Index

[Exercise 1: Exception 2](#_Toc65832091)

[Task 1 - Create MobilePhone Class 2](#_Toc65832092)

[Task 2 - Write a PhoneTest class to demonstrate exception handling 4](#_Toc65832093)

[Task 3 - Execute your program 5](#_Toc65832094)

[Exercise 2: Try-cath-finally 5](#_Toc65832095)

[Task 1 - Create TryCatchFinally Class 6](#_Toc65832096)

[Task 2 - Execute your program 6](#_Toc65832097)

[Exercise 3: Creating Your Own Exception Classes 6](#_Toc65832098)

[Task 1 - Create myException Class 6](#_Toc65832099)

[Task 2 - Create MyExceptionTest class 7](#_Toc65832100)

[Task 3 - Execute your program 7](#_Toc65832101)

[Exercise 4: Account in a Bank 8](#_Toc65832102)

[Task 1 - Create Account class 9](#_Toc65832103)

[Task 2 - Create AccountTest class 9](#_Toc65832104)

[Task 3 - Create InsufficientFundException class 10](#_Toc65832105)

[Task 4 - Create NegativeAmountException class 10](#_Toc65832106)

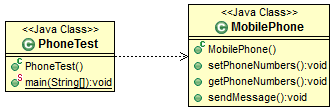
1. Exception

In this exercise, you will write a program to demonstrate exception handling

You have to create:

* MobilePhone class to store mobile phone numbers and send a message to one of numbers store in the array.
* PhoneTest class to demonstrate exception handling

Class diagram of the program:



## Create MobilePhone Class

### Import 2 libraries

* 1. **import** java.util.InputMismatchException;
  2. **import** java.util.Scanner;

### Variable to store the mobile numbers.

* 1. **int** phoneNumbers[];

### Variable to store the message text.

* 1. String message;

### Constructor

* 1. **public** MobilePhone() {
  2. message = "";
  3. }

### Method to construct an array and add numbers to it.

* 1. **public** **void** setPhoneNumbers() {
  2. Scanner input = **new** Scanner(System.*in*);
  3. **try** {
  4. System.*out*.println("Enter the number of mobile numbers to store:");
  5. **int** size = input.nextInt();
  6. phoneNumbers = **new** **int**[size];
  8. **for**(**int** index = 0;index < phoneNumbers.length;index++) {
  9. System.*out*.println("Enter a phone number:");
  10. phoneNumbers[index] = input.nextInt();
  11. }
  12. } **catch**(NegativeArraySizeException e) {
  13. System.*out*.println("Exception occurred - " +
  14. "Array size should be a positive value.");
  15. } **catch**(InputMismatchException e) {
  16. System.*out*.println("Exception occurred - Data type mismatch." +
  17. " Enter non-zero numeric value and try again.");
  18. } **catch**(Exception e) {
  19. System.*out*.println("Exception occurred - " + e.getMessage());
  20. }
  21. }

### Method to display the mobile numbers stored in the database(array).

* 1. **public** **void** getPhoneNumbers() {
  3. System.*out*.println("The mobile phone database consists of following
  4. phone numbers:");
  5. **for**(**int** index = 0;index < phoneNumbers.length;index++) {
  6. System.*out*.println(index + ". " + phoneNumbers[index]);
  7. }
  8. }

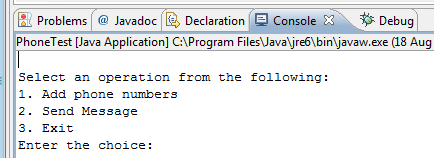
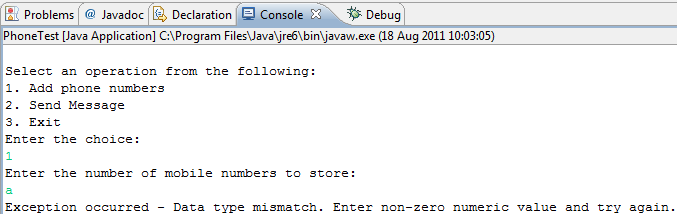
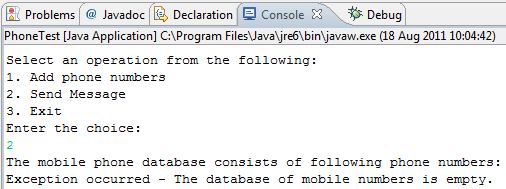
### Method to send a message to a mobile number based on the user input.

* 1. **public** **void** sendMessage() {
  2. Scanner input = **new** Scanner(System.*in*);
  3. **try** {
  4. getPhoneNumbers();
  5. System.*out*.println("Enter the index of phone number to which you
  6. want to send the message:");
  7. **int** index = input.nextInt();
  8. System.*out*.println("Enter the message text: ");
  9. // Remove enter key in buffer
  10. input.nextline();
  11. **this**.message = input.nextline();
  12. System.*out*.printf("\nSending message [%s] to [%d] ..... please
  13. wait\n", **this**.message, **this**.phoneNumbers[index]);
  14. System.*out*.printf("\nMessage successfully sent.");
  15. } **catch**(ArrayIndexOutOfBoundsException e) {
  16. System.*out*.println("Exception occurred - Invalid index.");
  17. } **catch**(InputMismatchException e) {
  18. System.*out*.println("Exception occurred - Data type mismatch.
  19. Check the data type and try again");
  20. } **catch**(NullPointerException e) {
  21. System.*out*.println("Exception occurred - The database of mobile
  22. numbers is empty.");
  23. } **catch**(Exception e) {
  24. System.*out*.println("Exception occurred - " + e.getMessage());
  25. }
  26. }
  27. }

## Write a PhoneTest class to demonstrate exception handling

* 1. **public** **static** **void** main(String[] args) {
  3. // Create an instance of MobilePhone class
  4. MobilePhone objNokia = **new** MobilePhone();
  6. Scanner input = **new** Scanner(System.*in*);
  8. // Variable to store user's choice
  9. **byte** choice ;
  11. // Iterate until the chooses to exit the application
  12. **do** {
  13. System.*out*.printf("\nSelect an operation from the following:");
  14. System.*out*.println("\n1. Add phone numbers \n2. Send Message\n3. Exit");
  16. // Accept the choice from the user
  17. System.*out*.println("Enter the choice: ");
  18. choice = input.nextByte();
  20. // Invoke the methods of MobilePhone class depending on the
  21. // operation selected by the user
  22. **if**(choice == 1) {
  23. objNokia.setPhoneNumbers();
  24. } **else** **if** (choice == 2) {
  25. objNokia.sendMessage();
  26. }
  27. } **while**(choice != 3);
  28. }

## Execute your program

* 1. 
  2. 
  3. 

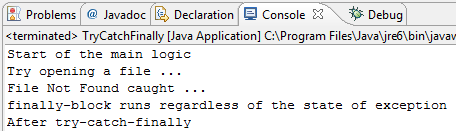
1. Try-catch-finally

In this exercise you will practice the structure try – catch-finally. Remember: If a catch-block catches that exception class or catches a superclass of that exception, the statement in that catch-block will be executed. The statements in the finally-block are then executed after that catch-block. The program continues into the next statement after the try-catch-finally, unless it is pre-maturely terminated or branch-out.

## Create TryCatchFinally Class

* 1. **import** java.util.Scanner;
  2. **import** java.io.File;
  3. **import** java.io.FileNotFoundException;
  4. **public** **class** TryCatchFinally {
  5. **public** **static** **void** main(String[] args) {
  6. **try** { // main logic
  7. System.*out*.println("Start of the main logic");
  8. System.*out*.println("Try opening a file ...");
  9. Scanner in = **new** Scanner(**new** File("test.in"));
  10. System.*out*.println("File Found, processing the file ...");
  11. System.*out*.println("End of the main logic");
  12. } **catch** (FileNotFoundException e) { // error handling separated from the main logic
  13. System.*out*.println("File Not Found caught ...");
  14. } **finally** { // always run regardless of exception status
  15. System.*out*.println("finally-block runs regardless of the state of exception");
  16. }
  17. // after the try-catch-finally
  18. System.*out*.println("After try-catch-finally");
  19. }
  20. }

## Execute your program



### Edit above code to understand how to try-catch-finally work (for example: add more type of exceptions or add a goto statement)

1. Creating Your Own Exception Classes

You can create your own Exception classes by extending from the class Exception or one of its subclasses. In this exercise, you will create your own exception named **myException** by Extending from the class Exception and create a **MyExceptionTest** to throw MyException.

## Create myException Class

* 1. **public** **class** MyException **extends** Exception{
  2. **public** MyException(String message) {
  3. **super**(message);
  4. }
  5. }

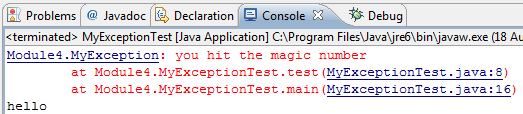
## Create MyExceptionTest class

* 1. **public** **static** **void** test(**int** number) **throws** MyException {
  2. **if** (number == 8) {
  3. **throw** (**new** MyException("you hit the magic number"));
  4. }
  5. System.*out*.println("hello"); // skip if exception triggered
  6. }

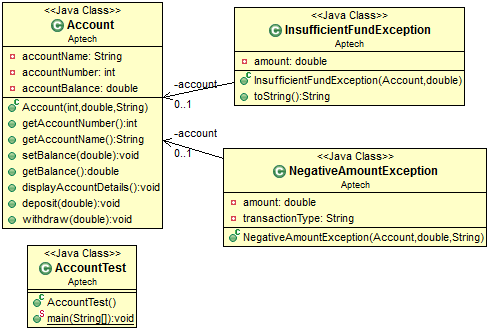
### Write code for main() method:

* 1. **public** **static** **void** main(String[] args) {
  2. **try** {
  3. *test*(9); // does not trigger exception
  4. *test*(8); // trigger exception
  5. } **catch** (MyException ex) { // exception handler
  6. ex.printStackTrace();
  7. }
  8. }

## Execute your program



1. Account in a Bank



* 1. The management of the A Bank is looking at automation as a means to save time and effort required in their work. In order to achieve this, the management has planned to computerize the following transactions:
* Creating a new account
* Withdrawing money from an account
* Depositing money in an account
  1. The CEO of the company and a team of experts have chosen your company to provide a solution for the same. Consider yourself to be a part of the team that implements the solution for designing the application.
  2. Create an application using exceptions and assertions to implement the transactions. The application should consist of the following classes.

### Account.java

Account Test.java

InsufficientFundException.java

NegativeAmountException.java

* 1. Each class has a specific purpose and functionality. The descriptions of each class are as follows.

## Create Account class

* 1. *(The Account class represents an actual bank account. It stores the following details of a bank account)*
* customerName
* accountNumber
* accountbalance
* void displayAccountDetails() : This method displays the details of the account
* void withdraw() : This method is used to withdraw money from an account. This method accepts the account number and the amount to be withdrawn from the account. The method then searches in the array of accounts for the account number. Use assertions for checking whether the account number and the amount to be withdrawn are positive. Also use an assertion to check if the array of accounts contains a minimum of one account record. The method also throws the user-defined exception InsufficientFund\*\*\*ception in case the amount to be withdrawn exceeds
* void deposit() :This method is used to deposit money in an account. The account number and the amount to be deposited in the account is accepted from the user. Use an assertion to check whether the account number is positive. The method searches for the account number and deposits the amount in the account if it exists. The displayAccountDetails() method is called if the operation succeeds. Use appropriate try catch blocks to handle all the possible exceptions that can be thrown due to the user inputs. A user-defined exception is thrown if the account number does not exist.

## Create AccountTest class

* 1. *(The AccountTest class is a java main class used to test the Account class. It creates an instance of* the Account class and displays *the following menu of options to the user)*
* Create a new account
* Withdraw Cash
* Deposit cash
* Exit
  1. The user can select any of the options and a corresponding method is invoked on the instance of the Bank class. Use an assertion to check for the control-flow invariant in case the user types an invalid option. The application exits when the Exit option is selected.

## Create InsufficientFundException class

* 1. This is a user-defined exception class derived from the base class Exception. This exception is thrown when the user tries to withdraw more money than the current account balance.

## Create NegativeAmountException class

* 1. This is a user-defined exception class derived from the base class Exception. This exception is thrown when the user tries to withdraw or deposit a negative amount.