**Q1. Write the following a functional interface and implement it using lambda:**

* **(1) First number is greater than second number or not Parameter (int ,int ) Return boolean**
* **(2) Increment the number by 1 and return incremented value Parameter (int) Return int**
* **(3) Concatination of 2 string Parameter (String , String ) Return (String)**
* **(4) Convert a string to uppercase and return . Parameter (String) Return (String)**

**Solution**

**@FunctionalInterface**

**interface FindGreater**

**{**

**boolean checkFirstIsGreater(int a, int b);**

**}**

**@FunctionalInterface**

**interface AddOne**

**{**

**int incrementByOne(int a);**

**}**

**@FunctionalInterface**

**interface Concat**

**{**

**String concat(String str1, String str2);**

**}**

**@FunctionalInterface**

**interface ConvertCase**

**{**

**String topUpper(String str1);**

**}**

**public class LambdaDemo {**

**public static void main(String[] args)**

**{**

**FindGreater findGreater=(a,b)->a>b;**

**System.*out*.println(findGreater.checkFirstIsGreater(1,2));**

**AddOne addOne=a->a+1;**

**System.*out*.println(addOne.incrementByOne(5));**

**Concat concat=(str1,str2)->str1+str2;**

**System.*out*.println(concat.concat("Surbhi","Garg"));**

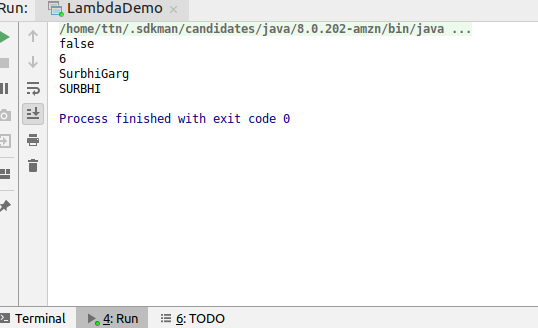
**ConvertCase convertCase=str1->str1.toUpperCase();**

**System.*out*.println(convertCase.topUpper("surbhi"));**

**}**

**}**

**Output**



**Q2. Create a functional interface whose method takes 2 integers and return one integer.**

**Solution**

**@FunctionalInterface**

**public interface BiFunction {**

**int perform(int a,int b);**

**}**

**Q3. Using (instance) Method reference create and apply add and subtract method and using (Static) Method reference create and apply multiplication method for the functional interface created.**

**Solution**

**public class BiFunctionDriver {**

**int add(int a,int b)**

**{**

**return a+b;**

**}**

**int subtract(int a,int b)**

**{**

**return a-b;**

**}**

**static int multiply(int a,int b)**

**{**

**return a\*b;**

**}**

**public static void main(String[] args) {**

**BiFunction adder=new BiFunctionDriver()::add;**

**BiFunction findDifference=new BiFunctionDriver()::subtract;**

**BiFunction multiply=BiFunctionDriver::*multiply*;**

**System.*out*.println(adder.perform(4,5));**

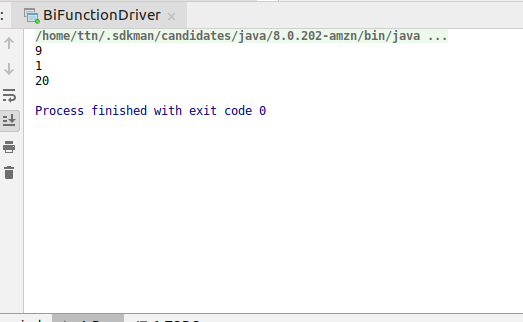
**System.*out*.println(findDifference.perform(5,4));**

**System.*out*.println(multiply.perform(4,5));**

**}**

**}**

**Output**



**Q4. Create an Employee Class with instance variables (String) name, (Integer)age, (String)city and get the instance of the Class using constructor reference**

**Solution**

**class Employee**

**{**

**int age;**

**String name;**

**String city;**

**public Employee(int age, String name, String city) {**

**this.age = age;**

**this.name = name;**

**this.city = city;**

**}**

**@Override**

**public String toString() {**

**return "Employee{" +**

**"age=" + age +**

**", name='" + name + '\'' +**

**", city='" + city + '\'' +**

**'}';**

**}**

**}**

**@FunctionalInterface**

**interface Generator**

**{**

**Employee createEmployee(int age,String name,String city);**

**}**

**public class ConstructorRefDemo {**

**public static void main(String[] args) {**

***//constructor reference***

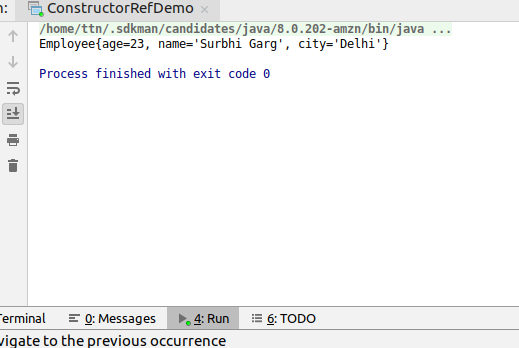
**Generator generator=Employee::new;**

**System.*out*.println(generator.createEmployee(23,"Surbhi Garg","Delhi"));**

**}**

**}**

**Output**



**Q5. Implement following functional interfaces from java.util.function using lambdas:**

**(1) Consumer (2) Supplier (3) Predicate (4) Function**

**Solution**

**import java.util.Arrays;**

**import java.util.List;**

**import java.util.function.Consumer;**

**import java.util.function.Function;**

**import java.util.function.Predicate;**

**import java.util.function.Supplier;**

**public class UtilFunctionalInterfacesDemo {**

**public static void main(String[] args) {**

***//1Consumer***

**Consumer<List<Integer>> consumer= e->e.forEach(System.*out*::println);**

**consumer.accept(Arrays.*asList*(3,4,5,6));**

***//2.Supplier***

**Supplier<Integer>supplier=()->5;**

**System.*out*.println(supplier.get());**

***//3.Predicate***

**Predicate<String>testString=e->e.length()>5;**

**System.*out*.println("Result of length>5 "+testString.test("Surbhi Garg"));**

***//4.Function***

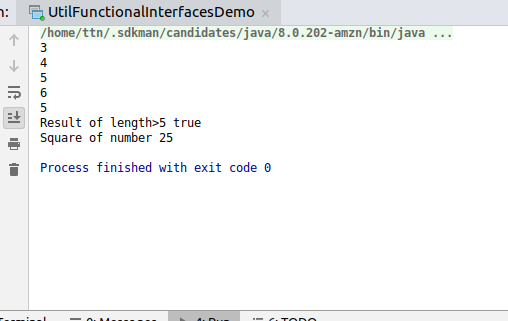
**Function<Integer,Integer>square=e->e\*e;**

**System.*out*.println("Square of number "+square.apply(5));**

**}**

**}**

**Output**



**Q6. Create and access default and static method of an interface**

**Solution**

**interface** Greeter

{

**default void** display(){

System.***out***.println(**"Hello World"**);

}

**static void** greet()

{

System.***out***.println(**"Good Morning"**);

}

}

**public class** DefaultAndStaticMethodDemo **implements** Greeter {

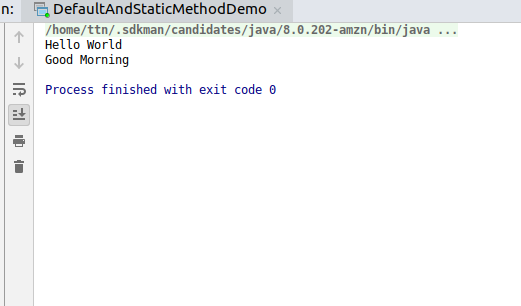
**public static void** main(String[] args) {

**new** DefaultAndStaticMethodDemo().display();

Greeter.*greet*();

}

**Output**



**Q7. Override the default method of the interface.**

**Solution**

**public class DefaultOverrideDemo implements Greeter {**

**public void display()**

**{**

**System.*out*.println("Overridden original display method");**

**}**

**public static void main(String[] args) {**

**new DefaultOverrideDemo().display();**

**}**

**}**

**Output**



**Q8. Implement multiple inheritance with default method inside interface.**

**Solution**

**interface StrengthTester**

**{**

**default void test()**

**{**

**System.*out*.println("Testing strength");**

**}**

**}**

**interface QualityTester**

**{**

**default void test()**

**{**

**System.*out*.println("Testing quality");**

**}**

**}**

**public class MultipleInheritanceDemo implements StrengthTester,QualityTester{**

**public void test()**

**{**

**StrengthTester.super.test();**

**QualityTester.super.test();**

**System.*out*.println("Overall test method");**

**}**

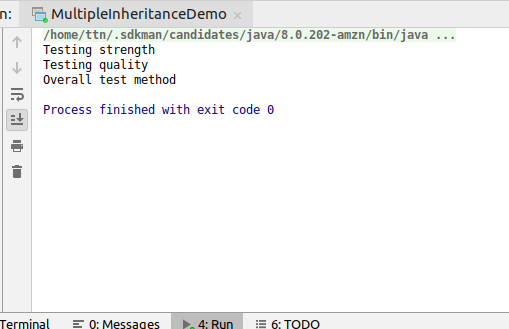
**public static void main(String[] args) {**

**new MultipleInheritanceDemo().test();**

**}**

**}**

**Output**



**Q9. Collect all the even numbers from an integer list.**

**import java.util.Arrays;**

**import java.util.List;**

**import java.util.stream.Collectors;**

**public class EvenNumbers {**

**public static void main(String[] args) {**

**List<Integer> integerList= Arrays.*asList*(1,2,3,4,5,6,7,8,9,10);**

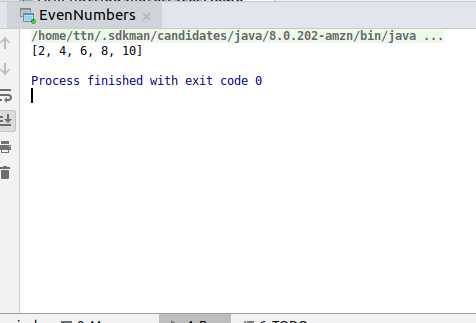
**List<Integer>evenNumbers=integerList.stream().filter(e->e%2==0).collect(Collectors.*toList*());**

**System.*out*.println(evenNumbers);**

**}**

**}**

**Output**



**Q10.Sum all the numbers greater than 5 in the integer list.**

**import java.util.Arrays;**

**import java.util.List;**

**import java.util.stream.Collectors;**

**public class SumEleGt5 {**

**public static void main(String[] args) {**

**List<Integer> integerList= Arrays.*asList*(1,2,3,4,5,6,7,8,9,10);**

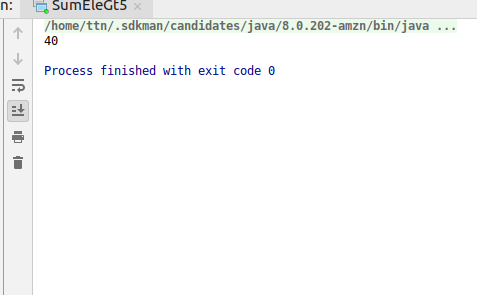
**System.*out*.println(integerList**

**.stream().filter(e->e>5).collect(Collectors.*summingInt*(e->e)));**

**}**

**}**

**Output**



**Q11. Find average of the number inside integer list after doubling it.**

**Solution**

**import java.util.Arrays;**

**import java.util.List;**

**public class DoubleAndAverage {**

**public static void main(String[] args) {**

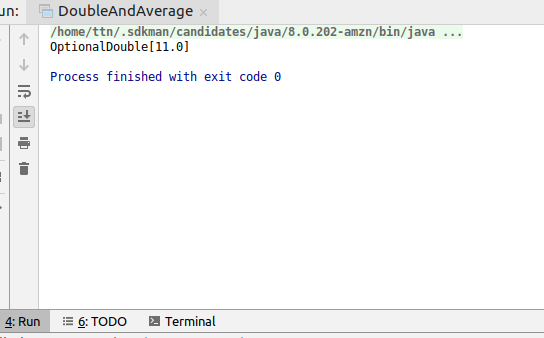
**List<Integer> integerList = Arrays.*asList*(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);**

**System.*out*.println(integerList.stream().mapToInt(e->e\*2).average());**

**}**

**}**

**Output**



**Q12. Find the first even number in the integer list which is greater than 3.**

**Solution**

**import java.util.Arrays;**

**import java.util.List;**

**import java.util.Optional;**

**public class FistEvenGt3 {**

**public static void main(String[] args) {**

**List<Integer> integerList = Arrays.*asList*(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);**

**Optional<Integer> result=integerList.stream().filter(e->e>3).filter(e->e%2==0).findFirst();**

**if(result.isPresent())**

**{**

**System.*out*.println("First number greater than 3 and even in the list is:"+result.get());**

**}**

**else**

**{**

**System.*out*.println("No such value");**

**}**

***//using logical operator within single filter***

**System.*out*.println(integerList.stream().filter(e->e>3&&e%2==0).findFirst());**

**}**

**}**

**Output**

