What is Java8?

* Java is the programming language used by millions of developers and numerous for their business solutions. With the Java 8 release, Java contributed in support for functional programming, enhancement of JavaScript engine, outstanding APIs for date time manipulation, new streaming API, etc. There are dozens of notable and valuable features in Java 8.

**What new features were added in Java 8?**

* **Lambda Expressions**− a new language feature allowing treating actions as objects
* **Method References** − enable defining Lambda Expressions by referring to methods directly using their names
* ***Optional*** − special wrapper class used for expressing optionality
* **Functional Interface** – an interface with maximum one abstract method, implementation can be provided using a Lambda Expression
* **Default methods** − give us the ability to add full implementations in interfaces besides abstract methods
* **Nashorn, JavaScript Engine** − Java-based engine for executing and evaluating JavaScript code
* ***Stream* API** − a special iterator class that allows processing collections of objects in a functional manner
* **Date API** − an improved, immutable JodaTime-inspired Date API

**What are main advantages of using Java 8?**

* More compact code
* Less boiler plate code
* More readable and reusable code
* More testable code
* Parallel operations

**What does the following lambda expression means ?**

* helloJava8 ( x-> x%2 )  
    
    
  Ans. helloJava8 receives an Integer as argument and then returns the modulus of that Integer.

**What are Default Methods ?**

* With Java 8, We can provide method definitions in the Interfaces that gets carried down the classes implementing that interface in case they are not overridden by the Class. Keyword "default" is used to mark the default method.

**What are advantages of lambda expression ?**

* Until Java 8 lists and sets were typically processed by client code obtaining an iterator from the collection, then using that to iterate over its elements and process each in turn. If the processing of different elements is to proceed in parallel, it was the responsibility of the client code, not the collection, to organise this.  
  **Java 8 makes it easier to distribute processing of collections over multiple threads.**  
  Collections can now organise their own iteration internally, transferring responsibility for parallelization from client code into library code.
* **Fewer lines of code.** As explained above the user has to only declare what is to be done in a declarative way.  
  **n -> System.out.println("Hello World " + n);**  
  So user has to type reduced amount of code.
* Using **Java 8 Lambda expressions higher efficiency** can be achieved. Using CPUs with multicores, user can take advantage of the multicore CPU’s by parallel processing of collections using lambda.

**Explain Java 8-Internal vs. External Iteration?**

* **External Iterators-** This Iterator is also known as active iterator or explicit iterator. For this type of iterator the control over iteration of elements is with the programmer. Which means that the programmer define when and how the next element of iteration is called.
* **Internal Iterators-** This Iterator is also known as passive iterator, implicit iterator or callback iterator. For this type of iterator the control over the iteration of elements lies with the iterator itself. The programmer only tells the iterator "What operation is to be performed on the elements of the collection". Thus the programmer only declares what is to be done and does not manage and control how the iteration of individual elements take place.

**What is a functional interface ?**

* @**Functional Interface** is a new interface added in Java 8 .  
  It indicates that the interface is to be used as a functional interface.package com.javainuse; **@FunctionalInterface** public interface Greetings { public void say Hello(String name); }   
  This annotation is **optional**. Even if not annotated with @FunctionalInterface, an interface can still be used as a functional interface.
* The interface for which we are writing the lambda function should have only a single method. if more than one method is specified for an interface annotated with we get a compiler exception  
  **Invalid '@FunctionalInterface' annotation; Greetings is not a functional interface**

**What is :: (double colon) operator-Method References in Java 8?**

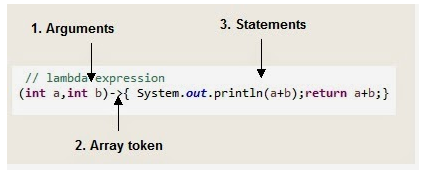
* Usually we use lambda expressions to create anonymous methods which return us the desired output. But sometimes lambda expressions do nothing but call an existing method. Because this lambda expression calls an existing method, **method reference** can be used here instead of Lambda function. Method reference is described using :: (double colon) symbol.

**What is Optional in Java 8?**

* Java 8 introduced a new **container class java.util.Optional<T>**. It wraps a single value, if that value is available. If the value is not available an empty optional should be returned. Thus it represents null value with absent value. This class has various utility methods like is Present() which helps users to avoid making use of null value checks. So instead of returning the value directly, a wrapper object is returned thus users can avoid the null pointer exception.

**Can you explain the syntax of Lambda expression?**

* So we can divide structure of Lambda expression to three parts:



**What is the difference between Collection API and Stream API?**

1.Collection API was introduced in jdk 1.2 while Stream API is introduced in jdk 1.8  
2. Collection objects are created eagerly while Stream API objects are created lazily.  
3. Iterate and Consume elements at any number of times for Collection object while iterate and consume elements only one time for Stream object.

**What is the difference between Iterator and Spliterator ?**

**1. Introduction :**Iterator was introduced in jdk 1.2 while Spliterator  is introduced in jdk 1.8  
**2. Use in API  :** Iterator is used for Collection API while Spliterator is used for Stream API  
**3. Parallel programming :**Iterator can be used for iterating the elements in Collection in sequential order while  Spliterator can be used for iterating the Stream elements in parallel or sequential order.  
**4. Universal Iterator :**Iteartor is universal iterator while Spliterator is not a universal iterator.

**What is Multiple Inheritance? How Java 8 supports Multiple Inheritance?**

* Multiple Inheritance means a class can inherit or extend characteristics and features from more than one parent class.
* In Java 7 or Earlier, Multiple Inheritance is not possible because Java follows “A class should extend one and only one class or abstract class” Rule. However, it’s possible to provide Multiple Implementation Inheritance using Interface because Java follows “A class can extend any number of Interfaces” Rule.
* However, Java 8 supports “Implementing Methods in Interfaces” by introducing new features: Default methods in Interface. Because of this feature, Java 8 supports Multiple Inheritance with some limitations

**Name few Java 8 annotations ?**

* **@FunctionalInterface annotation, introduced in Java SE 8, indicates that the type declaration is intended to be a functional interface, as defined by the Java Language Specification.  
    
  @Repeatable annotation, introduced in Java SE 8, indicates that the marked annotation can be applied more than once to the same declaration or type use. For more information, see Repeating Annotations.**

**What is StringJoiner ?**

* **StringJoiner is a util method to construct a string with desired delimiter. This has been introduced with from Java 8.  
    
  Sample Code  
    
  StringJoiner strJoiner = new StringJoiner(".");  
  strJoiner.add("Buggy").add("Bread");  
  System.out.println(strJoiner); // prints Buggy.Bread**