//Program on Type Parameter :

```java
package com.ravi.type_parameter_demo;

class Box<T>
{
    private T data;

    public Box(T data)   //Student data
    {
        super();
        this.data = data;
    }

    public T getData()
    {
        return data;
    }
}

public class TypeParameterDemo1 {
    public static void main(String[] args) {
        Box<Integer> intType = new Box<Integer>(15);
        System.out.println("Integer Type :" + intType.getData());

        Box<Double> douType = new Box<Double>(90.67);
        System.out.println("Double Type :" + douType.getData());

        Box<Character> charType = new Box<Character>('A');
        System.out.println("Character Type :" + charType.getData());

        Box<Student> studentType = new Box<Student>(new Student(111, "Scott"));
        System.out.println("Student type :"+studentType.getData());

    }

}

class Student
{
    private Integer id;
    private String name;

    public Student(Integer id, String name)
    {
        super();
        this.id = id;
        this.name = name;
    }

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

Program on Lambda Expression :

```java
package com.ravi.lambda;

import java.util.Scanner;

@FunctionalInterface
interface Predictable
{
    boolean predict(Integer num);
}

public class LambdaExample1
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the Age of the voter :");
        int ag = Integer.parseInt(sc.nextLine());

        Predictable p1 = age -> age>=18;
        System.out.println("Is Person Eligible for Voting ? :"+p1.predict(ag));
        sc.close();

    }

}
```

Working with predefined functional interfaces :

----

* In order to work with day to day programming and to write concise coding, Java software people has provided the following predefined functional interfrace which are available in java.util.function sub package.

1) Predicate<T>
2) Consumer<T>
3) Function<T,R>
4) Supplier<T>
5) BiPredicate<T,U>
6) BiConsumer<T,U>
7) BiFunction<T,R,U>
8) UnaryOperator<T>
9) BinaryOperator<T,T>

Note : We have total 43 predefined functional interfaces.

Predicate<T> Functional interface :

----

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

```
@FunctionalInterface
public interface Predicate<T>
{
    boolean test(T x);
}
```

* It contains an abstract method test() which accepts 'T' as a parameter and return type of this method is boolean.

* IT IS MAINLY USED TO VERIFY **"ONE ARGUMENT BOOLEAN EXPRESSION"**

//Programs :

```java
package com.ravi.predicate;

import java.util.Scanner;
import java.util.function.Predicate;

//Verify whether a number is even or odd
public class PredicateDemo1
{
    public static void main(String[] args)
    {
        Predicate<Integer> p1 =  num ->  num % 2==0;

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a Number :");
        int num = sc.nextInt();
        System.out.println("Is "+num+" Even number ?"+p1.test(num));
        sc.close();

    }

}
```

```java
package com.ravi.predicate;

import java.util.Scanner;
import java.util.function.Predicate;

//Verify whether my name starts with character 'R' or not ?

public class PredicateDemo2
{
    public static void main(String[] args)
    {
        Predicate<String> p2 = name -> name.startsWith("R");

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter your Name :");

        String name = sc.next();
        System.out.println("Is "+name+" starts with Character 'R' "+p2.test(name));
        sc.close();

    }

}
```

```java
package com.ravi.predicate;

import java.util.Scanner;
import java.util.function.Predicate;

//Verify whether the name is "Sachin" or not

public class PredicateDemo3
{
    public static void main(String[] args)
    {
        Predicate<String> p3 = name -> name.equalsIgnoreCase("Sachin");
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter your Name :");
        String name = sc.next();

        System.out.println("Are you Sachin ? "+p3.test(name));
        sc.close();
    }
}
```

Consumer<T>

----

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

```
@FunctionalInterface
public interface Consumer<T>
{
    void accept(T x);
}
```

* It has one abstract method accept() which takes 'T' as a parameter and returns nothing.

* IT IS USED TO ACCEPT/CONSUME THE VALUE WITHOUT RETURNING ANYTHING.

```java
package com.ravi.consumer;

import java.util.function.Consumer;

class Product
{
    private Integer id;
    private String name;

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

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

}

public class ConsumerDemo1
{
    public static void main(String[] args)
    {
        Consumer<Float> c1 = flt -> System.out.println("Float value is :"+flt);
        c1.accept(78.56F);

        Consumer<Boolean> c2 = bool -> System.out.println("Boolean value is :"+bool);
        c2.accept(true);

        Consumer<Product> c3 = prod -> System.out.println("Product object is :"+prod);
        c3.accept(new Product(222, "Laptop"));
    }

}
```