60.1.1. What are exceptions?

An exception by its very definition is something which is not normal.

In programming, if the program does not run as expected due to some abnormal event or situation, the error produced is called an exception.

Java provides easy to use constructs to handle such situations.

Even before we learn how to handle exceptions let us see an exception and try fixing it.

The below example code when run produces an exception called java.lang.ArithmeticException.

Click on **Submit** button to see the exception details as given below.

```
Caused by: java.lang.ArithmeticException: / by zero   // it has exception class name followed by
        at ExceptionDemo1.main(ExceptionDemo1.java:5) // it has the class/method name and the
        ... 6 more
```

After you click on `Submit` and see the exception message, replace the value `0` of the `divisor` variable with `2` to fix the problem.

Important: Please note that the blue animating arrow which is shown when you click **Submit**, is shown by our **intelligent error detection system**. When you start coding using a regular IDE during work, you will not be helped like this. You will only be provided the error information (stack trace) without this animating arrow. Hence, learning how to read and understand exception stack traces becomes an essential part of programming.

Note: Please don't change the package name.

Sample Test Cases

Explorer

Exceptio...

Submit

```
1  package q11317;
2  public class ExceptionDemo1 {
3      public static void main(String[] args) {
4
5          System.out.println("result = 17");
6      }
7  }
```

Terminal Test cases

v to read and understand exception stack trace?

00:37

exception occurs while the code is running, the JVM tries to provide as much information as regarding the line of code which triggered the exception.

ormation usually spans multiple lines which is called the exception stack trace.

n `Submit` button to see the exception stack trace produced while executing the code.

ry easy to fix the error in this scenario. However, we will first click on the `Submit` button and then learn d the stack trace information provided.

```
used by: java.lang.ArithmeticException: / by zero     // it contains the exception class name fol
        at ExceptionDemo2.divide(ExceptionDemo2.java:9)   // line - 9
        at ExceptionDemo2.main(ExceptionDemo2.java:5)     // line - 5
        ... 6 more
```

e exception stack trace contains two lines from the ExceptionDemo2 class.

he `top most line` in the **exception stack trace** which is from our code (meaning from a class written by s) is responsible for causing the exception.

When the statement in **line 9** in our code (ExceptionDemo2 class) is analyzed for the **ArithmeticException** with error message `/ by zero` , we can easily figure out that the value contained in the variable `divisor` is `0` (zero).

There can be some scenarios where the `top most line` is from a class available in Java standard classes or a class written by a third party library provider, in such cases we will have to ignore such lines till we find the lines which are from classes written by us.

Sample Test Cases

Explorer

Exceptio...

```
1  package q11318;
2  public class ExceptionDemo2 {
3      public static void main(String[] args) {
4
5          System.out.println("result = 17");
6      }
7  }
```

Terminal Test cases

.1.3. How to read and understand exception stack trace?

00:34

he exception stack trace can include classes which are not written by us. The below code when submitted ill terminate with an exception whose stack trace will contain method call information from Java classes in va.lang package.

s a learner it is easy to get lost when we see so many error lines.

lowever, it is extremely easy to pinpoint the line with error if we follow a simple thumb rule.

o start with let us click on `Submit` button to see the exception stack trace produced by the code.

Do not try to fix the code. First let us try to understand the exception stack trace we get when we click `Submit` .

```
Caused by: java.lang.NumberFormatException: For input string: "4a"  // notice the exception class n
        at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
        at java.lang.Integer.parseInt(Integer.java:580)
        at java.lang.Integer.parseInt(Integer.java:615)
        at ExceptionDemo3.convertAndAdd(ExceptionDemo3.java:10)
        at ExceptionDemo3.main(ExceptionDemo3.java:5)
        ... 6 more
```

The first thing we should read in the exception stack trace is the line starting with `Caused by:` which contains the **name of the exception class** and the **error message**.

In our case the name of the exception class is NumberFormatException

And the error message is `For input string: "4a"`

Sample Test Cases

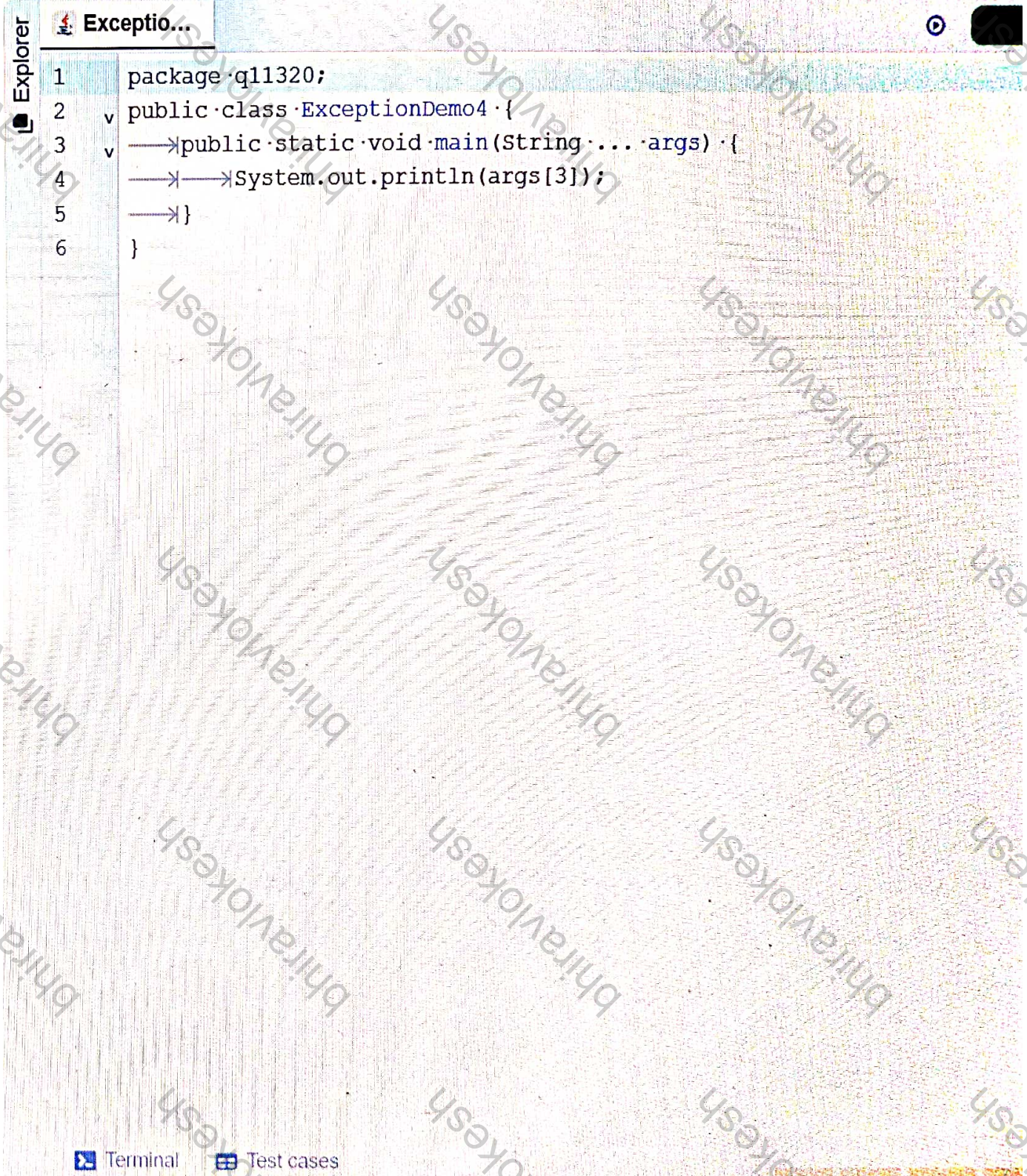
Explorer

Exceptio...

```
1  package q11319;
2  public class ExceptionDemo3 {
3      public static void main(String[] args) {
4
5          System.out.println("result = 7");
6      }
7  }
```

Terminal Test cases

Plots


Explorer
Exceptio...
1 package q11320;
2 public class ExceptionDemo4 {
3 public static void main(String ... args) {
4 System.out.println(args[3]);
5 }
6 }
Terminal
Test cases

- [ ] ParseException is an unchecked exception.
- [ ] NumberFormatException is an checked exception.
- [x] RuntimeException is an unchecked exception.
- [x] NullPointerException is an unchecked exception.
- [x] ArithmeticException is an unchecked exception.
- [ ] StackOverflowError is a checked exception.

Explorer

TryCatch...

```
1  package q11322;
2  public class TryCatchDemo {
3      public static void main(String[] args) {
4          System.out.println("Before sleep...");
5
6          System.out.println("After sleep...");
7      }
8  }
```

Terminal Test cases

- The program throws an unchecked exception
- (●) The compiler generates an error
- The program compiles successfully
- The program crashes at runtime

00:44

cked exception. It is thrown by JVM.

able to store the method call information, it class) of an `Exception` class. It is a

e `someMethod()`. And the `someMethod()`

shes the `main` method details in to a stack

main(String[] args)

shes the method information of
nt there are two entries in the **method call**

someMethod()

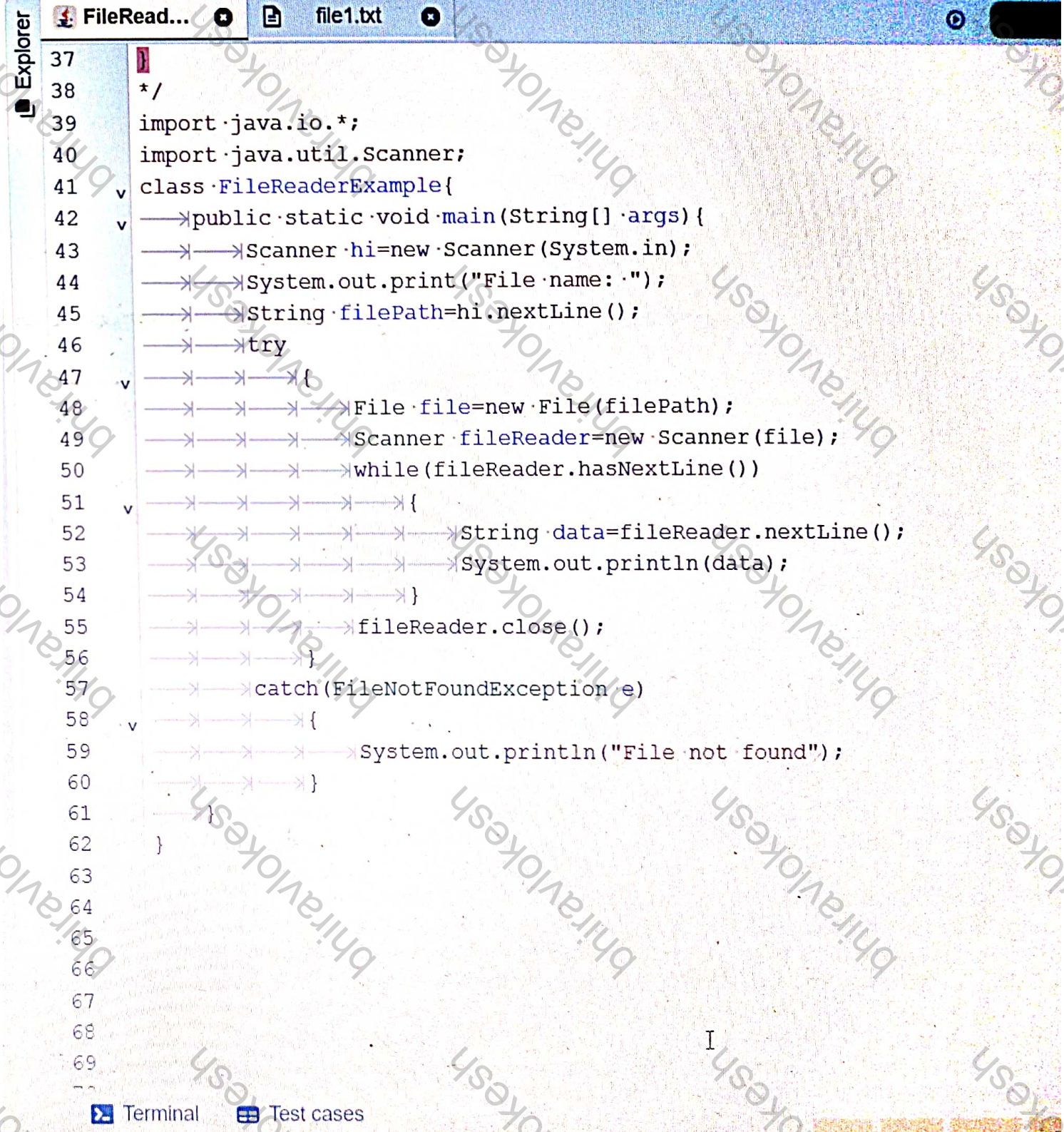
main(String[] args)

the method information of `someMethod()`

Explorer

StackOve...

```
1  package q35975;
2  public class StackOverflowErrorDemo {
3      ....
4      public static void main(String... args) {
5          System.out.println("Stack overflow occurred");
6      ....
7      }
8  }
```


Explorer
FileRead...
file1.txt
}
*/
import java.io.*;
import java.util.Scanner;
class FileReaderExample{
public static void main(String[] args){
Scanner hi=new Scanner(System.in);
System.out.print("File name: ");
String filePath=hi.nextLine();
try
{
File file=new File(filePath);
Scanner fileReader=new Scanner(file);
while(fileReader.hasNextLine())
{
String data=fileReader.nextLine();
System.out.println(data);
}
fileReader.close();
}
catch(FileNotFoundException e)
{
System.out.println("File not found");
}
}
}
Terminal
Test cases


Explorer
ArrayInde...
import java.util.Scanner;
public class ArrayIndexHandler {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
// Input array
int size = scanner.nextInt();
int[] numbers = new int[size];
for (int i = 0; i < size; i++) {
numbers[i] = scanner.nextInt();
}
// Input index
int index = scanner.nextInt();
// Array index validation
try{
int element=numbers[index];
System.out.println("Element at index "+index+" is: "+element);
}
catch(ArrayIndexOutOfBoundsException e){
System.out.println("Error: Index out of bounds");
}
}
}
Terminal
Test cases
‹ Prev
Reset
Submit
Next ›

- ○ Only the try block.
- ○ Only the catch block.
- ○ Only the finally block.
- ◉ Both the catch and finally blocks.

< Prev | Reset | Submit | Next >

r more `catch blocks`, then the `finally block` should be written only

re or in between the `catch blocks`.

✔ Valid

```
nceName1) {

nceName2) {
```

✔ Valid

```
eferenceName) {
```

✔ Valid

ry, `catch` and `finally` blocks given below:

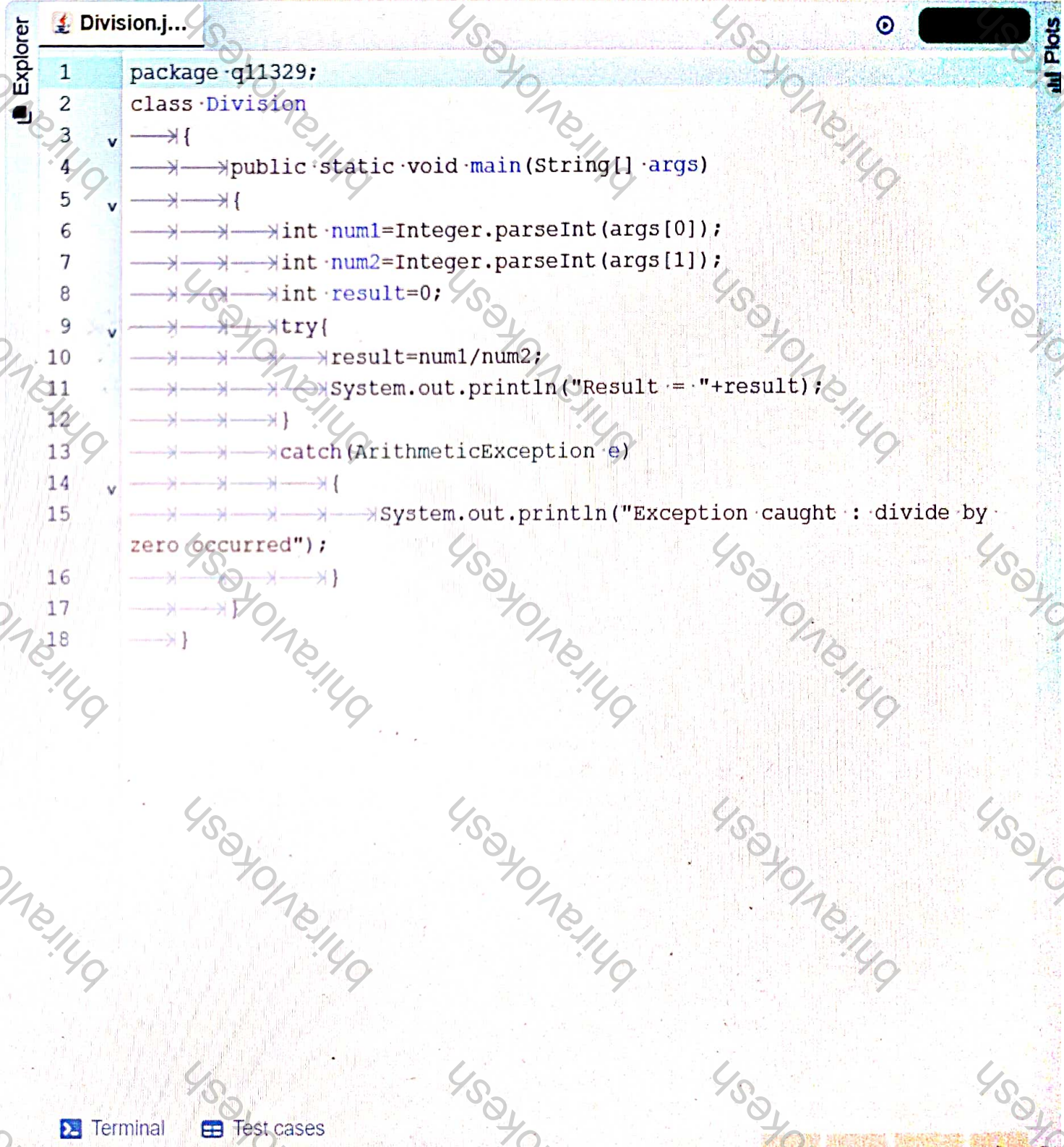
```
} finally {
    ...
} try (ExceptionClassName referenceName) {
    ...
}
```

```
try {
    ...
} finally (ExceptionClassName referenceName) {
    ...
} catch {
    ...
}
```

```
try {
    ...
} catch (ExceptionClassName referenceName) {
    ...
} finally {
    ...
}
```

```
try {
    ...
} finally {
    ...
}
```

- [ ] The code will print `3`.
- [ ] The code will print `34g`.
- [ ] The code will print `-1`.
- [x] The code will print `-2`.



Explorer

Division.j...

Plots

```
1  package q11329;
2  class Division
3  {
4      public static void main(String[] args)
5      {
6          int num1=Integer.parseInt(args[0]);
7          int num2=Integer.parseInt(args[1]);
8          int result=0;
9          try{
10             result=num1/num2;
11             System.out.println("Result = "+result);
12         }
13         catch(ArithmeticException e)
14         {
15             System.out.println("Exception caught : divide by
zero occurred");
16         }
17     }
18 }
```

Terminal Test cases

`ArithmeticException` **divided by zero by using try, catch and finally**

e class `MyFinallyBlock` which will receive **four** arguments and convert
wo into float values.

ocks separately for finding division of two **integers** and two **float** values.

ne arguments to the **main()** as **"10", "4", "10", "4"** then the program should

```
sion : 2

on : 2.5
```

ne arguments to the **main()** as **"5", "0", "3.8", "0.0"** then the program

```
on : Infinity
```

ckage name.

Explorer

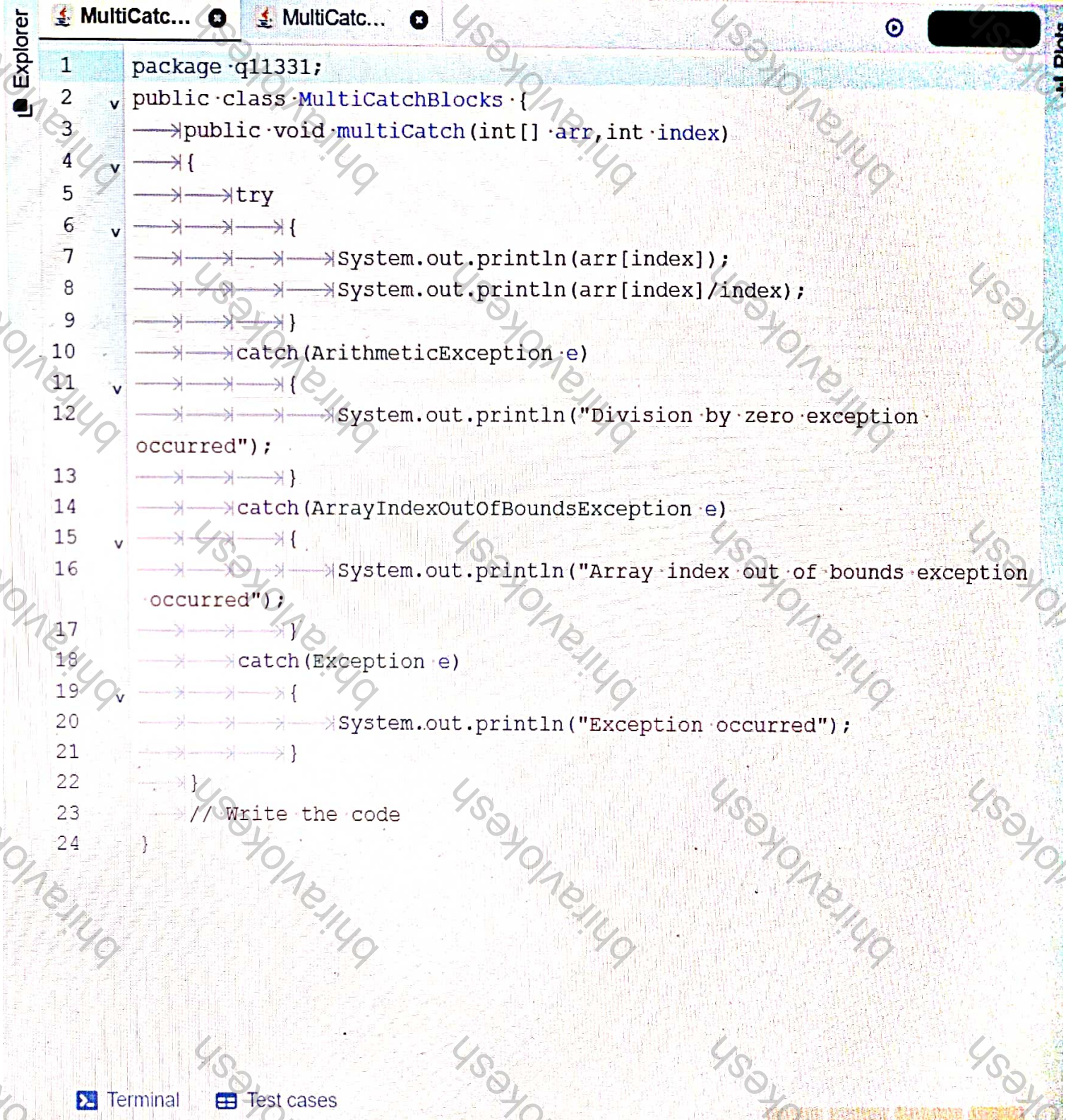
MyFinally...

```
1   package q11330;
2   public class MyFinallyBlock {
3       public static void main(String[] args)
4       {
5           int a=Integer.parseInt(args[0]);
6           int b=Integer.parseInt(args[1]);
7           float c=Float.parseFloat(args[2]);
8           float d=Float.parseFloat(args[3]);
9           try
10          {
11              try
12              {
13                  System.out.println("Result of integer values divi
    "+a/b);
14              }
15              catch(ArithmeticException e)
16              {
17                  System.out.println("Inside the 1st catch bloc
18              }
19              finally
20              {
21                  System.out.println("Inside the 1st finally block"
22              }
23              try
24              {
25                  System.out.println("Result of float values d
    : "+c/d);
26              }
27          catch(ArithmeticException e)
28          {
29              System.out.println("/by zero");
30          }
31          finally
```

Terminal Test cases

```
11          try
12          {
13              System.out.println("Result of integer values division
            :"+a/b);
14          }
15          catch(ArithmeticException e)
16          {
17              System.out.println("Inside the 1st catch block");
18          }
19          finally
20          {
21              System.out.println("Inside the 1st finally block");
22          }
23          try
24          {
25              System.out.println("Result of float values divisio
            n: "+c/d);
26          }
27          catch(ArithmeticException e)
28          {
29              System.out.println("/by zero");
30          }
31          finally
32          {
33              System.out.println("Inside the 2nd finally block");
34          }
35      }
36      catch(Exception e)
37      {
38      System.out.println("Exception");
39      }
40  }
41  }
```

Terminal Test cases

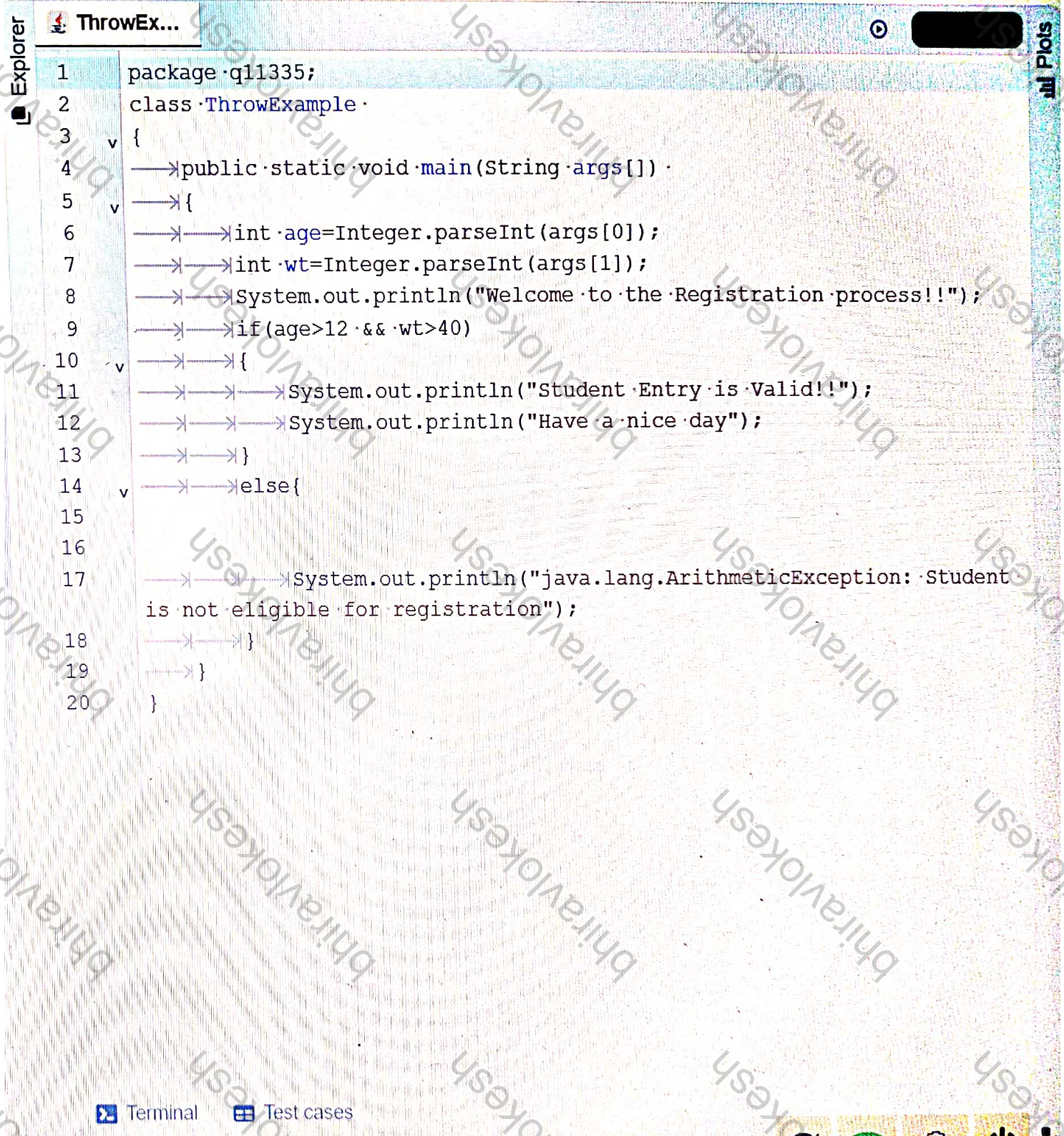

Explorer
MultiCatc...
MultiCatc...
1 package q11331;
2 public class MultiCatchBlocks {
3 public void multiCatch(int[] arr,int index)
4 {
5 try
6 {
7 System.out.println(arr[index]);
8 System.out.println(arr[index]/index);
9 }
10 catch(ArithmeticException e)
11 {
12 System.out.println("Division by zero exception occurred");
13 }
14 catch(ArrayIndexOutOfBoundsException e)
15 {
16 System.out.println("Array index out of bounds exception occurred");
17 }
18 catch(Exception e)
19 {
20 System.out.println("Exception occurred");
21 }
22 }
23 // Write the code
24 }
Terminal
Test cases

Successfully created st1.
st1: name = Ganga, age = 25
Could not create st2. Error message is: Invalid age: 1003. Valid range for age is between 0 and 999.

Could not create st1. Error message is: Invalid age: 25. Valid range for age is between 0 and 999.
Successfully created st2.
st2: name = Yamuna, age = 1003

Could not create st1. Error message is: Invalid age: 25. Valid range for age is between 0 and 999.
Could not create st2. Error message is: Invalid age: 1003. Valid range for age is between 0 and 999.

Successfully created st1.
st1: name = Ganga, age = 25
Successfully created st2.
st2: name = Yamuna, age = 1003


Explorer
ThrowEx...
Plots
1 package q11335;
2 class ThrowExample
3 {
4 public static void main(String args[])
5 {
6 int age=Integer.parseInt(args[0]);
7 int wt=Integer.parseInt(args[1]);
8 System.out.println("Welcome to the Registration process!!");
9 if(age>12 && wt>40)
10 {
11 System.out.println("Student Entry is Valid!!");
12 System.out.println("Have a nice day");
13 }
14 else{
15
16
17 System.out.println("java.lang.ArithmeticException: Student is not eligible for registration");
18 }
19 }
20 }
Terminal
Test cases

ows and throw - Square root calculator.

calculates the square root of a given number. The program should have the

lculateSquareRoot that takes a double value as a parameter.
areRoot method should throw a custom exception
mberException if the input number is negative.
, prompt the user to enter a number.
eSquareRoot method with the user's input and handle
berException using a try-catch block.
is valid (non-negative), print the square root; otherwise, print an error message.

with a partially completed Java program. Your task is to fill in the missing parts of the
ctionality as described in the question text.

Explorer

SquareR...

```
        System.out.print("Enter a number: ");
        double number = scanner.nextDouble();

        try {
            double squareRoot = calculateSquareRoot(number);
            System.out.println("Square root: " + squareRoot);
        } catch (NegativeNumberException e) {
            System.out.println(e.getMessage());
        }
    }
}
*/
import java.util.Scanner;
class SquareRootCalculator
{
    public static void main(String[] args)
    {
        Scanner hi=new Scanner(System.in);
        System.out.print("Enter a number: ");
        int number=hi.nextInt();
        double ss=Math.sqrt(number);
        System.out.println("Square root: "+ss);
    }
}
```

Terminal Test cases


Explorer
CustomE...
Plots
1 package q36057;
2 public class CustomExceptionExample {
3 public static void main(String[] args) {
4 System.out.println("Successfully created st1.");
5 System.out.println("st1 : name = Ganga, age = 25");
6 System.out.println("Could not create st2. Error message is : Invalid age : 1003. Valid range for age is between 0 and 999.");
7 System.out.println("Could not create st3. Error message is : Invalid name : Na. Name has to be a non-null value whose length is between 3 and 100 characters.");
8 }
9 }
Terminal
Test cases

Explorer

BankDem...

Plots

```
1   package q11337;
2   public class BankDemo {
3       public static void main(String [] args) {
4
5           System.out.println("Depositing $1000...");
6
7           System.out.println("Withdrawing $700...");
8
9           System.out.println("Withdrawing $600...");
10          System.out.println("Sorry, short of $300.0 in the account
    number 1001");
11      }
12  }
13
```

Terminal Test cases

Explorer

MyExcept...

```
1  package q11338;
2  public class MyException {
3      public static void main (String[] args) {
4          int number=Integer.parseInt(args[0]);
5          if(number>=25 && number<=45)
6          {
7              System.out.println("Given number : "+number);
8          }
9          else{
10             System.out.println("Please enter a number between 25 and 50");
11             System.out.println("q11338.NumberRangeException");
12         }
13     }
14 }
15
```

Terminal Test cases

‹ Prev Reset Submit Next ›

dRangeException that is thrown when a number is provided that
a method validateRange that takes two integer parameters:
uld throw the InvalidRangeException if the number is greater
prompts the user to enter a number and a maximum value, calls the
the exception appropriately.

y completed Java program. Your task is to fill in the missing parts of the
described in the question text.

```
25          range");
26          } catch (InvalidRangeException e) {
27              // System.out.println(e.getMessage());
28          // }
29      }
30  }
31  */
32  import java.util.Scanner;
33  class RangeValidator
34  {
35      public static void main(String[] args)
36      {
37          Scanner hi=new Scanner(System.in);
38          System.out.print("Number: ");
39          int num=hi.nextInt();
40          System.out.print("Maximum value: ");
41          int max=hi.nextInt();
42          if(num<max)
43          {
44              System.out.println("Number "+num+" is within the valid
    range");
45          }
46          else{
47              System.out.println("Number greater than the maximum
    value");
48          }
49      }
50
51
52
53
54
55
```

```
/*package q11358;
public class Student {
    private String id;
    private String name;
    public Student(String id, String name) {

        //fill in the missing code

    public String toString() {
        return "Student[ id = " + id + ", name=" + name + " ]";
    }
    public static void main(String[] args) {
        Student st1 = new Student("1007", "Ganga");
        System.out.println("st1 : " + st1);
    }
}*/
package q11358;
class Student{
    public static void main(String[] args){
        System.out.println("st1 : Student[ id = 1007, name=Ganga ]");
    }
}

    }
```

Terminal Test cases

- [x] In annotations @ indicates to compiler that what an annotation does.
- [x] Override is a predefined java annotation.
- [ ] We cannot define custom annotations in java
- [x] We can use multiple annotations in one declaration

- [x] Type annotations are supported only the release of Java SE 8
- [x] Type annotations supports improved analysis of Java programs
- [ ] Type annotations can support before release of Java SE 8

- [x] Annotations that are applied to another annotations are called meta annotations.
- [ ] Meta annotations are present in java.lang package
- [x] @Retention specifies that how a marked annotations are stored
- [x] ElementType.METHOD can be applied to a method-level annotation

- [ ] Delete the deprecation warning by using @Delete("deprecation")
- [x] Suppress the warning by using @SuppressWarnings("deprecation")
- [ ] Use @Suppress("deprecation")
- [ ] Use @SuppressWarnings("deprecated")

Explorer

Assertion...

```
1  package q11359;
2  public class AssertionDemo {
3      public static void main(String[] args) {
4          System.out.println("total = 3");
5      }
6  }
```

Terminal Test cases

- java.util
- java.io
- java.nio.file
- java.filesystem

- (●) InputStream and OutputStream
- ( ) Reader and Writer
- ( ) FileReader and FileWriter
- ( ) FileInputStream and FileOutputStream

File copied successfully!
Data written to the file!
Read data from the file: Hello, this is a test string.

File copied successfully!
Data written to the file!

Data written to the file!
Read data from the file: Hello, this is a test string.

File copied successfully!
Read data from the file: Hello, this is a test string.

directory represented by the path.

ng constructors:

```
presenting the specified pathname
le object from a parent path string and a child pa
object from a parent abstract path and a child pa
```

```
ts a file named "example.txt" in the current workin
mple.txt"); // Represents a file named "example.txt
```

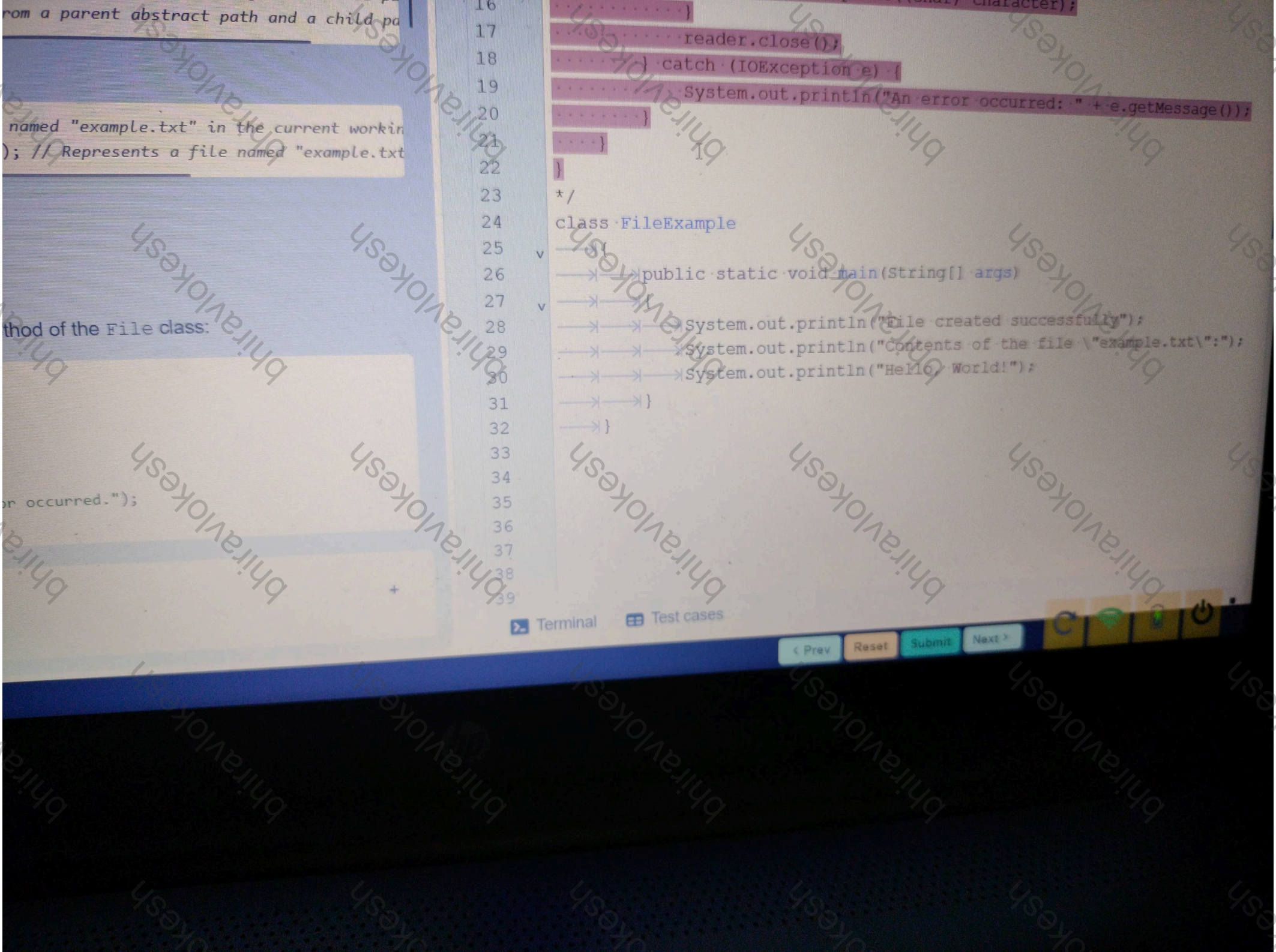
e() method of the File class:

```
ly.");

an error occurred.");
```

```
            FileReader reader = new FileReader(file);
            int character;
            System.out.println("Contents of the file \"" + file.getName
+ "\":");
            while ((character = reader.read()) != -1) {
                System.out.print((char) character);
            }
            reader.close();
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
}
*/
class FileExample
{
    public static void main(String[] args)
    {
        System.out.println("File created successfully");
        System.out.println("Contents of the file \"example.txt\":");
        System.out.println("Hello, World!");
    }
}
```

Terminal Test cases

< Prev Reset Submit Next >


rom a parent abstract path and a child pa
named "example.txt" in the current workin
); // Represents a file named "example.txt
thod of the File class:
or occurred.");
+
16 ....}
17 ..... reader.close();
18 ....} catch (IOException e) {
19 ..... System.out.println("An error occurred: " + e.getMessage());
20 .....}
21 ...}
22 }
23 */
24 class FileExample
25 {
26 public static void main(String[] args)
27 {
28 System.out.println("File created successfully");
29 System.out.println("Contents of the file \"example.txt\":");
30 System.out.println("Hello, World!");
31 }
32 }
33
34
35
36
37
38
39
Terminal
Test cases
‹ Prev
Reset
Submit
Next ›

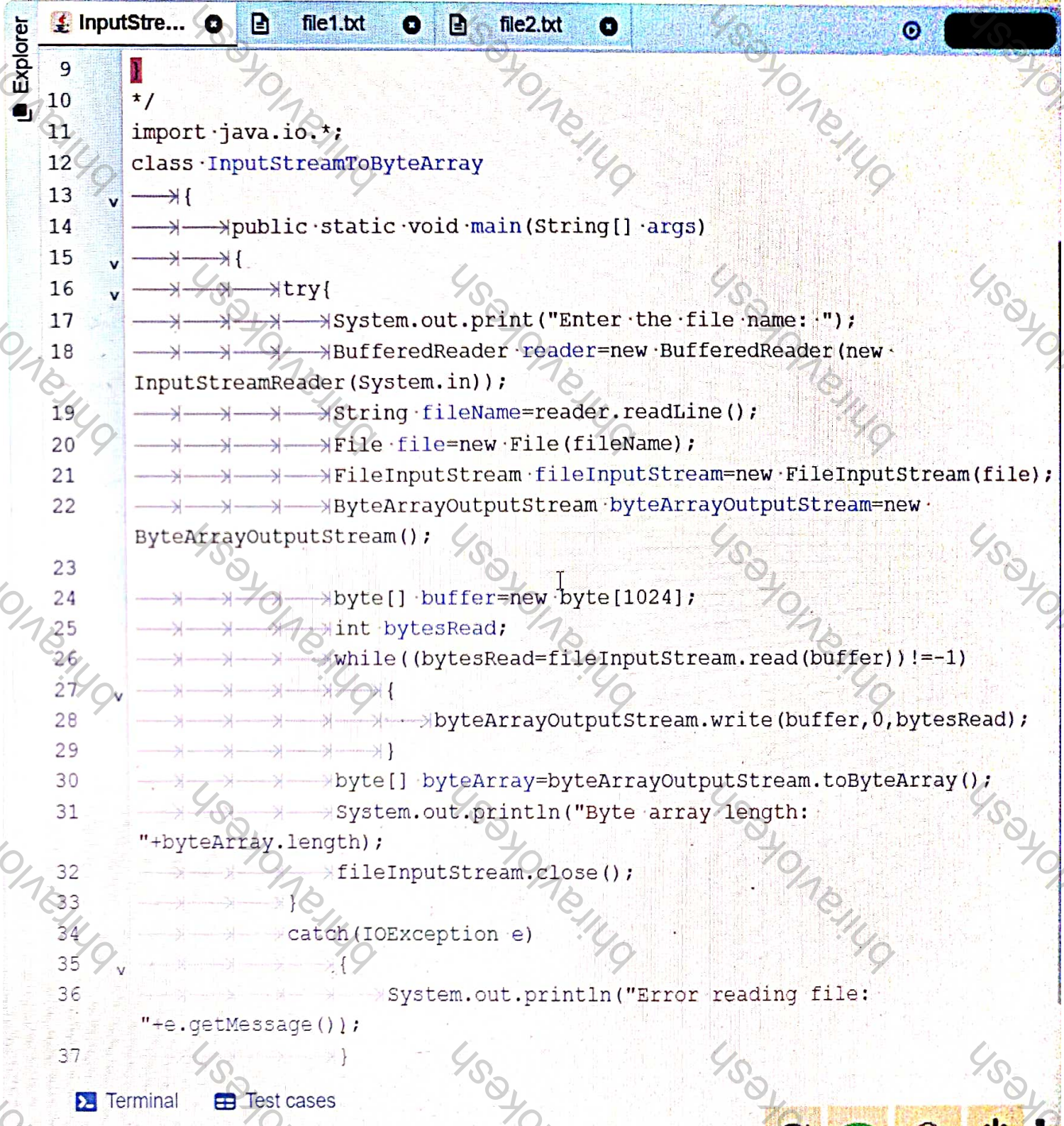
```
+ "\":");
            while ((character = reader.read()) != -1) {
                System.out.print((char) character);
            }
            reader.close();
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
}
*/
class FileExample
{
    public static void main(String[] args)
    {
        System.out.println("File created successfully");
        System.out.println("Contents of the file \"example.txt\":");
        System.out.println("Hello, World!");
    }
}
```

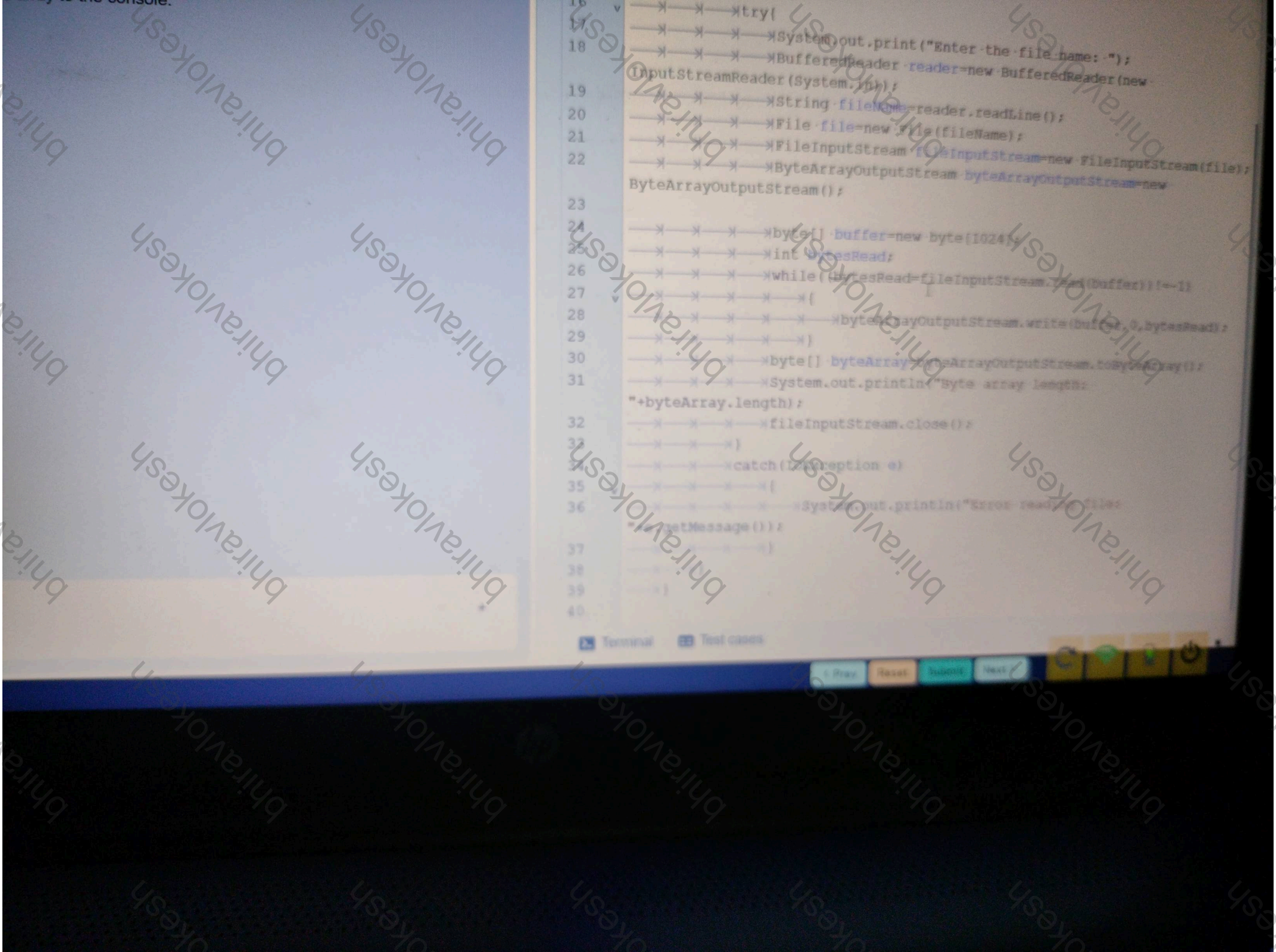
Terminal Test cases

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class FileAppendExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Step 1: Create a File object representing "example.txt"
        File file = new File("example.txt");

        try {

            System.out.println("Text appended to the file successfully");
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        } finally {
            scanner.close(); // Close the scanner
        }
    }
}
*/
import java.util.Scanner;
class FileAppendExample{
    public static void main(String[] args){
        Scanner hi=new Scanner(System.in);
        System.out.print("Text to append: ");
        String ss=hi.nextLine();
        System.out.println("Text appended to the file successfully");
    }
}
```

Terminal Test cases

- ○ OutputStream
- ○ OutputStreamWriter
- ◉ InputStream
- ○ InputStreamReader


Explorer
InputStre...
file1.txt
file2.txt
}
*/
import java.io.*;
class InputStreamToByteArray
{
public static void main(String[] args)
{
try{
System.out.print("Enter the file name: ");
BufferedReader reader=new BufferedReader(new InputStreamReader(System.in));
String fileName=reader.readLine();
File file=new File(fileName);
FileInputStream fileInputStream=new FileInputStream(file);
ByteArrayOutputStream byteArrayOutputStream=new ByteArrayOutputStream();
byte[] buffer=new byte[1024];
int bytesRead;
while((bytesRead=fileInputStream.read(buffer))!=-1)
{
byteArrayOutputStream.write(buffer,0,bytesRead);
}
byte[] byteArray=byteArrayOutputStream.toByteArray();
System.out.println("Byte array length: "+byteArray.length);
fileInputStream.close();
}
catch(IOException e)
{
System.out.println("Error reading file: "+e.getMessage());
}
Terminal
Test cases



```
        try{
            System.out.print("Enter the file name: ");
            BufferedReader reader=new BufferedReader(new
InputStreamReader(System.in));
            String fileName=reader.readLine();
            File file=new File(fileName);
            FileInputStream fileInputStream=new FileInputStream(file);
            ByteArrayOutputStream byteArrayOutputStream=new
ByteArrayOutputStream();

            byte[] buffer=new byte[1024];
            int bytesRead;
            while((bytesRead=fileInputStream.read(buffer))!=-1)
            {
                byteArrayOutputStream.write(buffer,0,bytesRead);
            }
            byte[] byteArray=byteArrayOutputStream.toByteArray();
            System.out.println("Byte array length:
"+byteArray.length);
            fileInputStream.close();
        }
        catch(IOException e)
            {
                System.out.println("Error reading file:
"+e.getMessage());
            }
    }
}
```

inal Test cases

- Byte streams perform automatic conversion from bytes to characters using the system's local character sets.
- Reader and Writer are the super classes for all byte streams.
- The count of characters returned by the read(char[] charArr) method depends on the total length of the character array.
- Closing streams after use is not necessary for resource management.

```
import java.nio.file.*;
public class ReaderWriterDemo {
    public static void main(String[] args) throws IOException {
        StringBuilder sb = new StringBuilder();
        sb.append("This text was written at 1 time\n");
        for (int i = 2; i <= 10; i++) {
            sb.append("This text was written at " + i + " times\n");
        }
    }
}
*/
package q35978;
class ReaderWriterDemo
{
    public static void main(String[] args)
    {
        System.out.println("This text was written at 1 time");
        for(int i=2;i<=10;i++)
        {
            System.out.println("This text was written at "+i+" times");
        }
        System.out.println();
    }
}
```

Terminal Test cases

- ○ All classes implementing the Serializable interface can participate in serialization.
- ◉ The Serializable interface has methods that need to be implemented.
- ○ The ObjectInputStream class is used to read objects from a byte stream.
- ○ If a class does not implement Serializable and an attempt is made to serialize its objects, a runtime exception will occur.

n and deserialization of objects in Java. The program defines a
le interface, allowing instances of the class to be serialized and

t1 and st2, with different attributes such as ID, name, age,
roceeds to serialize these objects to a file named
OutputStream.

details of st1 and st2 before they are serialized. After serialization,
an ObjectInputStream and assigns them to restoredSt1 and

toredSt1 and restoredSt2 after deserialization, demonstrating that
d.

Student objects to a file, and then deserializes them back into new
f object serialization and deserialization in Java.

Explorer

Serializati...

```
34      System.out.println("Before serialization st1 : Student[ id=CT1007, name=Ganga, age=25, seatingPosition=71, comments=Hard-Working ]");
35      System.out.println("Before serialization st2 : Student[ id=CT1008, name=Yamuna, age=26, seatingPosition=51, comments=Absent-Minded ]");
36      System.out.println("After deserialization st1 : Student[ id=CT1007, name=Ganga, age=25, seatingPosition=0, comments=null ]");
37      System.out.println("After deserialization st2 : Student[ id=CT1008, name=Yamuna, age=26, seatingPosition=0, comments=null ]");
38      {
39  }
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
```

Terminal Test cases

< Prev Reset Submit Next >

Explorer

Serializati...

```
34      System.out.println("Before serialization st1 : Student[ id=CT1007, name=Ganga, age=25, seatingPosition=71, comments=Hard-Working ]");
35      System.out.println("Before serialization st2 : Student[ id=CT1008, name=Yamuna, age=26, seatingPosition=51, comments=Absent-Minded ]");
36      System.out.println("After deserialization st1 : Student[ id=CT1007, name=Ganga, age=25, seatingPosition=0, comments=null ]");
37      System.out.println("After deserialization st2 : Student[ id=CT1008, name=Yamuna, age=26, seatingPosition=0, comments=null ]");
38  }
39  }
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
```

Terminal Test cases