Function<T,R> functional interface :

* It is a predefined functional interface available in java.util.function sub package.

* Function accepts two type parameters i.e T and R. 'T' represents "type of input" to the function where 'R' represents "return result".

```
@FunctionalInterface
public interface Function<T,R>
{
    public abstract R apply(T t);
}
```

* It has an abstract method apply() which accept T as a parameter and R is the return result.

* Here Input type and return result, both will be decided by developer.

//Program :

```
package com.ravi.function_demo;

import java.util.Scanner;
import java.util.function.Function;

//Finding the cube of a number

public class FunctionDemo1
{
    public static void main(String[] args)
    {
        Function<Integer, Integer> fn1 = num -> num*num*num;

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a Number :");
        int num = sc.nextInt();

        System.out.println("Cube of " +num+ " is :"+fn1.apply(num));
        sc.close();
    }

}
```

```
package com.ravi.function_demo;

import java.util.Scanner;
import java.util.function.Function;

//Finding the length of the given name

public class FunctionDemo2
{
    public static void main(String[] args)
    {
        Function<String, Integer> fn2 = name -> name.length();

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter Your Name :");
        String name = sc.nextLine();

        Integer length = fn2.apply(name);

        System.out.println("Length of "+name+ " is :"+length);
        sc.close();
    }

}
```

```
package com.ravi.function_demo;

import java.util.Scanner;
import java.util.function.Function;

//Verify whether my name is Virat Or not
public class FunctionDemo3
{
    public static void main(String[] args)
    {
        Function<String, Boolean> fn3 = str -> str.equalsIgnoreCase("Virat");

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your Name :");
        String name = sc.nextLine();
        System.out.println("Are you Virat ?"+fn3.apply(name));
        sc.close();
    }

}
```

Supplier<T>

* It is a predefined functional interface available in java.util.function sub package.

```
@FunctionalInterface
public interface Supplier<T>
{
    T get();
}
```

* It accepts abstract method get() which does not accept any parameter but the return type is T.

* It is mainly used to supply the given Type parameter.

```
package com.ravi.function_demo;

import java.util.function.Supplier;

public class SupplierDemo1
{
    public static void main(String[] args)
    {
        Supplier<String> s1 = () -> 100+""+200;
        System.out.println(s1.get());
    }

}
```

```
package com.ravi.supplier;

import java.util.function.Supplier;

class Employee
{
    private Integer id;
    private String name;
    private Double salary;

    public Employee(Integer id, String name, Double salary)
    {
        super();
        this.id = id;
        this.name = name;
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";
    }
}

public class SupplierDemo2
{
    public static void main(String[] args)
    {
        Supplier<Employee> s2 = () ->
        {
            return new Employee(111, "Alen", 65000d);
        };

        Employee obj = s2.get();
        System.out.println(obj);

    }

}
```

```
package com.ravi.supplier;

import java.util.Scanner;
import java.util.function.Supplier;

class Product
{
    private Integer id;
    private String name;
    private Double price;

    public Product(Integer id, String name, Double price)
    {
        super();
        this.id = id;
        this.name = name;
        this.price = price;
    }

    @Override
    public String toString()
    {
        return "Product [id=" + id + ", name=" + name + ", price=" + price + "]";
    }
}
public class SupplierDemo3
{
    public static void main(String[] args)
    {
        Supplier<Product> s3 = () ->
        {
            Scanner sc = new Scanner(System.in);
            System.out.print("Enter Product Id :");
            int id = Integer.parseInt(sc.nextLine());

            System.out.print("Enter Product Name :");
            String name = sc.nextLine();

            System.out.print("Enter Product Price :");
            double price = Double.parseDouble(sc.nextLine());

            return new Product(id, name, price);
        };

        Product obj = s3.get();
        System.out.println(obj);
    }

}
```

Can we develop our own Functional Interface ?

* Yes, We can develop our own functional interface by using Type Parameter.

```
package com.ravi.supplier;

//Custom Object
@FunctionalInterface
interface TriFunction<T,U,R>
{
    public abstract R myApply(T a, U b);
}

public class CustomFunctionalInterface
{
    public static void main(String[] args)
    {
        TriFunction<Integer, Integer, String> fn1 =   (a, b) -> a+""+b;
        System.out.println(fn1.myApply(100, 200));
    }
}
```