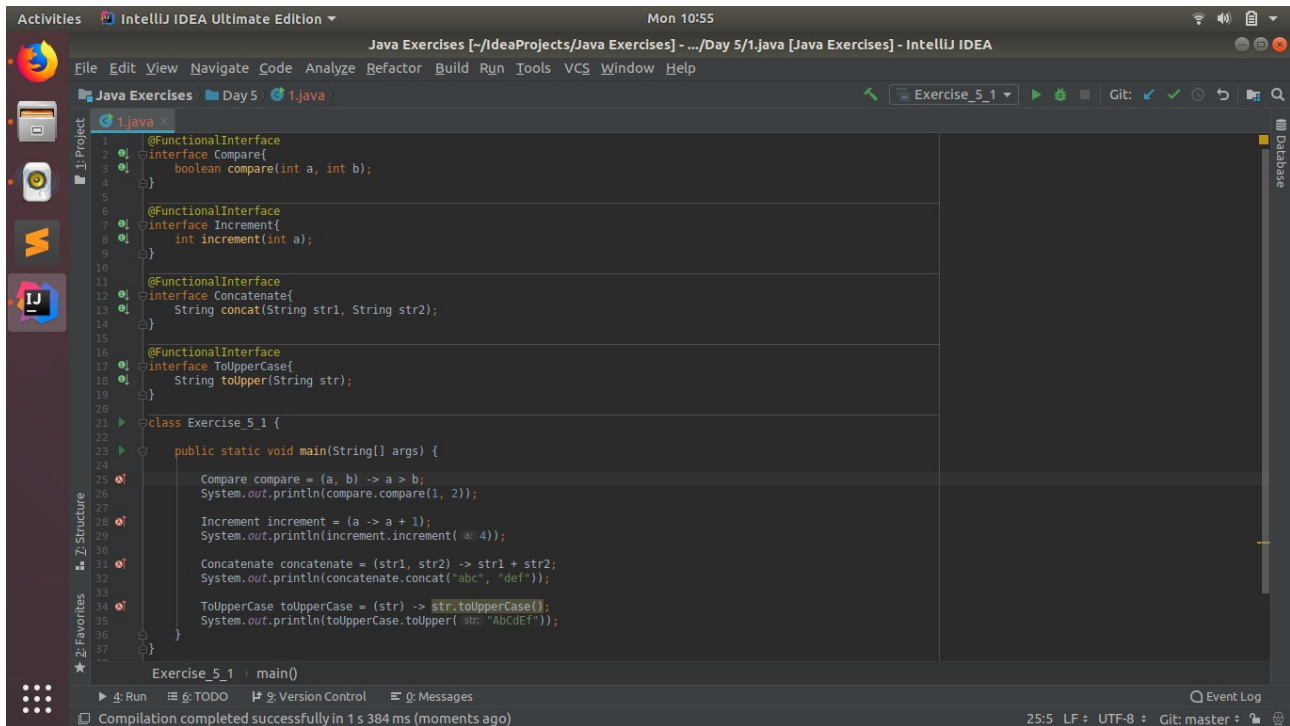


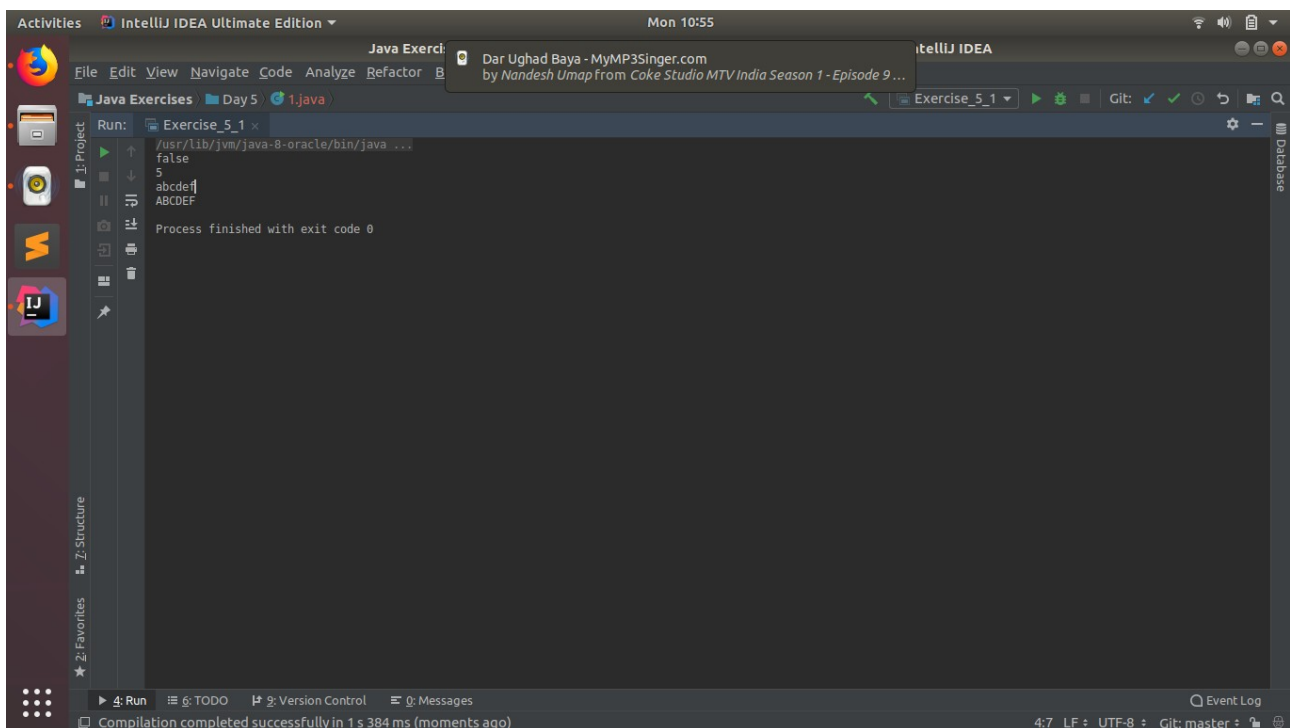
Q1. Write the following a functional interface and implement it using lambda:

- (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)

A1.



```
1 @FunctionalInterface
2 interface Compare{
3     boolean compare(int a, int b);
4 }
5
6 @FunctionalInterface
7 interface Increment{
8     int increment(int a);
9 }
10
11 @FunctionalInterface
12 interface Concatenate{
13     String concat(String str1, String str2);
14 }
15
16 @FunctionalInterface
17 interface ToUpperCase{
18     String toUpper(String str);
19 }
20
21 class Exercise_5_1 {
22     public static void main(String[] args) {
23
24         Compare compare = (a, b) -> a > b;
25         System.out.println(compare.compare(1, 2));
26
27         Increment increment = (a -> a + 1);
28         System.out.println(increment.increment( 4));
29
30         Concatenate concatenate = (str1, str2) -> str1 + str2;
31         System.out.println(concatenate.concat("abc", "def"));
32
33         ToUpperCase toUpperCase = (str) -> str.toUpperCase();
34         System.out.println(toUpperCase.toUpper( str: "AbCdEf"));
35     }
36 }
37
38 Exercise_5_1 > main()
```



```
Run: Exercise_5_1 x
/usr/lib/jvm/java-8-oracle/bin/java ...
false
5
abcdef
ABCDEF
Process finished with exit code 0
```

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

A2.

```
1 @FunctionalInterface
2 interface MyInterface {
3     int useless(int a, int b);
4 }
5
6 class Exercise_5_2 {
7
8     static int CreateInstance(int a, int b) { return a+b; }
9
10    public static void main(String[] args) {
11
12        MyInterface myInterface = Exercise_5_2::CreateInstance;
13        System.out.println(myInterface.useless(4, 5));
14    }
15 }
16
17
18
```

Run: Exercise\_5\_2 x  
/usr/lib/jvm/java-8-oracle/bin/java ...  
9  
Process finished with exit code 0

Compilation completed successfully in 1 s 319 ms (moments ago)

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.

A3.

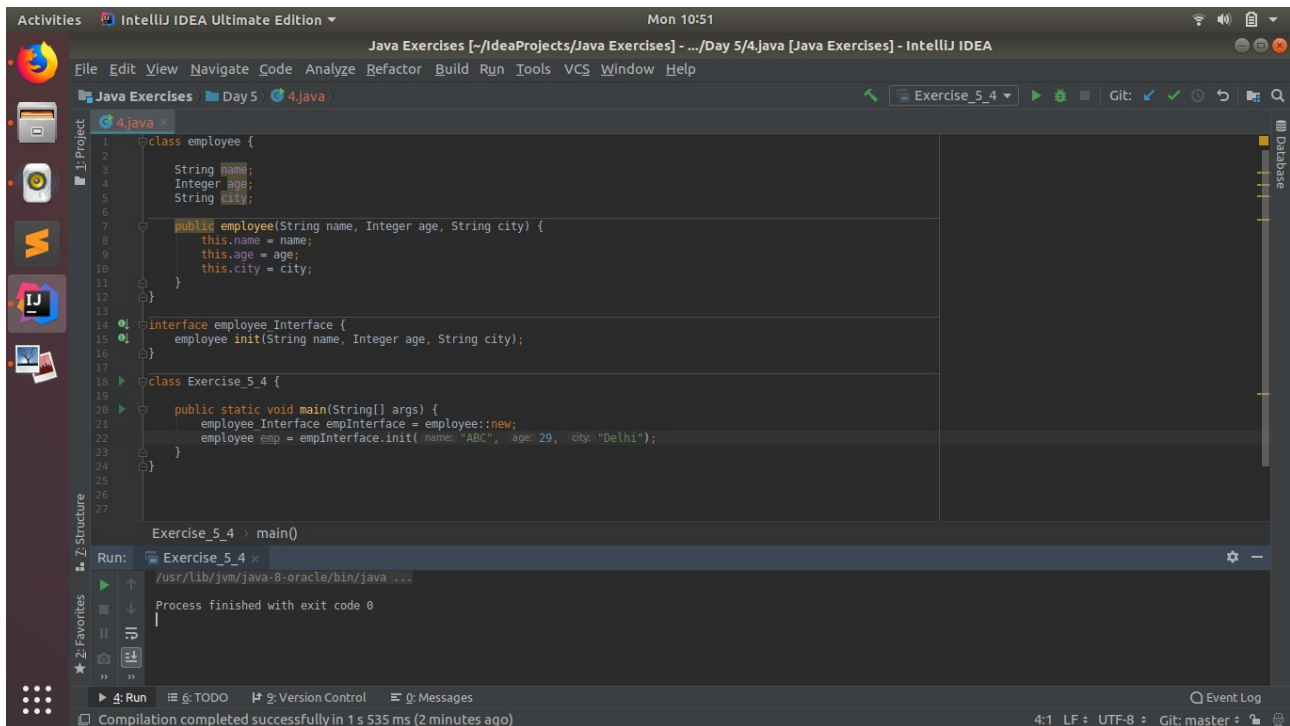
```
1 @FunctionalInterface
2 interface Operation {
3     int doIt(int a, int b);
4 }
5
6 class Exercise_5_3 {
7
8     static int mul(int a, int b) { return a * b; }
9
10    public static void main(String[] args) {
11
12        Operation addByInterface = (a, b) -> a + b;
13        System.out.println(addByInterface.doIt(4, 5));
14
15        Operation subByInterface = (a, b) -> a - b;
16        System.out.println(subByInterface.doIt(4, 5));
17
18        Operation mulByStatic = Exercise_5_3::mul;
19        System.out.println(mulByStatic.doIt(4, 5));
20    }
21 }
22
23
24
25
```

Run: Exercise\_5\_3 x  
/usr/lib/jvm/java-8-oracle/bin/java ...  
9  
-1  
20  
Process finished with exit code 0

Compilation completed successfully in 1 s 18 ms (moments ago)

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

A4.



```
1 class employee {
2     String name;
3     Integer age;
4     String city;
5
6     public employee(String name, Integer age, String city) {
7         this.name = name;
8         this.age = age;
9         this.city = city;
10    }
11 }
12
13
14 interface employee Interface {
15     employee init(String name, Integer age, String city);
16 }
17
18 class Exercise_5_4 {
19     public static void main(String[] args) {
20         employee Interface empInterface = employee::new;
21         employee emp = empInterface.init( name: "ABC", age: 29, city: "Delhi");
22     }
23 }
24
25
26
27
```

Run: Exercise\_5\_4 ×

/usr/lib/jvm/java-8-oracle/bin/java ...

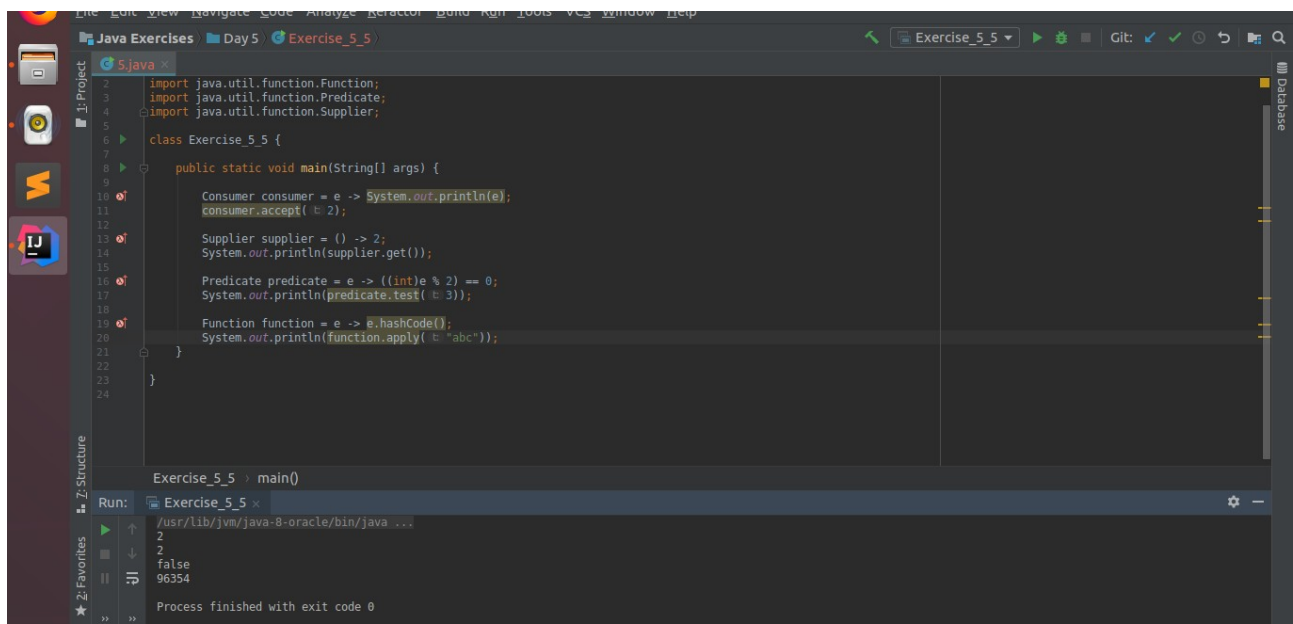
Process finished with exit code 0

4:1 LF UTF-8 Git: master

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

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

A5.



```
1 import java.util.function.Function;
2 import java.util.function.Predicate;
3 import java.util.function.Supplier;
4
5 class Exercise_5_5 {
6     public static void main(String[] args) {
7
8         Consumer consumer = e -> System.out.println(e);
9         consumer.accept(12);
10
11         Supplier supplier = () -> 2;
12         System.out.println(supplier.get());
13
14         Predicate predicate = e -> ((int)e % 2) == 0;
15         System.out.println(predicate.test(3));
16
17         Function function = e -> e.hashCode();
18         System.out.println(function.apply("abc"));
19     }
20 }
21
22
23
24
```

Run: Exercise\_5\_5 ×

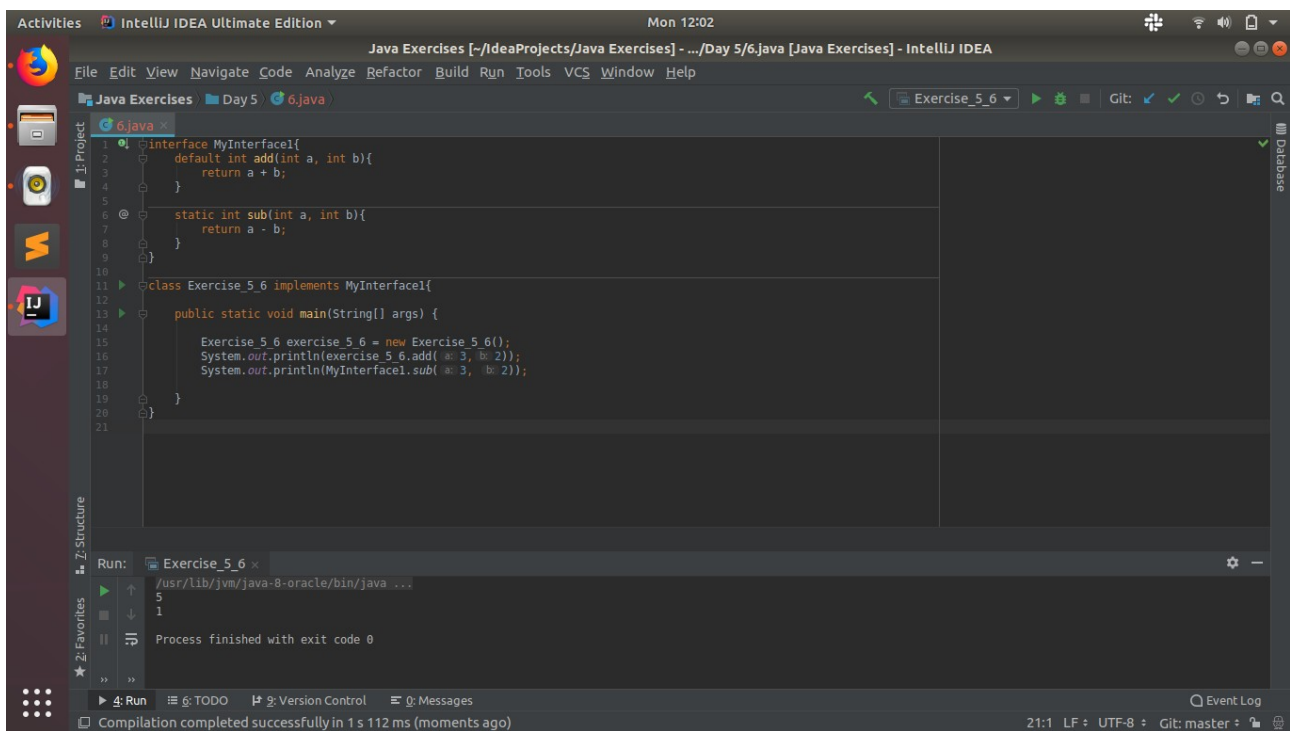
/usr/lib/jvm/java-8-oracle/bin/java ...

2  
2  
false  
96354

Process finished with exit code 0

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

A6.



The screenshot shows the IntelliJ IDEA interface with a project named 'Java Exercises'. The main editor displays the code for 'Exercise\_5\_6.java'. The code defines an interface 'MyInterface' with a default method 'add' and a static method 'sub'. The class 'Exercise\_5\_6' implements 'MyInterface' and has a 'main' method that calls both 'add' and 'sub'.

```
1 interface MyInterface {
2     default int add(int a, int b) {
3         return a + b;
4     }
5
6     static int sub(int a, int b) {
7         return a - b;
8     }
9 }
10
11 class Exercise_5_6 implements MyInterface {
12     public static void main(String[] args) {
13         Exercise_5_6 exercise_5_6 = new Exercise_5_6();
14         System.out.println(exercise_5_6.add(3, 2));
15         System.out.println(MyInterface.sub(3, 2));
16     }
17 }
18
19
20
21
```

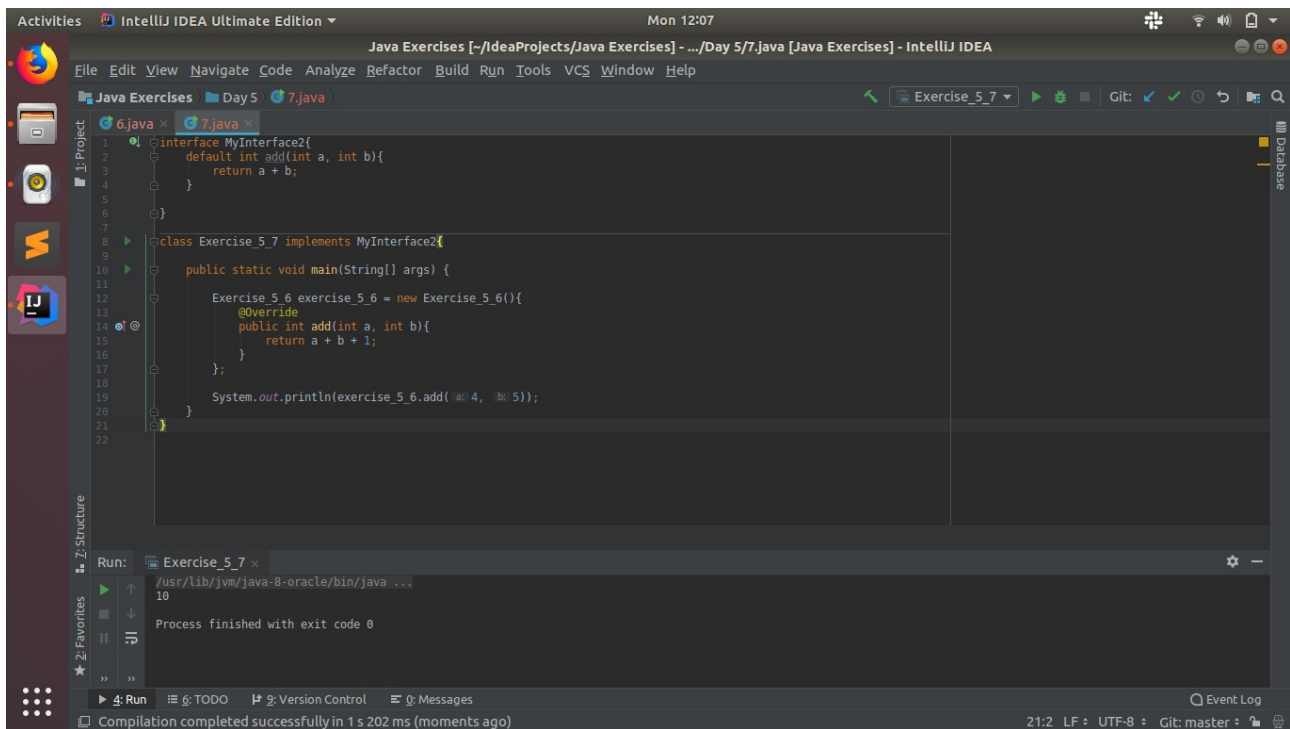
The Run window shows the output of the program:

```
Run: Exercise_5_6 x
/usr/lib/jvm/java-8-oracle/bin/java ...
5
1
Process finished with exit code 0
```

The status bar at the bottom indicates 'Compilation completed successfully in 1 s 112 ms (moments ago)'.

Q7. Override the default method of the interface.

A7.



The screenshot shows the IntelliJ IDEA interface with a project named 'Java Exercises'. The main editor displays the code for 'Exercise\_5\_7.java'. The code defines an interface 'MyInterface2' with a default method 'add'. The class 'Exercise\_5\_7' implements 'MyInterface2' and overrides the 'add' method. The 'main' method calls the 'add' method on an instance of 'Exercise\_5\_6'.

```
1 interface MyInterface2 {
2     default int add(int a, int b) {
3         return a + b;
4     }
5 }
6
7 class Exercise_5_7 implements MyInterface2 {
8     public static void main(String[] args) {
9         Exercise_5_6 exercise_5_6 = new Exercise_5_6();
10        @Override
11        public int add(int a, int b) {
12            return a + b + 1;
13        }
14        System.out.println(exercise_5_6.add(4, 5));
15    }
16 }
17
18
19
20
21
22
```

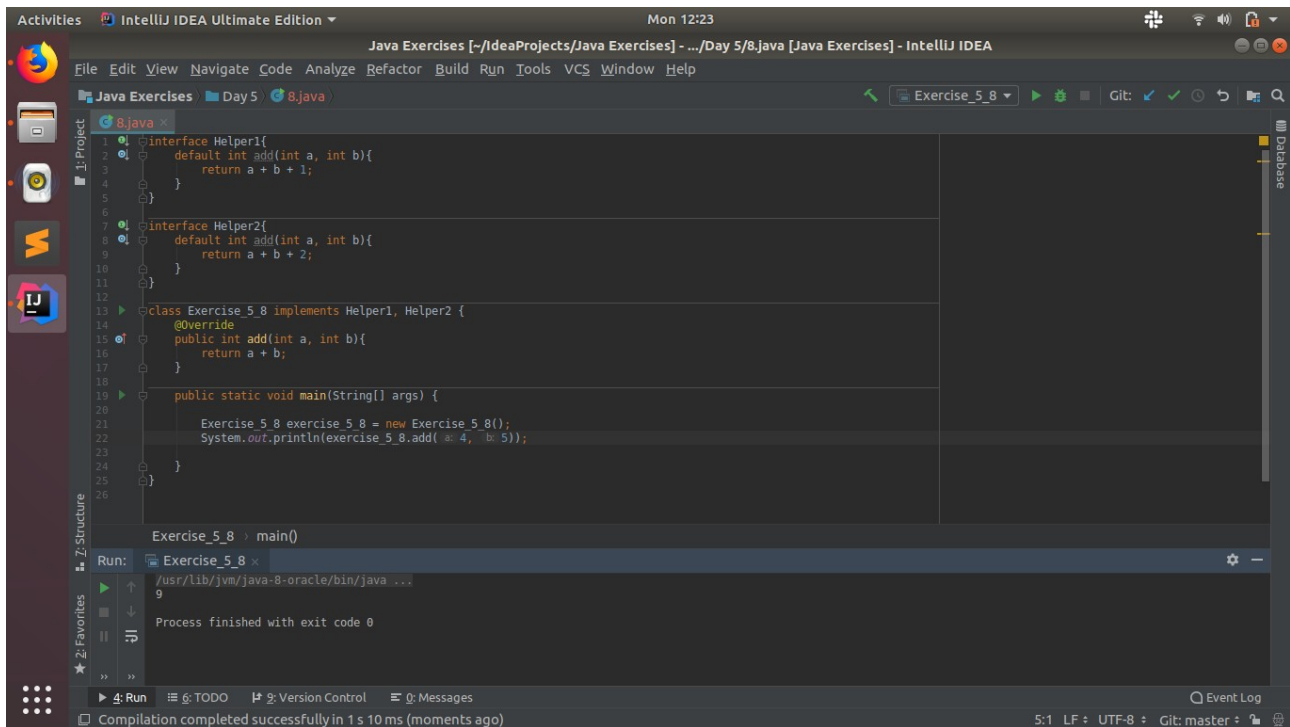
The Run window shows the output of the program:

```
Run: Exercise_5_7 x
/usr/lib/jvm/java-8-oracle/bin/java ...
10
Process finished with exit code 0
```

The status bar at the bottom indicates 'Compilation completed successfully in 1 s 202 ms (moments ago)'.

Q8. Implement multiple inheritance with default method inside interface.

A8.



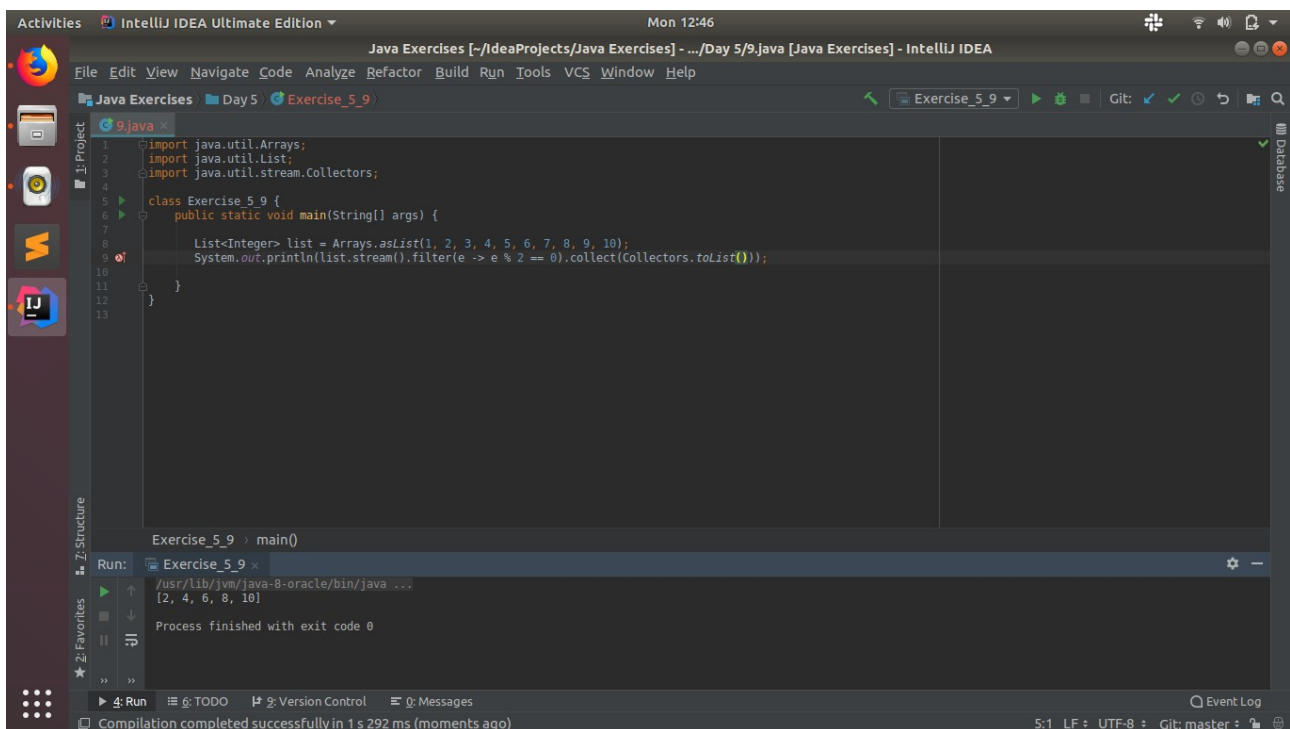
The screenshot shows the IntelliJ IDEA interface with a Java project named 'Java Exercises'. The main editor displays the code for 'Exercise\_5\_8.java'. It defines two interfaces, 'Helper1' and 'Helper2', each with a default method 'add'. The 'Exercise\_5\_8' class implements both interfaces and overrides the 'add' method. A 'main' method is also present, which creates an instance of 'Exercise\_5\_8' and calls the 'add' method with arguments 4 and 5. The 'Run' window at the bottom shows the execution output: '9'.

```
1 interface Helper1 {
2     default int add(int a, int b) {
3         return a + b + 1;
4     }
5 }
6
7 interface Helper2 {
8     default int add(int a, int b) {
9         return a + b + 2;
10    }
11 }
12
13 class Exercise_5_8 implements Helper1, Helper2 {
14     @Override
15     public int add(int a, int b) {
16         return a + b;
17     }
18
19     public static void main(String[] args) {
20
21         Exercise_5_8 exercise_5_8 = new Exercise_5_8();
22         System.out.println(exercise_5_8.add(4, 5));
23     }
24 }
25
26
```

Run: Exercise\_5\_8 x  
/usr/lib/jvm/java-8-oracle/bin/java ...  
9  
Process finished with exit code 0

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

A9.



The screenshot shows the IntelliJ IDEA interface with a Java project named 'Java Exercises'. The main editor displays the code for 'Exercise\_5\_9.java'. It imports 'java.util.Arrays', 'java.util.List', and 'java.util.stream.Collectors'. The 'Exercise\_5\_9' class has a 'main' method that creates a list of integers from 1 to 10, filters out the even numbers, and prints the resulting list. The 'Run' window at the bottom shows the execution output: '[2, 4, 6, 8, 10]'.

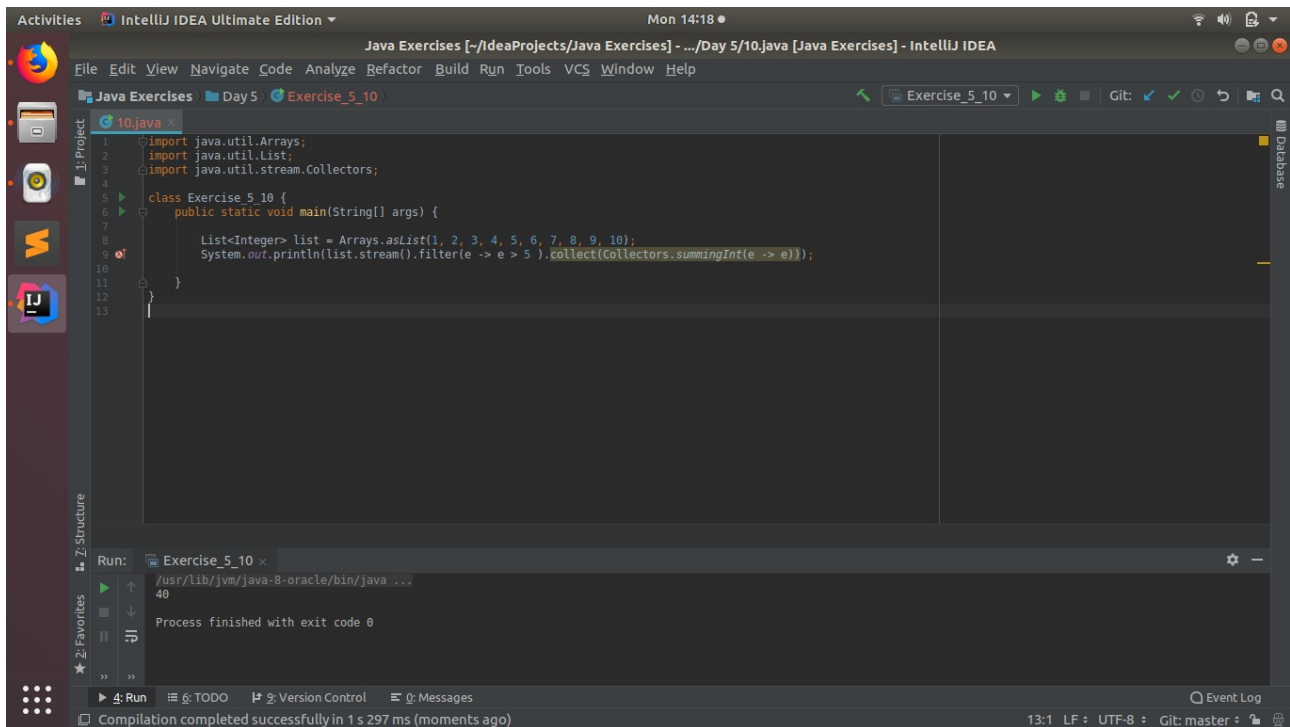
```
1 import java.util.Arrays;
2 import java.util.List;
3 import java.util.stream.Collectors;
4
5 class Exercise_5_9 {
6     public static void main(String[] args) {
7
8         List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
9         System.out.println(list.stream().filter(e -> e % 2 == 0).collect(Collectors.toList()));
10    }
11 }
12
13
```

Run: Exercise\_5\_9 x  
/usr/lib/jvm/java-8-oracle/bin/java ...  
[2, 4, 6, 8, 10]  
Process finished with exit code 0



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

A10.



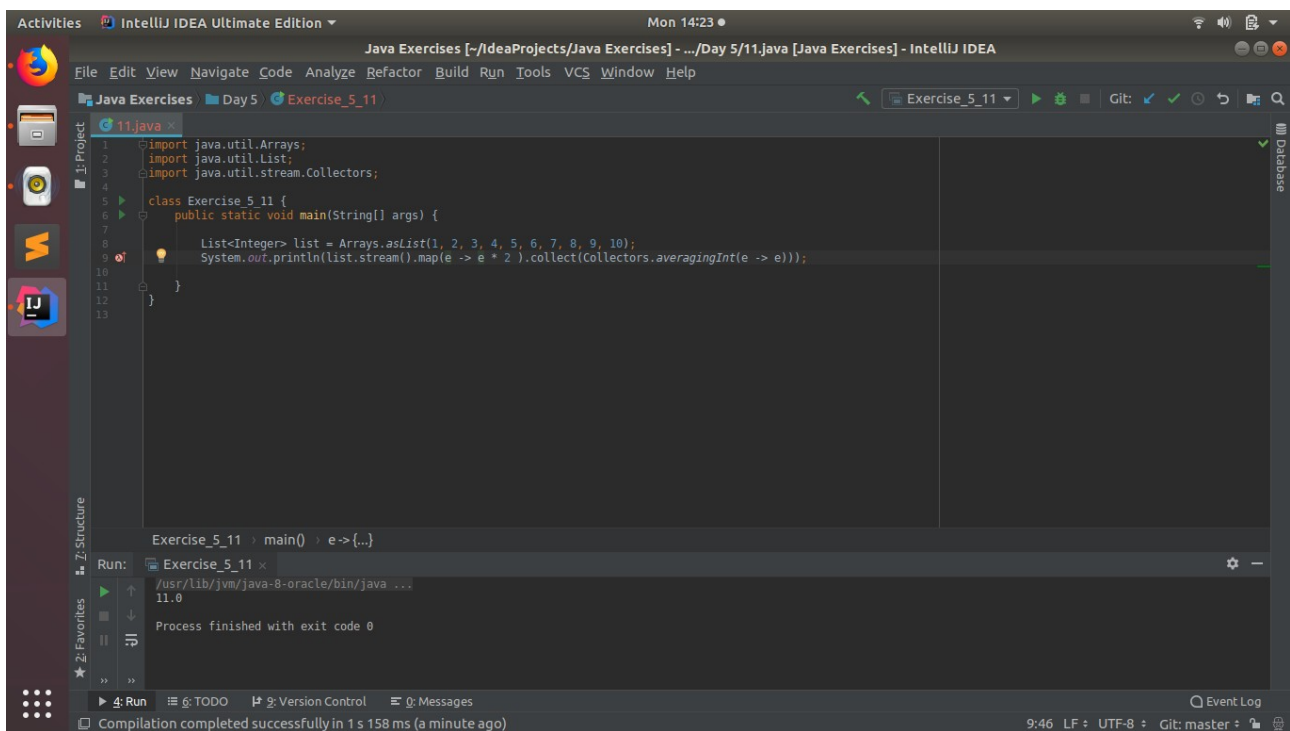
The screenshot shows the IntelliJ IDEA interface with the file `Exercise_5_10.java` open. The code defines a class `Exercise_5_10` with a `main` method that creates a list of integers from 1 to 10, filters for values greater than 5, and prints the sum. The Run window shows the output `40` and a successful completion message.

```
1 import java.util.Arrays;
2 import java.util.List;
3 import java.util.stream.Collectors;
4
5 class Exercise_5_10 {
6     public static void main(String[] args) {
7
8         List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
9         System.out.println(list.stream().filter(e -> e > 5 ).collect(Collectors.summingInt(e -> e)));
10
11     }
12 }
13
```

Run: Exercise\_5\_10 x  
40  
Process finished with exit code 0

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

A11.



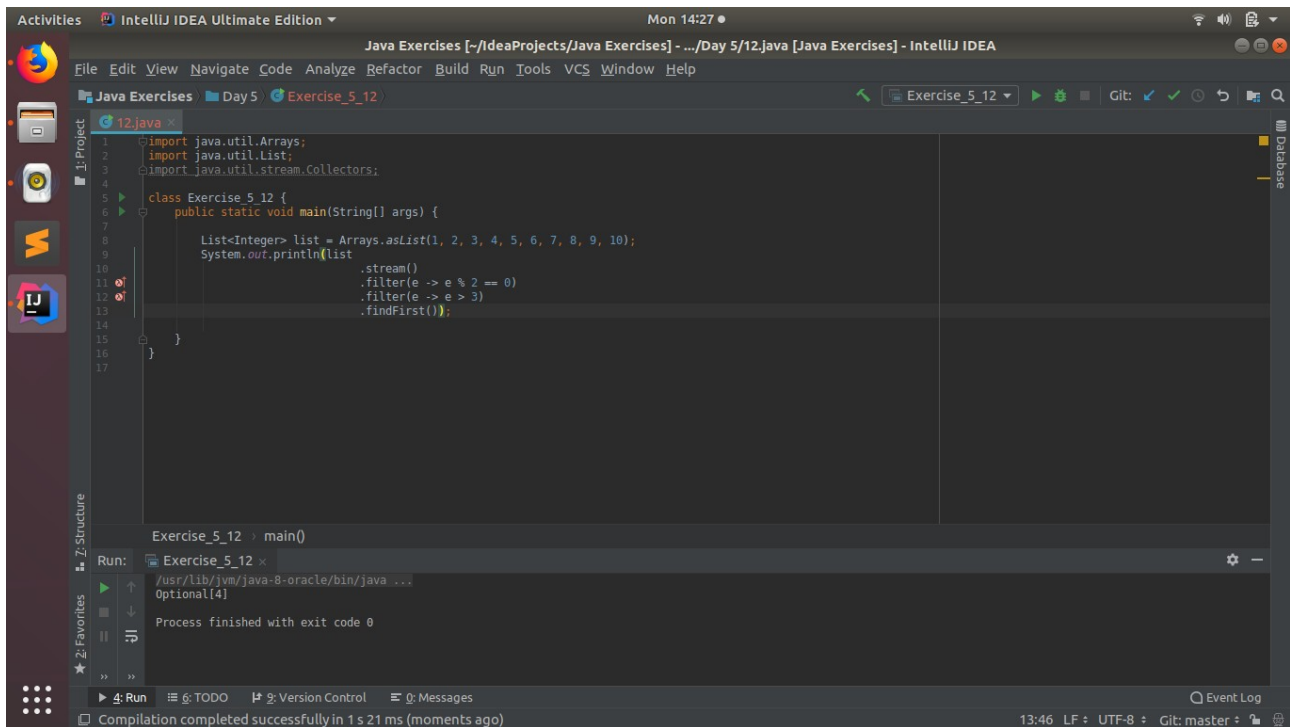
The screenshot shows the IntelliJ IDEA interface with the file `Exercise_5_11.java` open. The code defines a class `Exercise_5_11` with a `main` method that creates a list of integers from 1 to 10, doubles each element, and prints the average. The Run window shows the output `11.0` and a successful completion message.

```
1 import java.util.Arrays;
2 import java.util.List;
3 import java.util.stream.Collectors;
4
5 class Exercise_5_11 {
6     public static void main(String[] args) {
7
8         List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
9         System.out.println(list.stream().map(e -> e * 2 ).collect(Collectors.averagingInt(e -> e)));
10
11     }
12 }
13
```

Run: Exercise\_5\_11 x  
11.0  
Process finished with exit code 0

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

A12.



The screenshot displays the IntelliJ IDEA IDE interface. The main editor window shows the code for `Exercise_5_12.java`. The code imports `java.util.Arrays`, `java.util.List`, and `java.util.stream.Collectors`. It defines a class `Exercise_5_12` with a `main` method. Inside the `main` method, a list of integers is created using `Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)`. The list is then processed using a stream: `list.stream().filter(e -> e % 2 == 0).filter(e -> e > 3).findFirst()`. The `findFirst()` method is called, and the result is printed to the console using `System.out.println(list)`. The `Run` tab at the bottom shows the execution output: `/usr/lib/jvm/java-8-oracle/bin/java ... Optional[4]`, indicating that the first even number greater than 3 is 4. The status bar at the bottom indicates that the compilation was successful in 1 s 21 ms.

```
1 import java.util.Arrays;
2 import java.util.List;
3 import java.util.stream.Collectors;
4
5 class Exercise_5_12 {
6     public static void main(String[] args) {
7
8         List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
9         System.out.println(list
10             .stream()
11             .filter(e -> e % 2 == 0)
12             .filter(e -> e > 3)
13             .findFirst());
14     }
15 }
16
17
```

Run: Exercise\_5\_12 x

/usr/lib/jvm/java-8-oracle/bin/java ...  
Optional[4]

Process finished with exit code 0

Compilation completed successfully in 1 s 21 ms (moments ago)