

A5 Project - Project Lambda

A brief explanation of my study of project lambda, an openjdk project.

Saksham Barara, B18190

Project lambda of OpenJDK aims to introduce lambda expressions and some more features to the java language.

1 Introduction

The project introduces the following new features to java-

1. Lambda expressions
2. Method references
3. Expanded target typing and type inference
4. Default and Static methods in Interfaces

2. Lambda Expressions

Lambda expressions are anonymous methods, aimed at replacing anonymous classes with a different mechanism.

The syntax is described below,

```
(parameters) -> expression or  
(parameters) -> { statements; }
```

Using lambda expressions deals with the following problems,

1. inline classes are too bulky, lambda expressions help us write them in a concise manner
2. make it easier to distribute processing of collections over multiple threads easier by making iteration inside the collections possible

3. Functional Interfaces

Functional Interface is an interface having only a single abstract method (having no implementation). A lambda expression is an instance of a functional interface. The lambda expression does not have any information about the functional interface it is implementing, instead it is obtained from the context in which it is used. The type expected for the lambda expression is called the target type of the expression. A single lambda expression can have different target types in different contexts.

4. Target types

There are some rules for the target types of lambda expressions.

1. It should be a functional Interface.

2. The expression has the same number of parameters as the functional interface. The types of these parameters should also be the same.
3. The return value of the expression should be the same as the interface's return type.
4. The lambda expression should throw the same type of exceptions that are allowed by the functional interface.

Due to target typing, the compiler already knows what types the formal parameters of the lambda expressions will have so there is no need to mention them again.

The following contexts have target types.

- Variable declarations
- Assignment statements
- return statements
- array initializers
- method or constructor arguments
- Lambda expression bodies
- Conditional expressions
- Cast expressions

5. Lexical scoping

All names are interpreted as in the enclosing environment, except for those declared through formal parameters which shadow the names declared in the outer scope. The words `this` and `super` have the same meaning as just outside the lambda expression, i.e, that of the enclosing class.

6. Method references

Method references work in a similar way as compared to lambda expressions. The only difference is that instead of providing the implementation in the like body lambda expressions method references point to already existing implementations of methods.

7. Default Methods

Default methods are methods in an interface which have an implementation. They make it easy to build APIs as we will always have a default implementation of the method in the interface and the code will not throw any error if the method is called without any external implementation.

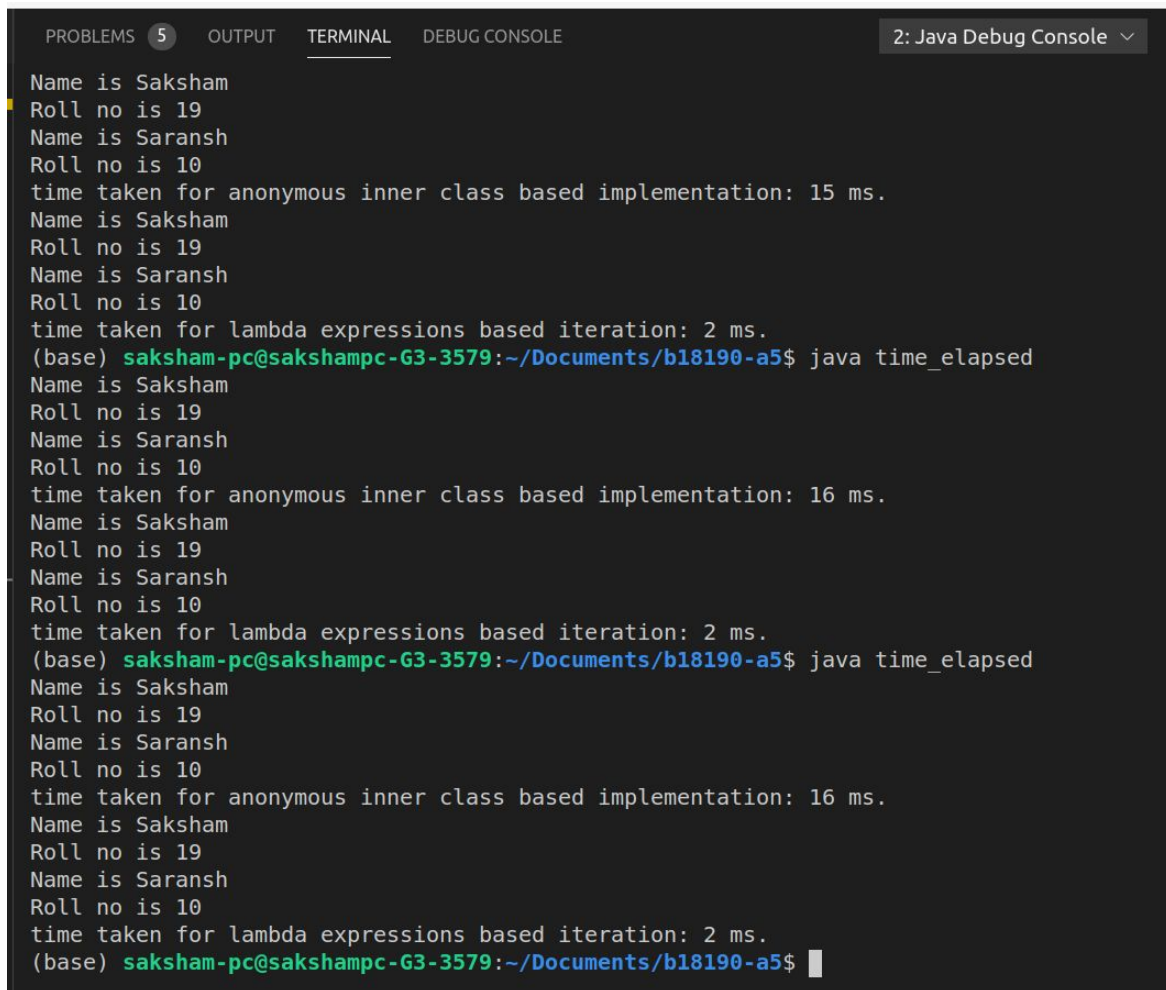
8. Benchmarks and Observations

I have written some code to test the features of project lambda.

- Test1.java has a simple basic lambda expression that is used inside the `ForEach` method of an `ArrayList`. The expression prints the square of the elements of the `ArrayList` to the system output.
- Test2.java uses a lambda expression(the one which returns the square of a number) implemented upon a custom functional interface. Notice that we do not need to provide types for input parameters of the lambda expression as they are inferred. In the case of Test2.java if commented out interface with `String` as types of formal parameters and return

type are used then we will get an error because it won't match with the type of values in arraylist.

- Test3.java shows a simple implementation of static method reference.
- Test4.java showcases the traversal of an array list first with anonymous inner classes and then with lambda expressions. The implementation with lambda expressions is very short and readable as compared to the anonymous inner classes.
- time_elapsed.java uses Instant and Duration classes in java to provide benchmarks for the code of Test4.java. The output shows that the execution time of lambda expression traversal is less than one with anonymous inner classes(refer to the below screenshot).



```
PROBLEMS 5 OUTPUT TERMINAL DEBUG CONSOLE 2: Java Debug Console v
Name is Saksham
Roll no is 19
Name is Saransh
Roll no is 10
time taken for anonymous inner class based implementation: 15 ms.
Name is Saksham
Roll no is 19
Name is Saransh
Roll no is 10
time taken for lambda expressions based iteration: 2 ms.
(base) saksham-pc@sakshampc-G3-3579:~/Documents/b18190-a5$ java time_elapsed
Name is Saksham
Roll no is 19
Name is Saransh
Roll no is 10
time taken for anonymous inner class based implementation: 16 ms.
Name is Saksham
Roll no is 19
Name is Saransh
Roll no is 10
time taken for lambda expressions based iteration: 2 ms.
(base) saksham-pc@sakshampc-G3-3579:~/Documents/b18190-a5$ java time_elapsed
Name is Saksham
Roll no is 19
Name is Saransh
Roll no is 10
time taken for anonymous inner class based implementation: 16 ms.
Name is Saksham
Roll no is 19
Name is Saransh
Roll no is 10
time taken for lambda expressions based iteration: 2 ms.
(base) saksham-pc@sakshampc-G3-3579:~/Documents/b18190-a5$
```

The output of time_elapsed.java after being executed a number of times

9. Conclusion

Lambda expression and method references help us to write better code which is easier to understand. Since the time of inclusion into the java language they have transformed the way in which people write object oriented code.