Exception Handling

Lecture 9

Dr. Tamer ABUHMED

Java Programming Course (SWE2023)

College of Computing

SUNG KYUN KWAN UNIVERSITY

Outline

- Introduction
- Exception handling Examples
- Application without Exception Handling (Example)
- Catch Exception Machanisim
  - Application with Exception Handling (Example)
- Uncaught exception
- When to Use Exception Handling?
- Exception Hierarchy
- Exception illustration Example
- Multi-catch in Java SE 7
- Assertions

Exception handling

- Exception—an indication of a problem that occurs during a program's execution.
  - The name "exception" implies that the problem occurs infrequently.
- With exception handling, a program can continue executing (rather than terminating) after dealing with a problem.
  - Mission-critical or business-critical computing.
  - Robust and fault-tolerant programs (i.e., programs that can deal with problems as they arise and continue executing).

Exception handling Examples

- `ArrayIndexOutOfBoundsException` occurs when an attempt is made to access an element past either end of an array.
- `ClassCastException` occurs when an attempt is made to cast an object that does not have an *is-a* relationship with the type specified in the cast operator.
- `NullPointerException` occurs when a `null` reference is used where an object is expected.
- Only classes that extend `Throwable` (package `java.lang`) directly or indirectly can be used with exception handling.

without Exception Handling

- Exceptions are thrown (i.e., the exception occurs) when a method detects a problem and is unable to handle it.
- Stack trace—information displayed at your *eclipse console* when an exception occurs and is not handled.
- Information includes:
  - The name of the exception in a descriptive message that indicates the problem that occurred
  - The method-call stack (i.e., the call chain) at the time it occurred. Represents the path of execution that led to the exception method by method.
- This information helps you debug the program.

Example: without Exception Handling

```
import java.util.Scanner;

public class DivideByZero
{
    // demonstrates throwing an exception when a
//divide-by-zero occurs
public static int quotient( int numerator, int
denominator )
    {
        return numerator / denominator;
} // end method quotient

    public static void main( String[] args )
    {
        Scanner scanner = new Scanner( System.in );
// scanner for input

        System.out.print( "Please enter an integer
numerator: " );
        int numerator = scanner.nextInt();
        System.out.print( "Please enter an integer
denominator: " );
        int denominator = scanner.nextInt();

        int result = quotient( numerator,
denominator );
        System.out.printf(
            "\nResult: %d / %d = %d\n", numerator,
denominator, result );
    } // end main
} // end class DivideByZeroNoExceptionHandling
```

Output1

```
Please enter an integer numerator: 5

Please enter an integer denominator: 0

Result: 5 / 3 = 1
```

Output2

```
Please enter an integer numerator: 5
Please enter an integer denominator: 0
Exception in thread "main"
java.lang.ArithmeticException: / by zero
at DivideByZero.quotient(DivideByZero.java:8)
at DivideByZero.main(DivideByZero.java:20)
```

Output3

```
Please enter an integer numerator: 5
Please enter an integer denominator: the
Exception in thread "main"
java.util.InputMismatchException
at java.util.Scanner.throwFor(Unknown Source)
at java.util.Scanner.next(Unknown Source)
at java.util.Scanner.nextInt(Unknown Source)
at java.util.Scanner.nextInt(Unknown Source)
at DivideByZero.main(DivideByZero.java:18)
```

Catch Exception

How to handle caught exception during the application running?

- try block encloses
  - code that might `throw` an exception
  - code that should not execute if an exception occurs.
- Consists of the keyword `try` followed by a block of code enclosed in curly braces.
- catch block (also called a catch clause or exception handler) catches and handles an exception.
  - Begins with the keyword `catch` and is followed by an exception parameter in parentheses and a block of code enclosed in curly braces.
- finally block is used for resource deallocation.
  - Placed after the `catch` block.
- At least one `catch` block or a finally block must immediately follow the `try` block.

Exceptions Methods

Method	Description
String getMessage()	Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.
synchronized Throwable getCause()	Returns the cause of the exception as represented by a Throwable object.
String toString()	Returns the name of the class concatenated with the result of getMessage()
void printStackTrace()	Prints the result of toString() along with the stack trace to System.err, the error output stream.

Example: with Exception Handling

```java
import java.util.InputMismatchException;
import java.util.Scanner;

public class DivideByZeroWithExceptionHandling
{
   // demonstrates throwing an exception when a divide-
by-zero occurs
public static int quotient( int numerator, int
denominator ) throws ArithmeticException
   {
      return numerator / denominator;
   } // end method quotient

public static void main( String[] args )
  {
   Scanner scanner = new Scanner( System.in );
// scanner for input
    boolean continueLoop = true;
// determines if more input is needed
  do
      {
         try // read two numbers and calculate quotient
         {
System.out.print( "Please enter an integer numerator: " );
int numerator = scanner.nextInt();
System.out.print( "Please enter an integer denominator: " );
```

```java
int denominator = scanner.nextInt();
int result = quotient( numerator, denominator );
System.out.printf( "\nResult: %d / %d = %d\n",
numerator, denominator, result );
continueLoop = false;
// input successful; end looping
   } // end try
catch ( InputMismatchException inputMismatchException )
  {
System.err.printf( "\nException: %s\n",
inputMismatchException );
scanner.nextLine();
// discard input so user can try again
System.out.println("You must enter integers. Please try
again.\n" );
} // end catch
catch ( ArithmeticException arithmeticException )
   {
System.err.printf( "\nException: %s\n",
arithmeticException );
System.out.println("Zero is an invalid denominator.
Please try again.\n" );
    } // end catch
  } while ( continueLoop ); // end do...while
 } // end main
} // end class DivideByZeroWithExceptionHandling
```

Example: with Exception Handling

Output

```
•  Please enter an integer numerator: 5
•  Please enter an integer denominator: 0
•  Zero is an invalid denominator. Please try again.

•  Please enter an integer numerator:
•  Exception: java.lang.ArithmeticException: / by zero
•  hello

•  Exception: java.util.InputMismatchException
•  You must enter integers. Please try again.
•  Please enter an integer numerator: 5
•  Please enter an integer denominator: 4

•  Result: 5 / 4 = 1
```

Uncaught exception

- Uncaught exception—one for which there are no matching `catch` blocks.
- Recall that previous uncaught exceptions caused the application to terminate early.
  - This does not always occur as a result of uncaught exceptions.
- Java uses a multithreaded model of program execution.
  - Each thread is a parallel activity.
  - One program can have many threads.
  - If a program has only one thread, an uncaught exception will cause the program to terminate.
  - If a program has multiple threads, an uncaught exception will terminate only the thread where the exception occurred.

Notes about Exception Handling

- When a `try` block terminates, local variables declared in the block go out of scope.
- When a `catch` block terminates, local variables declared within the `catch` block (including the exception parameter) also go out of scope.
- Any remaining `catch` blocks in the `try` statement are ignored, and execution resumes at the first line of code after the `try`…`catch` sequence.
- throws clause—specifies the exceptions a method throws.
  - Appears after the method's parameter list and before the method's body.
  - Contains a comma-separated list of the exceptions that the method will throw if various problems occur.

```
 try{
   int x =5;

   }catch(Exception e){
   int y =7;
   }
   finally {
   int z =10;
   }
 for (int x=0;i++;i++){

}
```

```
public static int quotient( int
numerator, int denominator )
throws ArithmeticException
```

When to Use Exception Handling?

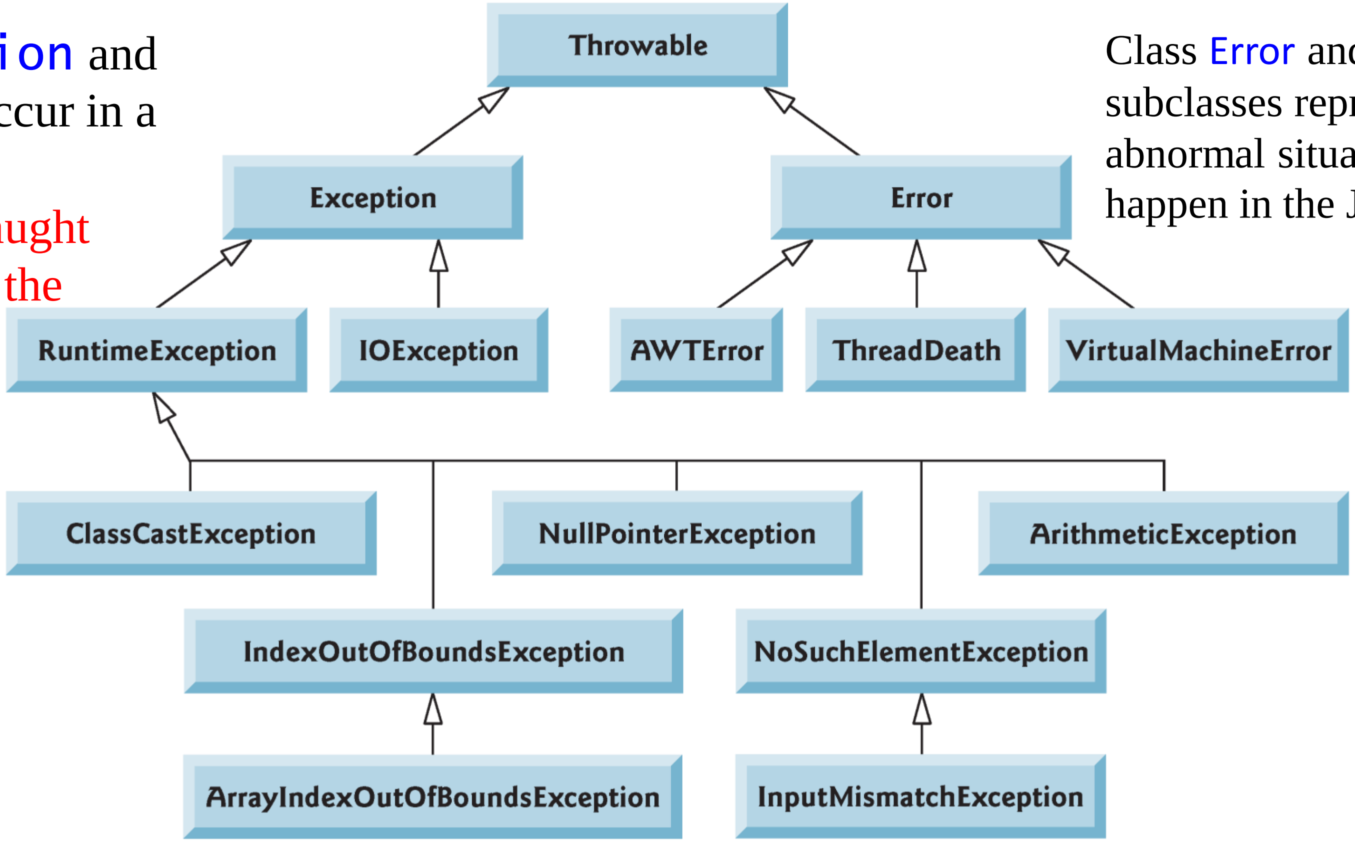
- Exception handling is designed to process synchronous errors, which occur when a statement executes.
- Common examples:
  - out-of-range array indices, arithmetic overflow, division by zero, invalid method parameters, thread interruption, unsuccessful memory allocation
- Exception handling is not designed to process problems associated with asynchronous events
  - disk I/O completions
  - network message arrivals
  - mouse clicks and keystrokes

Java Exception Hierarchy

- Exception classes inherit directly or indirectly from class Exception, forming an inheritance hierarchy.
  - Can extend this hierarchy with your own exception classes.

Class Exception and its subclasses occur in a Java program

These can be caught and handled by the application.

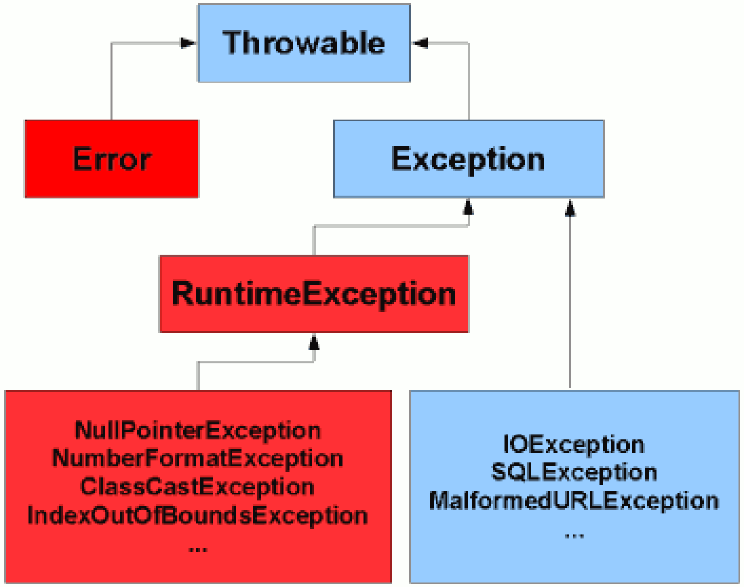


Class Error and its subclasses represent abnormal situations that happen in the JVM.

Checked vs unchecked exceptions

- Checked exceptions are subclasses of **Exception** class
- **Example of checked exceptions** are : **ClassNotFoundException**, **IOException**, **SQL Exception** and so on

- Unchecked Exceptions are subclasses of RuntimeException. Example of unchecked exceptions are : ArithmeticException, ArrayStoreException, ClassCastException and so on



Declaring you own Exception

- Keep the following points in mind when writing your own exception classes:
  - All exceptions must be a child of **Throwable.**
  - If you want to write a checked exception, extend the **Exception** class.
  - If you want to write a runtime exception, extend the **RuntimeException** class.

```
public class MyException extends Exception
{
   private double field1;
   public MyException (double input)
   {
      this.field1 = input;
   }
   public double getField1 ()
   {
      return field1;
   }
}
```

Example: Exception illustration

```
public class UsingExceptions
{
   public static void main( String[] args )
   {
try
      {
     throwException(); // call method throwException
      } // end try
   catch ( Exception exception ) // exception thrown by
throwException
      {
     System.err.println( "Exception handled in main" );
      } // end catch

      doesNotThrowException();
   } // end main


public static void throwException() throws Exception
   {
     try // throw an exception and immediately catch it
      {
   System.out.println( "Method throwException" );
     throw new Exception(); // generate exception
      } // end try
 catch ( Exception exception )
      {
System.err.println( "Exception handled in method
throwException" );
throw exception; // rethrow for further processing

// code here would not be reached; would cause
//compilation errors

   } // end catch
```

```
 finally // executes regardless of what occurs in
try...catch
  {
System.err.println("Finally executed throwException" );
      } // end finally
   } // end method throwException

   // demonstrate finally when no exception occurs
public static void doesNotThrowException()
   {
  try // try block does not throw an exception
      {
  System.out.println( "Method doesNotThrowException" );
      } // end try
  catch ( Exception exception ) // does not execute
      {
         System.err.println( exception );
      } // end catch
  finally {
System.err.println("Finally executed in
doesNotThrowException" );
      } // end finally

 System.out.println( "End of method
doesNotThrowException" );
   } // end method doesNotThrowException
} // end class UsingExceptions
```

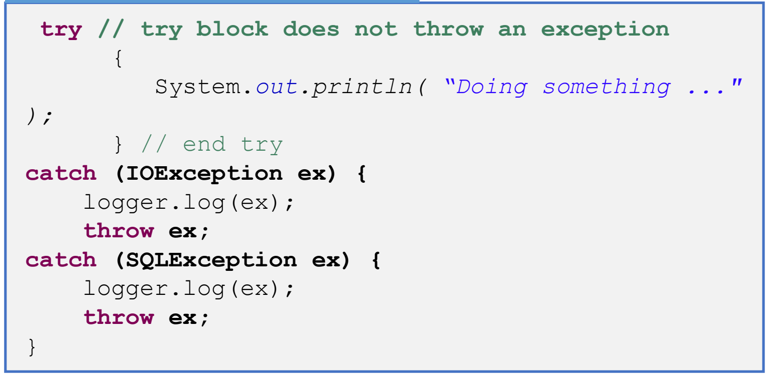
Output

```
Method throwException
Exception handled in method throwException
Finally executed in throwException
Method doesNotThrowException
Finally executed in doesNotThrowException
End of method doesNotThrowException
```

Multi-catch

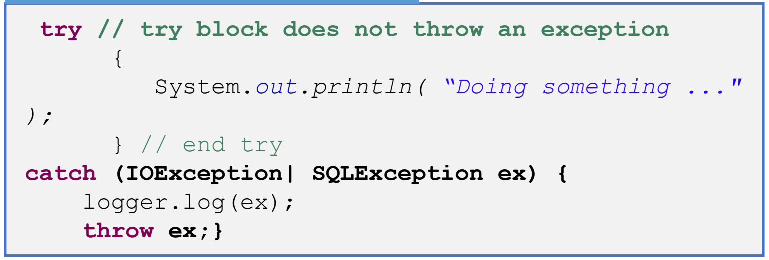
- Multi-catch: Handling Multiple Exceptions in One catch.
- If the bodies of several `catch` blocks are identical, you can use the new Java SE 7 multi-`catch` feature to catch those exception types in a single `catch` handler and perform the same task. This feature can reduce code duplication
- The syntax for a multi-`catch` header is:
  - `catch` (*Type1* | *Type2* | *Type3* e)

Before Java SE 7



```
  try // try block does not throw an exception
       {
          System.out.println( "Doing something ..."
);
       } // end try
catch (IOException ex) {
    logger.log(ex);
    throw ex;
catch (SQLException ex) {
    logger.log(ex);
    throw ex;
}
```

Java SE 7



```
  try // try block does not throw an exception
       {
          System.out.println( "Doing something ..."
);
       } // end try
catch (IOException| SQLException ex) {
    logger.log(ex);
    throw ex;}
```

Assertions

- Java includes two versions of the assert statement for validating assertions programatically.
- `assert` evaluates a `boolean` expression and, if `false`, throws an AssertionError (a subclass of `Error`).

  assert *expression*;

  - throws an `AssertionError` if *expression is* *`false`*.

  assert *expression1 : expression2;*

  - evaluates *expression1* and throws an `AssertionError` with *expression2* as the error message if expression1 is `false`.
- Can be used to programmatically implement preconditions and postconditions or to verify any other intermediate states that help you ensure your code is working correctly.

Assertions Example

```java
import java.util.Scanner;


public class AssertTest
{
 public static void main( String[] args )
   {
Scanner input = new Scanner( System.in );
System.out.print( "Enter a number between
0 and 10: " );
int number = input.nextInt();


// assert that the value is >= 0 and <= 10
assert ( number >= 0 && number <= 10 ) :
"bad number: " + number;


System.out.printf( "You entered %d\n",
number );
   } // end main
} // end class AssertTest
```

Code running commands

```
javac AssertTest.java
java -ea AssertTest
```

`-ea` **option is to** enable assertions

Output

```
Enter a number between 0 and 10: 55
Exception in thread "main"
java.lang.AssertionError: bad number: 55

at AssertTest.main(AssertTest.java:15)
```

Assertions Notes (1/2)

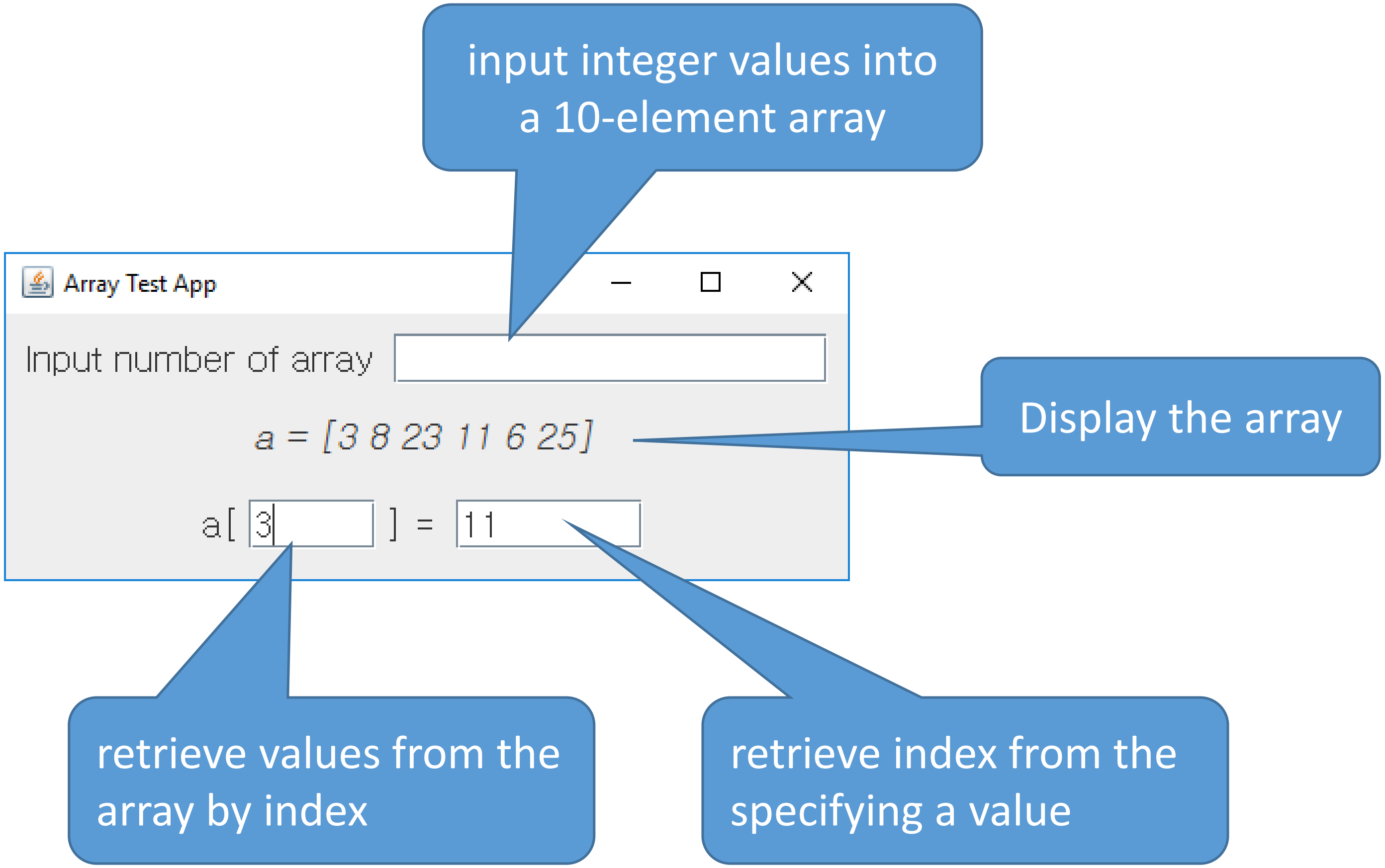
- You use assertions primarily for debugging and identifying logic errors in an application.
- You must explicitly enable assertions when executing a program
  - They reduce performance.
  - They are unnecessary for the program's user.
- To enable assertions, use the `java` command's `-ea` command-line option, as in

```
java -ea AssertTest
```

Assertions Notes (2/2)

- Users should not encounter any `AssertionError`s through normal execution of a properly written program.
  - Such errors should only indicate bugs in the implementation.
  - As a result, you should never catch an `AssertionError`.
  - Allow the program to terminate when the error occurs, so you can see the error message, then locate and fix the source of the problem.
- Since application users can choose not to enable assertions at runtime
  - You *should not* use `assert` to indicate runtime problems in production code.
  - You *should* use the exception mechanism for this purpose.

Exercise



Array Test App

Input number of array

a = [3 8 23 11 6 25]

a[3] = 11

input integer values into a 10-element array

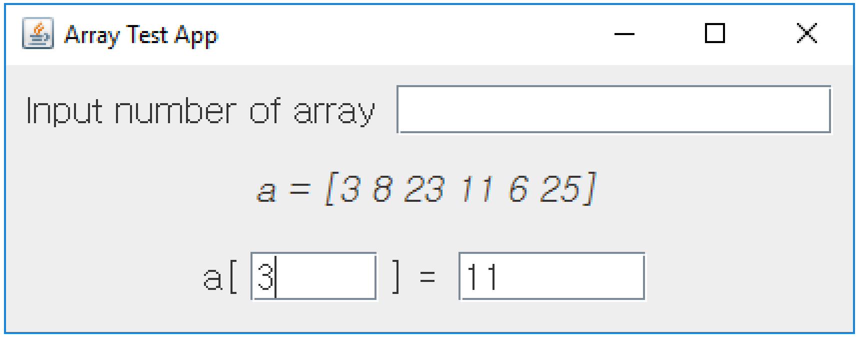
Display the array

retrieve values from the array by index

retrieve index from the specifying a value

Exercise: Design - MVC pattern

- Model:
  - int index = 0;
  - int array[] = new int[10];
- View:
  - JLabel lblArray and other 3 labels
- Controller:
  - JTextField txtInputField;
  - JTextField txtNumber;
  - JTextField txtIndex;



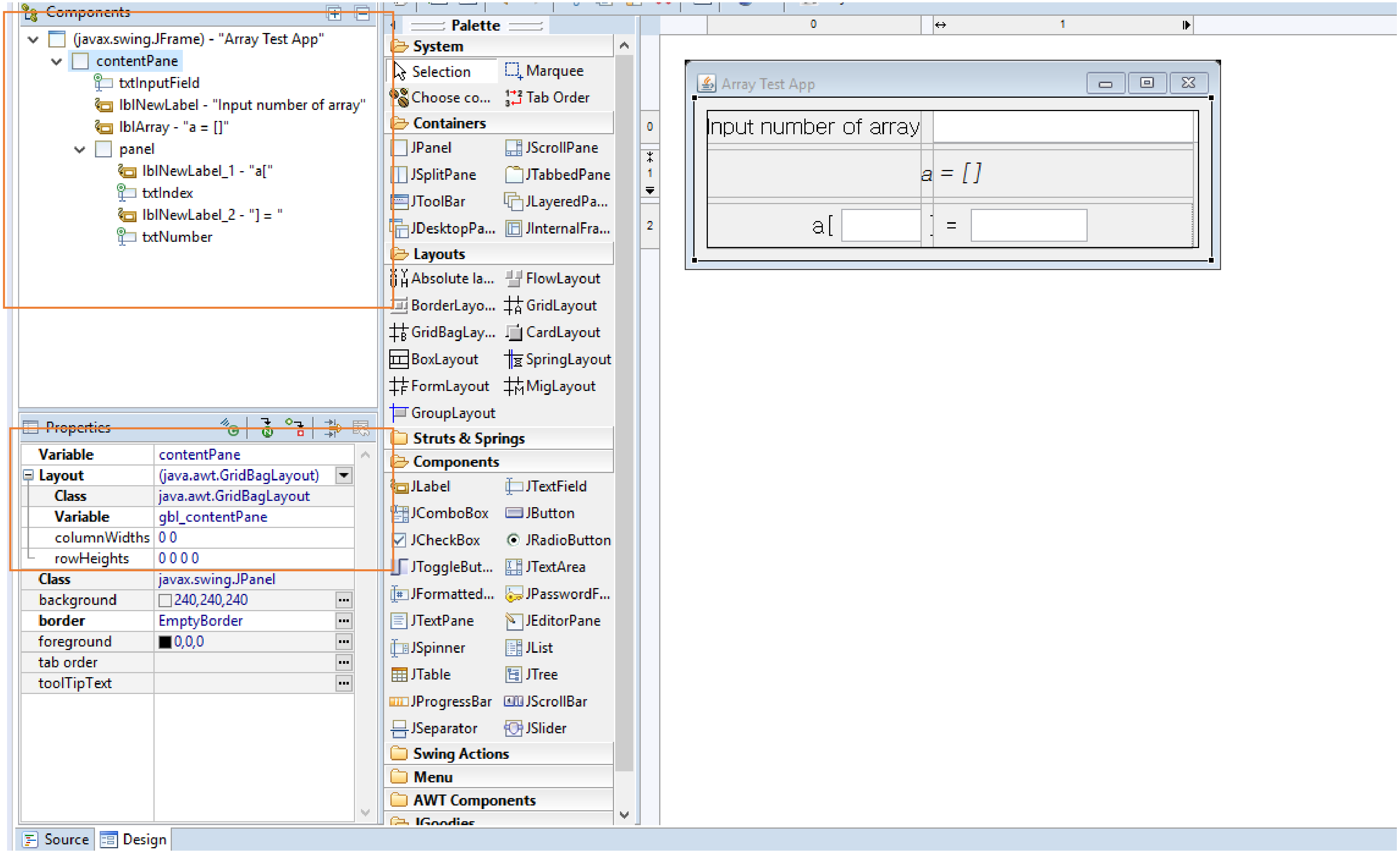
Array Test App

Input number of array

a = [3 8 23 11 6 25]

a[3] = 11

Exercise: View Implementation
Create the components



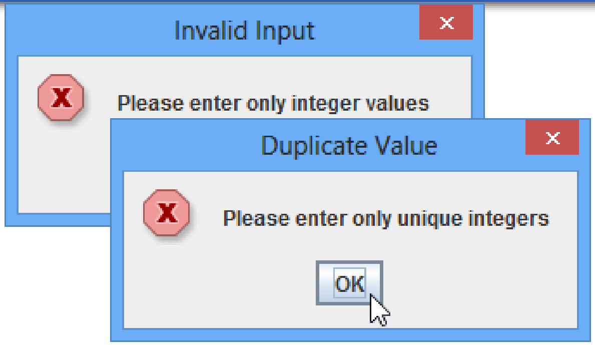
Components

- (javax.swing.JFrame) - "Array Test App"
  - contentPane
    - txtInputField
    - lblNewLabel - "Input number of array"
    - lblArray - "a = []"
    - panel
      - lblNewLabel_1 - "a["
      - txtIndex
      - lblNewLabel_2 - "] = "
      - txtNumber

Properties

Variable	contentPane
⊟ **Layout**	(java.awt.GridBagLayout)
Class	java.awt.GridBagLayout
Variable	gbl_contentPane
columnWidths	0 0
rowHeights	0 0 0 0
Class	javax.swing.JPanel
background	□ 240,240,240
border	EmptyBorder
foreground	■ 0,0,0
tab order	
toolTipText	

Exercise: Controller Implementation
Define the exceptions



Invalid Input

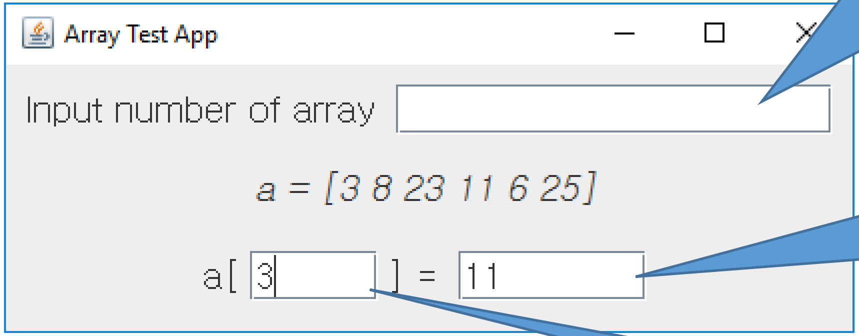
Please enter only integer values

Duplicate Value

Please enter only unique integers

Write catch handlers that catch:

- NumberFormatException
- ArrayIndexOutOfBoundsException
- DuplicateValueException (user define)



Array Test App

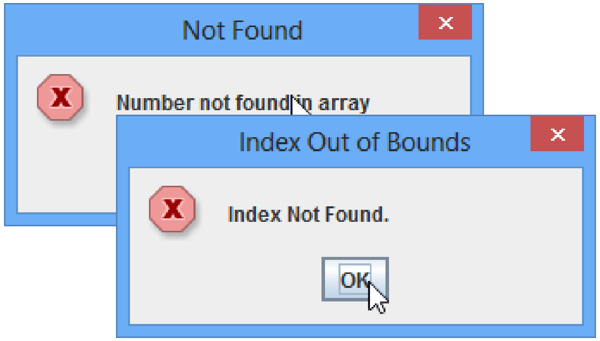
Input number of array

a = [3 8 23 11 6 25]

a[3] = 11

Write catch handlers that catch:

- NumberFormatException
- NumberNotFoundException(user define)



Not Found

Number not found in array

Index Out of Bounds

Index Not Found.

Write catch handlers that catch:

- NumberFormatException
- ArrayIndexOutOfBoundsException

Summary

- Introduction
- Exception handling Examples
- Application without Exception Handling (Example)
- Catch Exception Machanisim
  - Application with Exception Handling (Example)
- Uncaught exception
- When to Use Exception Handling?
- Exception Hierarchy
- Exception illustration Example
- Multi-catch in Java SE 7
- Assertions