**Java-8 – Primitive interfaces for Consumer**

* **IntConsumer:**

interface IntConsumer{

public void accept(int i);

}

IntConsumer c = i -> System.out.println(i \* i);

c.accept(10);

Example:

import java.util.function.IntConsumer;

class Test{

public static void main(String[] args){

IntConsumer c = I -> System.out.println(“The square of (i):”+(i\*i));

c.accept(10);

}

}

Similarly we have LongConsumer & DoubleConsumer

* **Others:**

ObjIntConsumer<T>

public void accept(T t, int i);

ObjLongConsumer<T>

public void accept(T t, long l);

ObjDoubleConsumer<T>

public void accept(T t, double d);

BiConsumer<Employee, Double> c = (e, d) -> e.salary = e.salary + d;

Employee e = new Employee(“durga”, 1000);

c.accept(2, 500.0);

For the above scenario, that is to increase the salary of the employee, we can use ObjDoubleConsumer instead of BiConsumer as it will impact the performance due to autoboxing and unboxing.

* **Example:**

import java.util.function.ObjDoubleConsumer;

import java.util.List;

import java.util.ArrayList;

class Employee{

String name;

double salary;

Employee(String name, double salary){

this.name = name;

this.salary = salary;

}

}

class Test{

public static void main(String[] args){

List<Employee> l = new ArrayList<>();

populate(l);

ObjDoubleConsumer<Employee> c = (e, d) -> e.salary = e.salary + d;

for(Employee e: l){

c.accept(e, 500);

}

for(Employee e: l){

System.out.println(“Employee name:”+e.name);

System.out.println(“Employee salary:”+e.salary);

System.out.println();

}

}

public static void populate(List<Employee> l){

l.add(new Employee(“Saravana”, 20000));

l.add(new Employee(“Gokul”, 10000));

}

}