Day-12 JA-111 Batch

Topics

- **Nested try block**
- **finally block and try with resource**
- **throw and throws**
- **User defined exception**

Nested try block

- try inside another try block
- The nested try block has its own catch block to handle the exception; if the catch block of nested try is not able to handle the exception then the catch block of outer try will handle it.

Syntax

```
try{//outer try
  ...
  try{//inner try
    ....
  }catch(Exceptiontype2 ex){
    ..
  }
  ..
}catch(Exceptiontype1 ex){
  ..
}
```

Consider a simple example of normal try-catch block which is written by developer "Alice"

An Example

```
package com.masai.sprint3;
import java.util.Scanner;

public class ExceptionDemo {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    //create an array of five elements
    int arr[] = {10, 30, 0, 53, 67};
    try {
      System.out.println("Inside try block");
      System.out.print("Enter the index of element to access ");
      int index = sc.nextInt();
      System.out.println("The element at index " + index + " is " +
arr[index]);
      System.out.println("Leaving try block");
    }catch(ArrayIndexOutOfBoundsException ex) {
      System.out.println("The index must be from 0 to " + (arr.length -
1));
```

```
        }
        System.out.println("Bye Bye");
        sc.close();
    }
}
```

Now assume a situation
In 2024, client is having a demand that he wants to take a part of array (starting index and ending index has to be provided input from user) find the addition of elements for this part of array and then find the quotient for elements of this part with addition to all elements of array

Same Input

```
Enter starting index: 0
Enter starting index: 2
```

Sample output

```
4 1 9 0 0
```

Explanation

```
Sum of the elements from index 0 to 2 is (10 + 30 + 5) = 45
45/10 -> 4
45/30 -> 1
45/5 -> 9
45/53 -> 0
45/53 -> 0
```

This time "alice" is not available and "Sam" has to do this. Sam has analyzed that array accessing is already done in the try block by Alice and she has already written the code handle ArrayIndexOutOfBoundsException so it is better to use nested try-catch block because no need to write catch block to handle the same exception again.

An Example

```
package com.masai.sprint3;
import java.util.Scanner;

public class ExceptionDemo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        //create an array of five elements
        int arr[] = {10, 30, 5, 53, 67};
        try {
            System.out.println("Inside outer try block");
            System.out.print("Enter the index of element to access ");
            int index = sc.nextInt();
            System.out.println("The element at index " + index + " is " +
arr[index]);

            try {
```

```
        System.out.println("Inside nested try block");
        System.out.print("Enter the starting index ");
        int startIndex = sc.nextInt();
        System.out.print("Enter the ending index ");
        int endIndex = sc.nextInt();

        //code to find sum
        int sum = 0;
        for(int i = startIndex; i <= endIndex; i++)
          sum = sum + arr[i];

        //code to find quotient
        for(int i = 0; i < arr.length; i++)
          System.out.println(sum/arr[i]);

        System.out.println("Leaving nested try block");
      }catch(ArithmeticException ex) {
        System.out.println("Inside inner catch block");
        System.out.println("The divisor must not be zero");
        System.out.println("Leaving inner catch block");
      }
      System.out.println("Leaving outer try block");
    }catch(ArrayIndexOutOfBoundsException ex) {
      System.out.println("Inside outer catch block");
      System.out.println("The index must be from 0 to " + (arr.length -
1));
      System.out.println("Leaving outer catch block");
    }
    System.out.println("Bye Bye");
    sc.close();
  }
}
```

Run-01

```
Inside outer try block
Enter the index of element to access 2
The element at index 2 is 5
Inside nested try block
Enter the starting index 0
Enter the ending index 2
4
1
9
0
0
Leaving nested try block
Leaving outer try block
Bye Bye
```

Run-02
```
Inside outer try block
Enter the index of element to access 2
The element at index 2 is 5
Inside nested try block
Enter the starting index 0
Enter the ending index 5
Inside outer catch block
The index must be from 0 to 4
Leaving outer catch block
Bye Bye
```

Run-03
```
Inside outer try block
Enter the index of element to access 5
Inside outer catch block
The index must be from 0 to 4
Leaving outer catch block
Bye Bye
```

Run-04 [When array has elements {10, 30, 0, 53, 67}]
```
Inside outer try block
Enter the index of element to access 2
The element at index 2 is 0
Inside nested try block
Enter the starting index 2
Enter the ending index 4
12
4
Inside inner catch block
The divisor must not be zero
Leaving inner catch block
Leaving outer try block
Bye Bye
```

If a method with try-catch in its body is called within try block then it is also equivalent to nested-try block. The same example can be written by creating a separate method for new assignment and then calling the method in try catch block

```
package com.masai.sprint3;
import java.util.Scanner;

public class ExceptionDemo {
  static void getQuotient(Scanner sc, int arr[]) {
    try {
      System.out.println("Inside nested try block");
      System.out.print("Enter the starting index ");
      int startIndex = sc.nextInt();
      System.out.print("Enter the ending index ");
      int endIndex = sc.nextInt();

      //code to find sum
      int sum = 0;
      for(int i = startIndex; i <= endIndex; i++)
        sum = sum + arr[i];

      //code to find quotient
      for(int i = 0; i < arr.length; i++)
        System.out.println(sum/arr[i]);

      System.out.println("Leaving nested try block");
    }catch(ArithmeticException ex) {
      System.out.println("Inside outer catch block");
      System.out.println("The divisor must not be zero");
      System.out.println("Leaving outer catch block");
    }
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    //create an array of five elements
    int arr[] = {10, 30, 5, 53, 67};
    try {
      System.out.println("Inside outer try block");
      System.out.print("Enter the index of element to access ");
      int index = sc.nextInt();
      System.out.println("The element at index " + index + " is " +
arr[index]);
      getQuotient(sc, arr);
      System.out.println("Leaving outer try block");
    }catch(ArrayIndexOutOfBoundsException ex) {
      System.out.println("Inside outer catch block");
      System.out.println("The index must be from 0 to " + (arr.length -
1));
      System.out.println("Leaving outer catch block");
    }
    System.out.println("Bye Bye");
    sc.close();
  }
}
```

The outputs will be same

The finally block

- It is executed always no matter exception takes place or not
- It will be executed even when control return from try block.
- If JVM halts then only it will not be executed.

- It is used for resource recovery.

An Example

```
try{
  code to open file
  ..
  code to read from/write to file (this code may generate exception)
  ..
  code to close file
}catch(Exception-type ref-name){
  ...
  code to handle exception
  ...
  code to close file (This code is actually duplicate.. i.e. this is WET
code which is not good)
}
```

Execution-pattern-1 (When no exception)

```
code to open file --> code to read from/write to file (no exception) -->
code to close file
```

Execution-pattern-2 (When exception occurs)

```
code to open file --> code to read from/write to file (exception occurs)
--> code to handle exception --> code to close file
```

We need a better place to write code to close the file (i.e. to recover resources) such that the file should be closed no matter exception is there or not. The same requirement can be addressed by finally block.

```
try{
  code to open file
  ..
  code to read to/write from file (this code may generate exception)
  ..
}catch(exception-type ref-name){
  ...
  code to handle exception
  ...
}finally{
     code to close file
}
```

Execution-pattern-1 (When no exception)

```
code to open file --> code to read to/write from file (no exception, try
block completed) --> code to close file (finally block will execute)
```

Execution-pattern-2 (When exception occurs)

```
code to open file --> code to read to/write from file (exception occurs)
--> code to handle exception (catch block)--> code to close file
```

```
(finally block will execute also)
```

Write code here to demonstrate calling of finally block.

```
package com.masai.sprint3;

public class FinallyDemo {
  static void throwExc(){
    try {
      System.out.println("throwExc: Inside try block");
      int a = 50/0;  //this line will throw exception
    }catch(ArithmeticException ex) {
      System.out.println("throwExc: Inside catch block");
    }finally {
      System.out.println("throwExc: Inside finally block");
    }
  }

  static void noExc(){
    try {
      System.out.println("noExc: Inside try block");
    }finally {
      System.out.println("noExc: Inside finally block");
    }
  }

  static void returnDemo(){
    try {
      System.out.println("returnDemo: Inside try block");
      return;
    }finally {
      System.out.println("returnDemo: Inside finally block");
    }
  }

  static void exitDemo() {
    try {
      System.out.println("exitDemo: Inside try block");
      System.exit(0);
    }finally {
      System.out.println("exitDemo: Inside try block");
    }
  }

  public static void main(String[] args) {
    throwExc();
    noExc();
    returnDemo();
    exitDemo();
  }
}
```

Output
```
throwExc: Inside try block
throwExc: Inside catch block
throwExc: Inside finally block
noExc: Inside try block
noExc: Inside finally block
returnDemo: Inside try block
returnDemo: Inside finally block
exitDemo: Inside try block
```

The try-with resource statement

So far we have seen about finally block that allow to prevent resource leak by ensuring that piece of code written inside finally must be executed regardless exception takes place or not. Java 7 provides one more tool to prevent resource leak, that is try-with-resource. The try-with-resource statement is a try block with one or more resource, resource refers to an object that must be closed after program finishes. The try-with-resource statement ensures that resources must be closed after try-with-resource finishes. Any object that implements the java.lang.AutoCloseable or java.io.Closeable interface can be used as a resource. Semi-colon(;) serve as a separator for multiple resources within try-with-resource statement.

```
try(code to open resource is here){
  code that may throw exception
}catch(ExceptionType-1 ex){
  code to handle exception
}
```

The throw and throws statement

Anu

Bhakti

Chitranshi

Say we have three developers that are **Anu, Bhakti** and **Chitranshi**. Look at the conversation among them

Anu: Hey Bhakti! I have created a new class that will help you to connect to database and perform

database related operations.
Bhakti: Ohh.. Can you provide me necessary details for the same
Anu: Yes! I will give you complete documentation (not code) in which all method prototypes are mentioned properly with details of parameters and nature of operations like for method to connect with database server. See documentation of a method-

```
public Connection connectToDatabase(String serverName, String username,
String password, String dbName)

  This method will connect to database
  Parameters
   1) serverName: the name/ip address of database server, for local
      system write localhost
   2) username: the username on the database server
   3) password: the password on the database server
   4) dbName: the name of database to connect on the database server

  Returns: An object of connection class upon successful connection to
  database, null if connection failed due to any reason.
```

Bhakti: Great! this is the same way documentation is provided by java also but connection failure can take place due to multiple reasons like server is not running, username & password provided is wrong, database does not exists etc etc. You are just returning null which given no details about the exact reason of failure. How i can figure out is fault is on my side or it is on server side.
Anu: Yes! you are right, i am getting different exception type for every failure reason but i don't know how to pass these to you such that you are caller of my method. Can you tell me what to do?
Bhakti: You can use throw keyword to throw the exception to caller from callee such that caller is now aware about the the cause of failure also he will do the required to make correction.
Anu: okay! great, thanks for guiding me so what i understand about throw keyword is

- It is used to throw the exception; JVM uses throw keyword to throw exception to our program when something abnormal takes place during the program execution.
- We can have exception object in two ways
  a. Grab it in catch block as it is created by JVM itself due to error in try block code
  b. We can create object using new keyword also because every exception type is a class

Bhakti: You are absolutely right. Improve this code and then Chitranshi will give you further feedbacks as i having next meeting.
Anu: okay
Chitranshi: Hey Anu! Can you Help me in connecting to database as i am new to database connection
Anu: Yes... i have developed a library, you just need to call the methods and you are done because all code is done by me.
Chitranshi: Hurray! great, can you tell me which method to call to connect to database
Anu: public Connection connectToDatabase(String serverName, String username, String password, String dbName)
[Say **chitranchi** is calling this method and by mistake she passed wrong combination of username and password]

Chitranshi: Hey Anu! This is unacceptable, you method is throwing some exception that i was unaware about so i didn't written any code to handle this exception this leads to crashing of my program. You must have told me for the exceptions that you method is throwing
Anu: Grrrrrr........ Bhakti told me to use throw keyword to throw the exception to the caller but she didn't told me how to inform the caller about the possible exception that can be thrown
Chitranshi: You must use throws clause with method signature that contains list of all exception types that your method is throwing
Anu: so my method signature must be public Connection connectToDatabase(String serverName, String username, String password, String dbName) throws ExceptionType-1, ExceptionType-2, ExceptionType-3
Chitranshi: Absolutely! This thing has to be used for every method
Anu: okay! great, thanks for guiding me so what i understand about throws clause is

- ➢ It is used to inform caller about the exceptions that a method can throw

Chitranshi: Yes Anu, but you should also remember

- ➢ If a method is throwing checked exception then it is mandatory to write the exception-type in the throws clause; no such restriction exists for unchecked type.
- ➢ Calling of method that throws checked exception must be either inside the try-catch block in caller or the caller should re-throw the exception again; no such restriction for unchecked exception.

An Example
```
import java.util.Scanner;

public class TryCatchDemo {
  static void printElement(int arr[], int index) {
    try {
      System.out.println("The element at index " + index + " is " +
arr[index]);
    }catch(ArrayIndexOutOfBoundsException ex) {
      System.out.println("Exception caught in printElement method");
      throw ex;//exception rethrown so caller can also do sth for this
exception
    }
  }

  static void checkAge(int age) throws IllegalArgumentException {
    if(age < 0 || age > 120) {
      IllegalArgumentException ex = new IllegalArgumentException("The
age must be between 0 and 120");
      throw ex;
    }
  }

  public static void main(String[] args) {
    int arr[] = {10, 29, 340};
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter index of array ");
    int index = sc.nextInt();
    try {
      printElement(arr, index);
      checkAge(155);
    }catch(ArrayIndexOutOfBoundsException ex) {
      System.out.println("Index must be from 0 to " + (arr.length - 1));
    }catch(IllegalArgumentException ex) {
      System.out.println(ex.getMessage());
    }

    sc.close();
  }
}
```

Run-01

```
The element at index 1 is 29
The age must be between 0 and 120
```

Creating Custom Exceptions

Say we have class with strength of 40 students so roll no of scholars will be from 1 to 40. Any roll number less than 1 and more than 40 will be considered as invalid roll number. For storing the roll number, We need to create variable of int data type, assigning value from -2147483468 to 2147483467 is perfectly valid according to the JVM i.e. assigning less than 1 or more than 40 to the roll number variable is no error according to the JVM i.e. no exception will be thrown from JVM side but according to our application an exception must be thrown when anybody assigns a value to roll no other than 1 to 40.

Solution: Create user defined exception (Also called custom exception)

Following are common steps for custom exception

- ➢ Create subclass of Exception class
- ➢ Provide a public constructor with String arguments
- ➢ Provide definition of public String toString() method

```
package com.masai.sprint3;
public class InvalidRollNumberException extends Exception{  //step-1
  public InvalidRollNumberException(String message){//step-2
    super(message);
  }

  @Override
  public String toString(){//step-3
    return "InvalidRollNumberException: " + getMessage();
  }
}
```

```
package com.masai.sprint3;
public class InvalidNameException extends Exception{//step-1
  public InvalidNameException(String message){//step-2
    super(message);
  }

  @Override
  public String toString(){//step-3
    return "InvalidNameException: " + getMessage();
  }
}

package com.hexaware.pack;
public class ExceptionDemoSeven{
  //say we are maintaining data of student
  private int rollNo;
  private String name;

  public void setRollNo(int rollNo) throws InvalidRollNumberException{
    if(rollNo < 0 || rollNo > 40){
      //throw InvalidRollNumberException
      InvalidRollNumberException rnExc = new
InvalidRollNumberException("Roll number must be from 1 to 40");
      throw rnExc;
    }
    this.rollNo = rollNo;
  }

  public int getRollNo(){
    return rollNo;
  }

  public void setName(String name) throws InvalidNameException{
    if(name == null || name.length() < 3){
      InvalidNameException nmExc = new InvalidNameException("Name is
required and must be of minimum 3 characters");
      throw nmExc;
    }
    this.name = name;
  }

  public String getName(){
    return name;
  }

  @Override
  public String toString(){
    return "Student[Roll no " + rollNo + ", name = " + name + "]";
  }
  public static void main(String args[]){
    ExceptionDemoSeven st = new ExceptionDemoSeven();
    try{
      st.setRollNo(25);
      st.setName("");
      System.out.println(st);
    }catch(InvalidRollNumberException | InvalidNameException ex){
      System.out.println(ex);
    }
  }
}
```

Output

```
InvalidNameException: Name is required and must be of minimum 3 characters
```

Some important Points

1. If a method in super class throws no exception/unchecked exception then while overriding in sub-class method cannot throw a checked exception. Yet it can throw unchecked exception or no exception

Case-1

```
class A{
        public void show(){
                …
        }
}

class B extends A{
        public void show() throws ArrayIndexOutOfBoundsException{  //okay
                …
        }
}
```

Case-2

```
class P{
  public void show() throws ArrayIndexOutOfBoundsException{
    …
  }
}

class Q extends P{
  public void show(){  //okay
    …
  }
}
```

Case-3

```
class X{
  public void show() throws ClassCastException{
    …
  }
}

class Y extends X{
  public void show() throws ArrayIndexOutOfBoundsException{  //okay
    …
  }
}
```

Case-4

```
class X{
  public void show() throws ClassCastException{
    …
  }
}

class Y extends X{
  public void show() throws FileNotFoundException{  //Error
    …
  }
}
```

2. If a method in super class throws checked exception then while overriding in sub-class method can throw checked exception of same of sub-type but not super-type.
   Tip: FileNotFoundException and IOException both are checked exception defined in java.io package

Case-1

```
class X{
  void show() throws IOException{
    ..
  }
}

class Y extends X{
  void show() throws FileNotFoundException{  //okay
    ..
  }
}
```

Case-2

```
class P{
  void show() throws FileNotFoundException{
    ..
  }
}

class Q extends P{
  void show(){  //Okay
    ..
  }
}
```

Case-3

```
class P{
  void show() throws FileNotFoundException{
    ..
  }
}

class Q extends P{
  void show() throws IOException{  //Error
    ..
  }
}
```

You Activity

Write a program to implement the LIFO system
1. Create class OverflowException to represent the overflow of stack
2. Create class UnderflowException to represent the underflow of stack
3. Create an interface Stack with following operations

```
interface Stack{
  void push(int element) throws OverflowException;
  int pop() throws UnderflowException;
  int peek() throws UnderflowException;
  static String displayStackElements(int stack[]){
    //write code to convert the stack elements to comma separate String
sequence from index 0 to top
  }
}
```

4. create implementing class

```
class MyStack implements Stack{
  private int stack[];
  private int top;
  final static int MAX_SIZE = 5;

  MyStack(){
    stack = new int[MAX_SIZE];
    top = -1;
  }
  ...
  ...

  public String toString(){
    return Stack.displayStackElements(stack);
  }
}

import java.util.Scanner;

class Demo{
  public static void main(String args[]){
    Scanner sc = new Scanner(System.in);
    int choice = 0;
    Stack stack = new MyStack();
    do{
      System.out.println("1. Push element to stack");
      System.out.println("2. pop element from stack");
      System.out.println("3. get top element of stack");
      System.out.println("4. display element of stack");
      System.out.println("5. Bye Exit");
      System.out.print("Enter selection ");
      choice = sc.nextInt();
      int element = -1;
      try{
        switch(choice){
          case 1:
            System.out.print("Enter element ");
            element = sc.nextInt();
            stack.push(element);
            break;
          case 2:
            element = stack.pop();
            System.out.print("The popped element is " + element);
            break;
          case 3:
            element = stack.peek();
            System.out.print("The peek element is " + element);
            break;
          case 4:
            System.out.print(stack);
            break;
          case 5:
            System.out.println("Bye bye");
            break;
          default:
            System.out.println("Invalid Selection! try again");
        }
      }catch(UnderflowException | OverflowException ex){
          System.out.println(ex.getMessage());
        }
    }while(choice != 5);
    sc.close();
  }
}
```

Sample Output

```
1. Push element to stack
2. pop element from stack
3. get top element of stack
4. display element of stack
5. Bye Exit
Enter selection 1
Enter element 10
1. Push element to stack
2. pop element from stack
3. get top element of stack
4. display element of stack
5. Bye Exit
Enter selection 1
Enter element 20
1. Push element to stack
2. pop element from stack
3. get top element of stack
4. display element of stack
5. Bye Exit
Enter selection 1
Enter element 30
1. Push element to stack
2. pop element from stack
3. get top element of stack
4. display element of stack
5. Bye Exit
Enter selection 1
Enter element 40
1. Push element to stack
2. pop element from stack
3. get top element of stack
4. display element of stack
5. Bye Exit
Enter selection 1
Enter element 50
1. Push element to stack
2. pop element from stack
3. get top element of stack
4. display element of stack
5. Bye Exit
Enter selection 1
Enter element 20
No space in stack to add element
1. Push element to stack
2. pop element from stack
3. get top element of stack
4. display element of stack
5. Bye Exit
Enter selection 3
The peek element is 50
1. Push element to stack
2. pop element from stack
3. get top element of stack
4. display element of stack
5. Bye Exit
Enter selection 4
10 20 30 40 50
1. Push element to stack
2. pop element from stack
3. get top element of stack
4. display element of stack
5. Bye Exit
Enter selection 2
The popped element is 50
1. Push element to stack
2. pop element from stack
3. get top element of stack
4. display element of stack
5. Bye Exit
Enter selection 2
The popped element is 40
1. Push element to stack
2. pop element from stack
3. get top element of stack
4. display element of stack
5. Bye Exit
Enter selection 2
The popped element is 30
1. Push element to stack
2. pop element from stack
3. get top element of stack
4. display element of stack
5. Bye Exit
Enter selection 2
The popped element is 20
1. Push element to stack
2. pop element from stack
3. get top element of stack
4. display element of stack
5. Bye Exit
```

```
Enter selection 2
The popped element is 10
1. Push element to stack
2. pop element from stack
3. get top element of stack
4. display element of stack
5. Bye Exit
Enter selection 2
No element in stack to pop
The popped element is 10
1. Push element to stack
2. pop element from stack
3. get top element of stack
4. display element of stack
5. Bye Exit
Enter selection 5
Bye bye
```