Primitive Type Functional Interfaces

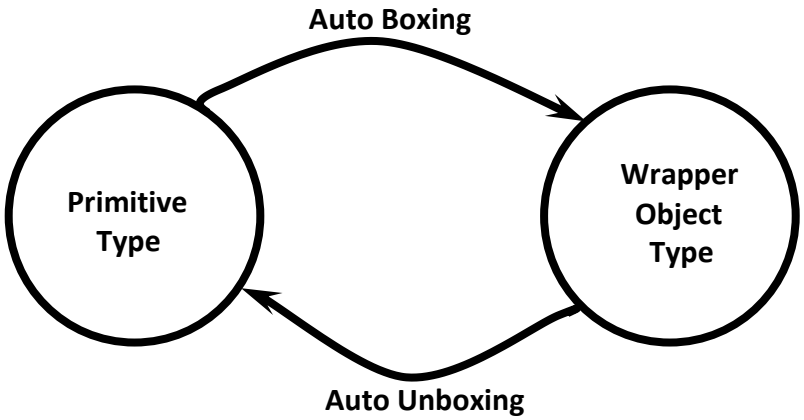
What is the Need of Primitive Type Functional Interfaces?

1. Autoboxing:

Automatic conversion from primitive type to Object type by compiler is called autoboxing.

2. Autounboxing:

Automatic conversion from object type to primitive type by compiler is called autounboxing.



3. In the case of generics, the type parameter is always object type and no chance of passing primitive type.

```
ArrayList<Integer> l = new ArrayList<Integer>();//valid
ArrayList<int> l = new ArrayList<int>();//invalid
```

Need of Primitive Functional Interfaces:

In the case of normal Functional interfaces (like Predicate, Function etc) input and return types are always Object types. If we pass primitive values then these primitive values will be converted to Object type (Auboxing), which creates performance problems.

```
1)  import java.util.function.Predicate;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        int[] x ={0,5,10,15,20,25};
```

```
7)        Predicate<Integer> p=i->i%2==0;
8)        for (int x1 : x)
9)        {
10)         if(p.test(x1))
11)         {
12)           System.out.println(x1);
13)         }
14)       }
15)    }
16) }
```

In the above examples,6 times autoboxing and autounboxing happening which creates Performance problems.

To overcome this problem primitive functional interfaces introduced, which can always takes primitive types as input and return primitive types.Hence autoboxing and autounboxing won't be required,which improves performance.

Primitive Type Functional Interfaces for Predicate:

The following are various primitive Functional interfaces for Predicate.
1.IntPredicate-->always accepts input value of int type
2.LongPredicate-->always accepts input value of long type
3.DoublePredicate-->always accepts input value of double type

1.

```
1)  interface IntPredicate
2)  {
3)    public boolean test(int i);
4)    //default methods: and(),or(),negate()
5)  }
```

2.

```
1)  interface LongPredicate
2)  {
3)    public boolean test(long l);
4)    //default methods: and(),or(),negate()
5)  }
```

3.

```
1)  interface DoublePredicate
2)  {
3)    public boolean test(double d);
4)    //default methods: and(),or(),negate()
5)  }
```

Demo Program for IntPredicate:

```
1)  import java.util.function.IntPredicate;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        int[] x ={0,5,10,15,20,25};
7)        IntPredicate p=i->i%2==0;
8)        for (int x1 : x)
9)        {
10)         if(p.test(x1))
11)         {
12)           System.out.println(x1);
13)         }
14)       }
15)    }
16) }
```

In the above example, autoboxing and autounboxing won't be performed internally.Hence performance wise improvements are there.

Primitive Type Functional Interfaces for Function:

The following are various primitive Type Functional Interfaces for Function

1. IntFunction: can take int type as input and return any type
public R apply(int i);

2. LongFunction: can take long type as input and return any type
public R apply(long i);

3. DoubleFunction: can take double type as input and return any type
public R apply(double d);

4. ToIntFunction: It can take any type as input but always returns int type
public int applyAsInt(T t)

5. ToLongFunction: It can take any type as input but always returns long type
public long applyAsLong(T t)

6. ToDoubleFunction: It can take any type as input but always returns double type
public int applyAsDouble(T t)

7. IntToLongFunction: It can take int type as input and returns long type
public long applyAsLong(int i)

8. IntToDoubleFunction: It can take int type as input and returns long type
public double applyAsDouble(int i)

9. LongToIntFunction: It can take long type as input and returns int type
public int applyAsInt(long i)

10. LongToDoubleFunction: It can take long type as input and returns double type
public int applyAsDouble(long i)

11. DoubleToIntFunction: It can take double type as input and returns int type
public int applyAsInt(double i)

12. DoubleToLongFunction: It can take double type as input and returns long type
public int applyAsLong(double i)

13. ToIntBiFunction:return type must be int type but inputs can be anytype
public int applyAsInt(T t, U u)

14. ToLongBiFunction:return type must be long type but inputs can be anytype
public long applyAsLong(T t, U u)

15. ToDoubleBiFunction: return type must be double type but inputs can be anytype
public double applyAsDouble(T t, U u)

Demo Program to find square of given integer by using Function:

```
1)  import java.util.function.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        IntFunction<Integer,Integer> f=i->i*i;
7)        System.out.println(f.apply(4));
8)     }
9)  }
```

Demo Program to find square of given integer by using IntFunction:

```
1)  import java.util.function.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        IntFunction<Integer> f=i->i*i;
7)        System.out.println(f.apply(5));
8)     }
9)  }
```

Demo Program to find length of the given String by using Function:

```
1)  import java.util.function.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        Function<String,Integer> f=s->s.length();
7)        System.out.println(f.apply("durga"));
8)     }
9)  }
```

Demo Program to find length of the given String by using ToIntFunction:

```
1)  import java.util.function.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        ToIntFunction<String> f=s->s.length();
7)        System.out.println(f.applyAsInt("durga"));
8)     }
9)  }
```

Demo Program to find square root of given integer by using Function:

```
1)  import java.util.function.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        Function<Integer,Double> f=i->Math.sqrt(i);
7)        System.out.println(f.apply(7));
8)     }
9)  }
```

```
1)  import java.util.function.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        IntToDoubleFunction f=i->Math.sqrt(i);
7)        System.out.println(f.applyAsDouble(9));
8)     }
9)  }
```

Primitive Version for Consumer:

The following 6 primitive versions available for Consumer:

1. IntConsumer
public void accept(int value)

2. LongConsumer
public void accept(long value)

3. DoubleConsumer
public void accept(double value)

4. ObjIntConsumer<T>
public void accept(T t,int value)

5. ObjLongConsumer<T>
public void accept(T t,long value)

6. ObjDoubleConsumer<T>
public void accept(T t,double value)

Demo Program for IntConsumer:

```
1)  import java.util.function.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        IntConsumer c= i->System.out.println("The Square of i:"+(i*i));
7)        c.accept(10);
8)     }
9)  }
```

Output: The Square of i:100

Demo Program to increment employee Salary by using ObjDoubleConsumer:

```
1)  import java.util.function.*;
2)  import java.util.*;
3)  class Employee
4)  {
5)     String name;
6)     double salary;
7)     Employee(String name,double salary)
8)     {
9)        this.name=name;
```

```
10)       this.salary=salary;
11)    }
12) }
13) class Test
14) {
15)    public static void main(String[] args)
16)    {
17)       ArrayList<Employee> l= new ArrayList<Employee>();
18)       populate(l);
19)       ObjDoubleConsumer<Employee> c=(e,d)->e.salary=e.salary+d;
20)       for(Employee e:l)
21)       {
22)          c.accept(e,500.0);
23)       }
24)       for(Employee e:l)
25)       {
26)          System.out.println("Employee Name:"+e.name);
27)          System.out.println("Employee Salary:"+e.salary);
28)          System.out.println();
29)       }
30)
31)    }
32)    public static void populate(ArrayList<Employee> l)
33)    {
34)       l.add(new Employee("Durga",1000));
35)       l.add(new Employee("Sunny",2000));
36)       l.add(new Employee("Bunny",3000));
37)       l.add(new Employee("Chinny",4000));
38)    }
39) }
```

Output:

```
Employee Name:Durga
Employee Salary:1500.0

Employee Name:Sunny
Employee Salary:2500.0

Employee Name:Bunny
Employee Salary:3500.0

Employee Name:Chinny
Employee Salary:4500.0
```

Primitive Versions for Supplier:

The following 4 primitive versions available for Supplier:

1. IntSupplier

public int getAsInt();

2. LongSupplier

public long getAsLong()

3. DoubleSupplier

public double getAsDouble()

4. BooleanSupplier

public boolean getAsBoolean()

Demo Program to generate 6 digit random OTP by using IntSupplier:

```
1)  import java.util.function.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        IntSupplier s=()->(int)(Math.random()*10);
7)        String otp="";
8)        for(int i =0;i<6;i++)
9)        {
10)          otp=otp+s.getAsInt();
11)       }
12)       System.out.println("The 6 digit OTP: "+otp);
13)    }
14) }
```

Output: The 6 digit OTP: 035716

UnaryOperator<T>:

- If input and output are same type then we should go for UnaryOperator
- It is child of Function<T,T>

```
1)  import java.util.function.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
```

```
        Function<Integer,Integer> f=i->i*i;
        System.out.println(f.apply(5));
    }
}
```

```
import java.util.function.*;
class Test
{
    public static void main(String[] args)
    {
        UnaryOperator<Integer> f=i->i*i;
        System.out.println(f.apply(6));
    }
}
```

The primitive versions for UnaryOperator:

IntUnaryOperator:
public int applyAsInt(int)

LongUnaryOperator:
public long applyAsLong(long)

DoubleUnaryOperator:
public double applyAsDouble(double)

Demo Program-1 for IntUnaryOperator:

```
import java.util.function.*;
class Test
{
    public static void main(String[] args)
    {
        IntUnaryOperator f=i->i*i;
        System.out.println(f.applyAsInt(6));
    }
}
```

Demo Program-2 for IntUnaryOperator:

```
import java.util.function.*;
class Test
{
    public static void main(String[] args)
    {
```

```
6)        IntUnaryOperator f1=i->i+1;
7)        System.out.println(f1.applyAsInt(4));
8)
9)        IntUnaryOperator f2=i->i*i;
10)       System.out.println(f2.applyAsInt(4));
11)
12)       System.out.println(f1.andThen(f2).applyAsInt(4));
13)
14)    }
15) }
```

Output:

```
5
16
25
```

BinaryOperator:

It is the child of BiFunction<T,T,T>

BinaryOperator<T>
public T apply(T,T)

```
1)  import java.util.function.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        BiFunction<String,String,String> f=(s1,s2)->s1+s2;
7)        System.out.println(f.apply("durga","software"));
8)     }
9)  }
```

```
1)  import java.util.function.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        BinaryOperator<String> b=(s1,s2)->s1+s2;
7)        System.out.println(b.apply("durga","software"));
8)     }
9)  }
```

The primitive versions for BinaryOperator:

1. IntBinaryOperator

public int applyAsInt(int i,int j)

2. LongBinaryOperator

public long applyAsLong(long l1,long l2)

3. DoubleBinaryOperator

public double applyAsLong(double d1,double d2)

```
1)  import java.util.function.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        BinaryOperator<Integer> b=(i1,i2)->i1+i2;
7)        System.out.println(b.apply(10,20));
8)     }
9)  }
```

```
1)  import java.util.function.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        IntBinaryOperator b=(i1,i2)->i1+i2;
7)        System.out.println(b.applyAsInt(10,20));
8)     }
9)  }
```