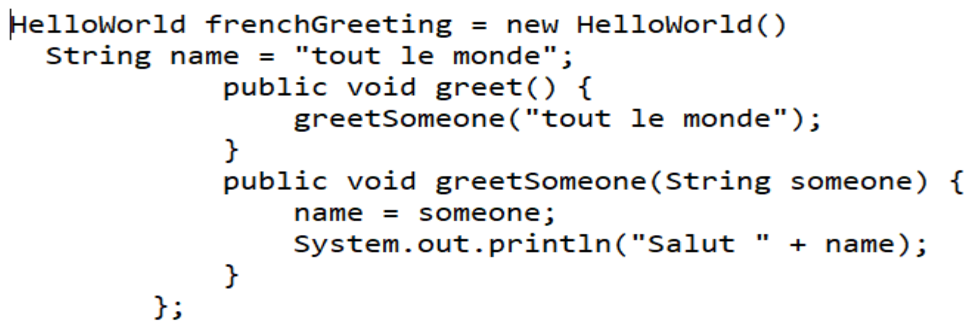
EXPERIMENT NO. 16

1.What is anonymous class ?

Ans –

Anonymous classes enable you to make your code more concise. They enable you to declare and instantiate a class at the same time. They are like local classes except that they do not have a name. Use them if you need to use a local class only once . Consider the instantiation of the frenchGreeting object:



```
HelloWorld frenchGreeting = new HelloWorld()
  String name = "tout le monde";
            public void greet() {
                greetSomeone("tout le monde");
            }
            public void greetSomeone(String someone) {
                name = someone;
                System.out.println("Salut " + name);
            }
        };
```

The anonymous class expression consists of the following:

1.The new operator

2.The name of an interface to implement or a class to extend. In this example, the anonymous class is implementing the interface HelloWorld. Parentheses that contain the arguments to a constructor, just like a normal class instance creation expression.

Note: When you implement an interface, there is no constructor, so you use an empty pair of parentheses, as in this example.

3.A body, which is a class declaration body. More specifically, in the body, method declarations are allowed but statements are not.

2. What is anonymus function ?

Ans –

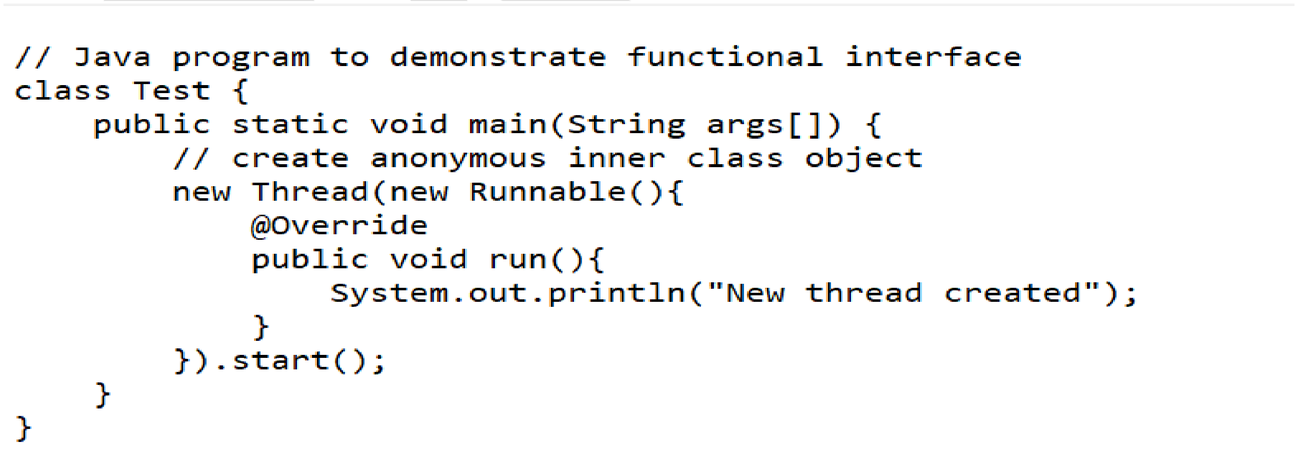
Anonymous function is a function definition that is not bound to an identifier. These are a form of nested function, in allowing access to variables in the scope of the containing function (non-local functions). This means anonymous functions need to be implemented using closures. Simply, lambda is an anonymous function which can be passed around in a concise way . A lambda expression represents an anonymous function. It comprises of a set of parameters , a lambda operator (->) and a function body.

3.What is functional interface ? Give an example.

Ans –

A functional interface is an interface that contains only one abstract method. They can have only one functionality to exhibit. From Java 8 onwards, lambda expressions can be used to represent the instance of a functional interface. A functional interface can have any number of default methods. Runnable, ActionListener, Comparable are some of the examples of functional interfaces. Before Java 8, we had to create anonymous inner class objects or implement these interfaces.

Eg-



```
// Java program to demonstrate functional interface
class Test {
    public static void main(String args[]) {
        // create anonymous inner class object
        new Thread(new Runnable(){
            @Override
            public void run(){
                System.out.println("New thread created");
            }
        }).start();
    }
}
```

4.What is lambda expression in java ? Give an example of zero ,one and multiple parameter function using lambda expression.

Ans –

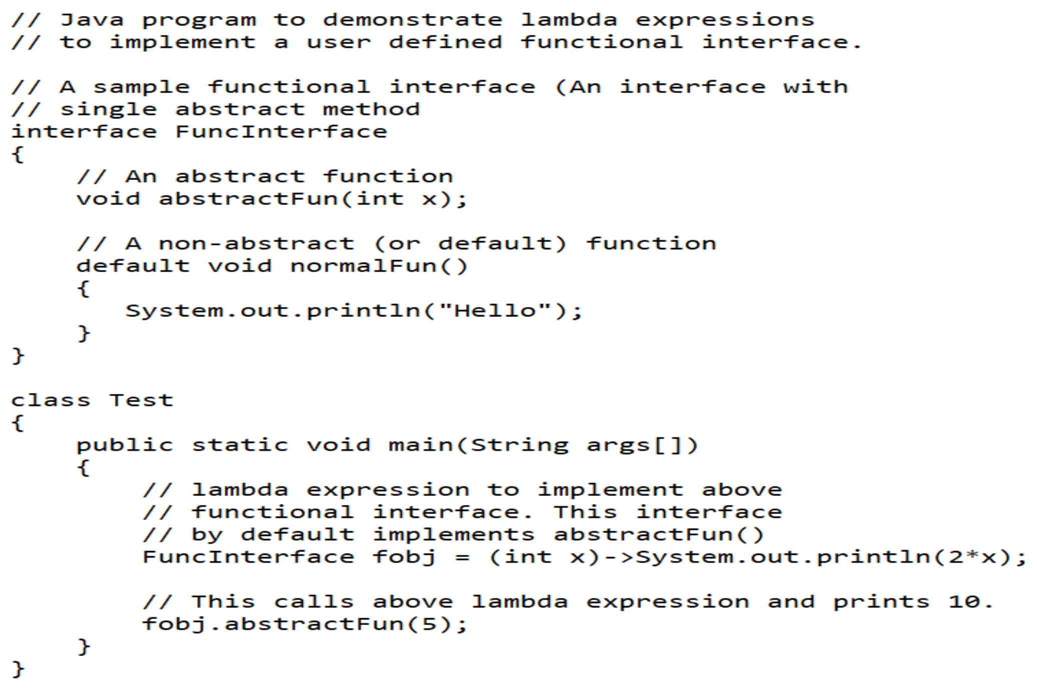
Lambda expressions basically express instances of functional interfaces (An interface with single abstract method is called functional interface. An example is java.lang.Runnable). lambda expressions implement the only abstract function and therefore implement functional interfaces

lambda expressions are added in Java 8 and provide below functionalities.

Enable to treat functionality as a method argument, or code as data.

A function that can be created without belonging to any class.

A lambda expression can be passed around as if it was an object and executed on demand.



```
// Java program to demonstrate lambda expressions
// to implement a user defined functional interface.

// A sample functional interface (An interface with
// single abstract method
interface FuncInterface
{
    // An abstract function
    void abstractFun(int x);

    // A non-abstract (or default) function
    default void normalFun()
    {
       System.out.println("Hello");
    }
}

class Test
{
    public static void main(String args[])
    {
        // lambda expression to implement above
        // functional interface. This interface
        // by default implements abstractFun()
        FuncInterface fobj = (int x)->System.out.println(2*x);

        // This calls above lambda expression and prints 10.
        fobj.abstractFun(5);
    }
}
```

- Zero parameter:

  () -> System.out.println("Zero parameter lambda");

- One parameter:–

(p) -> System.out.println("One parameter: " + p);

It is not mandatory to use parentheses, if the type of that variable can be inferred from the context

- Multiple parameters :

(p1, p2) -> System.out.println("Multiple parameters: " + p1 + ", " + p2);