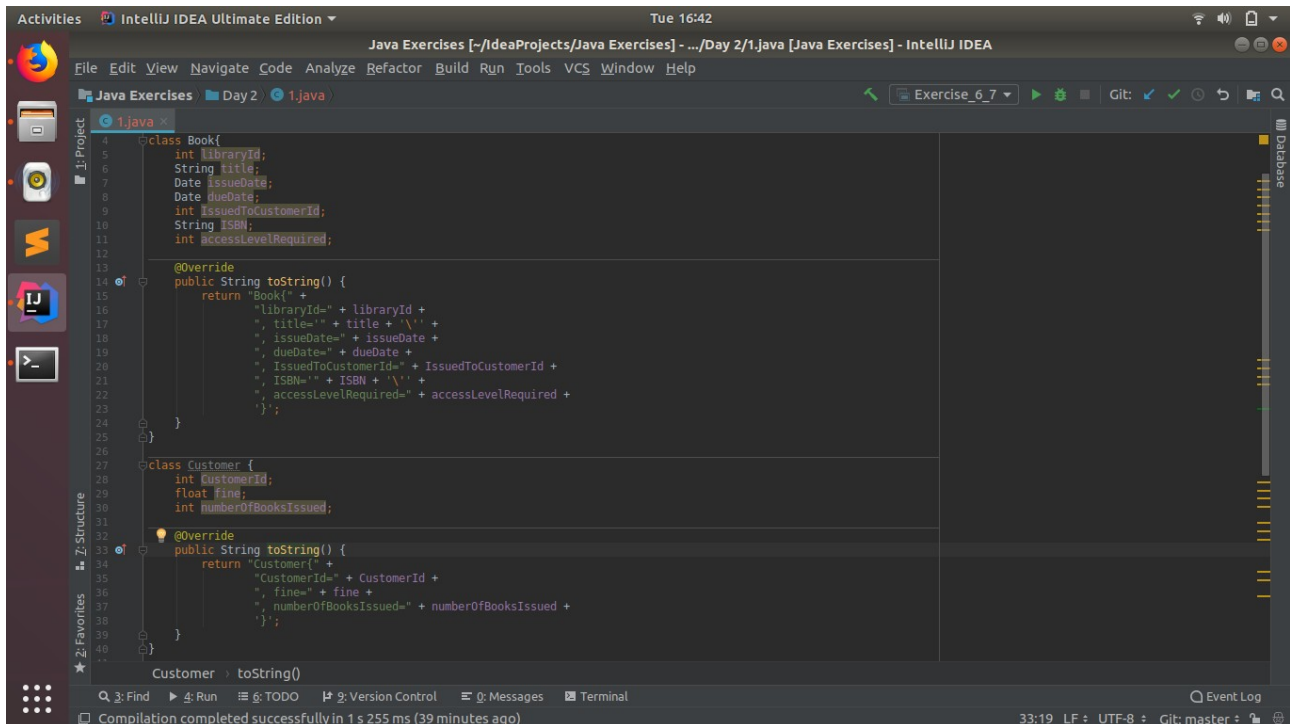


Q1. Create Java classes having suitable attributes for Library management system. Use OOPs concepts in your design. Also try to use interfaces and abstract classes.

A1.



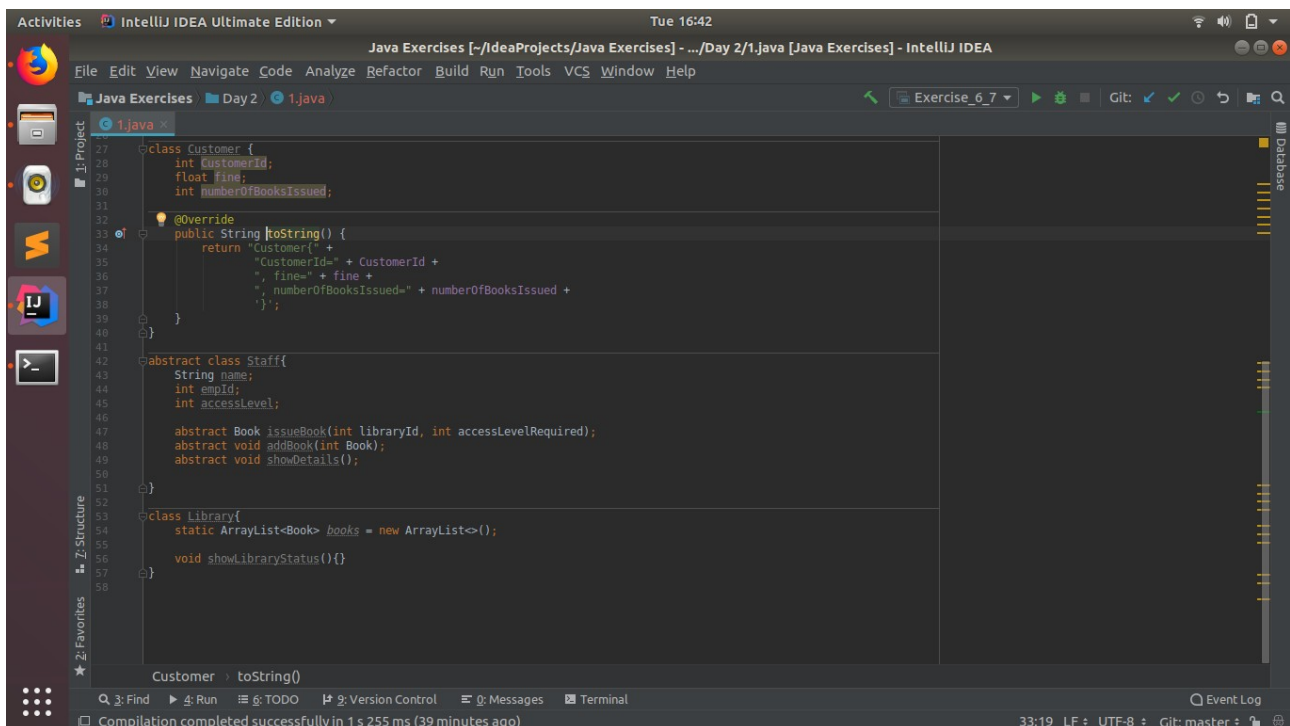
The screenshot shows the IntelliJ IDEA interface with a Java file named '1.java'. The code defines two classes: 'Book' and 'Customer'. The 'Book' class has attributes: 'libraryId' (int), 'title' (String), 'issueDate' (Date), 'dueDate' (Date), 'issuedToCustomerId' (int), 'ISBN' (String), and 'accessLevelRequired' (int). It overrides the 'toString()' method to return a formatted string of these attributes. The 'Customer' class has attributes: 'customerId' (int), 'fine' (float), and 'numberOfBooksIssued' (int). It also overrides the 'toString()' method to return a formatted string of these attributes. The status bar at the bottom indicates 'Compilation completed successfully in 1 s 255 ms (39 minutes ago)'.

```
class Book {
    int libraryId;
    String title;
    Date issueDate;
    Date dueDate;
    int issuedToCustomerId;
    String ISBN;
    int accessLevelRequired;

    @Override
    public String toString() {
        return "Book{" +
            "libraryId=" + libraryId +
            ", title=" + title + '\'' +
            ", issueDate=" + issueDate +
            ", dueDate=" + dueDate +
            ", issuedToCustomerId=" + issuedToCustomerId +
            ", ISBN=" + ISBN + '\'' +
            ", accessLevelRequired=" + accessLevelRequired +
            '}';
    }
}

class Customer {
    int customerId;
    float fine;
    int numberOfBooksIssued;

    @Override
    public String toString() {
        return "Customer{" +
            "customerId=" + customerId +
            ", fine=" + fine +
            ", numberOfBooksIssued=" + numberOfBooksIssued +
            '}';
    }
}
```



The screenshot shows the IntelliJ IDEA interface with a Java file named '1.java'. The code defines two classes: 'Staff' and 'Library'. The 'Staff' class is an abstract class with attributes: 'name' (String), 'empId' (int), and 'accessLevel' (int). It defines three abstract methods: 'issueBook(int libraryId, int accessLevelRequired)', 'addBook(int Book)', and 'showDetails()'. The 'Library' class has a static attribute: 'books' (ArrayList<Book>) and a method: 'showLibraryStatus()'. The status bar at the bottom indicates 'Compilation completed successfully in 1 s 255 ms (39 minutes ago)'.

```
abstract class Staff {
    String name;
    int empId;
    int accessLevel;

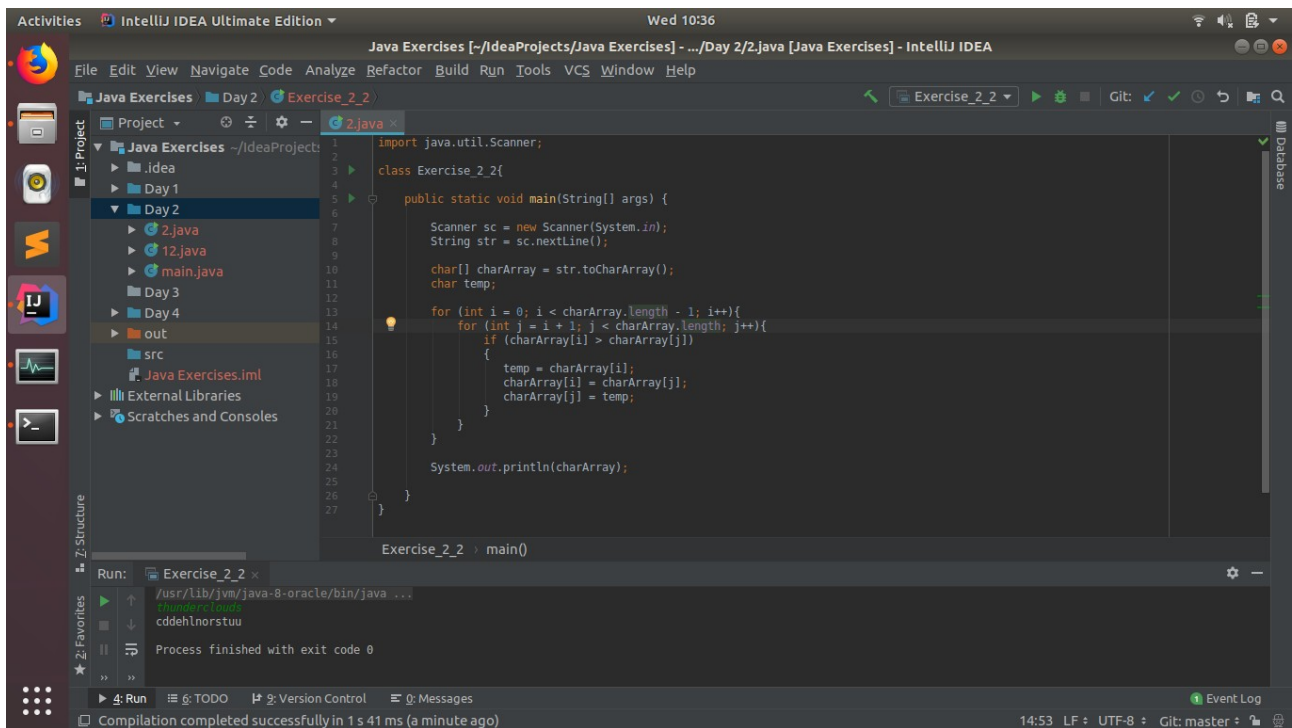
    abstract Book issueBook(int libraryId, int accessLevelRequired);
    abstract void addBook(int Book);
    abstract void showDetails();
}

class Library {
    static ArrayList<Book> books = new ArrayList<>();

    void showLibraryStatus() {}
}
```

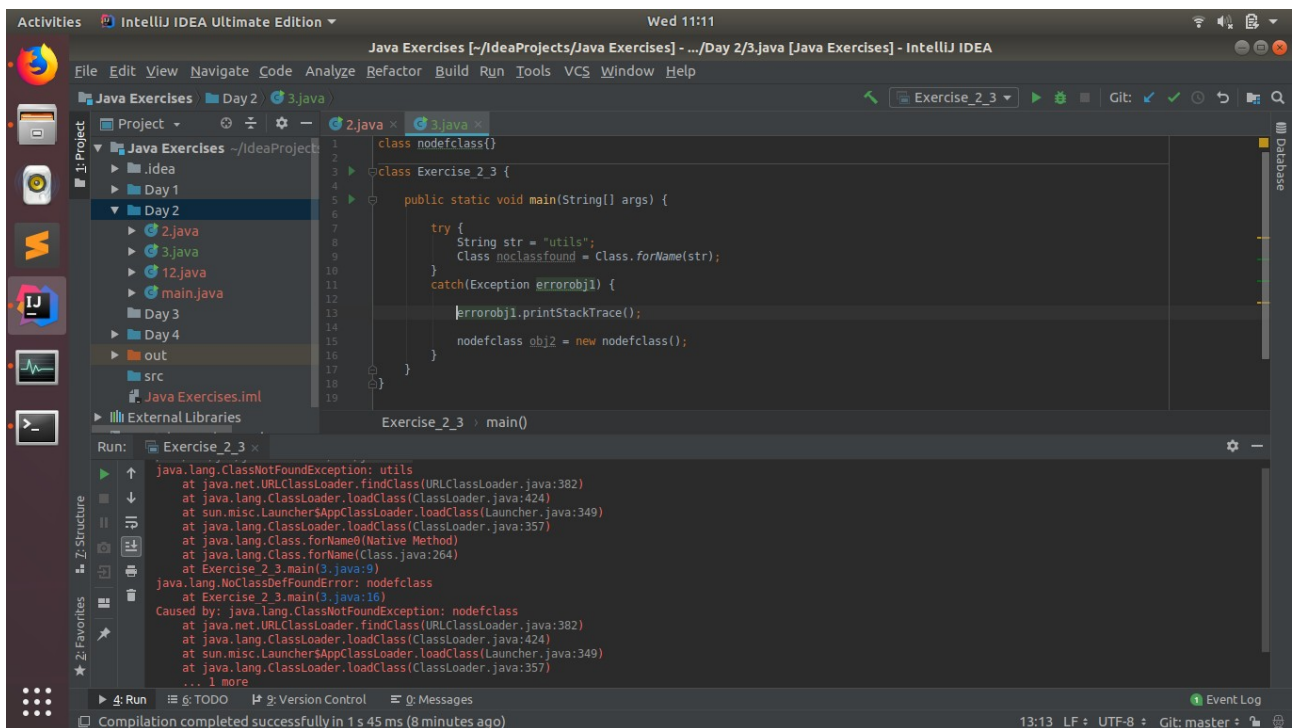
Q2. WAP to sorting string without using string Methods?.

A2.



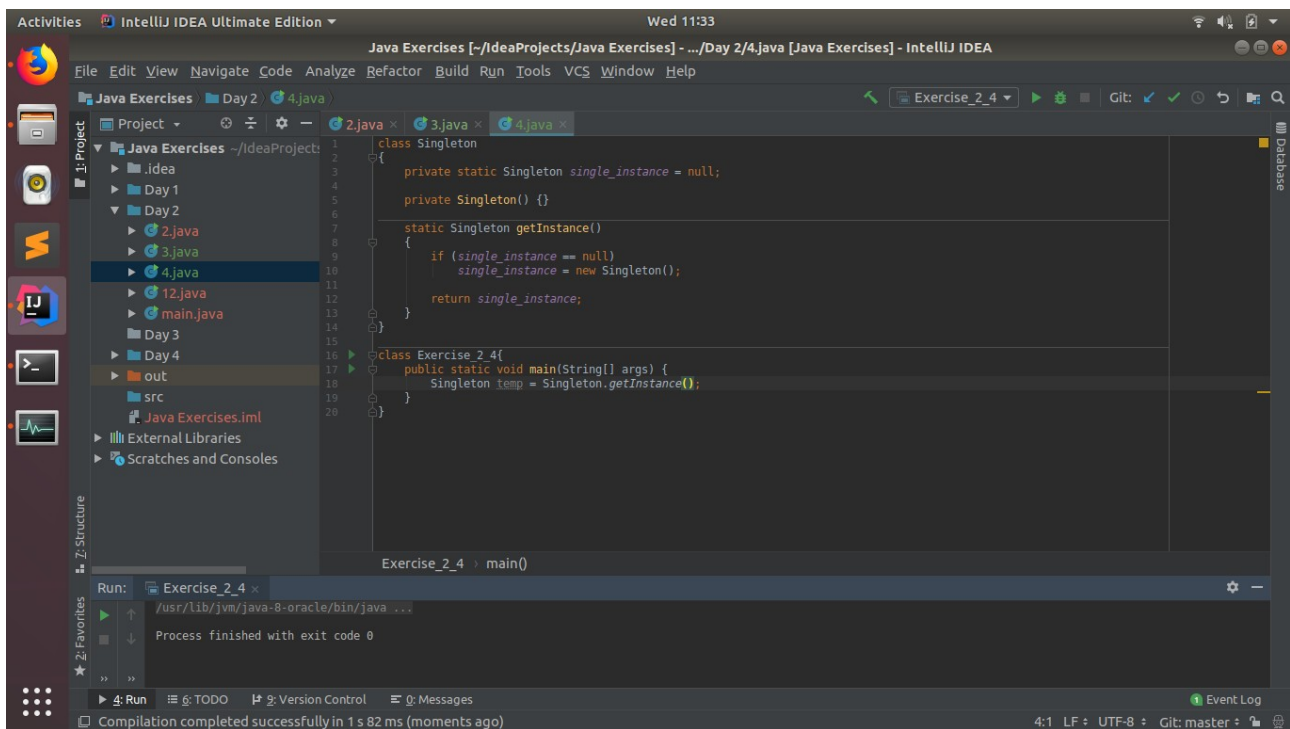
Q3. WAP to produce NoClassDefFoundError and ClassNotFoundException exception.

A3.



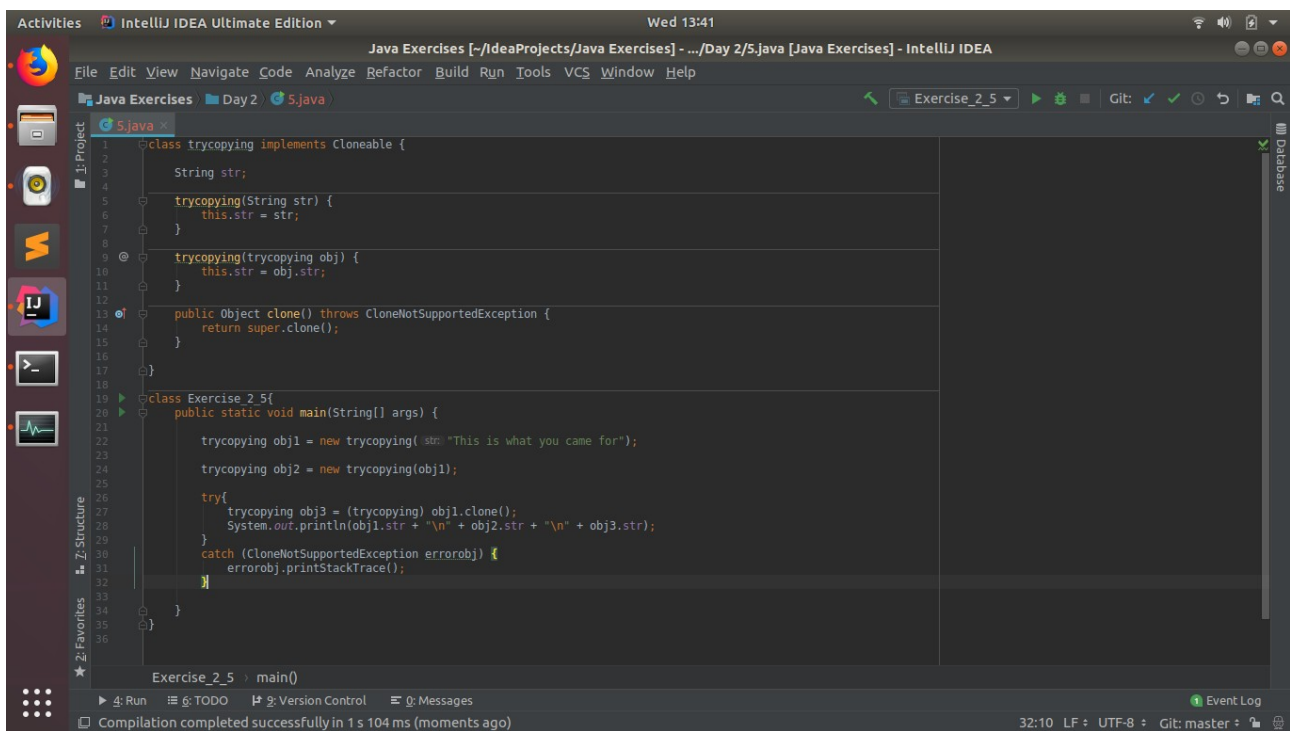
Q4. WAP to create singleton class.

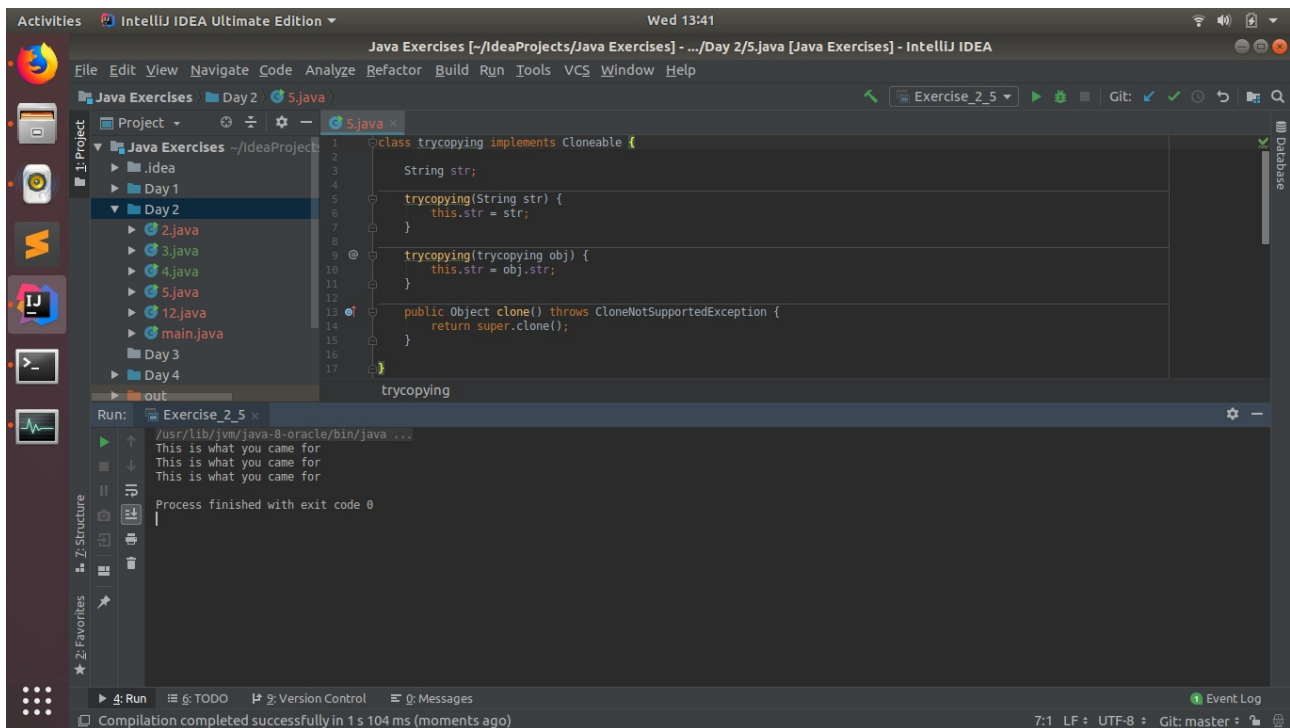
A4.



Q5. WAP to show object cloning in java using cloneable and copy constructor both.

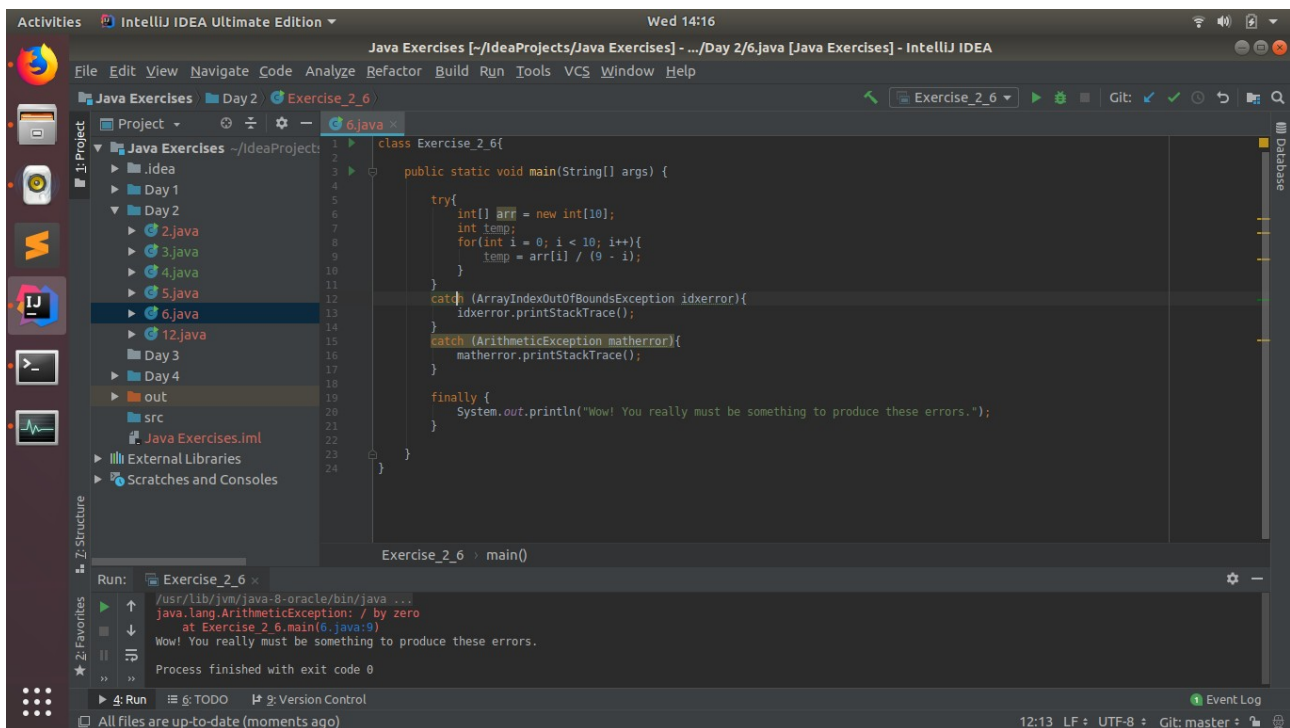
A5.





Q6. WAP showing try, multi-catch and finally blocks.

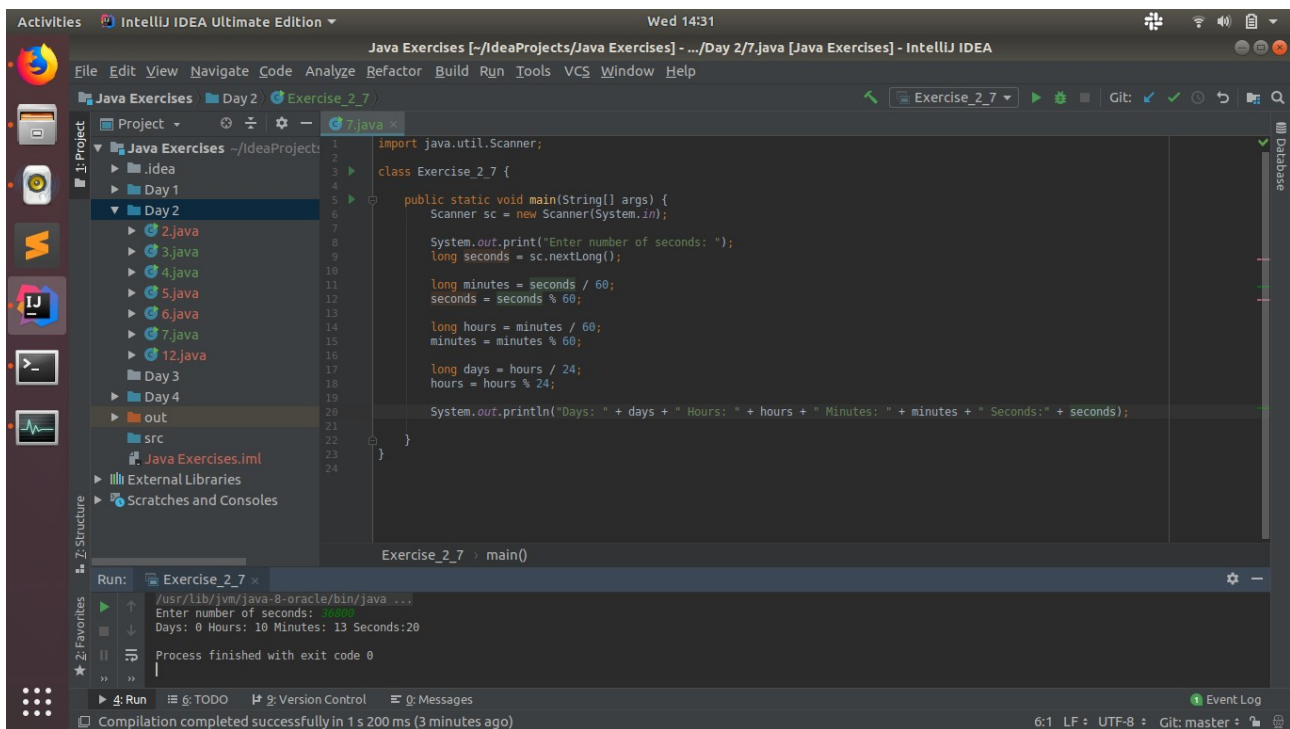
A6.





Q7. WAP to convert seconds into days, hours, minutes and seconds.

A7.



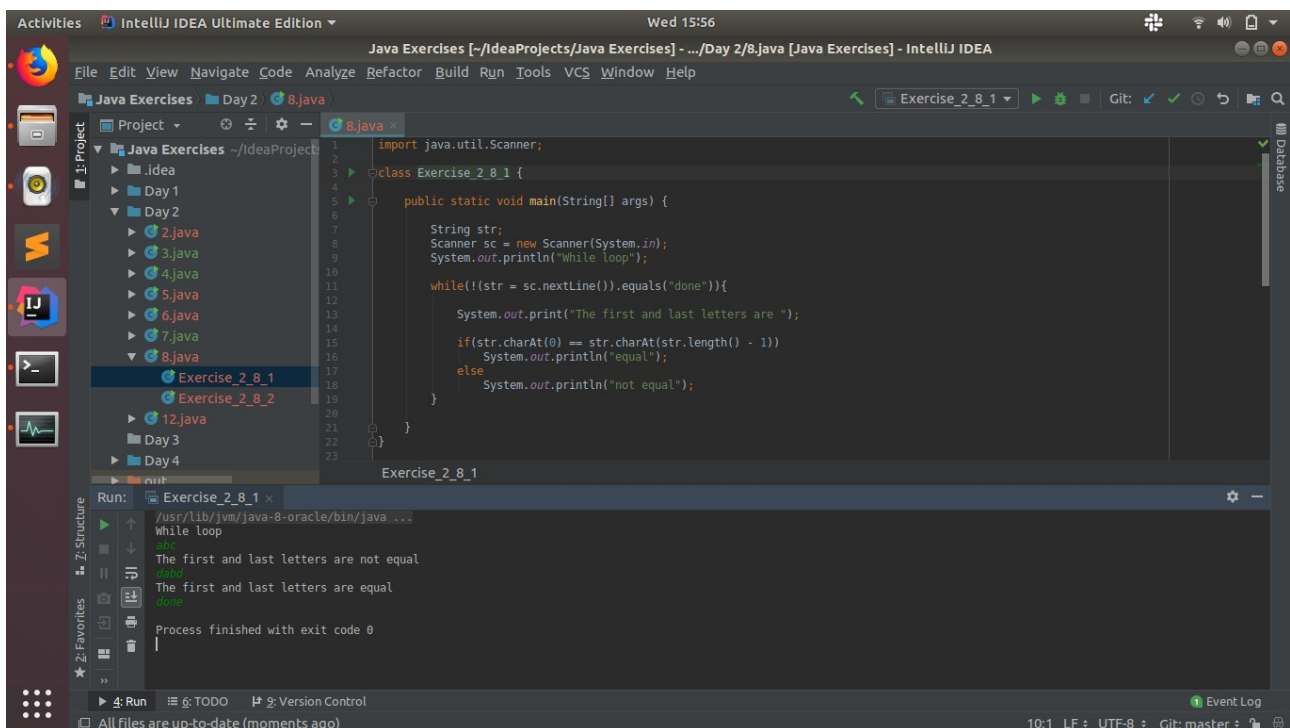
The screenshot shows the IntelliJ IDEA interface with a Java project named 'Java Exercises'. The file explorer on the left shows a directory structure with 'Day 2' containing files '2.java' through '12.java'. The main editor displays the code for 'Exercise\_2\_7.java'. The code imports 'java.util.Scanner' and defines a class 'Exercise\_2\_7' with a 'main' method. The 'main' method prompts the user to enter the number of seconds, reads the input, and then calculates the equivalent days, hours, minutes, and seconds using integer division and modulus. The output is printed as 'Days: 0 Hours: 10 Minutes: 13 Seconds: 20'. The Run window at the bottom shows the execution of 'Exercise\_2\_7' with the same output.

```
1 import java.util.Scanner;
2
3 class Exercise_2_7 {
4
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7
8         System.out.print("Enter number of seconds: ");
9         long seconds = sc.nextLong();
10
11         long minutes = seconds / 60;
12         seconds = seconds % 60;
13
14         long hours = minutes / 60;
15         minutes = minutes % 60;
16
17         long days = hours / 24;
18         hours = hours % 24;
19
20         System.out.println("Days: " + days + " Hours: " + hours + " Minutes: " + minutes + " Seconds: " + seconds);
21     }
22 }
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Q8. WAP to read words from the keyboard until the word done is entered. For each word except done, report whether its first character is equal to its last character. For the required loop, use a

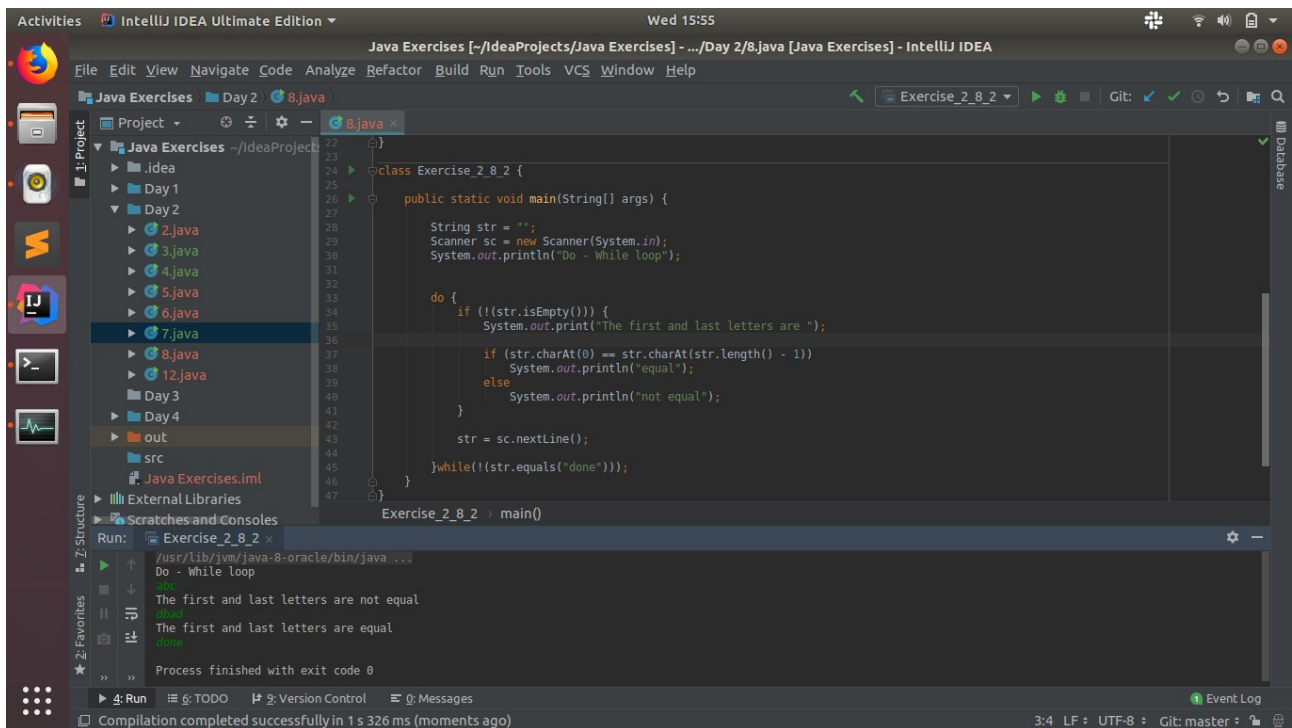
- a) while statement
- b) do-while statement

A8.



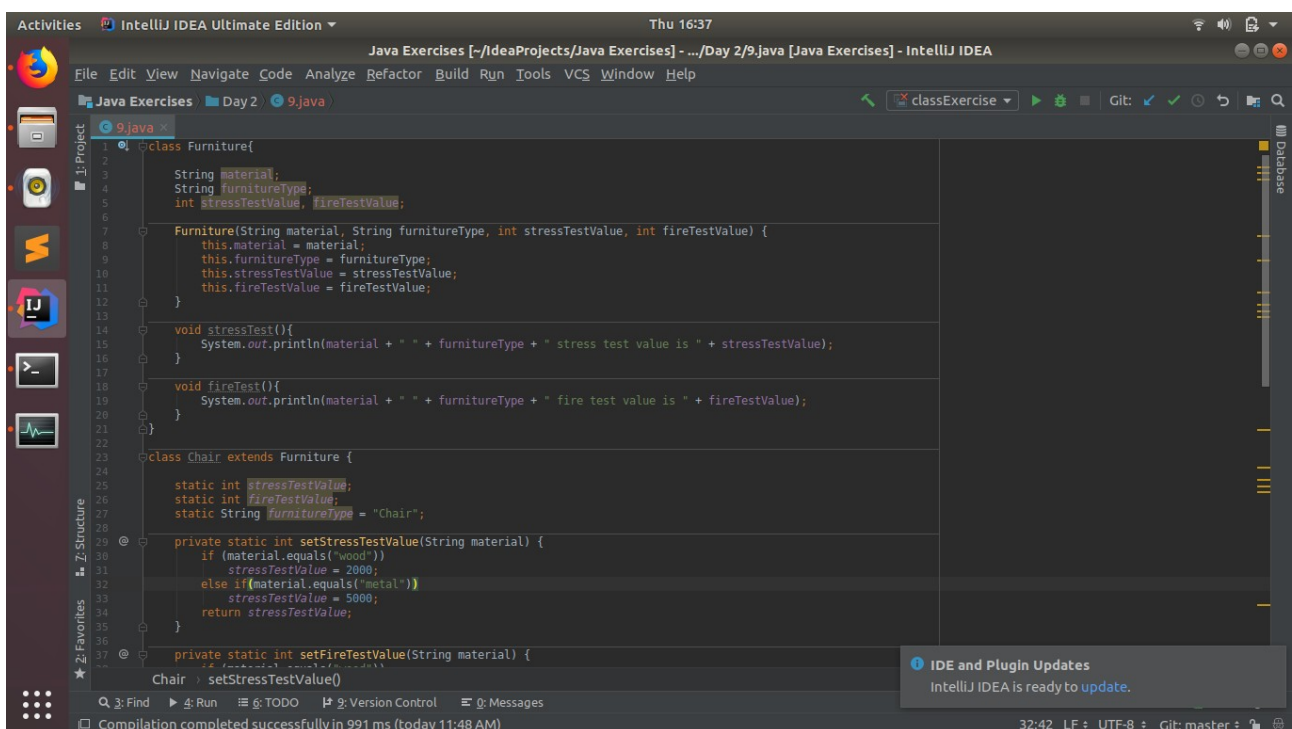
The screenshot shows the IntelliJ IDEA interface with a Java project named 'Java Exercises'. The file explorer on the left shows a directory structure with 'Day 2' containing files '2.java' through '12.java'. The main editor displays the code for 'Exercise\_2\_8\_1.java'. The code imports 'java.util.Scanner' and defines a class 'Exercise\_2\_8\_1' with a 'main' method. The 'main' method prompts the user to enter a word, reads the input, and then checks if the first and last characters are equal. If they are equal, it prints 'The first and last letters are equal'; otherwise, it prints 'The first and last letters are not equal'. The loop continues until the user enters 'done'. The Run window at the bottom shows the execution of 'Exercise\_2\_8\_1' with the output 'The first and last letters are not equal' and 'The first and last letters are equal'.

```
1 import java.util.Scanner;
2
3 class Exercise_2_8_1 {
4
5     public static void main(String[] args) {
6         String str;
7         Scanner sc = new Scanner(System.in);
8         System.out.println("while loop");
9
10        while(!(str = sc.nextLine()).equals("done")){
11
12            System.out.print("The first and last letters are ");
13
14            if(str.charAt(0) == str.charAt(str.length() - 1))
15                System.out.println("equal");
16            else
17                System.out.println("not equal");
18        }
19    }
20 }
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```



Q9. Design classes having attributes for furniture where there are wooden chairs and tables, metal chairs and tables. There are stress and fire tests for each products.

A9.



```

class Chair extends Furniture {
    static int stressTestValue;
    static int fireTestValue;
    static String furnitureType = "Chair";

    private static int setStressTestValue(String material) {
        if (material.equals("wood"))
            stressTestValue = 2000;
        else if (material.equals("metal"))
            stressTestValue = 5000;
        return stressTestValue;
    }

    private static int setFireTestValue(String material) {
        if (material.equals("wood"))
            fireTestValue = 250;
        else if (material.equals("metal"))
            fireTestValue = 600;
        return fireTestValue;
    }

    Chair(String material) {
        super(material, furnitureType, setStressTestValue(material), setFireTestValue(material));
    }
}

class Table extends Furniture {
    static int stressTestValue;
    static int fireTestValue;
    static String furnitureType = "Table";

    private static int setStressTestValue(String material) {
        if (material.equals("wood"))
            stressTestValue = 3000;
        else if (material.equals("metal"))
            stressTestValue = 6000;
        return stressTestValue;
    }
}

```

```

class Chair extends Furniture {
    static int stressTestValue;
    static int fireTestValue;
    static String furnitureType = "Chair";

    private static int setStressTestValue(String material) {
        if (material.equals("wood"))
            stressTestValue = 2000;
        else if (material.equals("metal"))
            stressTestValue = 5000;
        return stressTestValue;
    }

    private static int setFireTestValue(String material) {
        if (material.equals("wood"))
            fireTestValue = 250;
        else if (material.equals("metal"))
            fireTestValue = 600;
        return fireTestValue;
    }

    Chair(String material) {
        super(material, furnitureType, setStressTestValue(material), setFireTestValue(material));
    }
}

class Table extends Furniture {
    static int stressTestValue;
    static int fireTestValue;
    static String furnitureType = "Table";

    private static int setStressTestValue(String material) {
        if (material.equals("wood"))
            stressTestValue = 3000;
        else if (material.equals("metal"))
            stressTestValue = 6000;
        return stressTestValue;
    }

    private static int setFireTestValue(String material) {
        if (material.equals("wood"))
            fireTestValue = 350;
        else if (material.equals("metal"))
            fireTestValue = 650;
        return fireTestValue;
    }

    Table(String material) {
        super(material, furnitureType, setStressTestValue(material), setFireTestValue(material));
    }
}

```

Q10. Design classes having attributes and method(only skeleton) for a coffee shop. There are three different actors in our scenario and i have listed the different actions they do also below

\* Customer

- Pays the cash to the cashier and places his order, get a token number back
  - Waits for the intimation that order for his token is ready
  - Upon intimation/notification he collects the coffee and enjoys his drink
- ( Assumption: Customer waits till the coffee is done, he wont timeout and cancel the order.

Customer always likes the drink served. Exceptions like he not liking his coffee, he getting wrong coffee are not considered to keep the design simple.)

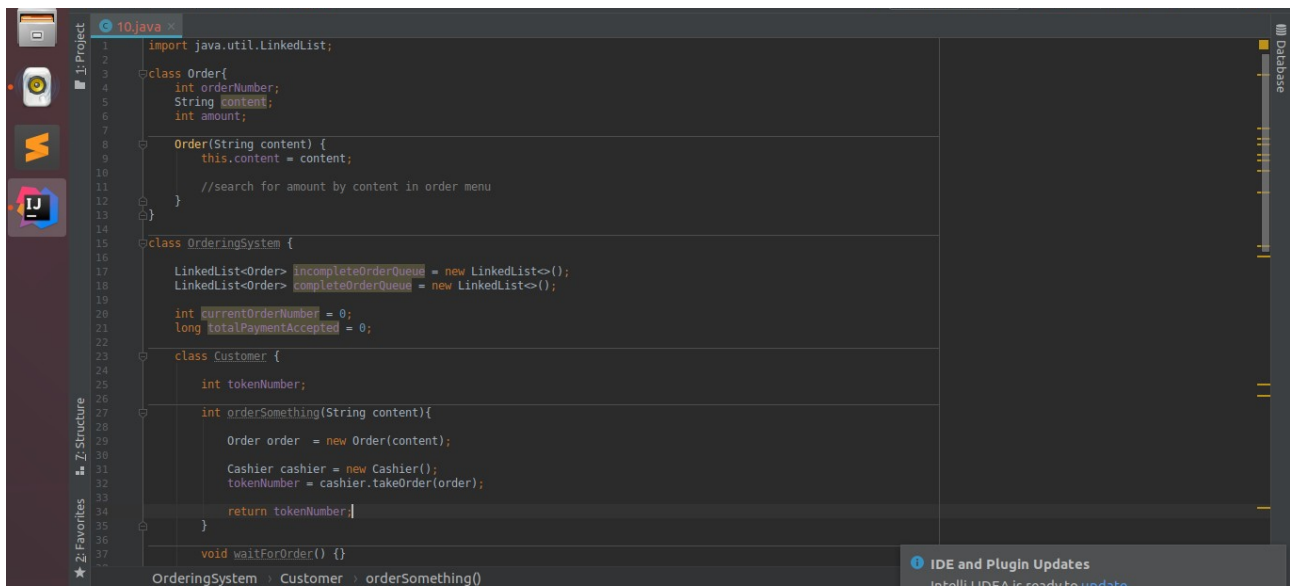
\* Cashier

- Takes an order and payment from the customer
  - Upon payment, creates an order and places it into the order queue
  - Intimates the customer that he has to wait for his token and gives him his token
- ( Assumption: Token returned to the customer is the order id. Order queue is unlimited. With a simple modification, we can design for a limited queue size)

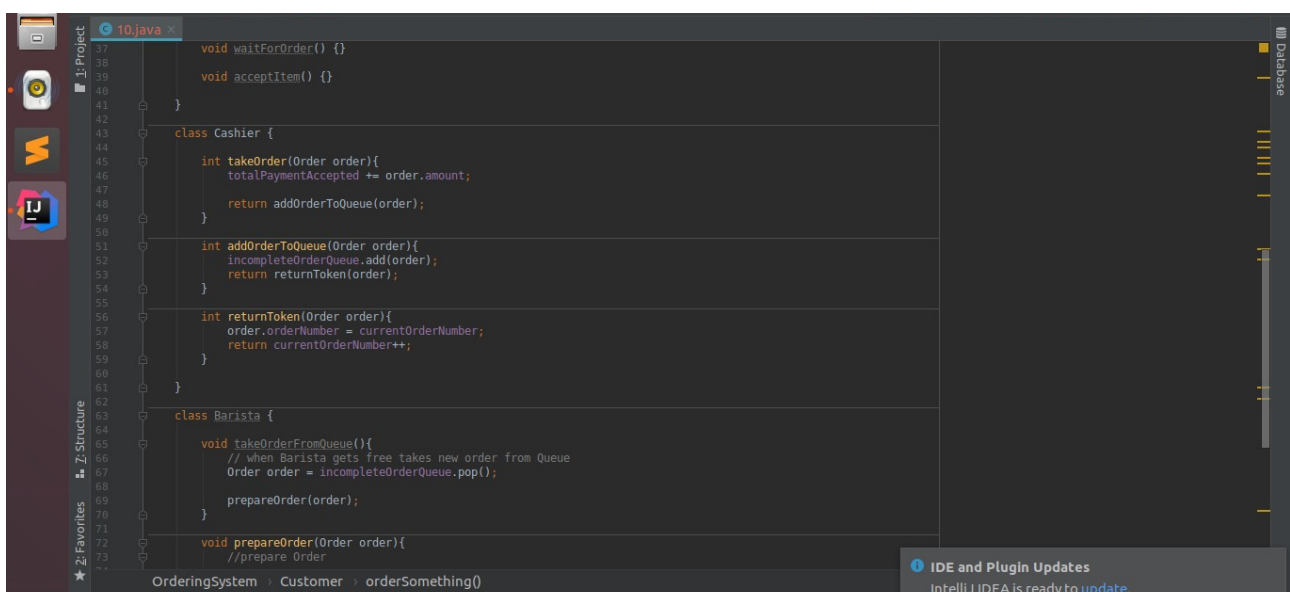
\* Barista

- Gets the next order from the queue
- Prepares the coffee
- Places the coffee in the completed order queue
- Places a notification that order for token is ready

A10.

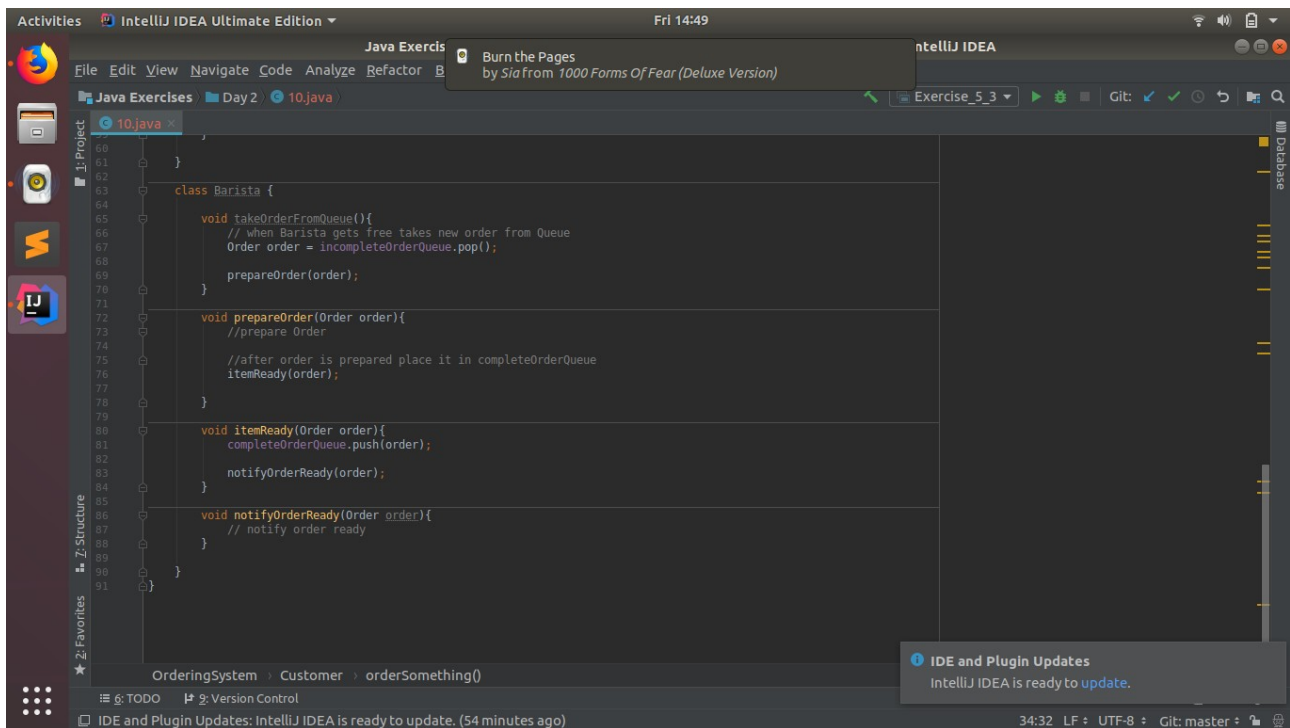


```
1 import java.util.LinkedList;
2
3 class Order {
4     int orderNumber;
5     String content;
6     int amount;
7
8     Order(String content) {
9         this.content = content;
10        //search for amount by content in order menu
11    }
12 }
13
14 class OrderingSystem {
15
16     LinkedList<Order> incompleteOrderQueue = new LinkedList<>();
17     LinkedList<Order> completeOrderQueue = new LinkedList<>();
18
19     int currentOrderNumber = 0;
20     long totalPaymentAccepted = 0;
21
22     class Customer {
23
24         int tokenNumber;
25
26         int orderSomething(String content){
27             Order order = new Order(content);
28
29             Cashier cashier = new Cashier();
30             tokenNumber = cashier.takeOrder(order);
31
32             return tokenNumber;
33         }
34
35         void waitForOrder() {}
36     }
37
38     void orderSomething()
```



```
37     void waitForOrder() {}
38
39     void acceptItem() {}
40 }
41
42 class Cashier {
43
44     int takeOrder(Order order){
45         totalPaymentAccepted += order.amount;
46
47         return addOrderToQueue(order);
48     }
49
50     int addOrderToQueue(Order order){
51         incompleteOrderQueue.add(order);
52         return returnToken(order);
53     }
54
55     int returnToken(Order order){
56         order.orderNumber = currentOrderNumber;
57         return currentOrderNumber++;
58     }
59 }
60
61 class Barista {
62
63     void takeOrderFromQueue(){
64         // when Barista gets free takes new order from Queue
65         Order order = incompleteOrderQueue.pop();
66
67         prepareOrder(order);
68     }
69
70     void prepareOrder(Order order){
71         //prepare Order
72     }
73 }
```



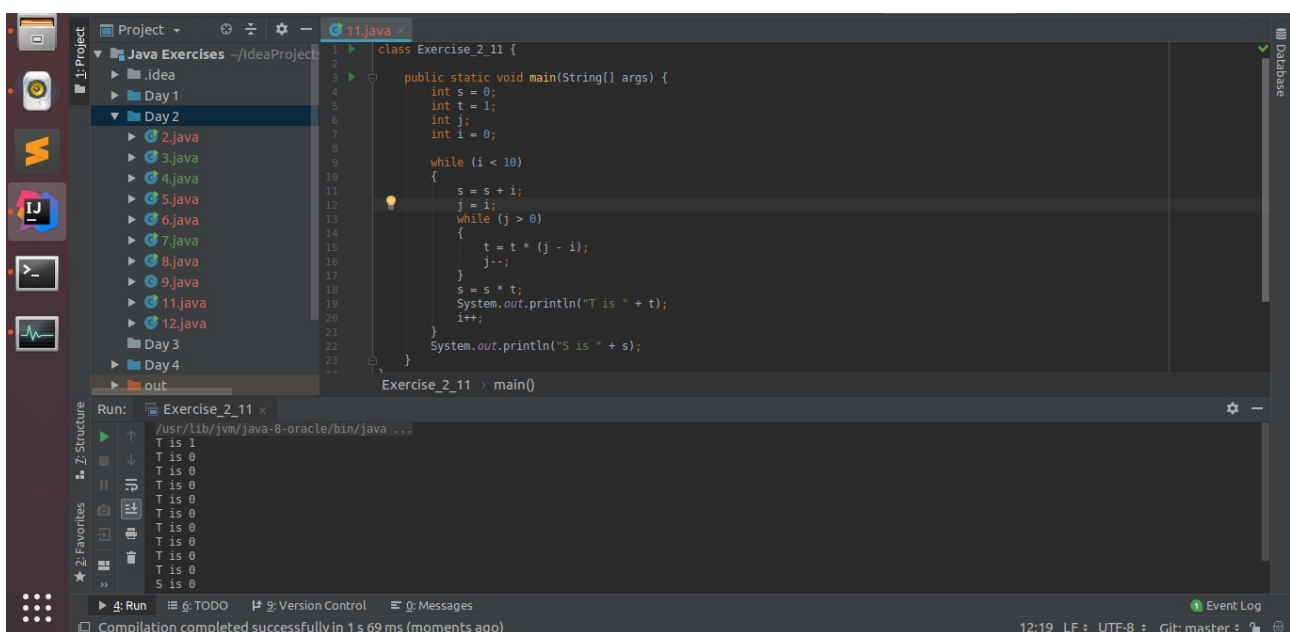


Q11. Convert the following code so that it uses nested while statements instead of for statements:

```

int s = 0;
int t = 1;
for (int i = 0; i < 10; i++)
{
    s = s + i;
    for (int j = i; j > 0; j--)
    {
        t = t * (j - i);
    }
    s = s * t;
    System.out.println("T is " + t);
}
System.out.println("S is " + s);

```



Q12.What will be the output on new Child(); ?

```
class Parent extends Grandparent {
```

```
    {  
    System.out.println("instance - parent");  
    }
```

```
    public Parent() {  
    System.out.println("constructor - parent");  
    }
```

```
    static {  
    System.out.println("static - parent");  
    }
```

```
}
```

```
class Grandparent {
```

```
    static {  
    System.out.println("static - grandparent");  
    }
```

```
    {  
    System.out.println("instance - grandparent");  
    }
```

```
    public Grandparent() {  
    System.out.println("constructor - grandparent");  
    }
```

```
}
```

```
class Child extends Parent {
```

```
    public Child() {  
    System.out.println("constructor - child");  
    }
```

```
    static {  
    System.out.println("static - child");  
    }
```

```
    {  
    System.out.println("instance - child");  
    }
```

```
}
```

IntelliJ IDEA Ultimate Edition

Java Exercises [~/IdeaProjects/Java Exercises] - .../Day 2/12.java [Java Exercises] - IntelliJ IDEA

```
1 class Parent extends Grandparent {
2
3     System.out.println("instance - parent");
4 }
5
6 public Parent() {
7     System.out.println("constructor - parent");
8 }
9
10 static {
11     System.out.println("static - parent");
12 }
13
14
15 class Grandparent {
16
17     static {
18         System.out.println("static - grandparent");
19     }
20
21
22     System.out.println("instance - grandparent");
23 }
24
25 public Grandparent() {
26     System.out.println("constructor - grandparent");
27 }
28
29
30 class Child extends Parent {
31
32     public Child() {
33         System.out.println("constructor - child");
34     }
35
36
37     static {
38         System.out.println("static - child");
39     }
40
41
42     System.out.println("instance - child");
43 }
44
45 class Exercise_2_12 {
46     public static void main(String[] args) {
47         Child c = new Child();
48     }
49 }
50
```

Run: 4 Run 5 TODO 6 Version Control 7 Messages 8 Event Log

Compilation completed successfully in 1 s 33 ms (a minute ago)

IntelliJ IDEA Ultimate Edition

Java Exercises [~/IdeaProjects/Java Exercises] - .../Day 2/12.java [Java Exercises] - IntelliJ IDEA

```
19     System.out.println("static - grandparent");
20 }
21
22     System.out.println("instance - grandparent");
23 }
24
25 public Grandparent() {
26     System.out.println("constructor - grandparent");
27 }
28
29
30 class Child extends Parent {
31
32     public Child() {
33         System.out.println("constructor - child");
34     }
35
36
37     static {
38         System.out.println("static - child");
39     }
40
41
42     System.out.println("instance - child");
43 }
44
45 class Exercise_2_12 {
46     public static void main(String[] args) {
47         Child c = new Child();
48     }
49 }
50
```

Run: 4 Run 5 TODO 6 Version Control 7 Messages 8 Event Log

Compilation completed successfully in 1 s 33 ms (a minute ago)

Run: Exercise\_2\_12 x

```
static - grandparent
static - parent
static - child
instance - grandparent
constructor - grandparent
instance - parent
constructor - parent
instance - child
constructor - child
Process finished with exit code 0
```

Run: 4 Run 5 TODO 6 Version Control 7 Messages 8 Event Log

Compilation completed successfully in 1 s 33 ms (a minute ago)

Q13. Create a custom exception that do not have any stack trace

A13.

