**Java Lambda Expressions Interview Questions And Answers**

1. What is lambda expression?

Lambda expression in itself is an anonymous method i.e. a method with no name which is used to provide implementation of the method defined by a functional interface.

A new syntax and operator has been introduced in Java for Lambda expressions.

General form of lambda expression

(optional) (Arguments) -> body

Give some examples of lambda expressions.

A very simple example of lambda expression would be-

() -> 5

This lambda expression takes no parameter (that's why empty parenthesis) and returns the constant value 5.

The above lambda expression is equivalent to the following Java code-

int getValue(){

return 5;

}

Another example which has 2 parameters-

// concatenating two strings, it has 2 string parameters and they are concatenated in lambda body

(String s1, String s2) -> s1+s2;

1. What is a functional interface?

A functional interface is an interface with only one abstract method. A functional interface is also known as SAM type where SAM stands for (Single Abstract Method).

An example of functional interface with in Java would be Runnable interface which has only one method run().

public interface Runnable {

public void run();

}

1. How do you use lambda expression with functional interface?

Lambda expression provides implementation of the abstract method defined by the functional interface.

1. How target type is inferred for the lambda expression?

Lambda expression doesn't have a type of its own. A lambda expression provides implementation of the abstract method defined by the functional interface. Thus the functional interface specifies its target type.

Lambda supports "target typing" which infers the object type from the context in which it is used.

To infer that object type from the context -

The parameter type of the abstract method and the parameter type of the lambda expression must be compatible. For Example, if the abstract method in the functional interface specifies one int parameter, then the lambda should also have one int parameter explicitly defined or implicitly inferred as int by the context.

Its return type must be compatible with the method's type.

Lambda expression can throw only those exceptions which are acceptable to the method.

(int x, int y) -> x+y; or (x, y) -> x + y; which one of these is a valid lambda expression?

Both of them are valid lambda expressions if used in a correct context.

With the first one (int x, int y) -> x+y; we know that the parameters must be of type int.

In case of (x, y) -> x + y; if used in a correct context type can be inferred from the context in which the lambda expression is executed.

(int x, y) -> x + y; is this a valid lambda expression?

You can't have lambda expression where type for only one of the parameter is explicitly declared so this lambda expression is invalid.

1. What is block lambda expression?

A block lambda is the lambda expression where the right side of the lambda expression is a block of code.

As example -

IMyFunc myFactorialFunc = (num) -> {

int fact = 1;

for(int i = 1; i <= num; i++){

fact = i \* fact;

}

return fact;

};

Where as (String s1, String s2) -> s1+s2; is a single expression lambda.

1. Can we have a generic functional interface?

Since lambda expression doesn't have type parameters of its own so it can't be generic. But the functional interface that specifies the target type for the lambda expression can be generic.

1. What about inbuilt functional interfaces?

With Java 8 many new functional interfaces are being defined, in fact there is a whole new package java.util.function added with many functional interfaces. The interfaces in this package are general purpose functional interfaces used by the JDK, and are available to be used by user code as well.

Comparator method is a functional interface but I see a lot of other methods in Comparator method then how is it a Single Abstract method interface?

From Java 8 it is possible for an interface to have default methods and static methods so, in a functional interface there may be other default and static methods but there must be only one abstract method.

A functional interface can specify Object class public methods too in addition to the abstract method. That interface will still be a valid functional interface. The public Object methods are considered implicit members of a functional interface as they are automatically implemented by an instance of functional interface.

1. What is functional interface annotation?

Java 8 also introduces an annotation @FunctionalInterface to be used with functional interface. Annotating an interface with @FunctionalInterface indicates that an interface type declaration is intended to be a functional interface.

1. Is it mandatory to mark functional interface with @FunctionalInterface annotation?

It is not mandatory to mark functional interface with @FunctionalInterface annotation, it is more of a best practice to do that and also gives a surety that no other abstract method will be added accidentally to the functional interface. Because it will result in compiler error if any other abstract method is added to a functional interface which is annotated with @FunctionalInterface annotation.

1. What is effective final in Java? What is variable capture?

When a lambda expression uses an assigned local variable from its enclosing space there is an important restriction. A lambda expression may only use local variable whose value doesn't change. That restriction is referred as "variable capture".

The local variables that a lambda expression may use are known as "effectively final". An effectively final variable is one whose value does not change after it is first assigned.

1. Can lambda expression throw exception? Is there any restriction in lambda expression exception handling?

A lambda expression can throw an exception but lambda expression must throw exception compatible with those specified in the throws clause of the functional interface method.

If a lambda expression body throws an exception, the throws clause of the functional interface method must declare the same exception type or its supertype.

1. What is method reference in Java 8?

Lambda expressions can be used to call an existing method. Java 8 provides another feature called method reference that provides a clearer alternative to refer to the existing method by name.

General form of Method reference- ClassName (or object)::methodName

class name or instance is separated from the method name by a double colon. The :: is a new separator (known as double colon operator) that has been added in Java 8.

1. Give an example of method reference.

A very simple example of method reference would be how we call System.out.println.

Suppose you are using a forEach statement to print all the elements of the list then the lambda expression for the same would be written as-

myList.forEach(x -> System.out.println(x))

Using method reference it can be written as -

list.forEach(System.out::println);

1. What are the types of method reference?

There are four kinds of method references:

|  |  |  |
| --- | --- | --- |
| Kind | Example | Syntax |
| Reference to a static method | ContainingClass::staticMethodName | ClassName::methodName |
| Reference to an instance method of a particular object | containingObject::instanceMethodName | objRef::methodName |
| Reference to an instance method of an arbitrary object of a particular type | ContainingType::methodName | ClassName::instanceMethodName |
| Reference to a constructor | ClassName::new | classname::new |

15) What is optional, and what is it best used for ?

Optional is a new container class defined in the java.util package, and used to represent optional values that either exist or do not exist. Optional’s chief benefit is avoiding null checks, and there are no longer any “NullPointerException” results at run-time.

16) What is Type Inference?

Type inference helps the compiler determine the argument types by looking at each method invocation and corresponding declaration.

17) Why are default methods needed in the interface?

Default methods let you add new functionality to your libraries’ interfaces and ensure binary compatibility with older code written for the interfaces