==================================================================================

**HW-1: 12-09-2022**

**Q1. List down the important features of Java 1.8.**

Java 8 or JDK 1.8 is the most significant expansion of the Java language yet. Java 8’s new features such as Lambda Expressions, Stream APIs, Nashorn, Compact Profiles, new Time APIs increase the expressive power of the platform and make it easier for developers to take advantage of modern, multicore processors. This article gives an overview of the new features in Java 8 with links to in-depth tutorials for the most important of them.

**New Features in Java 8**

1. [**Lambda Expressions**](https://www.javabrahman.com/java-8/lambda-expressions-java-8-explained-examples/) enable you to treat functionality as a method argument, or code as data. Lambda expressions let you express instances of single-method interfaces (referred to as [**Functional Interfaces**](https://www.javabrahman.com/java-8/functional-interfaces-java-8/)) more compactly.
2. [**Method references**](https://www.javabrahman.com/java-8/java-8-method-references-tutorial-examples/) provide easy-to-read lambda expressions for methods that already have a name. [**Constructor References**](https://www.javabrahman.com/java-8/constructor-references-java-8-simplified-tutorial/) are the equivalent forms of representation for constructors.
3. [**Default methods**](https://www.javabrahman.com/java-8/default-methods-in-java-8-with-examples/) enable new functionality to be added to the interfaces of libraries and ensure binary compatibility with code written for older versions of those interfaces.
4. With default methods in Java 8, [**multiple inheritance of behavior**](https://www.javabrahman.com/java-8/java-8-multiple-inheritance-of-behavior-from-interfaces-using-default-methods/) is now possible in Java, and it is important to understand the [**conflict resolution rules**](https://www.javabrahman.com/java-8/java-8-multiple-inheritance-conflict-resolution-rules-and-diamond-problem/) which resolve *Diamond Problem* and other conflict scenarios.
5. Java 8’s **new package** **[java.util.function](https://www.javabrahman.com/java-8/java-8-java-util-function-package-tutorial/)** provides many useful functional interfaces for the most common scenarios. The 4 most important functional interface among them are – [**Predicate**](https://www.javabrahman.com/java-8/java-8-java-util-function-predicate-tutorial-with-examples/), [**Consumer**](https://www.javabrahman.com/java-8/java-8-java-util-function-consumer-tutorial-with-examples/), [**Function**](https://www.javabrahman.com/java-8/java-8-java-util-function-function-tutorial-with-examples/) and [**Supplier**](https://www.javabrahman.com/java-8/java-8-java-util-function-supplier-tutorial-with-examples/).
6. [**Repeating Annotations**](https://www.javabrahman.com/java-8/java-8-repeating-annotations-tutorial/) provide the ability to apply the same annotation type more than once to the same declaration or type use.
7. New **java.util.stream** package provides a new [**Streams API**](https://www.javabrahman.com/java-8/java-8-streams-api-tutorial-with-examples/) to support functional-style operations on streams of elements. The Stream API is integrated into the Collections API.
8. Java 8’s [**new Collector interface**](https://www.javabrahman.com/java-8/java-8-java-util-stream-collector-basics-tutorial-with-examples/) and its multiple predefined implementations provide an efficient way to terminate the Stream operations and collect the result in a collection.
9. **A**[**new Date-Time package**](https://www.javabrahman.com/java-8/overview-of-java-8-new-date-time-api-java-time-package-tutorial/)**– java.time** – with a new comprehensive set of date and time utilities.
10. Java 8’s **new enhanced methods in Collections API** are covered in a series of 4 tutorials – **[Iterable.forEach() & Iterator.remove()](https://www.javabrahman.com/java-8/java-8-iterable-foreach-iterator-remove-methods-tutorial-with-examples/)**,  **[Collection.removeIf()](https://www.javabrahman.com/java-8/java-8-collection-removeif-method-tutorial-with-examples/)**,  **[List.sort() & List.replaceAll()](https://www.javabrahman.com/java-8/new-features-in-java-8/www.javabrahman.com/java-8/java-8-list-sort-list-replaceall-methods-tutorial-with-examples/)**,  and new [**multi-value map**](https://www.javabrahman.com/java-8/java-8-maps-computeifabsent-computeifpresent-getordefault-methods-tutorial-with-examples/) methods.
11. Java 8 has introduced [**new internal iterators**](https://www.javabrahman.com/java-8/java-8-internal-iterators-vs-external-iterators/) based on declarative functional programming style.
12. Comparator interface has undergone a major upgrade in Java 8 with new methods leveraging Java 8’s functional programming features, comparator chaining, in-built null handling, and many more such features. [**Java 8 Comparator tutorial**](https://www.javabrahman.com/java-8/the-complete-java-8-comparator-tutorial-with-examples/) covers these new features in depth.
13. **Nashorn Javascript Engine enhanced** to provide a version of javascript which would run within the JVM
14. **Standard Encoding and Decoding Base64**,**Parallel Array Sorting** and **Unsigned Arithematic Support**.
15. **JDBC 4.2** with new features and notably the JDBC-ODBC bridge has been removed.
16. **Concurrency related important changes** which include –
    * Changes to ConcurrentHashMap to support aggregate operations based on the newly added streams facility and lambda expression.
    * Addition of classes to the java.util.concurrent.atomic package to support scalable updatable variables.
    * Support for a common pool in ForkJoinPool.
    * New **StampedLock** class to to provide a capability-based lock with three modes for controlling read/write access
17. **Type Annotations** provide the ability to apply an annotation anywhere a type is used, not just on a declaration. Used with a pluggable type system, this feature enables improved type checking of your code.
18. **Improved Type Inference** and **Method Parameter Reflection**.
19. **Compact Profiles** contain predefined subsets of the Java SE platform and enable applications that do not require the entire Platform to be deployed and run on small devices.
20. **Improved internationalization** including support for Unicode 6.2.0, new Calendar and Locale APIs, Adoption of Unicode CLDR Data and the java.locale.providers System Property and Ability to Install a Custom Resource Bundle as an Extension.

---------------------------------------------------------

**Q2.** **Differentiate between C++ & Java.**

|  |  |  |
| --- | --- | --- |
| **Comparison Index** | **C++** | **Java** |
| **Platform-independent** | C++ is platform-dependent. | Java is ***platform-independent.*** |
| **Mainly used for** | C++ is mainly used for ***system programming.*** | Java is mainly used ***for application programming.*** It is widely used in Windows-based, web-based, enterprise, and mobile applications. |
| **Goto** | C++ supports the [goto](https://www.javatpoint.com/cpp-goto-statement) statement. | Java ***doesn't support the goto statement.*** |
| **Multiple inheritance** | C++ supports multiple inheritance. | Java ***doesn't support multiple inheritance through class.*** *It can be achieved by using*[*interfaces in java*](https://www.javatpoint.com/interface-in-java)*.* |
| **Operator Overloading** | C++ supports [operator overloading](https://www.javatpoint.com/cpp-overloading). | Java ***doesn't support operator overloading.*** |
| **Pointers** | C++ supports [pointers](https://www.javatpoint.com/cpp-pointers). You can write a pointer program in C++. | ***Java supports pointer internally***. *However,* ***you can't write*** *the pointer program in java*. It means java has restricted pointer support in java. |
| **Compiler and Interpreter** | C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent. | ***Java uses both compiler and interpreter.*** Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent. |
| **Call by Value and Call by reference** | C++ supports both call by value and call by reference. | ***Java supports call by value only***. There is no call by reference in java. |
| **Structure and Union** | C++ supports structures and unions. | Java ***doesn't support structures and unions.*** |
| **Thread Support** | C++ doesn't have built-in support for threads. *It relies on third-party libraries for thread support.* | Java ***has built-in***[***thread***](https://www.javatpoint.com/multithreading-in-java)***support.*** |
| **Documentation comment** | C++ doesn't support documentation comments. | Java ***supports documentation comment (/\*\* ... \*/)*** to create documentation for java source code. |
| **Virtual Keyword** | C++ supports virtual keyword so that we can decide whether or not to override a function. | ***Java has no virtual keyword***. We can override all non-static methods by default. In other words, non-static methods are virtual by default. |
| **unsigned right shift >>>** | C++ doesn't support >>> operator. | ***Java supports unsigned right shift >>> operator*** that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator. |
| **Inheritance Tree** | C++ always creates a new inheritance tree. | Java always uses a single inheritance tree because all classes are the child of the Object class in Java. The Object class is the root of the [inheritance](https://www.javatpoint.com/inheritance-in-java) tree in java. |
| **Hardware** | C++ is nearer to hardware. | ***Java is not so interactive with hardware.*** |
| **Object-oriented** | C++ is an object-oriented language. However, in the C language, a single root hierarchy is not possible. | Java is also an [object-oriented](https://www.javatpoint.com/java-oops-concepts) language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object. |

**Note**

* Java ***doesn't support default arguments*** like C++.
* Java ***does not support header files like C++.*** J***ava uses the import keyword to include different classes and methods.***

C vs C++ vs Java

The languages are based on each other but still, they are different in design and philosophy. The following table describes the major differences between C, C++, and Java. It will help you to select which language you have to learn.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.N.** | **Basis** | **C** | **C++** | **Java** |
| **1** | **Origin** | The C language is based on BCPL. | The C++ language is based on the C language. | The Java programming language is based on both C and C++. |
| **2** | **Programming Pattern** | It is a procedural language. | It is an object-oriented programming language. | It is a pure object-oriented programming language. |
| **3** | **Approach** | It uses the top-down approach. | It uses the bottom-up approach. | It also uses the bottom-up approach. |
| **4** | **Dynamic or Static** | It is a static programming language. | It is also a static programming language. | It is a dynamic programming language. |
| **5** | **Code Execution** | The code is executed directly. | The code is executed directly. | The code is executed by the JVM. |
| **6** | **Platform Dependency** | It is platform dependent. | It is platform dependent. | It is platform-independent because of byte code. |
| **7** | **Translator** | It uses a compiler only to translate the code into machine language. | It also uses a compiler only to translate the code into machine language. | Java uses both compiler and interpreter and it is also known as an interpreted language. |
| **8** | **File Generation** | It generates the .exe, and .bak, files. | It generates .exe file. | It generates .class file. |
| **9** | **Number of Keyword** | There are **32** keywords in the C language. | There are **60** keywords in the C++ language. | There are **52** keywords in the Java language. |
| **10** | **Source File Extension** | The source file has a .c extension. | The source file has a .cpp extension. | The source file has a .java extension. |
| **11** | **Pointer Concept** | It supports pointer. | It also supports pointer. | Java does not support the pointer concept because of security. |
| **12** | **Union and Structure Datatype** | It supports union and structure data types. | It also supports union and structure data types. | It does not support union and structure data types. |
| **13** | **Pre-processor Directives** | It uses pre-processor directives such as #include, #define, etc. | It uses pre-processor directives such as #include, #define, #header, etc. | It does not use directives but uses packages. |
| **14** | **Constructor/ Destructor** | It does not support constructor and destructor. | It supports both constructor and destructor. | It supports constructors only. |
| **15** | **Exception Handling** | It does not support exception handling. | It supports exception handling. | It also supports exception handling. |
| **16** | **Memory Management** | It uses the calloc(), malloc(), free(), and realloc() methods to manage the memory. | It uses new and delete operator to manage the memory. | It uses a garbage collector to manage the memory. |
| **17** | **Overloading** | It does not support the overloading concept. | Method and operator overloading can be achieved. | Only method overloading can be achieved. |
| **18** | **goto Statement** | It supports the goto statement. | It also supports the goto statement. | It does not support the goto statements. |
| **19** | **Used for** | It is widely used to develop drivers and operating systems. | It is widely used for system programming. | It is used to develop web applications, mobile applications, and windows applications. |
| **20** | **Array Size** | An array should be declared with size. For example, int num[10]. | An array should be declared with size. | An array can be declared without declaring the size. For example, int num[]. |