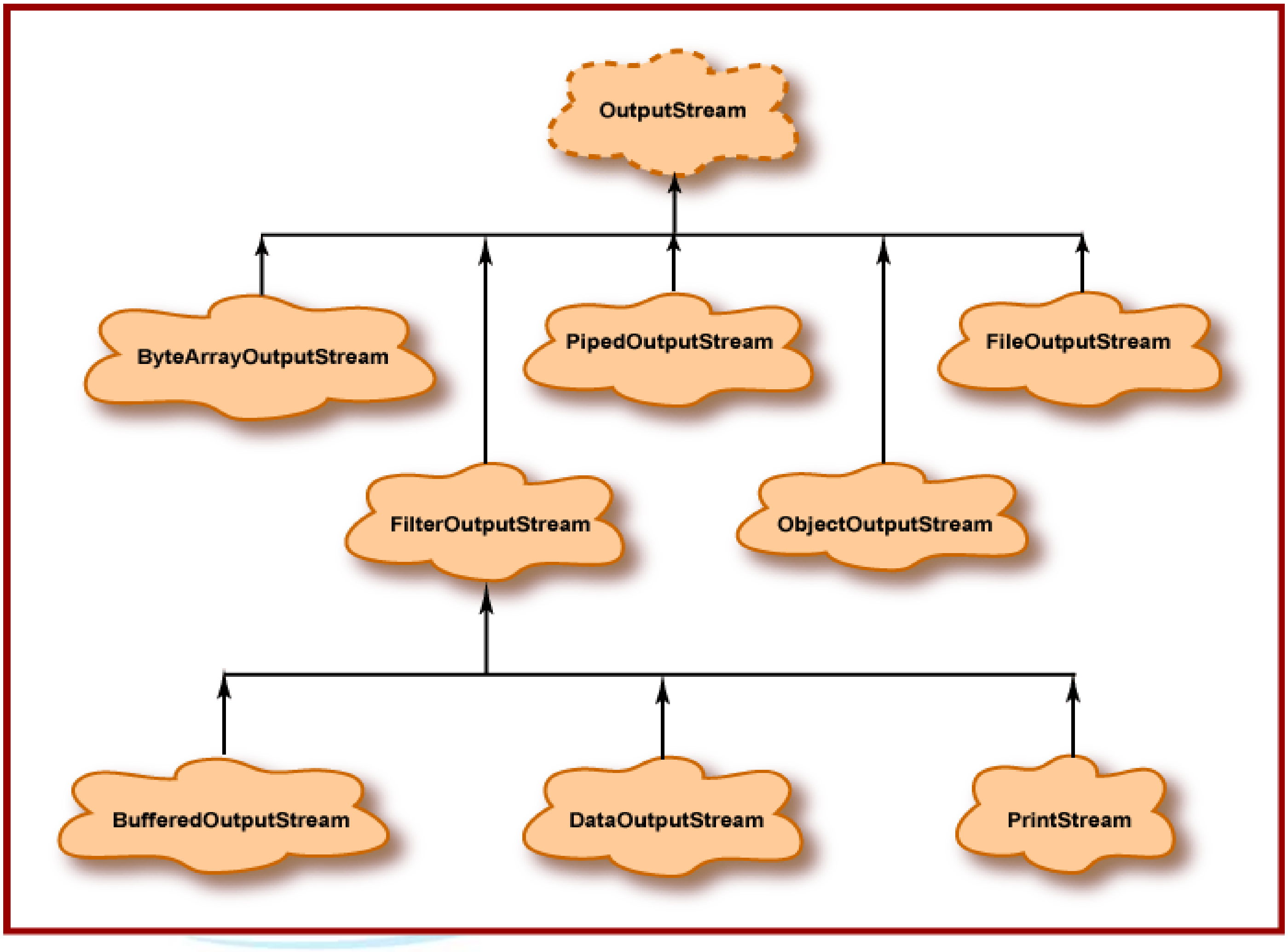

Writer
(abstract)
CharArrayWriter
StringWriter
PipedWriter
BufferedWriter
OutputStreamWriter
FilterWriter
(abstract)
PrintWriter
FileWriter

Byte Stream

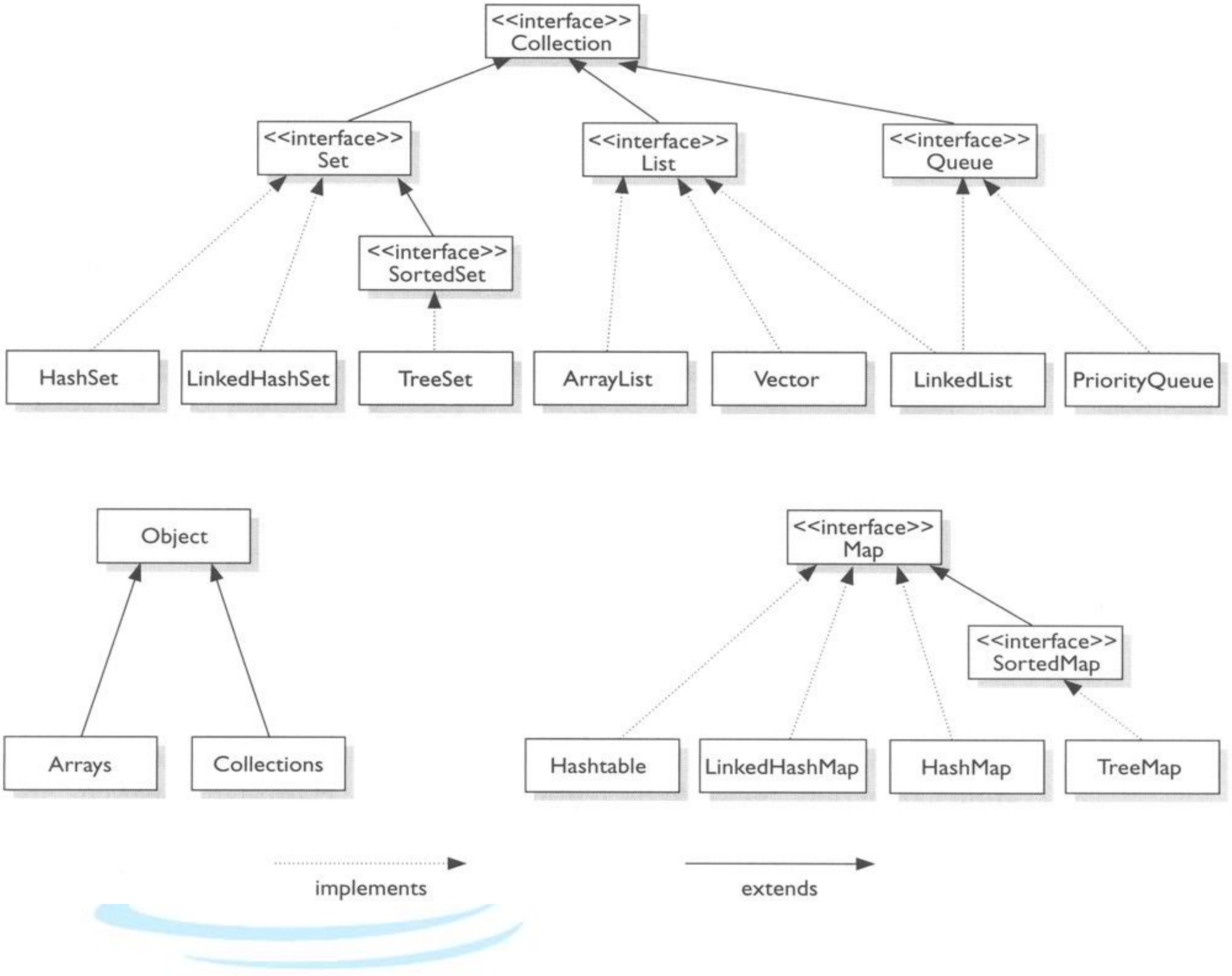
InputStream



OutputStream



Collection


<<interface>>
Collection
<<interface>>
Set
<<interface>>
List
<<interface>>
Queue
<<interface>>
SortedSet
HashSet
LinkedHashSet
TreeSet
ArrayList
Vector
LinkedList
PriorityQueue
Object
Arrays
Collections
<<interface>>
Map
<<interface>>
SortedMap
Hashtable
LinkedHashMap
HashMap
TreeMap
implements
extends

List

- ArrayList :- Ordered but not Sorted
- Vector :- same as ArrayList but vector methods are synchronized.
- LinkedList :- Ordered not sorted (peek(), poll(), push())

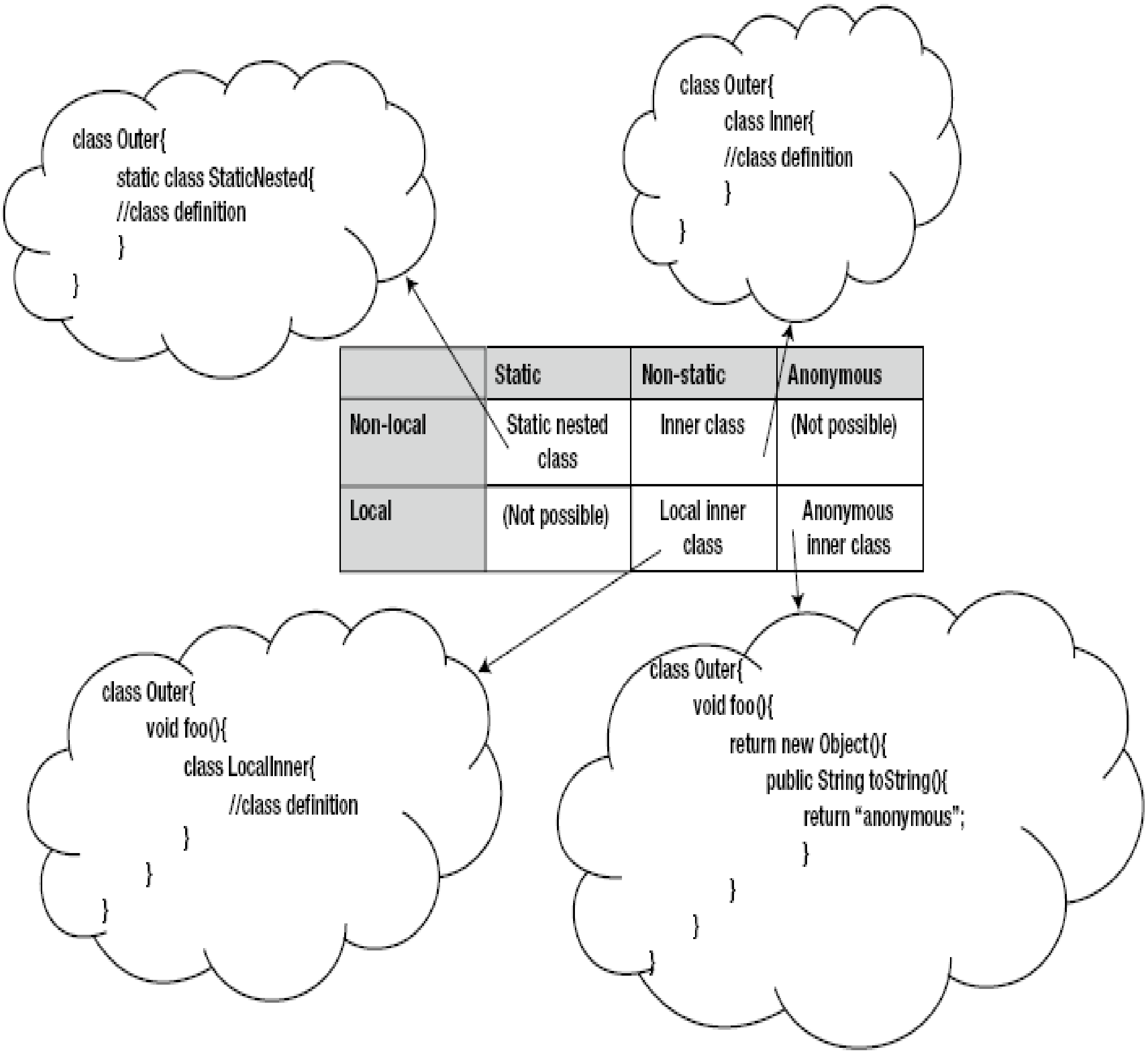
Set

- HashSet :- Unorder, Unsorted type of set.
- LinkedHashSet :- Ordered type of set.
- TreeSet :- Sorted type of set

Map

- HashMap :- Unorder, unsorted type of map. Allow one null key & multiple null Values.
- HashTable :- synchronized type. Null key or value not allowed.
- LinkedHashMap :- ordered , faster iteration.
- TreeMap :- Sorted type of map.

Inner class



	Static	Non-static	Anonymous
Non-local	Static nested class	Inner class	(Not possible)
Local	(Not possible)	Local inner class	Anonymous inner class

```
class Outer{
        static class StaticNested{
        //class definition
        }
}
```

```
class Outer{
        class Inner{
        //class definition
        }
}
```

```
class Outer{
        void foo(){
                class LocalInner{
                        //class definition
                }
        }
}
```

```
class Outer{
        void foo(){
                return new Object(){
                        public String toString(){
                                return "anonymous";
                        }
                }
        }
}
```