Object Oriented Programming

Exception Handling and String Handling

Exception Handling

- A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code
- When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error
- That method may choose to handle the exception itself,or pass it on. Either way, at some point, the exception is caught and processed

Exception Handling

- Exceptions can be generated by the Java run-time system, or they

can be manually generated by your code

- Exceptions thrown by Java relate to fundamental errors that violate the rules of the Java language or the constraints of the Java execution environment
- Manually generated exceptions are typically used to report some error condition to the caller of a method

Exception Handling

- Java exception handling is managed via five keywords: try, catch, throw, throws, and finally
- If an exception occurs within the try block, it is thrown
- Your code can catch this exception (using catch) and handle it in some rational manner

Exception Handling

- System generated exceptions are automatically thrown by the Java run-time system
- To manually throw an exception, use the keyword throw
- Any exception that is thrown out of a method must be specified as such by a throws clause
- Any code that absolutely must be executed after a try block completes is put in a finally block

Exception Handling

- This is the general form of an exception-handling block:

```
try {
    // block of code to monitor for errors
}

catch (ExceptionType1 exOb) {
    // exception handler for ExceptionType1
}

catch (ExceptionType2 exOb) {
    // exception handler for ExceptionType2
}
// ...
finally {
    // block of code to be executed after try block ends
}
```

Exception types

- All exception types are subclasses of the built-in class Throwable
- Thus, Throwable is at the top of the exception class hierarchy
- Immediately below Throwable are two subclasses that partition exceptions into two distinct branches, exception and error.

Exception types

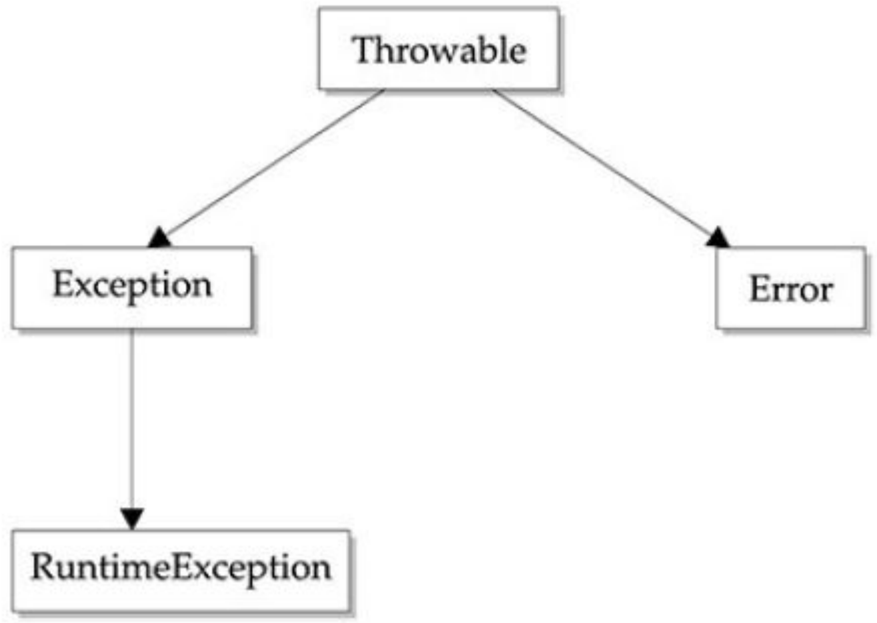
- One branch is headed by Exception
- This class is used for exceptional conditions that user programs should catch
- This is also the class that you will subclass to create your own custom exception types.

Exception types

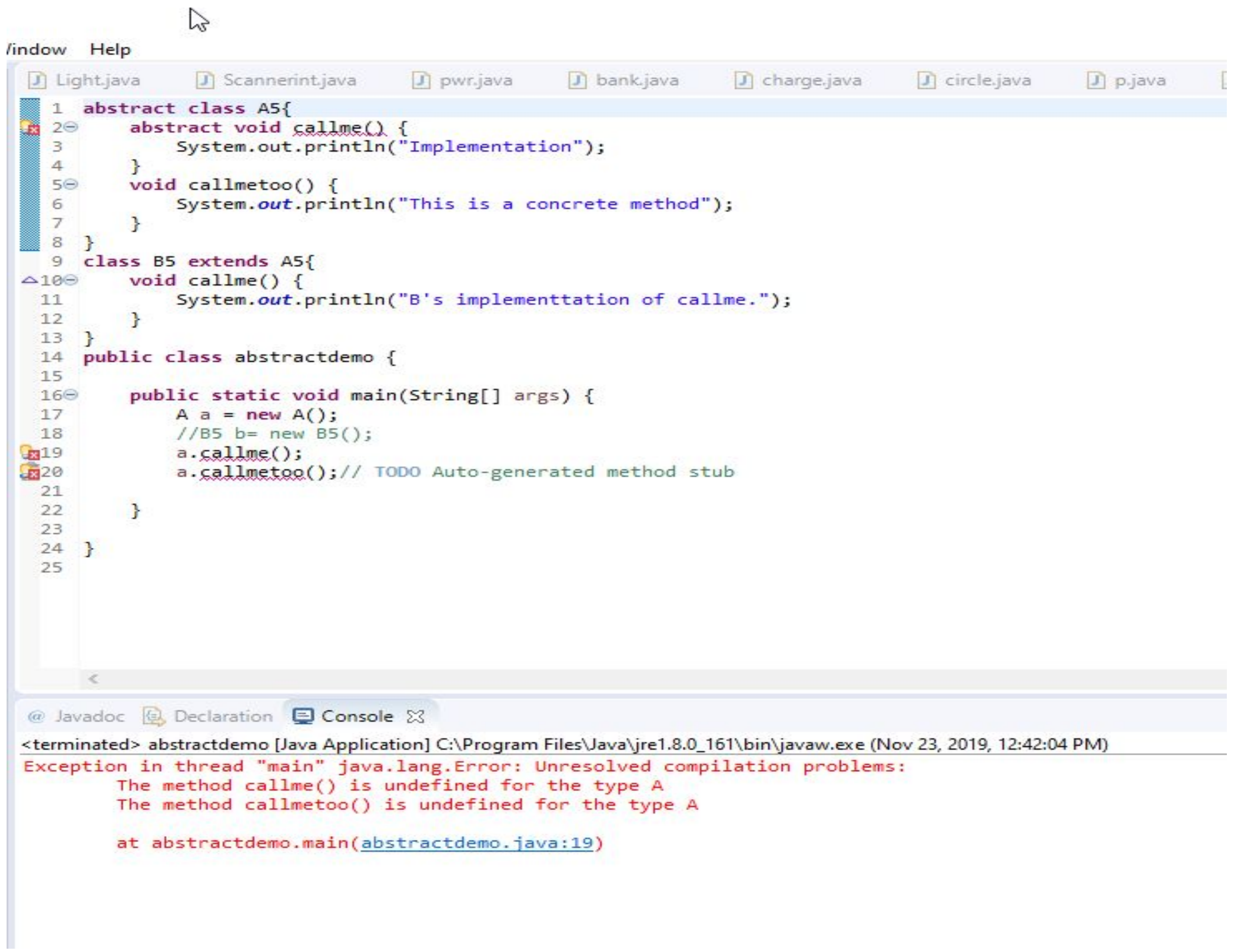
- There is an important subclass of Exception, called RuntimeException
- Exceptions of this type are automatically defined for the programs that you write and include things such as division by zero and invalid array indexing
- The other branch is topped by Error, which defines exceptions that are not expected to be caught under normal circumstances by your program and is related to java runtime environment. For example: Stack overflow

Exception types

The top-level exception hierarchy is shown here:



Example



/indow Help

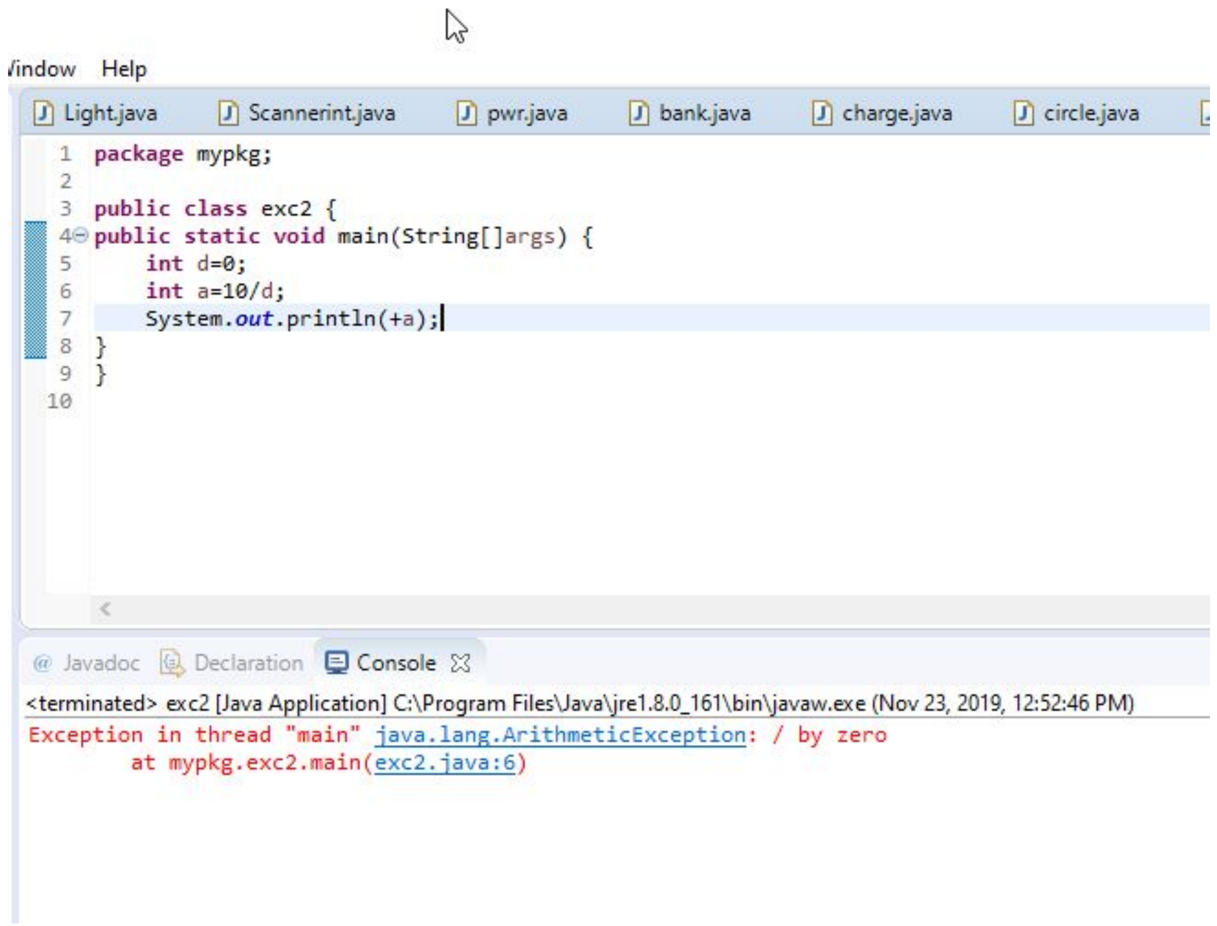
Light.java Scannerint.java pwr.java bank.java charge.java circle.java p.java

```java
abstract class A5{
    abstract void callme() {
        System.out.println("Implementation");
    }
    void callmetoo() {
        System.out.println("This is a concrete method");
    }
}
class B5 extends A5{
    void callme() {
        System.out.println("B's implementtation of callme.");
    }
}
public class abstractdemo {

    public static void main(String[] args) {
        A a = new A();
        //B5 b= new B5();
        a.callme();
        a.callmetoo();// TODO Auto-generated method stub

    }

}

```

Javadoc Declaration Console

```
<terminated> abstractdemo [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 23, 2019, 12:42:04 PM)
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
        The method callme() is undefined for the type A
        The method callmetoo() is undefined for the type A

        at abstractdemo.main(abstractdemo.java:19)
```

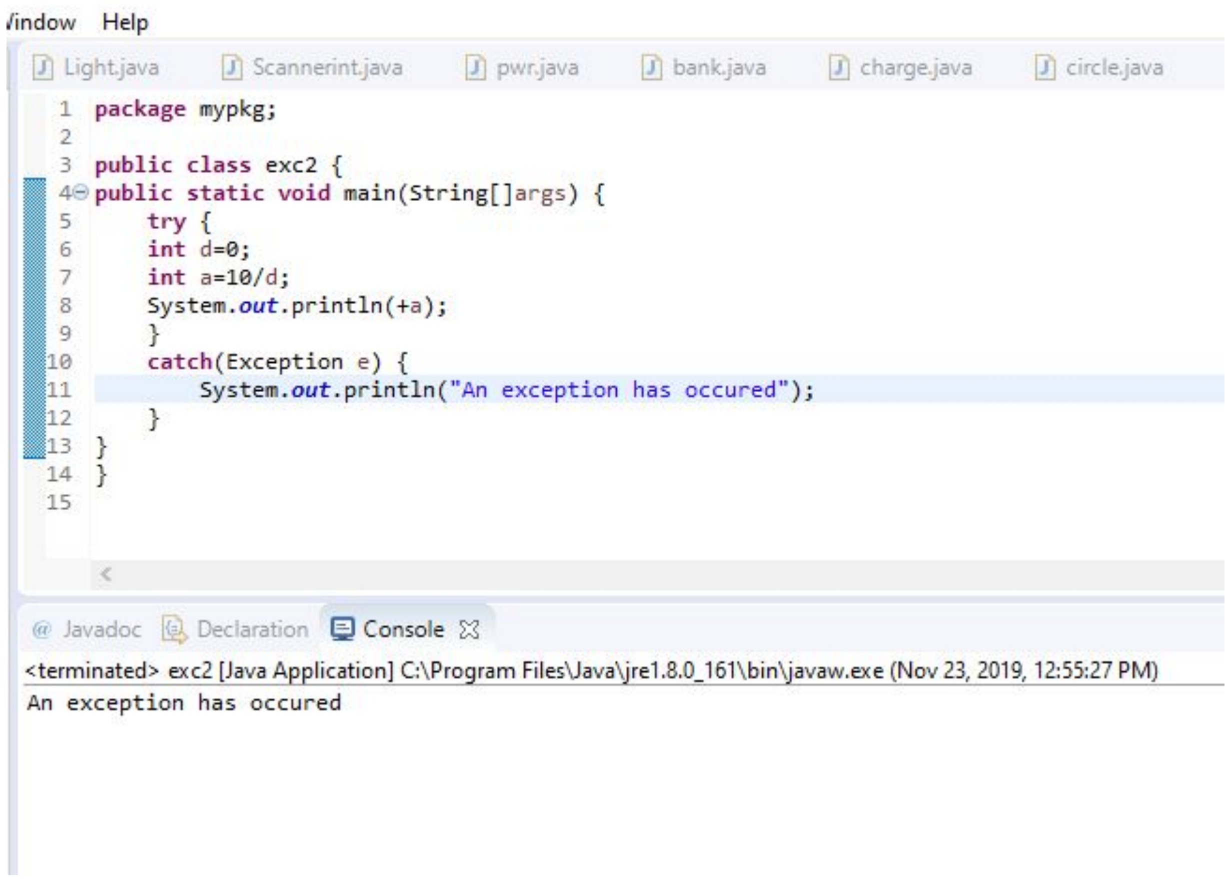
Example



```java
package mypkg;

public class exc2 {
public static void main(String[]args) {
     int d=0;
     int a=10/d;
     System.out.println(+a);
}
}

```

```
<terminated> exc2 [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 23, 2019, 12:52:46 PM)
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at mypkg.exc2.main(exc2.java:6)
```

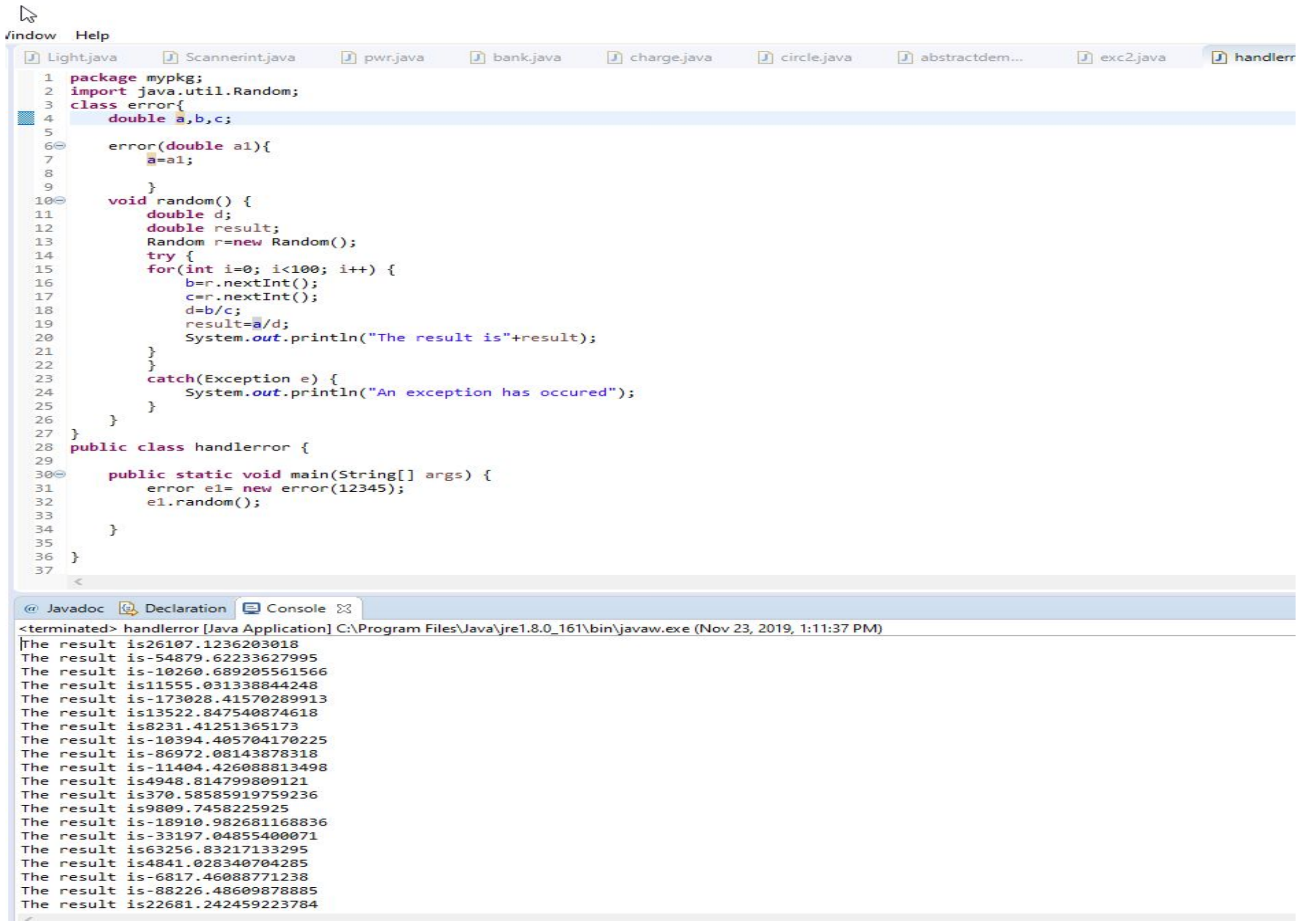
Example



```java
package mypkg;

public class exc2 {
public static void main(String[]args) {
    try {
    int d=0;
    int a=10/d;
    System.out.println(+a);
    }
    catch(Exception e) {
        System.out.println("An exception has occured");
    }
}
}
```

```
<terminated> exc2 [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 23, 2019, 12:55:27 PM)
An exception has occured
```

Example


Window Help
Light.java Scannerint.java pwr.java bank.java charge.java circle.java abstractdem... exc2.java handlerr
Javadoc Declaration Console

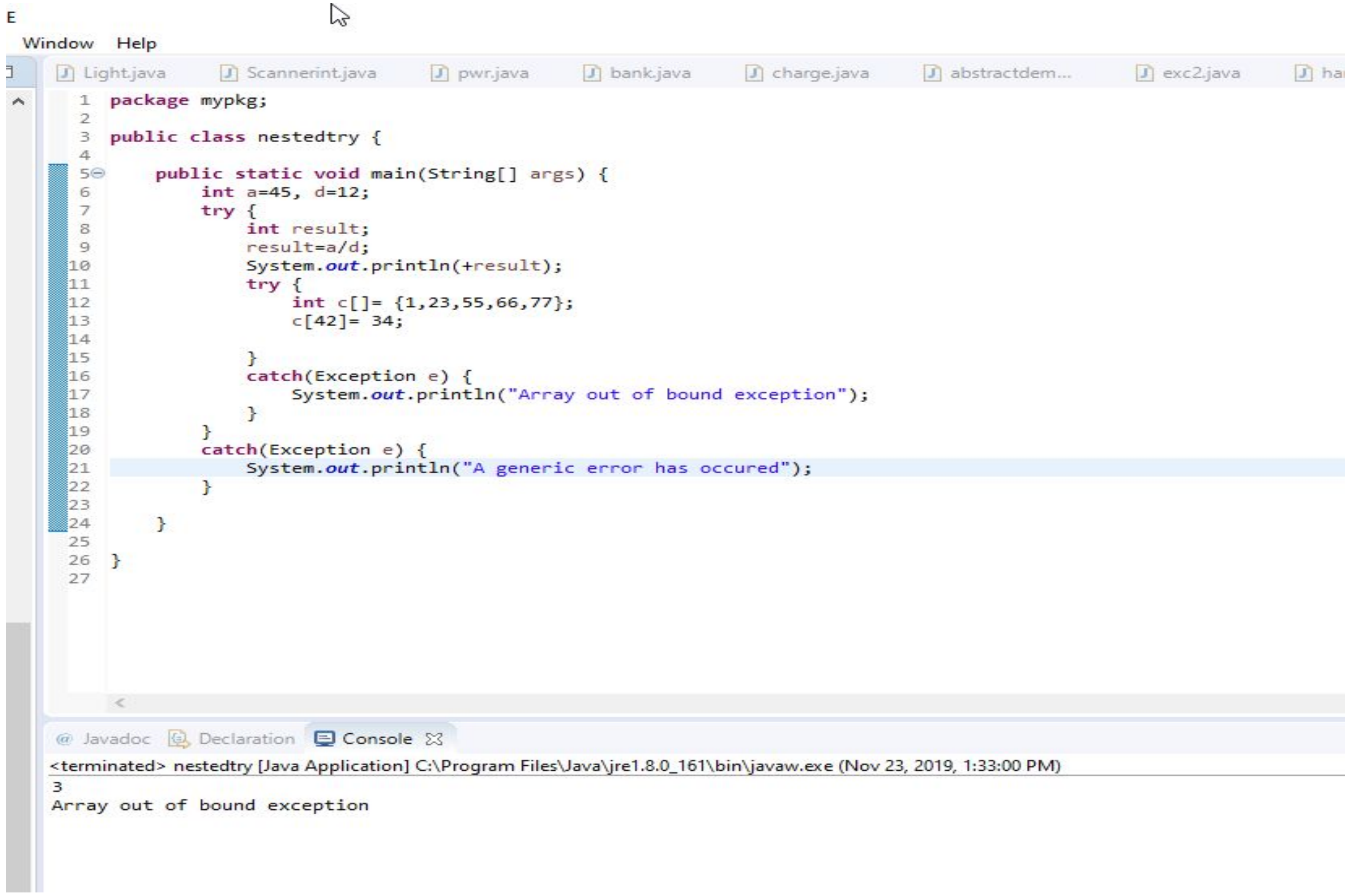
```java
package mypkg;
import java.util.Random;
class error{
    double a,b,c;

    error(double a1){
        a=a1;

        }
    void random() {
        double d;
        double result;
        Random r=new Random();
        try {
        for(int i=0; i<100; i++) {
            b=r.nextInt();
            c=r.nextInt();
            d=b/c;
            result=a/d;
            System.out.println("The result is"+result);
        }
        }
        catch(Exception e) {
            System.out.println("An exception has occured");
        }
    }
}
public class handlerror {

    public static void main(String[] args) {
        error e1= new error(12345);
        e1.random();

    }

}
```

```
<terminated> handlerror [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 23, 2019, 1:11:37 PM)
The result is26107.1236203018
The result is-54879.62233627995
The result is-10260.689205561566
The result is11555.031338844248
The result is-173028.41570289913
The result is13522.847540874618
The result is8231.41251365173
The result is-10394.405704170225
The result is-86972.08143878318
The result is-11404.426088813498
The result is4948.814799809121
The result is370.58585919759236
The result is9809.7458225925
The result is-18910.982681168836
The result is-33197.04855400071
The result is63256.83217133295
The result is4841.028340704285
The result is-6817.46088771238
The result is-88226.48609878885
The result is22681.242459223784
```

Nested try

- The try statement can be nested. That is, a try statement can be inside the block of another try
- Each time a try statement is entered, the context of that exception is pushed on the stack
- If an inner try statement does not have a catch handler for a particular exception, the stack is unwound and the next try statement’s catch handlers are inspected for a match. This continues until one of the catch statements succeeds.

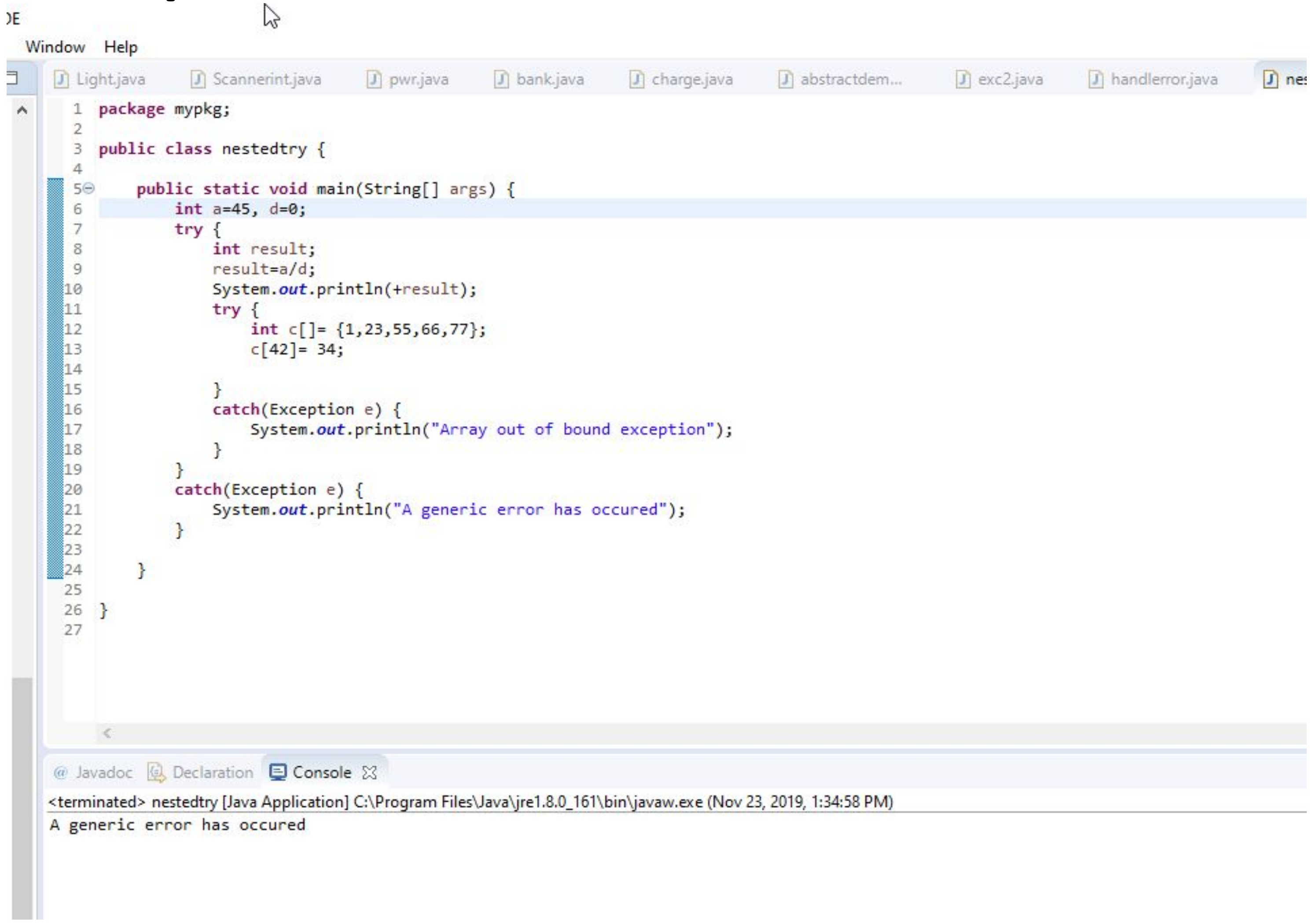
Example



```
package mypkg;

public class nestedtry {

    public static void main(String[] args) {
        int a=45, d=12;
        try {
            int result;
            result=a/d;
            System.out.println(+result);
            try {
                int c[]= {1,23,55,66,77};
                c[42]= 34;

            }
            catch(Exception e) {
                System.out.println("Array out of bound exception");
            }
        }
        catch(Exception e) {
            System.out.println("A generic error has occured");
        }

    }

}
```

```
<terminated> nestedtry [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 23, 2019, 1:33:00 PM)
3
Array out of bound exception
```

Example



```
package mypkg;

public class nestedtry {

    public static void main(String[] args) {
        int a=45, d=0;
        try {
            int result;
            result=a/d;
            System.out.println(+result);
            try {
                int c[]= {1,23,55,66,77};
                c[42]= 34;

            }
            catch(Exception e) {
                System.out.println("Array out of bound exception");
            }
        }
        catch(Exception e) {
            System.out.println("A generic error has occured");
        }

    }

}
```

```
<terminated> nestedtry [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 23, 2019, 1:34:58 PM)
A generic error has occured
```

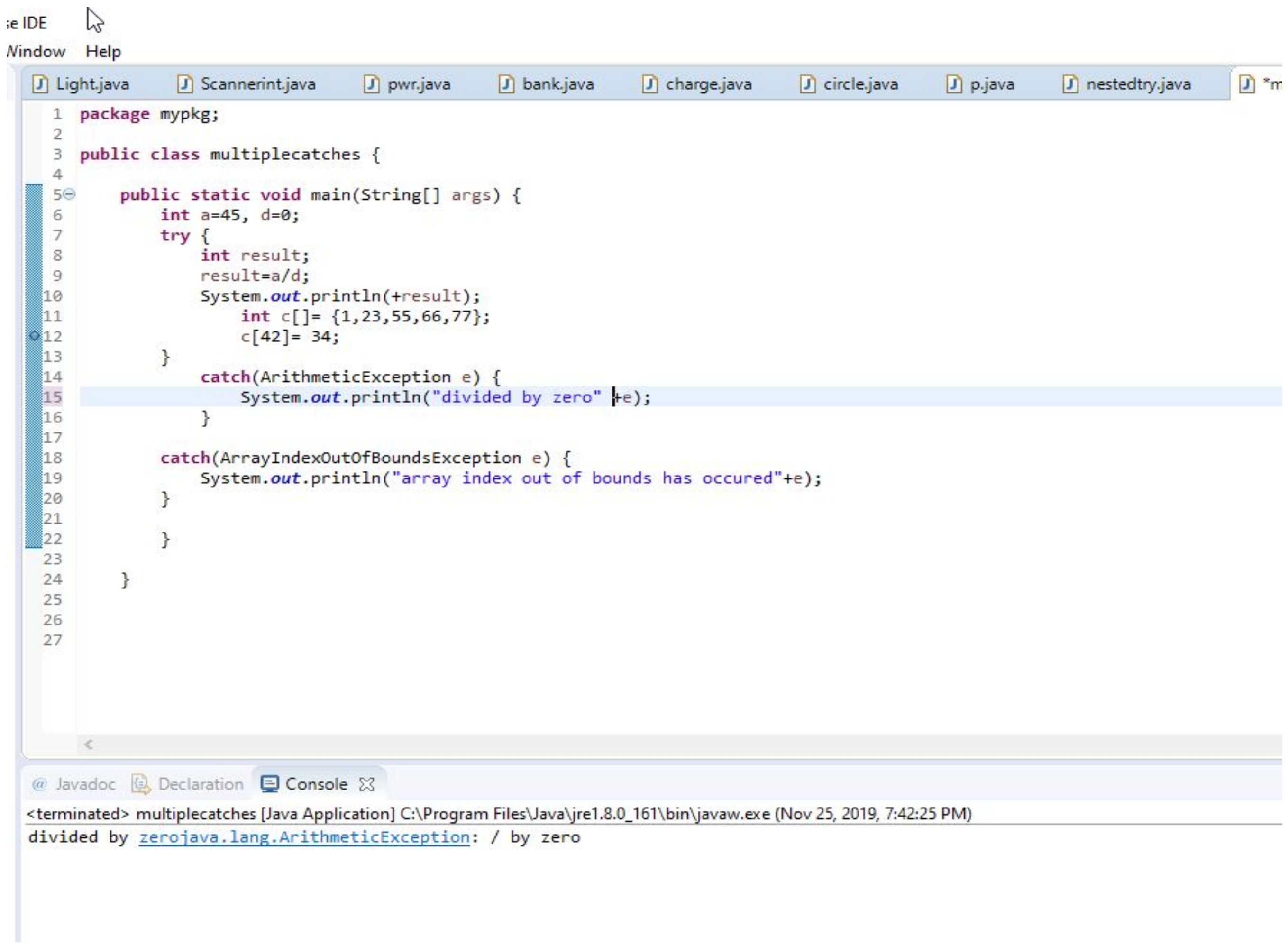
Multiple catch statement

- In some cases, more than one exception could be raised by a single piece of code.
- To handle this type of situation, you can specify two or more catch

clauses, each catching a different type of exception

- When an exception is thrown, each catch statement is inspected in order, and the first one whose type matches that of the exception is executed
- After one catch statement executes, the others are bypassed, and execution continues after the try / catch block

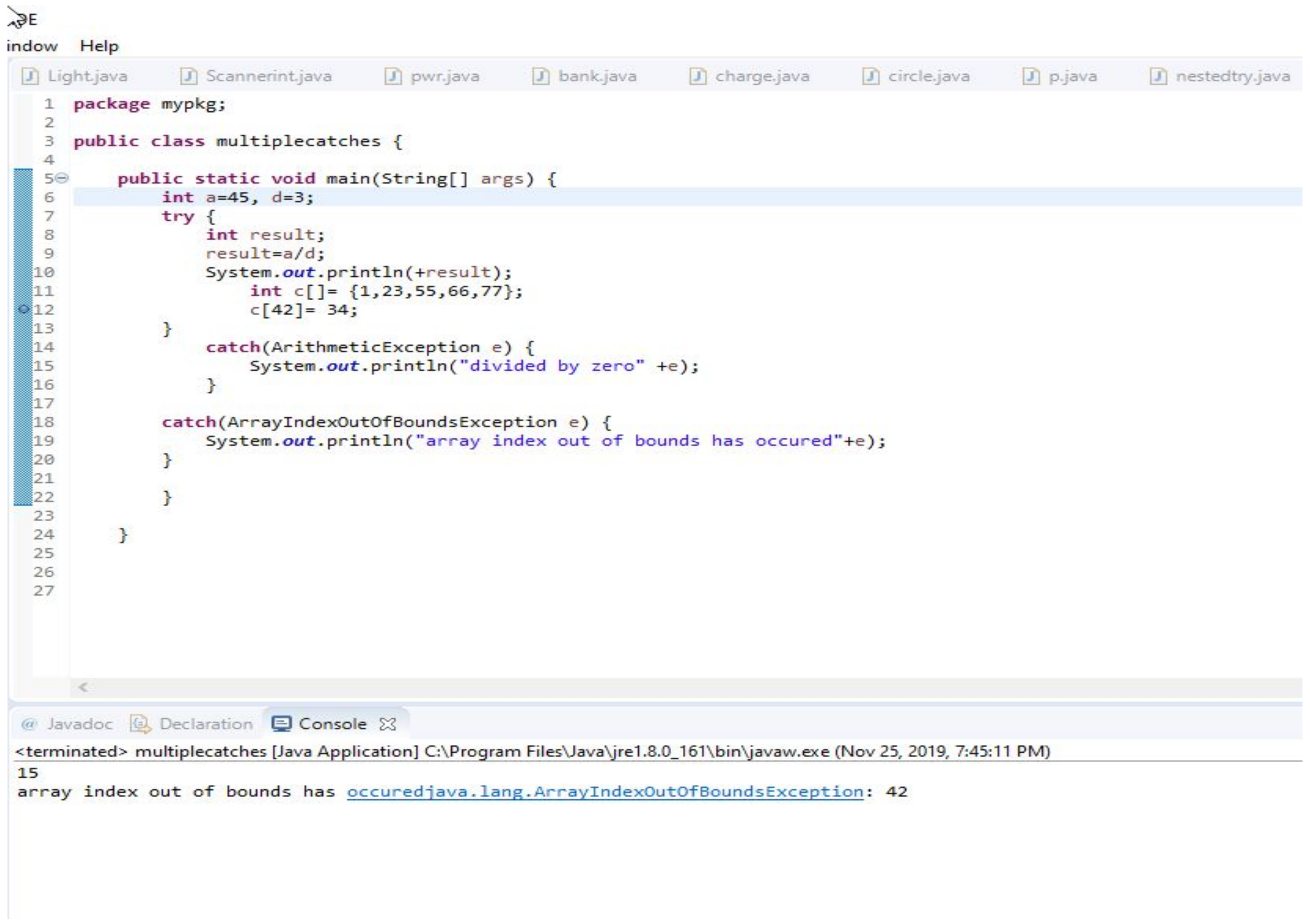
Multiple catch statement



```
package mypkg;

public class multiplecatches {

    public static void main(String[] args) {
        int a=45, d=0;
        try {
            int result;
            result=a/d;
            System.out.println(+result);
                int c[]= {1,23,55,66,77};
                c[42]= 34;
        }
            catch(ArithmeticException e) {
                System.out.println("divided by zero" +e);
            }

        catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("array index out of bounds has occured"+e);
        }

        }

    }
```

```
<terminated> multiplecatches [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 25, 2019, 7:42:25 PM)
divided by zerojava.lang.ArithmeticException: / by zero
```

Multiple catch statement



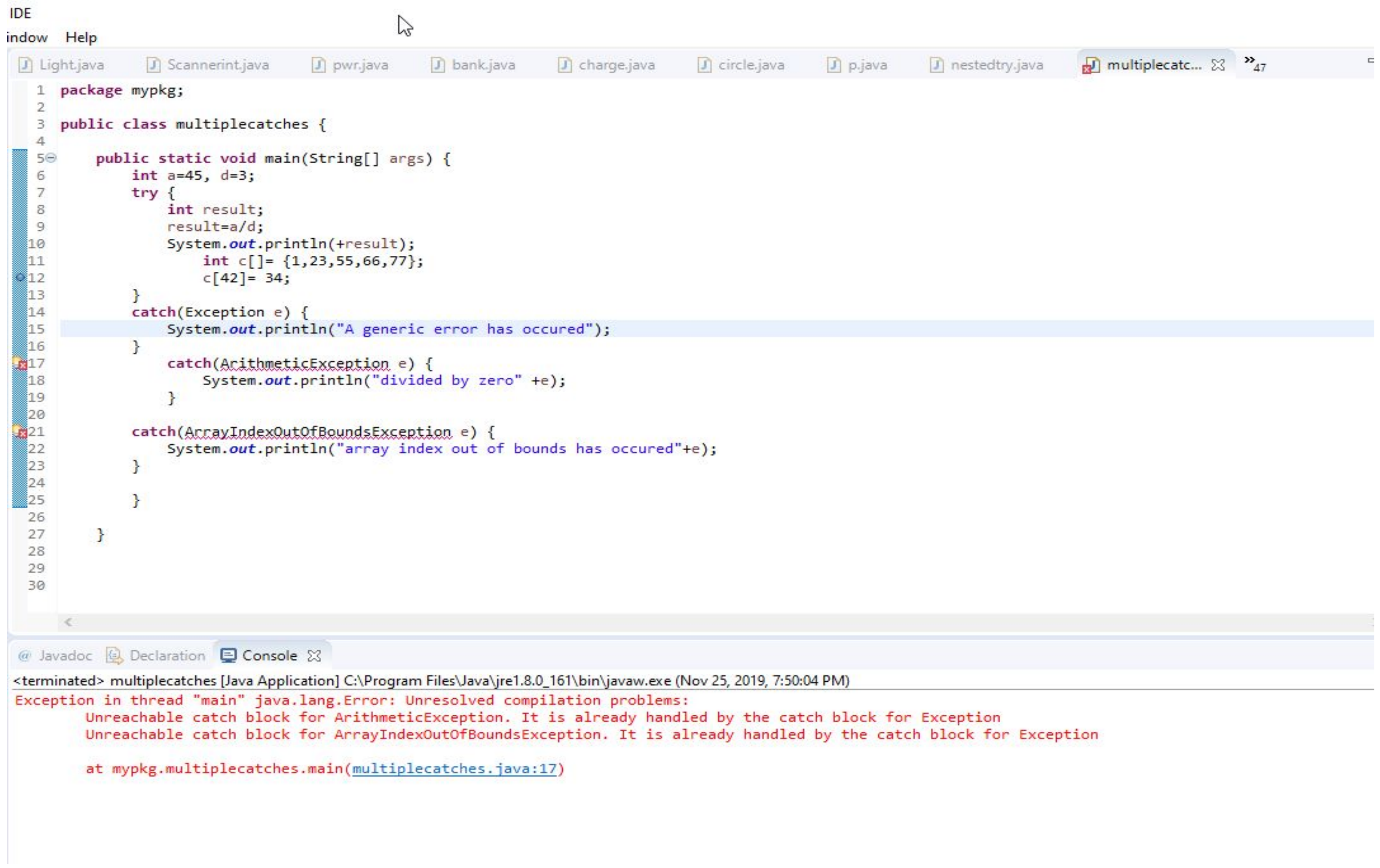
```java
package mypkg;

public class multiplecatches {

    public static void main(String[] args) {
        int a=45, d=3;
        try {
            int result;
            result=a/d;
            System.out.println(+result);
                int c[]= {1,23,55,66,77};
                c[42]= 34;
        }
            catch(ArithmeticException e) {
                System.out.println("divided by zero" +e);
            }

        catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("array index out of bounds has occured"+e);
        }

        }

    }
```

```
<terminated> multiplecatches [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 25, 2019, 7:45:11 PM)
15
array index out of bounds has occuredjava.lang.ArrayIndexOutOfBoundsException: 42
```

Multiple catch statement

- When you use multiple catch statements, it is important to remember that exception subclasses must come before any of their superclasses
- This is because a catch statement that uses a superclass will catch exceptions of that type plus any of its subclasses
- Thus, a subclass would never be reached if it came after its superclass. Further, in Java, unreachable code is an error

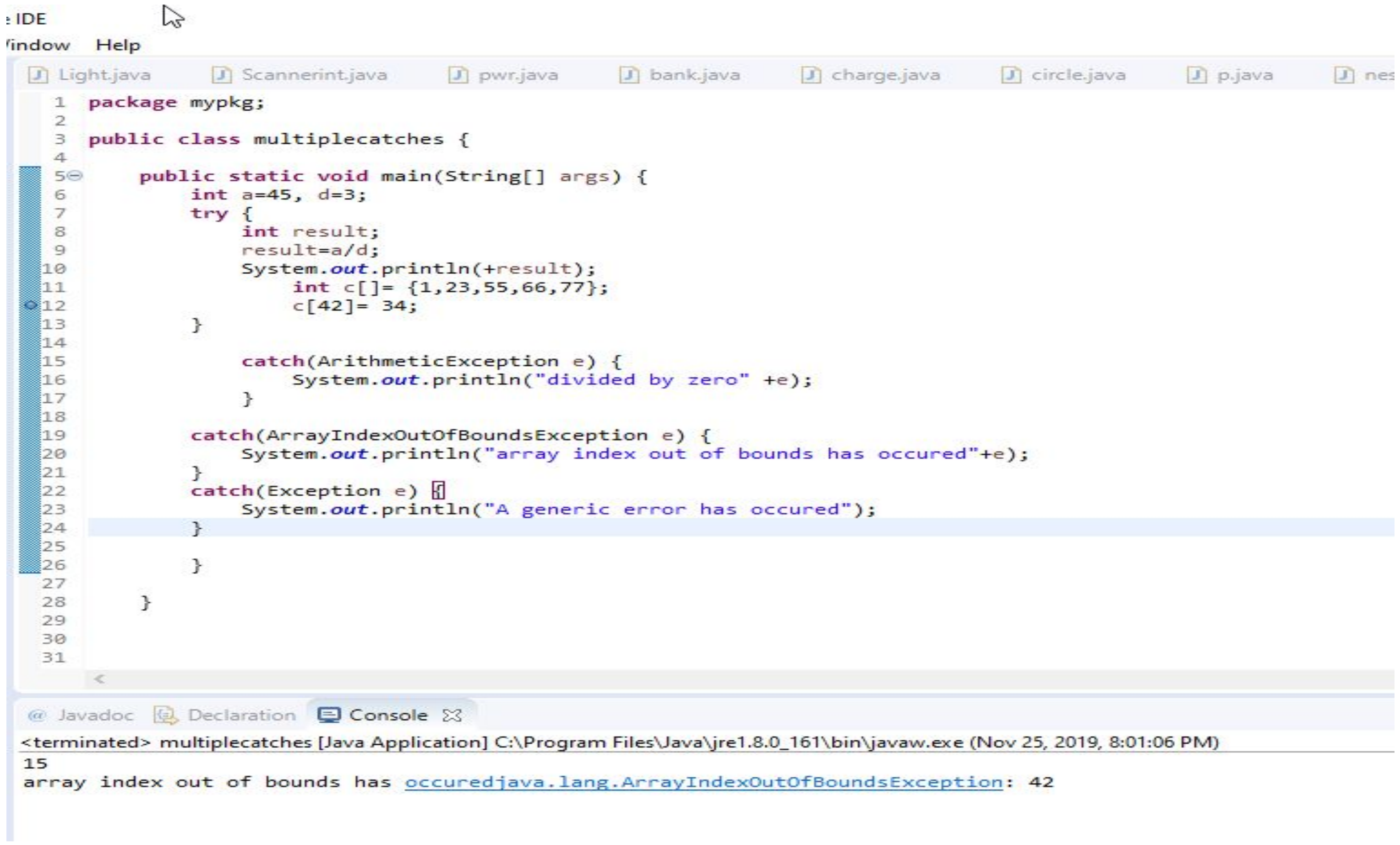
Multiple catch statement



```java
package mypkg;

public class multiplecatches {

    public static void main(String[] args) {
        int a=45, d=3;
        try {
            int result;
            result=a/d;
            System.out.println(+result);
                int c[]= {1,23,55,66,77};
                c[42]= 34;
        }
        catch(Exception e) {
            System.out.println("A generic error has occured");
        }
            catch(ArithmeticException e) {
                System.out.println("divided by zero" +e);
            }

        catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("array index out of bounds has occured"+e);
        }

        }

    }
```

```
<terminated> multiplecatches [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 25, 2019, 7:50:04 PM)
Exception in thread "main" java.lang.Error: Unresolved compilation problems: 
	Unreachable catch block for ArithmeticException. It is already handled by the catch block for Exception
	Unreachable catch block for ArrayIndexOutOfBoundsException. It is already handled by the catch block for Exception

	at mypkg.multiplecatches.main(multiplecatches.java:17)
```

Multiple catch statement



```java
package mypkg;

public class multiplecatches {

    public static void main(String[] args) {
        int a=45, d=3;
        try {
            int result;
            result=a/d;
            System.out.println(+result);
                int c[]= {1,23,55,66,77};
                c[42]= 34;
        }

            catch(ArithmeticException e) {
                System.out.println("divided by zero" +e);
            }

        catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("array index out of bounds has occured"+e);
        }
        catch(Exception e) {
            System.out.println("A generic error has occured");
        }

        }

    }
```

```
<terminated> multiplecatches [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 25, 2019, 8:01:06 PM)
15
array index out of bounds has occuredjava.lang.ArrayIndexOutOfBoundsException: 42
```

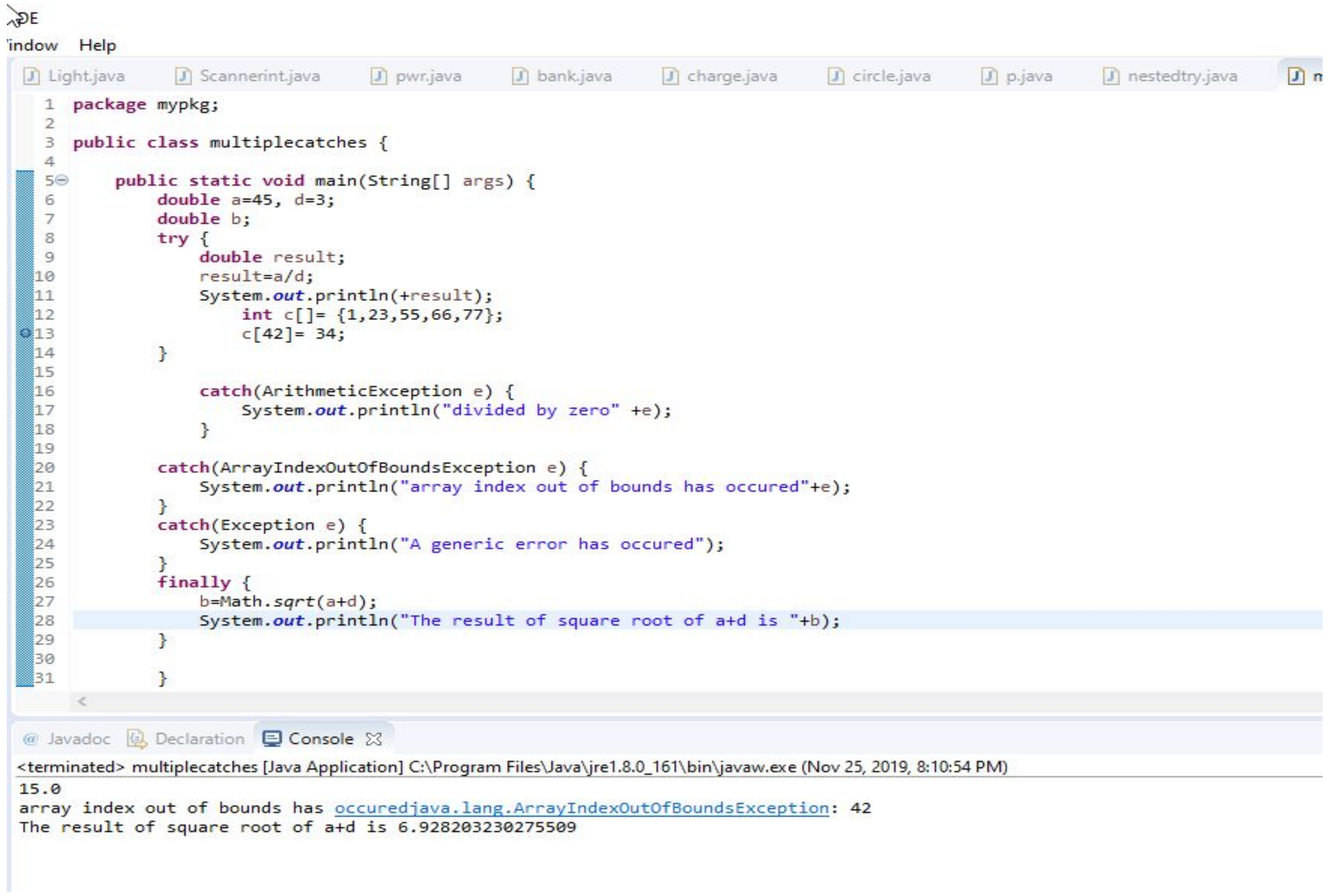
finally

- When exceptions are thrown, execution in a method takes a rather abrupt, nonlinear path that alters the normal flow through the method. Depending upon how the method is coded
- it is even possible for an exception to cause the method to return prematurely
- This could be a problem in some methods. For example, if a method opens a file upon entry and closes it upon exit, then you will not want the code that closes the file to be bypassed by the exceptionhandling mechanism. The finally keyword is designed to address this contingency

finally

- finally creates a block of code that will be executed after a try /catch block has completed and before the code following the try/catch block
- The finally block will execute whether or not an exception is thrown
- If an exception is thrown, the finally block will execute even if no catch statement matches the exception
- Each try statement requires at least one catch or a finally clause.

Example



```java
package mypkg;

public class multiplecatches {

    public static void main(String[] args) {
        double a=45, d=3;
        double b;
        try {
            double result;
            result=a/d;
            System.out.println(+result);
                int c[]= {1,23,55,66,77};
                c[42]= 34;
        }

            catch(ArithmeticException e) {
                System.out.println("divided by zero" +e);
            }

        catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("array index out of bounds has occured"+e);
        }
        catch(Exception e) {
            System.out.println("A generic error has occured");
        }
        finally {
            b=Math.sqrt(a+d);
            System.out.println("The result of square root of a+d is "+b);
        }

        }
```

```
<terminated> multiplecatches [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 25, 2019, 8:10:54 PM)
15.0
array index out of bounds has occuredjava.lang.ArrayIndexOutOfBoundsException: 42
The result of square root of a+d is 6.928203230275509
```

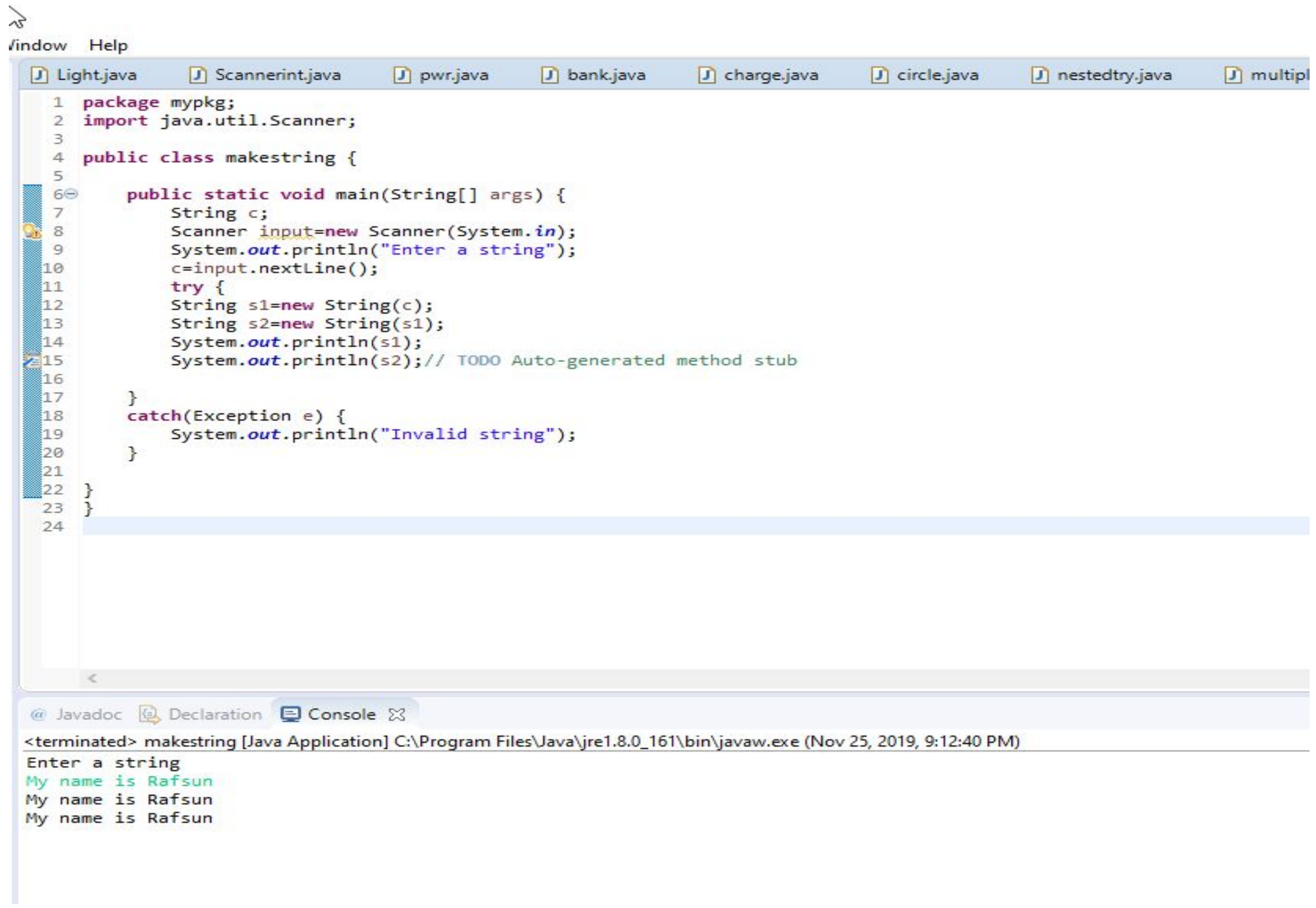
String constructors

- The String class supports several constructors. To create an empty String, call the default constructor. For example,

```
String s = new String();
```

will create an instance of String with no characters in it.

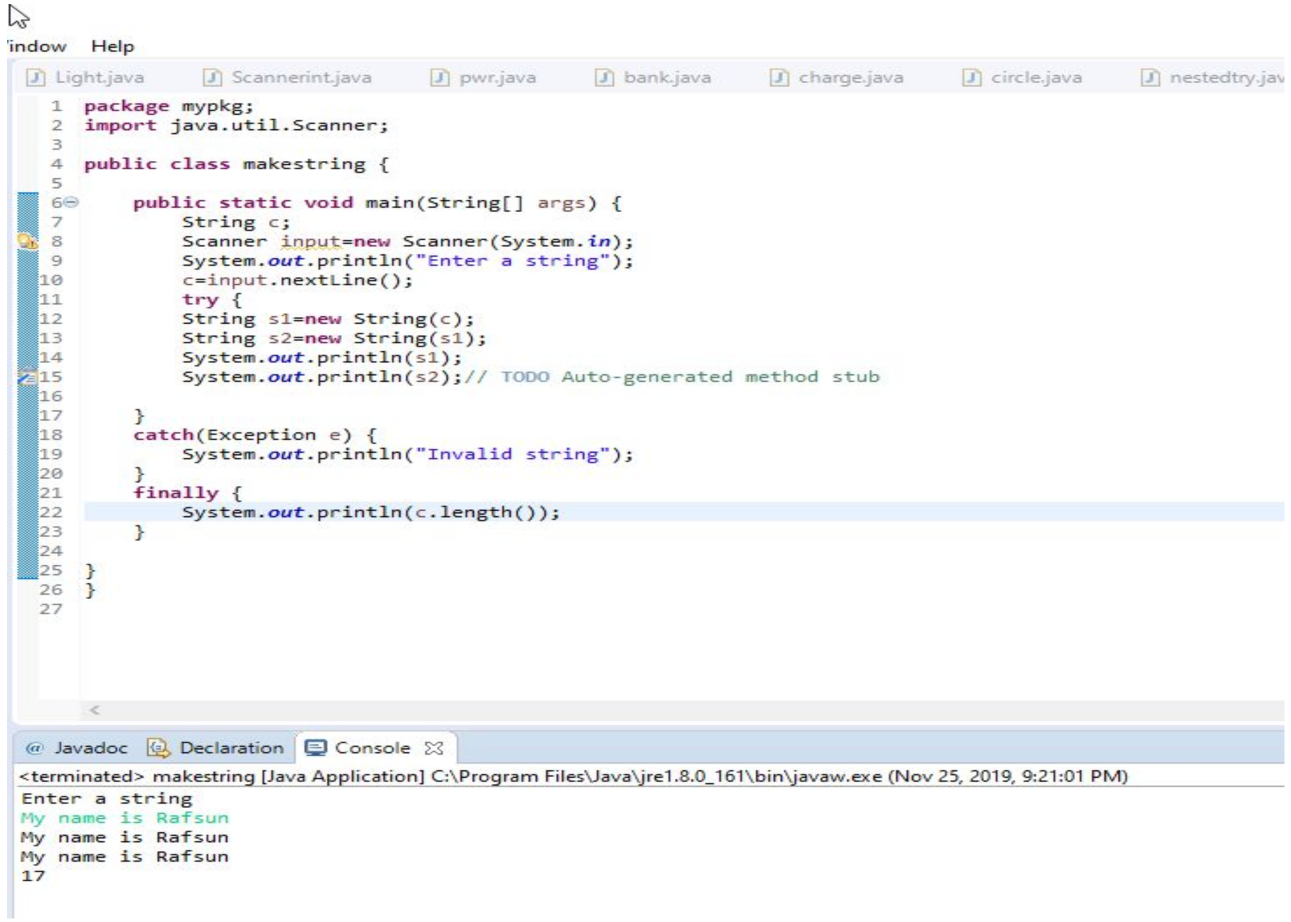
String example



```java
package mypkg;
import java.util.Scanner;

public class makestring {

    public static void main(String[] args) {
        String c;
        Scanner input=new Scanner(System.in);
        System.out.println("Enter a string");
        c=input.nextLine();
        try {
        String s1=new String(c);
        String s2=new String(s1);
        System.out.println(s1);
        System.out.println(s2);// TODO Auto-generated method stub

    }
    catch(Exception e) {
        System.out.println("Invalid string");
    }

}
}
```

```
<terminated> makestring [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 25, 2019, 9:12:40 PM)
Enter a string
My name is Rafsun
My name is Rafsun
My name is Rafsun
```

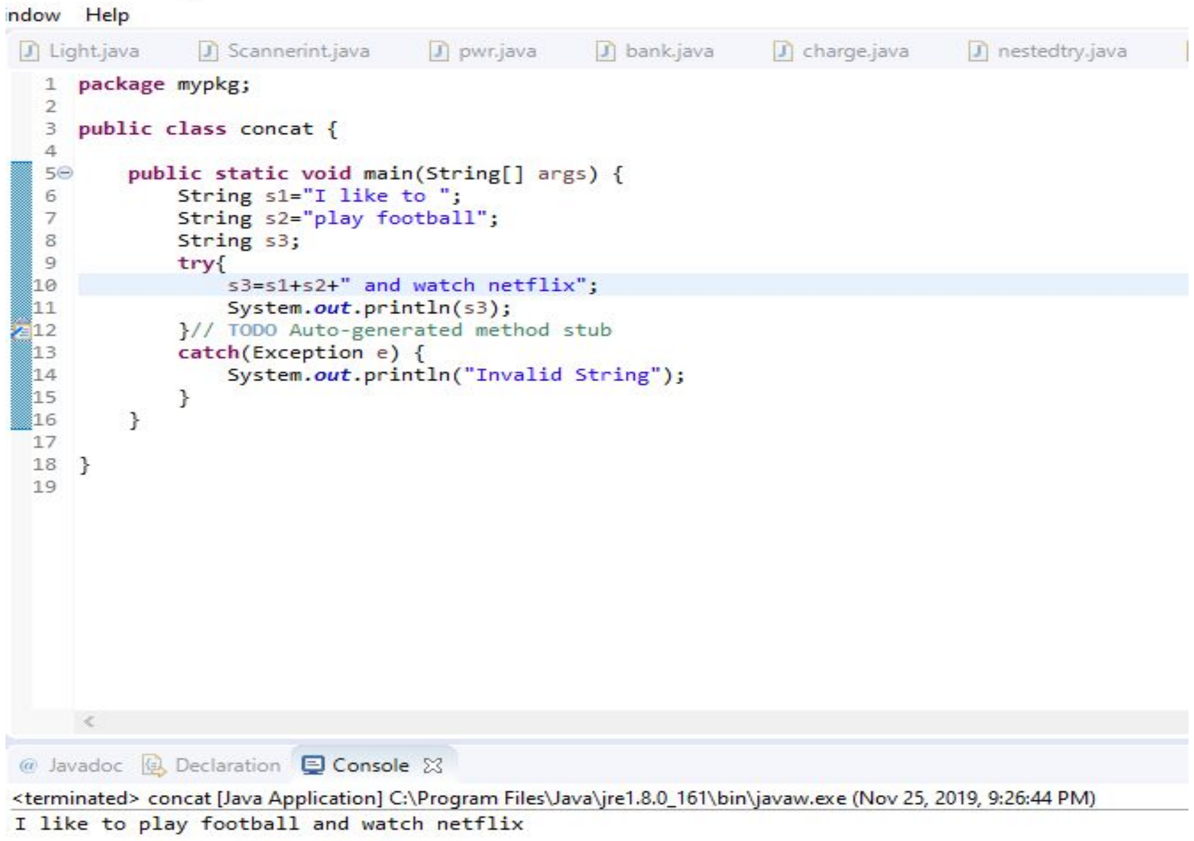
String example



```java
package mypkg;
import java.util.Scanner;

public class makestring {

    public static void main(String[] args) {
        String c;
        Scanner input=new Scanner(System.in);
        System.out.println("Enter a string");
        c=input.nextLine();
        try {
        String s1=new String(c);
        String s2=new String(s1);
        System.out.println(s1);
        System.out.println(s2);// TODO Auto-generated method stub

    }
    catch(Exception e) {
        System.out.println("Invalid string");
    }
    finally {
        System.out.println(c.length());
    }

}
}
```

```
<terminated> makestring [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 25, 2019, 9:21:01 PM)
Enter a string
My name is Rafsun
My name is Rafsun
My name is Rafsun
17
```

String example



```
package mypkg;

public class concat {

    public static void main(String[] args) {
        String s1="I like to ";
        String s2="play football";
        String s3;
        try{
            s3=s1+s2+" and watch netflix";
            System.out.println(s3);
        }// TODO Auto-generated method stub
        catch(Exception e) {
            System.out.println("Invalid String");
        }
    }

}
```

```
<terminated> concat [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 25, 2019, 9:26:44 PM)
I like to play football and watch netflix
```

getchars()

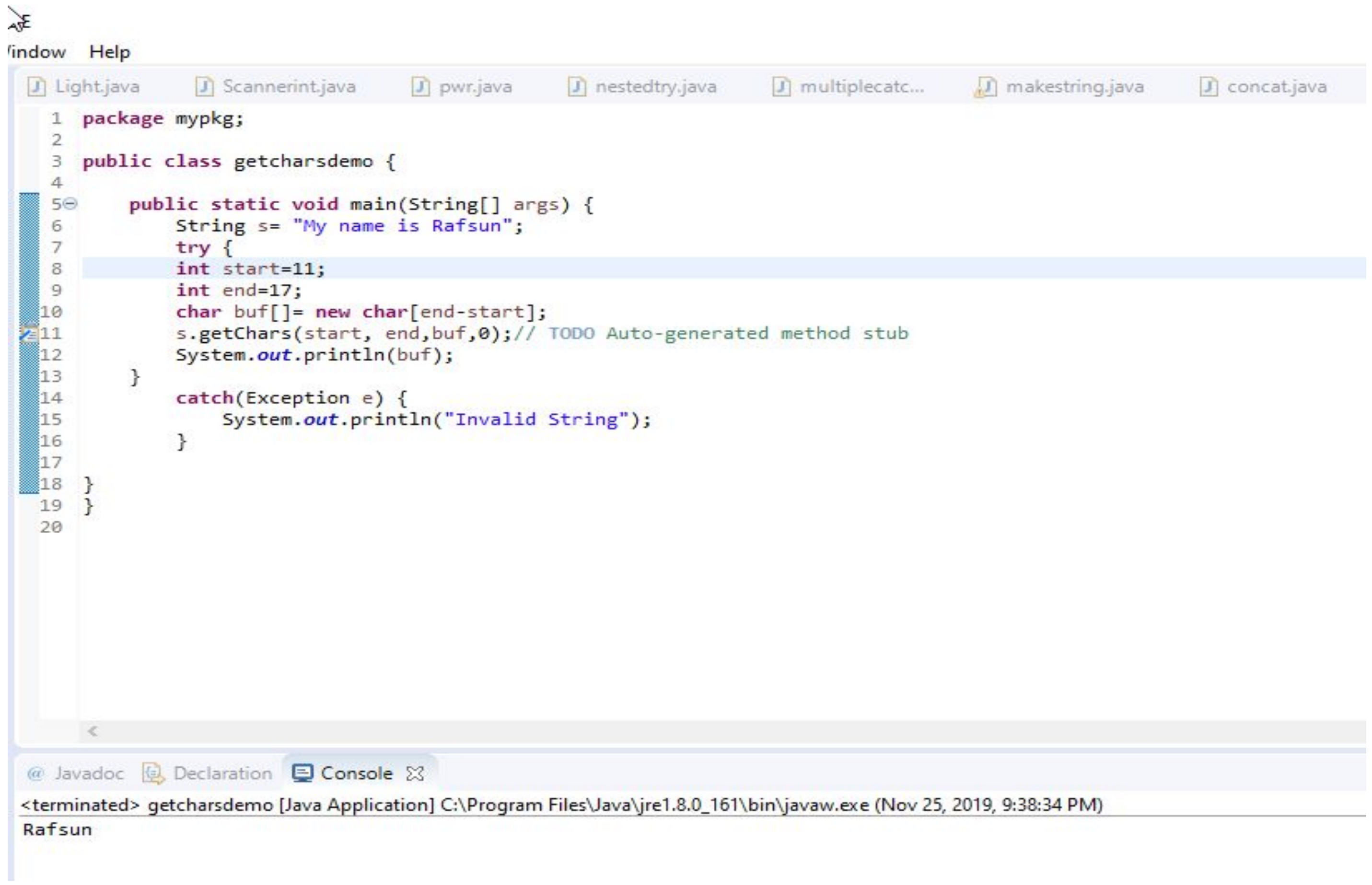
- If you need to extract more than one character at a time, you can use the

  getChars( ) method. It has this general form:

  void getChars(int sourceStart, int sourceEnd, char target[ ], int targetStart)

  Here, sourceStart specifies the index of the beginning of the substring, and sourceEnd specifies an index that is one past the end of the desired substring.

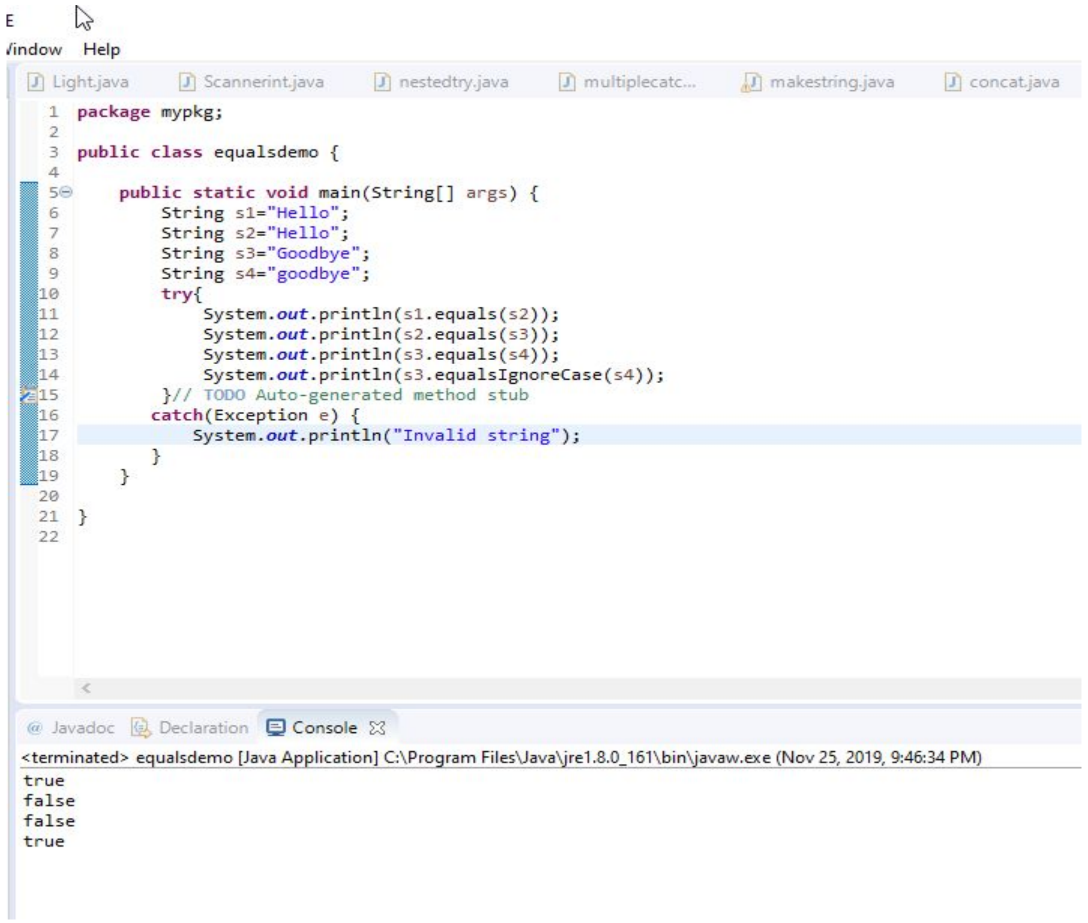
String example



```java
package mypkg;

public class getcharsdemo {

    public static void main(String[] args) {
        String s= "My name is Rafsun";
        try {
        int start=11;
        int end=17;
        char buf[]= new char[end-start];
        s.getChars(start, end,buf,0);// TODO Auto-generated method stub
        System.out.println(buf);
    }
        catch(Exception e) {
            System.out.println("Invalid String");
        }

}
}

```

```
<terminated> getcharsdemo [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 25, 2019, 9:38:34 PM)
Rafsun
```

String example



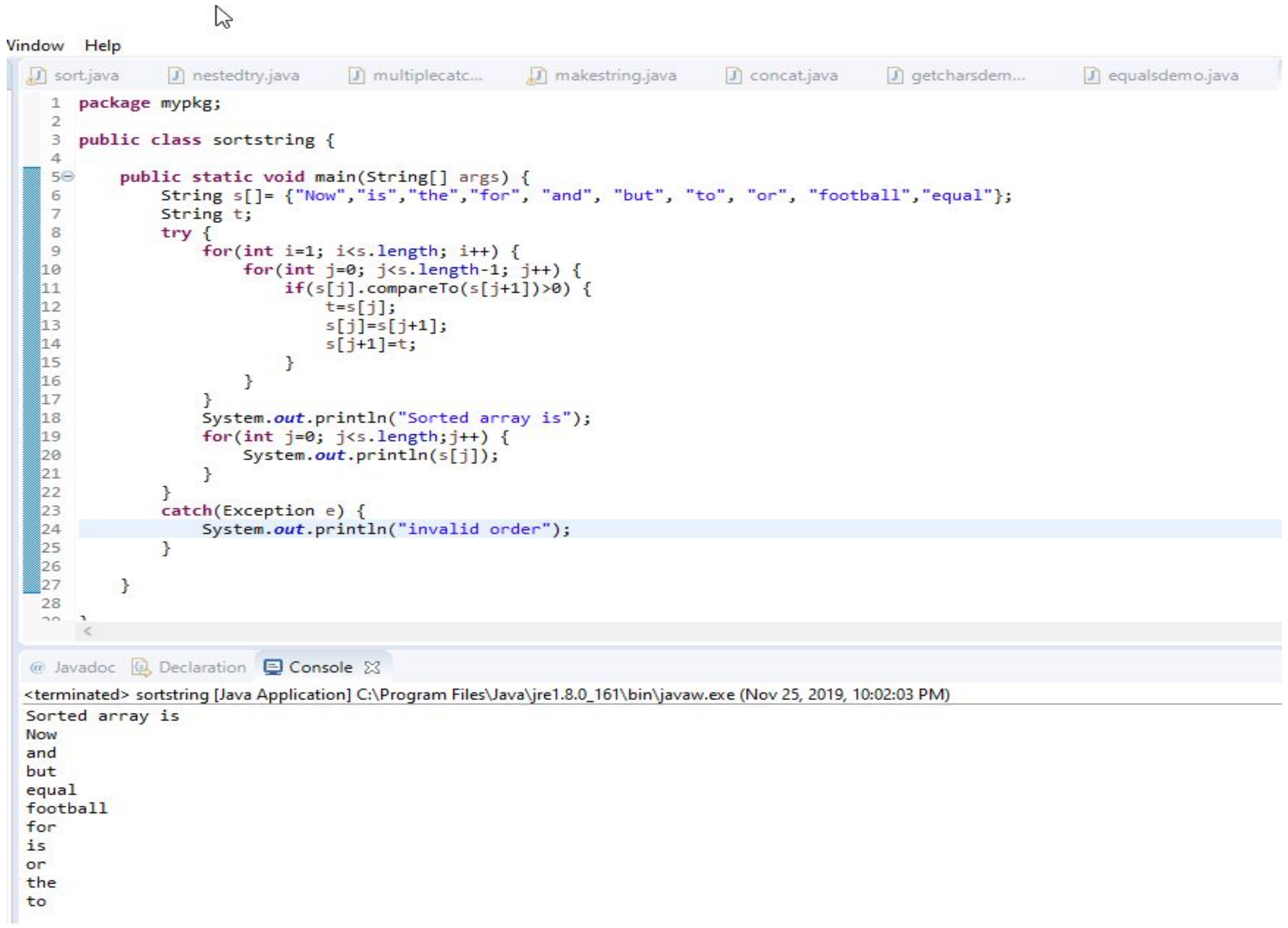
```java
package mypkg;

public class equalsdemo {

	public static void main(String[] args) {
		String s1="Hello";
		String s2="Hello";
		String s3="Goodbye";
		String s4="goodbye";
		try{
			System.out.println(s1.equals(s2));
			System.out.println(s2.equals(s3));
			System.out.println(s3.equals(s4));
			System.out.println(s3.equalsIgnoreCase(s4));
		}// TODO Auto-generated method stub
		catch(Exception e) {
			System.out.println("Invalid string");
		}
	}

}
```

```
<terminated> equalsdemo [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 25, 2019, 9:46:34 PM)
true
false
false
true
```

compareTo()

- Often, it is not enough to simply know whether two strings are identical
- For sorting applications, you need to know which is less than, equal to, or greater than the next
- The method compareTo( ) serves this purpose

Example



```java
package mypkg;

public class sortstring {

    public static void main(String[] args) {
        String s[]= {"Now","is","the","for", "and", "but", "to", "or", "football","equal"};
        String t;
        try {
            for(int i=1; i<s.length; i++) {
                for(int j=0; j<s.length-1; j++) {
                    if(s[j].compareTo(s[j+1])>0) {
                        t=s[j];
                        s[j]=s[j+1];
                        s[j+1]=t;
                    }
                }
            }
            System.out.println("Sorted array is");
            for(int j=0; j<s.length;j++) {
                System.out.println(s[j]);
            }
        }
        catch(Exception e) {
            System.out.println("invalid order");
        }

    }

}
```

```
<terminated> sortstring [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 25, 2019, 10:02:03 PM)
Sorted array is
Now
and
but
equal
football
for
is
or
the
to
```

Searching a string

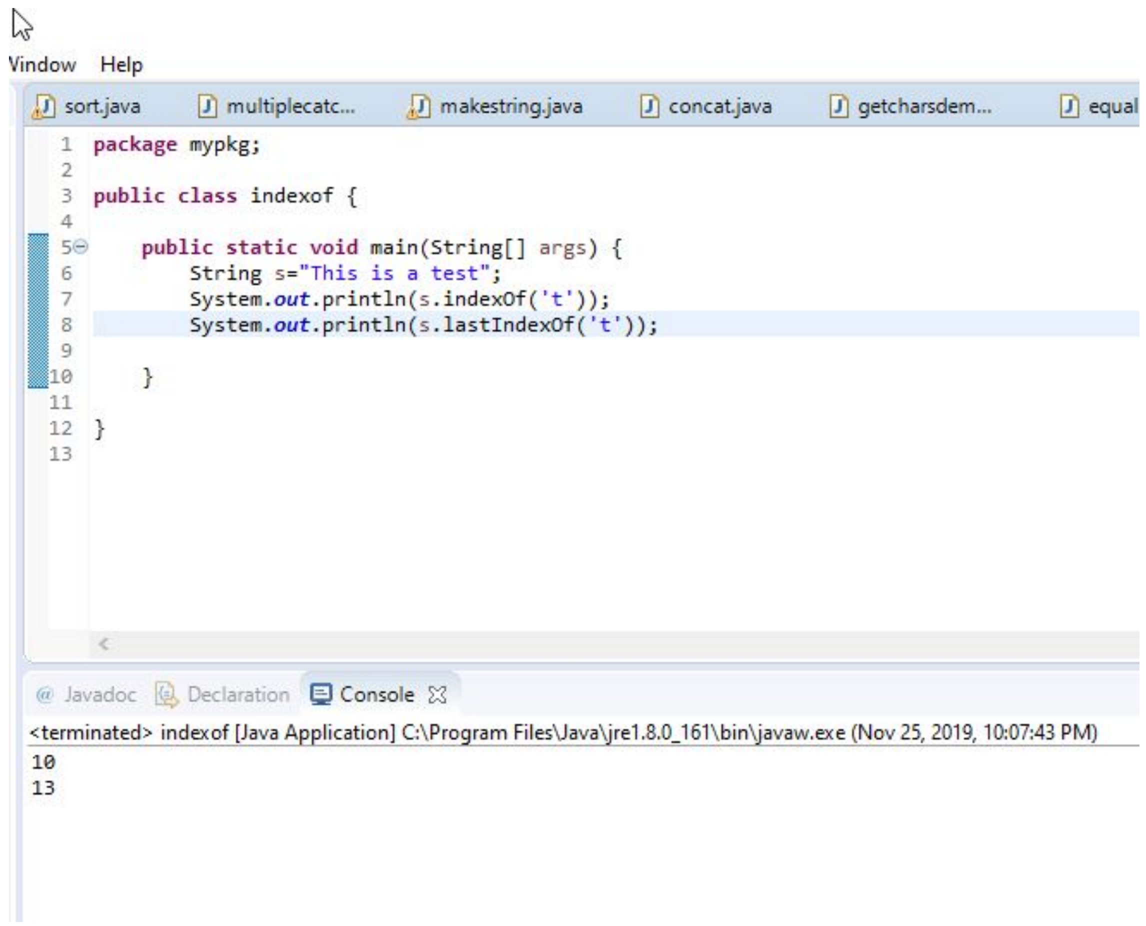
- The String class provides two methods that allow you to search a string for a specified character or substring:
  - indexOf( ) Searches for the first occurrence of a character or

substring.

  - lastIndexOf( ) Searches for the last occurrence of a character or

substring.

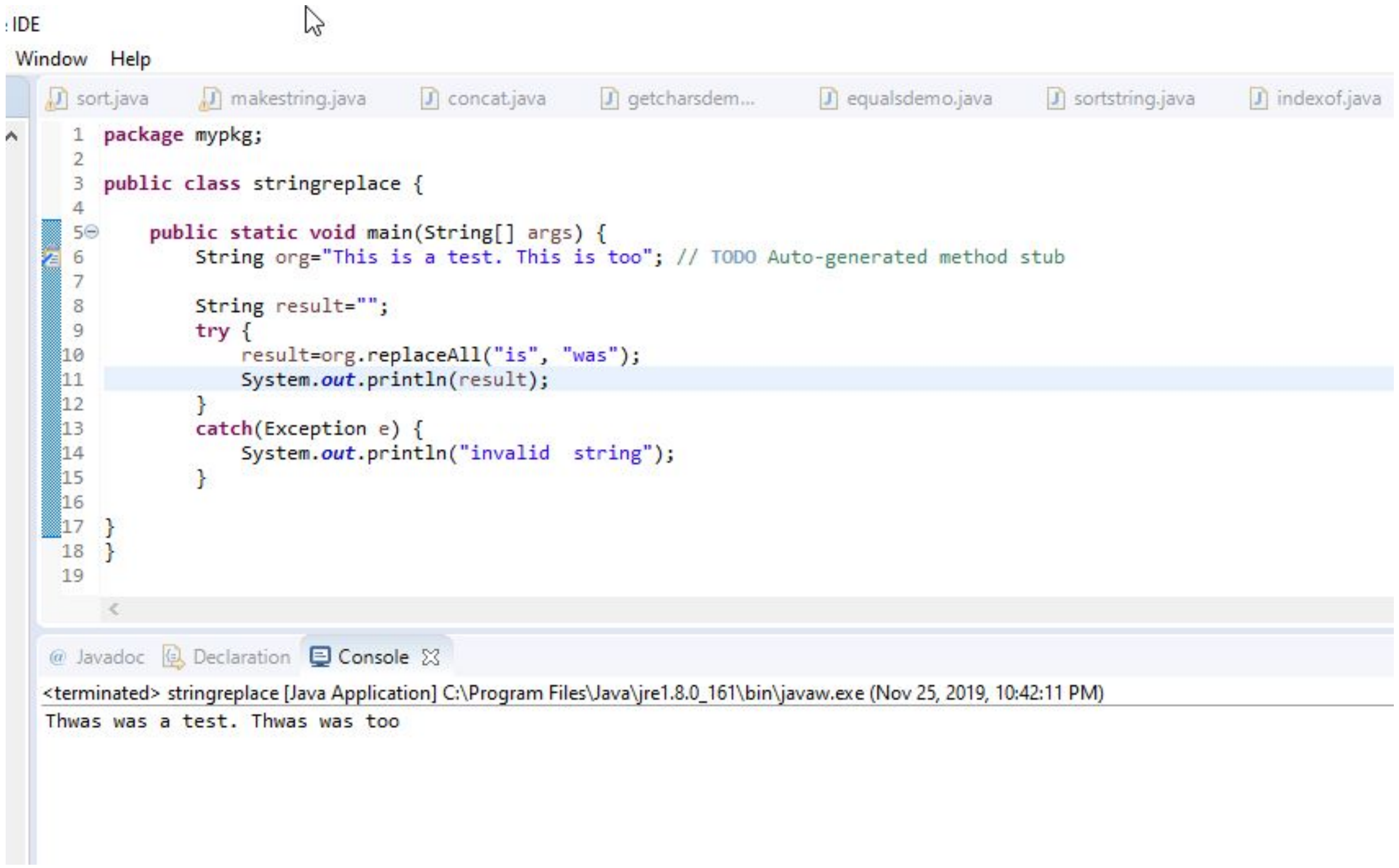
String example



```
package mypkg;

public class indexof {

    public static void main(String[] args) {
        String s="This is a test";
        System.out.println(s.indexOf('t'));
        System.out.println(s.lastIndexOf('t'));

    }

}
```

```
<terminated> indexof [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 25, 2019, 10:07:43 PM)
10
13
```

Replacing a substring



```
package mypkg;

public class stringreplace {

    public static void main(String[] args) {
        String org="This is a test. This is too"; // TODO Auto-generated method stub

        String result="";
        try {
            result=org.replaceAll("is", "was");
            System.out.println(result);
        }
        catch(Exception e) {
            System.out.println("invalid  string");
        }

}
}

```

```
<terminated> stringreplace [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 25, 2019, 10:42:11 PM)
Thwas was a test. Thwas was too
```