Date : 16/12/2020
Spring Boot 9AM
Mr. RAGHU

---

*) Java 8 has provided one package ' java.util.function ' that contains
   pre-defined functional interfaces. Those can be used to write Lambda
   Expressions and Method References.

https://docs.oracle.com/javase/8/docs/api/java/util/function/package-summary.html

---Ex#1:----------

```
package in.nareshit.raghu;

interface Math {
   String add(int a,int b);
}

public class Test {

        public static void main(String[] args) {
                //Interface ob = (params) -> { body  }
                Math m = (a,b) -> a+b;

                //call method
                int result = m.add(10, 20);
                System.out.println(result);
        }
}
```

*) Above Math (FI) is having one abstract method
   that takes 2 inputs and returns one value.
*) java has provided BiFunction(I) that matches to above
   requirement.

```
   interface BiFunction<T, U, R> {
      R apply(T t, U u);
   }
```

   T = 1st param data type
   U = 2nd param data type
   R = output DataType

*** Do not compare dataTypes, as they decided at runtime.
   Just compare no.of inputs(arguments) and return type is void/non-void

---Ex#2-------

```
package in.nareshit.raghu;

import java.util.function.BiFunction;

public class Test {
```

```
        public static void main(String[] args) {
                //Interface ob = (params) -> { body  }
                BiFunction<Integer, Double, Double> m = (a,b) -> a+b;

                //call method
                Double result = m.apply(10, 20.25);
                System.out.println(result);
        }
}
```

==========

Interface name --Compare --? NO
abstract method name --> ? No
DataTypes ---? NO

No.of method params--? YES
Return Type is void/non-void ---? YES

--Ex#3----------

```
package in.nareshit.raghu;

interface Sample {
    String show();
}

public class Test {

        public static void main(String[] args) {
                //Interface ob = (params) -> { body  }
                Sample ob = () -> { return "WELCOME"; };
                //call method
                String msg = ob.show();
                System.out.println(msg);
        }
}
```

Find out a function interface that takes 0 input and given 1 output (non-void)

--Ex#4--------------

```
package in.nareshit.raghu;

import java.util.function.Supplier;

public class Test {

        public static void main(String[] args) {
                //Interface ob = (params) -> { body  }
                Supplier<String> ob = () -> { return "WELCOME"; };
                //call method
                String msg = ob.get();
                System.out.println(msg);
        }
}
```

Spring Java based configuration

```
@Configuration
public class ____ {

   // 1 object = 1 method
   @Bean
   public <className/InterfaceName> <objectName>(){
      //logic
      return ob;
   }

}
```

CommandLineRunner(CLR) as Lambda Expression

*) Here CLR is a functional interface. So, we can define one Lambda Expression
   By using java based configuration.

#1. Create Spring Starter Project
 Name : SpringBoot2RunnerasLambda
 Package: in.nareshit.raghu

#2. Java configuration file

```
package in.nareshit.raghu.config;

import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AppConfig {

        @Bean
        public CommandLineRunner clrOb() {
                // interface ob = () -> { }
                /*
                CommandLineRunner c = (args) -> {
                        System.out.println("FROM LAMDBA BASIC
EXAMPLE");
                };
                return c;
                */

                return (args) -> System.out.println("FROM LAMDBA BASIC
EXAMPLE");

        }
}
```

***) run() method is called as start class when we start/run our main class.

==========

Config Properties using class and object (HAS-A Relation)

In this case use below syntax to define keys in properties

```
   prefix.hasAVariable.variable=<value>
```

#1 Create Spring Boot application
Name : SpringBoot2RunnerConfigPropsHasARelation
Package: in.nareshit.raghu

#2. Model class

```
package in.nareshit.raghu.model;

public class Product {

	private int pid;
	private String pname;
	private double pcost;

	public int getPid() {
		return pid;
	}
	public void setPid(int pid) {
		this.pid = pid;
	}
	public String getPname() {
		return pname;
	}
	public void setPname(String pname) {
		this.pname = pname;
	}
	public double getPcost() {
		return pcost;
	}
	public void setPcost(double pcost) {
		this.pcost = pcost;
	}
	@Override
	public String toString() {
		return "Product [pid=" + pid + ", pname=" + pname + ",
pcost=" + pcost + "]";
	}


}
```

#3. Runner class

```
package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;
import
org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Product;

@Component
@ConfigurationProperties(prefix = "my.app")
```

```
public class ObjectConfigPropesRunner
        implements CommandLineRunner
{

        private Product pob; //HAS-A

        public void run(String... args) throws Exception {
                System.out.println(pob);
        }

        public Product getPob() {
                return pob;
        }

        public void setPob(Product pob) {
                this.pob = pob;
        }


}

#4 application.properties
my.app.pob.pid=10
my.app.pob.pname=PEN
my.app.pob.pcost=500.2
```

==========

```
Task#1
 Define one CommandLine Runner
 create variables for student (sid,sname,sfee, subjects:List<String>,
grade)
 read data from properties files using @ConfigurationProperties

Task#2
 Define manual properties files (sample.properties, admin.properties)
 Create one class Information(id,code,model)
 Read data using @Value
 id,code <--- Read from sample.properties
 model <--- admin.properties

Task#3
 Consider given class as pre-defined (you only create it under
project)

 class JpaData {
   String dialect;
   boolean showSql;
   String ddlAuto;
 }

 Create this object using Java based Configuration and Read and print
using
 Starter class/Runner class.

*) Task: CODE: 16122020 TO: javabyraghu@gmail.com
```
