Java Lambda Expressions:

**1) Definition**

Java 8 Lambda Expressions can be defined as methods without names i.e anonymous functions. Like methods, they can have parameters, a body, a return type and possible list of exceptions that can be thrown. But unlike methods, neither they have names nor they are associated with any particular class.

**2) Lambda Syntax**

(Parameters) -> Expression

OR

(Parameters) -> { Statements }

Lambda syntax consist of three parts – list of parameters, an arrow mark and a body. The body of a lambda can be an expression or a set of statements. If it is set of statements, they must be enclosed within curly braces { }. Return type and possible list of exceptions that can be thrown are not explicitly mentioned in a lambda. They are implicitly applied.

**3) Where To Use Lambda Expressions?**

Lambda expressions are used where an instance of functional interface is expected. Functional interface is an interface which has only one abstract method. Functional interfaces can have any number of default methods. But, they must have only one abstract method.

@FunctionalInterface

public interface Comparator

{

    int compare(T o1, T o2);       //Only one abstract method

}

@FunctionalInterface

public interface Runnable

{

    public abstract void run();   //Only one abstract method

}

@FunctionalInterface

public interface ActionListener extends EventListener

{

    public void actionPerformed(ActionEvent e);  //Only One abstract method

}

Before Java 8, anonymous inner classes are used to implement functional interfaces. After Java 8, you can use lambda expressions to implement functional interfaces.

**4**) **How To Use Lambda Expressions?**

Lambda expressions are used to implement functional interfaces. Before Java 8, anonymous inner classes are used to implement functional interfaces. Let’s explore this with some before Java 8 and after Java 8 code.

Before Java 8 : Implementation of *Comparator* interface using anonymous inner class

Comparator<Student> idComparator = new Comparator<Student>() {

            @Override

            public int compare(Student s1, Student s2) {

                return s1.getID()-s2.getID();

            }

        };

After Java 8 : Implementation of *Comaparator* interface using lambda expressions

|  |  |
| --- | --- |
| 1 | Comparator<Student> idComparator = (Student s1, Student s2) -> s1.getID()-s2.getID(); |

Before Java 8 : Implementation of Runnable interface using anonymous inner class

Runnable r = new Runnable() {

            @Override

            public void run() {

                System.out.println("Runnable Implementation Using Anonymous Inner Class");

            }

        };

After Java 8 : Implementation of Runnable interface using lambda expressions

Runnable r = () -> System.out.println("Runnable Implementation Using Lambda Expressions");

Before Java 8 : Implementation Of ActionListener interface using anonymous inner class

Label label = new Label();

Button button = new Button("Send");

button.addActionListener(new ActionListener() {

    @Override

    public void actionPerformed(ActionEvent e) {

        label.setText("Sent");

    }

});

After Java 8 : Implementation of ActionListener interface using lambda expressions

Label label = new Label();

Button button = new Button("Send");

button.addActionListener((ActionEvent e) -> label.setText("Sent..."));

