**Java Version Updates**

**Java 1.5 :**

1. Generics : Generic in Java is added to provide compile time type-safety of code and removing risk of ClassCastException at [runtime](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html) which was quite frequent error in Java code, for those who doesn’t know what is type-safety at compile time, it’s just a check by compiler that correct Type is used in correct place and there should not be any ClassCastException.  
   *Example* :

List<Integer> lis = new ArrayList<Integer>();

1. Autoboxing/unboxing : Automatic conversion from primitive types to their object equivalent is auto boxing. Reverse process is auto unboxing

*Example* :

int i = 0;

i = new Integer(5); // auto-unboxing

Integer i2 = 5; // autoboxing

1. Static Import :

*Example*:

Import static System.out;

Class ABC

{

Public void show()

{

//System.out.println(“”);

out.println(“”);

}

}

1. Annotations: Annotations in Java is all about adding [meta-data](http://www.javabeat.net/forums/index.php?topic=139.0) facility to the Java Elements. Like Classes, [Interfaces](http://www.javabeat.net/forums/index.php?topic=139.0) or Enums, Annotations define a type in Java and they can be applied to several Java Elements. Tools which will read and interpret the Annotations will implement a lot of functionalities from the meta-information obtained.
2. Type safe Enums: A Java Enum is a special Java type used to define collections of constants. More precisely, a Java enum type is a special kind of Java class. An enum can contain constants, methods etc.
3. Varargs :

*Example*

public static void main(String args[])

{ add(5,6,7);}

public static void add(int…values)//varargs

{}

1. Enhanced for loops:

*Example:*

For(int I : values)

{

}

**Java 1.7:**

1. Dynamic Language Support(Diamond operator,Type Interference for Generic Instance),

*Example:*

Prior JDK 7

Map<String, List<String>> employeeRecords = new HashMap<String, List<String>>();

List<Integer> primes = new ArrayList<Integer>();

In JDK 7

Map<String, List<String>> employeeRecords = new HashMap<>();

List<Integer> primes = new ArrayList<>();

1. String in Switch
2. Automatic Resource Management – Before 1.7, we need to close resource in finally block, but now its taken care automatically with try with resources.
3. Fork Join Framework[Taking advantage of multiple processors in modern servers.To use all the available processing power to enhance the performance of applications],
4. Underscore in numeric literals,

*Example:*

int billion = 1\_000\_000\_000; // 10^9

long creditCardNumber = 1234\_4567\_8901\_2345L; //16 digit number

long ssn = 777\_99\_8888L;

double pi = 3.1415\_9265;

float pif = 3.14\_15\_92\_65f;

double pi = 3.\_1415\_9265; // underscore just after decimal point

long creditcardNum = 1234\_4567\_8901\_2345\_L; //underscore at the end of number

long ssn = \_777\_99\_8888L; //undersocre at the beginning

1. Catching multiple exception Type in Single Catch block,

*Example*

Before 1.7

try {

......

} catch(ClassNotFoundException ex) {

ex.printStackTrace();

} catch(SQLException ex) {

ex.printStackTrace();

}

In JDK 7, you could use one single catch block, with exception types separated by '|'.

try {

......

} catch(ClassNotFoundException|SQLException ex) {

ex.printStackTrace();

}

1. Binary Literals with prefix 'ob':

In JDK 7, you can express literal values in binary with prefix '0b' (or '0B') for integral types (byte, short, int and long), similar to C/C++ language. Before JDK 7, you can only use octal values (with prefix '0') or hexadecimal values (with prefix '0x' or '0X').

int mask = 0b01010000101;

or even better

int binary = 0B0101\_0000\_1010\_0010\_1101\_0000\_1010\_0010;

1. G1 Garbage collector - JDK 7 introduced a new Garbage Collector known as G1 Garbage Collection, which is short form of garbage first. G1 garbage collector performs clean-up where there is most garbage. To achieve this it split [Java heap memory](http://javarevisited.blogspot.sg/2013/04/what-is-maximum-heap-size-for-32-bit-64-JVM-Java-memory.html) into multiple regions as opposed to 3 regions in the prior to Java 7 version (new, old and permgen space).
2. More precise rethrowing of exceptions
3. Java NIO 2.0 Package- New File system API complements older one and provides several useful method checking, deleting, copying, and moving files. for example, now you can check if a file is hidden in Java. You can also create symbolic and hard links from Java code. JDK 7 new file API is also capable of searching for files using wild cards. You also get support to watch a directory for changes.

**Java 8 :**

1. Functional Programming – Instead of focusing on objects, we focus on behavior. Imperative programming[what to do and how to do], Declarative programming[what to do]

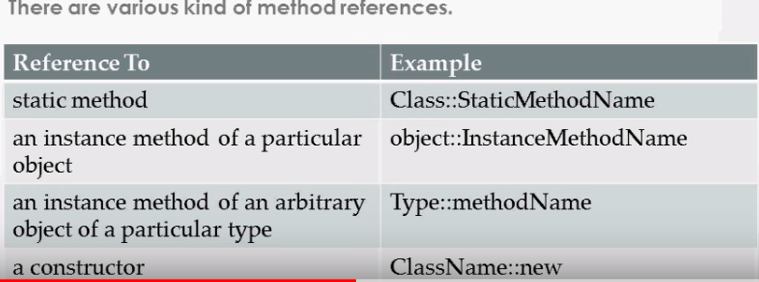
Functional Interface – Interface which has only one method is called as Functional Interface.Java 8 has introduced functional interface in java.util.function

1. Lambda Expression – Remove boiler plate code which we use in Anonymous Inner class. It is used with functional Interfaces

Syntax : comma separated list of parameters enclosed in parenthesis of method, arrow, A body which consists of a single expression or a statement block

(a,b) -> a+b;

1. Stream API – Whenever we have mutable variables and when 2 threads are trying to access the variables, we get concurrency problems. To overcome the concurrency problems we have stream API. It helps to achieve functional programming
2. Define methods in Interface
3. forEach() method : have added in java.lang.Iterable to iterate each element. This is the default method of java.lang.Iterable. java.lang.Iterable is the super interface of List and Collection so forEach() is available in List and Collection to use.
4. Date and Time API
5. Method References: This can be used when lambda expression does nothing but just calls a method which is already defined. :: operator is used. It works with functional interface only.



Interface in Java 8:

1. Methods in interface are by default public abstract till java 1.7. we can just declare methods till 1.7
2. In Java 1.8, Methods can be defined in the Interface by using the **default** keyword, else it will not work in Java 8. Example, List Interface has got number of new methods like stream(). Instead of declaring the stream() in List interface, they have defined it so that it doesn’t break the java code which was built in older versions if they have used List.
3. Then what is the difference between Interface and abstract class in Java 1.8?
   1. Abstract classes are similar to interfaces. You cannot instantiate them, and they may contain a mix of methods declared with or without an implementation. However, with abstract classes, you can declare fields that are not static and final, and define public, protected, and private concrete methods. With interfaces, all fields are automatically public, static, and final, and all methods that you declare or define (as default methods) are public. In addition, you can extend only one class, whether or not it is abstract, whereas you can implement any number of interfaces.
4. We will have the diamond problem when a method is defined in 2 interfaces and when the class implements both the interfaces. In that case, we will have to define the same method again in the implemented class to avoid the diamond problem
5. Class extends an interface and another class. Both the extended class and interface has the same defined method[show()]. When the child class calls the same method[show()], parent class implementation will be called rather than the interface one. class will have more power than interface.
6. When to try to define a method in an interface which is already defined in the Object class, it will be an error.
7. Static methods can also be defined in Java 8. It can be called with the interface name directly.

For Each in Java 8:

1. External loops – Until Java 1.8, we were using external for loops. This is like you have a box of items, you put your hands to take the items one by one

**for**(**int** i=0;i<values.size();i++)

{

System.***out***.println(values.get(i));

}

**for**(**int** i:values)

{

System.***out***.println(i);

}

1. Internal loops – In Java 1.8, we have internal for loops which is very fast compared to the previous ones.

values.forEach(i -> System.***out***.println(i));

Stream API:

1. Stream methods will create multiple threads based on the CPU core.

values.forEach(i ->System.***out***.println(i));

values.parallelStream().forEach(i -> System.***out***.println(i)); // This will create multiple threads based on CPU core

1. It has number of methods in it. Intermediate methods[**Intermediate operations are lazy. This means that they will be invoked only if it is necessary for the terminal operation execution.**] and Terminate methods[The API has many terminal operations which aggregate a stream to a type or to a primitive]
2. Streams used can’t be reused whereas in list this is not the case

values.parallelStream().forEach(i -> System.***out***.println(i));// First time it will work

values.parallelStream().forEach(i -> System.***out***.println(i)); // Second time, exception

1. Only we reduce mutation, only if we reduce mutation, we achieve concurrency. Only we achieve concurrency, the system will be fast. To write a code without achieving mutation, You can take advantage of stream methods
   1. *Example :values.stream().map(i->i\*2).reduce(0,(c,e)->c+e);*

Stream() – method in Collection interface to convert your collection(values) to a stream

Filter – Intermediate method. Which will filter the data based on the criteria mentioned.

Map – Intermediate method. Will apply the logic and convert to the specified output

Reduce – Terminal method. Take 2 parameters and do the aggregation

Findfirst – Terminal method. Picks the first element

DateTime API:

1. Problems with old Date API,
   1. you have 2 packages util & sql. Same class in 2 packages.
   2. If we need to format any date, we need to import java.text
   3. Date classes are not thread safe.
2. New Date API,
   1. it is immutable
   2. Packages have changed. All Date APIs are in java.time.

**Java 9:**

1. JigSaw
   1. Modularity : Rt.jar : This is a runtime.jar. It has 4500 classes. This class is used by all java classes at present. Do we need all the classes in rt.jar? The answer is No.Then why do we need to all the classes in the Jar not a part of it which we actually need. For this reason, modules are created. JDK 9 has approximately 94 modules, so we can use the modules we need and ignore others.
   2. Creating modules : By default, all packages in a module are private. If you need to export a module, you have to say exports in module-info.java. Similarly if you need any other module,you have to say requires in module-info.java
2. REPL : Read Eval Print Loop with the help of Jshell. Introductive console. This is to test and teach java code easier
   1. Jshell commands: /list, /edit,/help, /set feedback, /vars, /methods, /types, /history, /imports, /help shortcuts,/open <filename>, /save <filename.jsh>, /!, /drop <vriable name>, /exit
3. Enhancement in Stream API
4. Define private methods in interface
5. HTTP 2 protocol
6. Enhancement in garbage collector

**Java 10:**

1. Parallel full GC for G1 – G1 is the default GC in java 9. But when the concurrent collections had problems with GC, we need a full clean up. This brings the need for full GC.
2. Local Variable Type Inference. Var is a new keyword that can be used only with local variables

var list = new ArrayList<String>();

1. Application class Data Sharing – Class Data sharing : When JVM starts, there is a shared archive file created by JRE and this loads the classes from this file. In JDK 10, application class files are also placed on the shared archive
2. Root Certificates – The main motive is to get the open jdk and oracle jdk builds the same
3. Consolidated the JDK forest into a single repository. In JDK 9, there were 8 respoistories
4. Removed the native header generation tool – javah. This can be done by Javac
5. Heap Allocation on alternate Memory Devices
6. Shorter start up times in Jshell REPL – JDK 10 shares its compiler instances
7. API for creating unmodifiable collections - JDK 10 has created copy() on list, set and map. These methods will create unmodifiable list
8. Improved Byte code generation for enhanced for Loop
9. Clean GC Interface
10. Support for multiple stylesheets in Javadoc
11. Thread Local Handshakes