Lambda Expressions

- It provides a clear and concise way to represent one method interface using an expression
- useful in collection library
- Before lambda expression, anonymous inner class was the only option to implement the method

Why to use Lamda Expression

1. To provide the implementation of Functional interface.
2. Less coding.

Java Anonymous inner class

- A class that have no name is known as anonymous inner class in java.
- It should be used if you have to override method of class or interface.

Java anonymous inner class example using interface

```
interface Eatable
{
 void eat();
}
class TestAnnonymousInner1
{
 public static void main(String args[])
{
 Eatable e=new Eatable(){
  public void eat(){System.out.println("nice fruits");}
 };
 e.eat();
 }
}
```

Internal class generated by the compiler

```
import java.io.PrintStream;
static class TestAnonymousInner1$1 implements Eatable
{
TestAnonymousInner1$1(){}
void eat(){System.out.println("nice fruits");}
}
```

```
Eatable p=new Eatable()
{
void eat(){System.out.println("nice fruits");}
};
```

- A class is created but its name is decided by the compiler which implements the Eatable interface and provides the implementation of the eat() method.
- An object of Anonymous class is created that is referred by p reference variable of Eatable type.

Java Lambda Expression Syntax

- **1) Argument-list:** It can be empty or non-empty as well.
- **2) Arrow-token:** It is used to link arguments-list and body of expression.
- **3) Body:** It contains expressions and statements for lambda expression.

without Lambda Expression

```
interface Drawable{
    public void draw();
}
public class LambdaExpressionExample
 {
    public static void main(String[] args) {
        int width=10;

        //without lambda, Drawable implementation using anonymous class
        Drawable d=new Drawable(){
            public void draw(){System.out.println("Drawing "+width);}
        };
        d.draw();
    }
}
```

with Lambda Expression

```
interface Drawable{
    public void draw();
}

public class LambdaExpressionExample {
    public static void main(String[] args) {
        int width=10;

        //with lambda
        Drawable d2=()->{                    //Public and method name draw is
                  repeated in anonymous class
            System.out.println("Drawing "+width);        // return type is also removed
                      because it is implicit from the code                    itself
        };
        d2.draw();
    }
}
```

Lambda Expression Example: No Parameter

```
interface Sayable{
    public String say();
}
public class LambdaExpressionExample{
public static void main(String[] args) {
    Sayable s=()->{
        return "I have nothing to say.";
    };
    System.out.println(s.say());
}
}
```

Single Parameter

```
interface Sayable{
   public String say(String name);
}

public class LambdaExpressionExample{
   public static void main(String[] args) {

      // Lambda expression with single parameter.
      Sayable s1=(name)->{
         return "Hello, "+name;
      };
      System.out.println(s1.say(“World"));

      // You can omit function parentheses
      Sayable s2= name ->{
         return "Hello, "+name;
      };
      System.out.println(s2.say(“World"));
   }
}
```

Multiple Parameters

```
interface Addable{
   int add(int a,int b);
}

public class LambdaExpressionExample{
   public static void main(String[] args) {

      // Multiple parameters in lambda expression
      Addable ad1=(a,b)->(a+b);
      System.out.println(ad1.add(10,20));

      // Multiple parameters with data type in lambda expression
      Addable ad2=(int a,int b)->(a+b);
      System.out.println(ad2.add(100,200));
   }
}
```

with or without return keyword

```
package lambdaExample;

interface Addable{
   int add(int a,int b);
}

public class lambdaExpression {
   public static void main(String[] args) {

      // Lambda expression without return keyword.
      Addable ad1=(a,b)->(a+b);
      System.out.println(ad1.add(10,20));

      // Lambda expression with return keyword.
      Addable ad2=(int a,int b)->{
                     return (a+b);
                     };
      System.out.println(ad2.add(100,200));
   }
}
```

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
class Product{
    int id;
    String name;
    float price;
    public Product(int id, String name, float price) {
        super();
        this.id = id;
        this.name = name;
        this.price = price;
    }
}
```

```
public class Lambda ExpressionExample{
   public static void main(String[] args) {
      List<Product> list=new ArrayList<Product>();


      //Adding Products
      list.add(new Product(1,"HP Laptop",25000f));
      list.add(new Product(3,"Keyboard",300f));
      list.add(new Product(2,"Dell Mouse",150f));
System.out.println("Sorting on the basis of name...");

      // implementing lambda expression
      Collections.sort(list,(p1,p2)->{
      return p1.name.compareTo(p2.name);
      });
      for(Product p:list){
         System.out.println(p.id+" "+p.name+" "+p.price);
      }

   }
}
```

Lamda as object

```
public interface MyComparator
 {

    public boolean compare(int a1, int a2);

}
MyComparator myComparator = (a1, a2) -> return a1 > a2;

boolean result = myComparator.compare(2, 5);
```