

# Chuyên đề mở rộng

## More on Functional Programming

### &

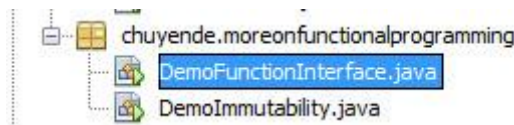
## Additional Features of Java 8

### Mục tiêu

- ✓ Tạo được các Functional Interface sử dụng để đơn giản hóa mã nguồn bằng cấu trúc lambda
- ✓ Xác định tính bất biến của đối tượng trong Java
- ✓ Sử dụng engine Nashorn để gọi và thực thi Javascript từ mã nguồn Java.

### Bài thực hành số 1:

**Yêu cầu:** tạo lớp DemoFunctionInterface, trong đó tạo các phương thức demo với các Functional Interface được giới thiệu trong Java 8.



### Code tham khảo:

```
package chuyende.moreonfunctionalprogramming;
```

```
import java.util.function.BiPredicate;  
import java.util.function.Function;  
import java.util.function.IntConsumer;  
import java.util.function.Supplier;  
import java.util.function.UnaryOperator;
```

```
/**  
 *  
 * @author minhvuvc  
 */  
public class DemoFunctionInterface {  
  
    /**  
     * Demo với FI Function<T,R>
```

\* CHẤP NHẬN 1 ĐỐI SỐ VÀ TẠO RA 1 KẾT QUẢ

\*/

```
private void demoFunction() {
```

```
    // 1. Trường minh
```

```
    Function<Integer, String> f = new Function<Integer, String>() {
```

```
        @Override
```

```
        public String apply(Integer t) {
```

```
            return "#" + t; // Trả về số nguyên thêm tiền tố #
```

```
        }
```

```
    };
```

```
    // 2. Lambda
```

```
    Function<Integer, Integer> f1 = (t) -> {
```

```
        return t * t; // Trả về bình phương số nguyên
```

```
    };
```

```
    System.out.println(f1.andThen(f).apply(5)); // Thực hiện f1 >> f
```

```
    System.out.println(f.compose(f1).apply(3)); // Trước khi thực hiện f thì làm  
    TRƯỚC f1
```

```
}
```

```
/**
```

```
* CHẤP NHẬN 1 ĐỐI SỐ VÀ KHÔNG TRẢ VỀ KẾT QUẢ NÀO
```

```
*/
```

```
private void demoConsumer() {
```

```
    // 1. Trường minh
```

```
    IntConsumer ic = new IntConsumer() {
```

```
        @Override
```

```
        public void accept(int value) {
```

```
            throw new UnsupportedOperationException("Not supported yet."); //To  
change body of generated methods, choose Tools | Templates.
```

```
        }
```

```
    };
```

```
    // 2. Lambda
```

```
    IntConsumer ic1 = (number) -> {
```

```
        System.out.println("Cube = " + (Math.pow(number, 3)));
```

```
    };
```

```
    ic1.accept(3);
```

```
}
```

```
/**
```

```
* NHẬN VÀO ĐỐI SỐ VÀ TRẢ VỀ GIÁ TRỊ BOOLEAN
```

```

*/
private void demoPredicate() {
    // 1. Tường minh
    BiPredicate<Integer, String> bp = new BiPredicate<Integer, String>() {
        @Override
        public boolean test(Integer t, String u) {
            throw new UnsupportedOperationException("Not supported yet."); //To
change body of generated methods, choose Tools | Templates.
        }
    };

    // 2. Lambda
    BiPredicate<Integer, String> bp1 = (t, u) -> {
        return t == Integer.parseInt(u); // Kiểm tra số (dạng chuỗi) có bằng số
nguyên?.
    };

    System.out.println(bp1.test(6, "8") ? "Match" : "Not match");
}

/**
 * THỰC HIỆN PHÉP TÍNH TRÊN THAM SỐ ĐẦU VÀO VÀ TRẢ VỀ KẾT
QUẢ
 */
private void demoOperator() {
    // 1. Tường minh
    UnaryOperator<Float> uo = new UnaryOperator<Float>() {
        @Override
        public Float apply(Float t) {
            throw new UnsupportedOperationException("Not supported yet."); //To
change body of generated methods, choose Tools | Templates.
        }
    };

    // 2. Lambda
    UnaryOperator<Float> uo1 = (t) -> {
        return new Float(t * t * Math.PI); // Trả về diện tích hình tròn.
    };

    System.out.println("Square of Circle has radius is 5: " + uo1.apply(5f));
}

/**
 * KHÔNG CÓ THAM SỐ ĐẦU VÀO VÀ TRẢ VỀ KẾT QUẢ

```

```

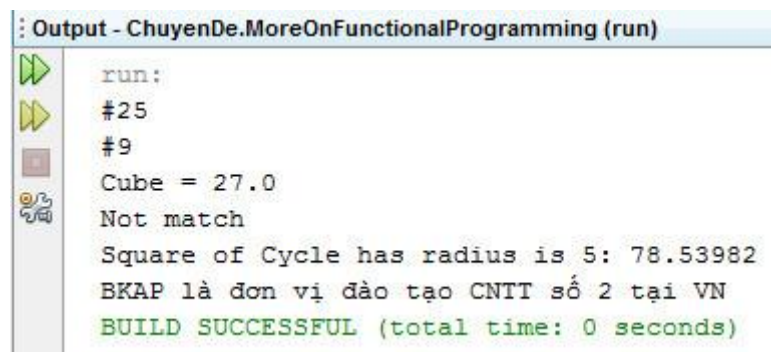
*/
private void demoSupplier() {
    // 1. Thường mình
    Supplier<String> s = new Supplier<String>() {
        @Override
        public String get() {
            throw new UnsupportedOperationException("Not supported yet."); //To
change body of generated methods, choose Tools | Templates.
        }
    };

    // 2. Lambda
    Supplier<String> s1 = () -> {
        return "BKAP là đơn vị đào tạo CNTT số 2 tại VN";
    };

    System.out.println(s1.get());
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    DemoFunctionInterface dfi = new DemoFunctionInterface();
    dfi.demoFunction();
    dfi.demoConsumer();
    dfi.demoPredicate();
    dfi.demoOperator();
    dfi.demoSupplier();
}
}

```



The screenshot shows the 'Output - ChuyenDe.MoreOnFunctionalProgramming (run)' window. It displays the following text:

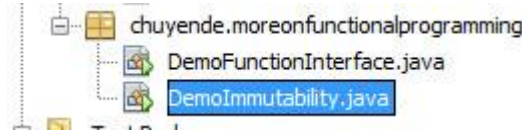
```

run:
#25
#9
Cube = 27.0
Not match
Square of Cycle has radius is 5: 78.53982
BKAP là đơn vị đào tạo CNTT số 2 tại VN
BUILD SUCCESSFUL (total time: 0 seconds)

```

## Bài thực hành số 2:

**Yêu cầu:** tạo lớp DemoImmutability, trong đó tạo inner class Person và hàm changeName(), changeString(). Truyền tham số tương ứng để xác định tính bất biến của đối tượng trong Java



### Code tham khảo:

```
package chuyende.moreonfunctionalprogramming;
```

```
/**
```

```
*
```

```
* @author minhvufo
```

```
*/
```

```
public class DemoImmutability {
```

```
    class Person {
```

```
        String name;
```

```
        int age;
```

```
        public Person(String name, int age) {
```

```
            this.name = name;
```

```
            this.age = age;
```

```
        }
```

```
    }
```

```
    private void changeName(Person per) {
```

```
        per.name = "Tiểu màn thầu";
```

```
    }
```

```
    private void changeString(String str) {
```

```
        str = "New String";
```

```
    }
```

```
    private void demoFixity() {
```

```
        Person p = new Person("Minh", 0);
```

```
        changeName(p);
```

```
        System.out.println("Name: " + p.name);
```

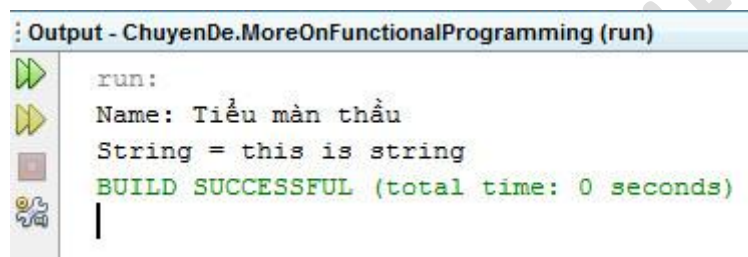
```
        String test = "this is string";
```

```

changeString(test);
System.out.println("String = " + test);
}

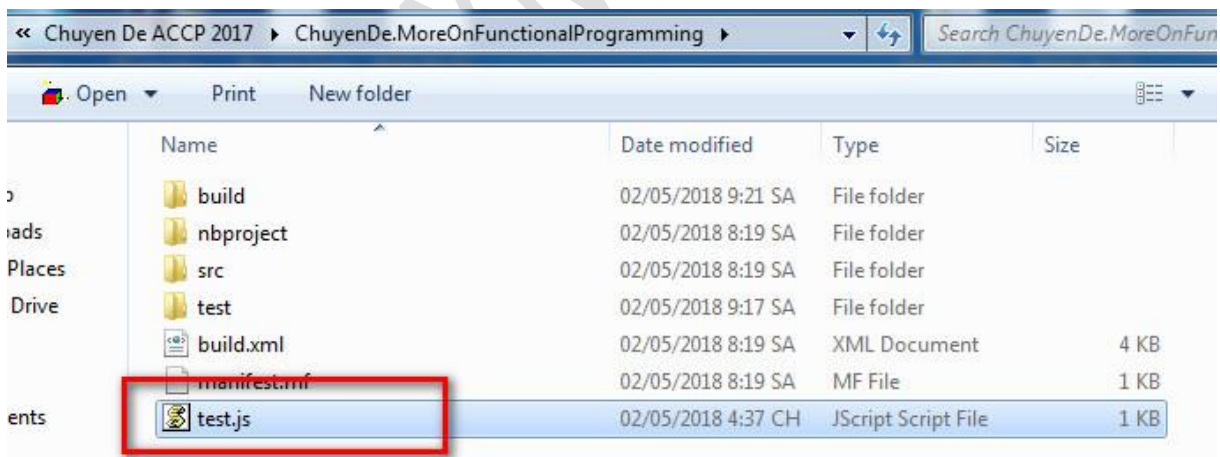
/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    DemoImmutability main = new DemoImmutability();
    main.demoFixity();
}
}

```

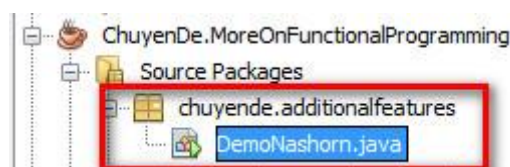


### Bài thực hành số 3:

**Yêu cầu:** Trong thư mục gốc project, tạo file javascript tên là test.js. Tạo file java trong project như hình. Viết mã nguồn gọi và thực thi javascript từ Java.



*File javascript*



*File Java*

## Code tham khảo:

### test.js

```
var a = 3;
var b = 5;
print('Hi, Vietnam Vodich ' + (a + b));

var func1 = function(name){
    print("Welcome " + name + " to the world of Javascript!");
    return "End call....";
}
```

### DemoNashorn.java

```
package chuyende.additionalfeatures;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.script.Invocable;
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
import javax.script.ScriptException;

/**
 *
 * @author minhvuvc
 */
public class DemoNashorn {

    /**
     * Hàm gọi và thực thi mã Javascript từ file
     */
    private void runScript() {
        try {
            ScriptEngineManager scriptEngineManager = new ScriptEngineManager();
            ScriptEngine nashorn =
scriptEngineManager.getEngineByName("nashorn");

            nashorn.eval(new FileReader("test.js"));

            Invocable invocable = (Invocable) nashorn;
            Object obj = invocable.invokeFunction("func1", "MinhVT");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
        System.out.println(obj);
    } catch (FileNotFoundException ex) {
        Logger.getLogger(DemoNashorn.class.getName()).log(Level.SEVERE,
null, ex);
    } catch (ScriptException ex) {
        Logger.getLogger(DemoNashorn.class.getName()).log(Level.SEVERE,
null, ex);
    } catch (NoSuchMethodException ex) {
        Logger.getLogger(DemoNashorn.class.getName()).log(Level.SEVERE,
null, ex);
    }
}

/**
 * @param args the command line arguments
 */
public static void main(String[] args) {
    DemoNashorn dn = new DemoNashorn();
    dn.runScript();

    // ===== Chạy script trực tiếp =====

    System.out.println("=====
=====");
    ScriptEngineManager scriptEngineManager = new ScriptEngineManager();
    ScriptEngine nashorn = scriptEngineManager.getEngineByName("nashorn");

    String name = "Bachkhoa-Aptech";
    Integer result = null;

    try {
        nashorn.eval("print('' + name + '')"); //
        result = (Integer) nashorn.eval("10 + 2");
        System.out.println("-----");
        String jsScript = "var a = 3, b = 2;"
            + "print('Sum = ' + (a + b))";
        nashorn.eval(jsScript);

    } catch (ScriptException e) {
        System.out.println("Error executing script: " + e.getMessage());
    }
    System.out.println(result.toString());

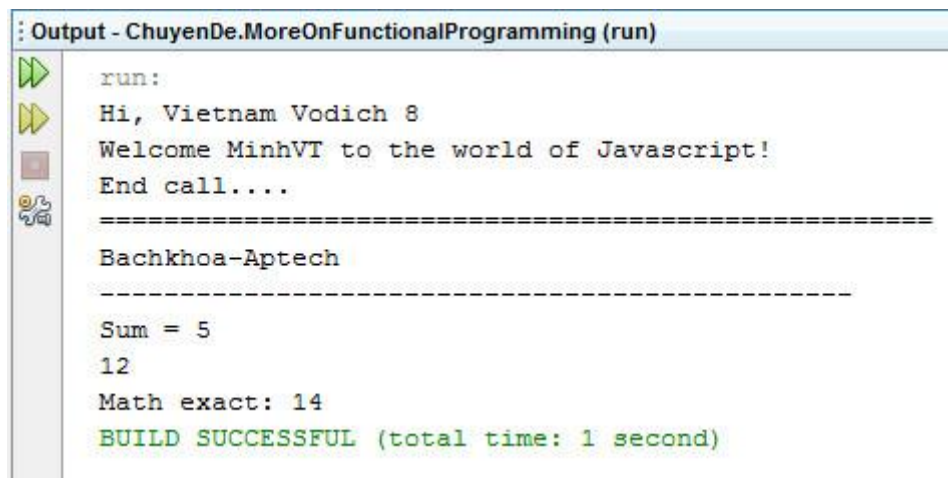
    int ex1 = 9;
    int ex2 = 5;
```



```

    System.out.println("Math exact: " + Math.addExact(ex1, ex2));
}
}

```



The screenshot shows the output of a Java program. The window title is "Output - ChuyenDe.MoreOnFunctionalProgramming (run)". The output text is as follows:

```

run:
Hi, Vietnam Vodich 8
Welcome MinhVT to the world of Javascript!
End call....
=====
Bachkhoa-Aptech
-----
Sum = 5
12
Math exact: 14
BUILD SUCCESSFUL (total time: 1 second)

```