Day-11 JA-111 Batch

Topics

- **Regular Expressions**
- **Exception Handling Introduction**
- **The try-catch block & multiple catch statements**

Regular Expressions

A regular expression is a sequence of characters that forms a search pattern. When you search for data in a text, you can use this search pattern to describe what you are searching for. A regular expression can be a single character, or a more complicated pattern.

The main important application areas of Regular Expression are:

- To implement validation logic.
- To develop Pattern matching applications.
- To develop translators like compilers, interpreters etc.

Here are the some symbols-

Regular Expression	**Description**
^regex	Finds regex that must match at the beginning of the line.
regex$	Finds regex that must match at the end of the line. $ Checks if a line end follows.
X|Z	Finds X or Z.
XZ	Finds X directly followed by Z.
[abc]	Set definition, can match the letter a or b or c.
[abc][vz]	Set definition, can match a or b or c followed by either v or z.
[a-z]	Ranges: matches a letter between a and z
[^abc]	When a caret appears as the first character inside square brackets, it negates the pattern. i.e. this pattern matches any character except a or b or c.

The backslash \ is an escape character in Java Strings. That means backslash has a predefined meaning in Java. You have to use double backslash \\ to define a single backslash. If you want to define \w, then you must be using \\w in your regex. If you want to use backslash as a literal, you have to type \\\\ as \ is also an escape character in regular expressions.

Meta Characters

Regular Expression	**Description**
\d	Any digit, short for [0-9]
\D	A non-digit, short for [^0-9]
\s	A whitespace character, short for [\t\n\x0b\r\f]
\S	A non-whitespace character, short for ^[\t\n\x0b\r\f]
\w	A word character, short for [a-zA-Z_0-9]
\W	A non-word character [^\w]
\S+	Several non-whitespace characters

Qualifiers

The quantifiers specify the number of occurrences of a character.

Qualifier	Description
.	Matches any character
*	Occurs zero or more times, is short for {0,} e.g. X* finds no or several letter X, .* finds any character sequence
+	Occurs one or more times, is short for {1,} e.g. X+ Finds one or several letter X
?	Occurs no or one times, ? is short for {0,1}. e.g. X? finds no or exactly one letter X
{X}	Occurs X number of times, {} describes the order of the preceding liberal e.g. \d{3} searches for three digits, .{10} for any character sequence of length 10.
{X,}	Occurs minimum X number of times e.g. \d{3,} searches for minimuum three digits, .{10,} for any character sequence of minimum length 10.
{X,Y}	Occurs between X and Y times, e.g. \d{1,4} means \d must occur at least once and at a maximum of four.

The **java.util.regex** package primarily consists of the following three classes −

Pattern Class

A Pattern object is a compiled representation of a regular expression. The Pattern class provides no public constructors. To create a pattern, you must first invoke one of its public static compile() methods, which will then return a Pattern object. These methods accept a regular expression as the first argument.

- static Pattern compile(String regex): Compiles the given regular expression into a pattern.
- Matcher matcher(CharSequence input): Creates a matcher that will match the given input against this pattern.

Matcher Class

A Matcher object is the engine that interprets the pattern and performs match operations against an input string. Like the Pattern class, Matcher defines no public constructors. You obtain a Matcher object by invoking the matcher() method on a Pattern object.

- boolean matches(): Attempts to match the entire region against the pattern.
- boolean find(): it attempts to find the next match and returns true if it is available otherwise returns false.
- int start(): returns the starting index of the matched subsequence.
- int end(): returns the ending index of the matched subsequence. This index is excluded for subsequence
- String group(): returns the matched subsequence.

PatternSyntaxException

A PatternSyntaxException object is an unchecked exception that indicates a syntax error in a regular expression pattern.

An Example

```
package com.masai.sprint3;
import java.util.regex.*;

class SimpleRegExDemo{
  public static void main(String[] args) {
    Pattern p;
    Matcher m;

    int count = 0;
    p = Pattern.compile("ab");
    m = p.matcher("abbbabbaba");
    while (m.find()) {
      count++;
      System.out.println("The " + m.group() + " is from " + m.start() +
" to " + (m.end() - 1));
    }
    System.out.println("The no of occurrences of is " + count);

    System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");

    p = Pattern.compile("\\s");
    String[] s = p.split("Hello\thow are\nyou");
    for(String s1:s)
      System.out.println(s1);
    System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");

    String expressions[] = {"[a-z]", "[0-9]", "[abc]", "[^abc]", "[^a-
zA-Z0-9]"};
    for(String currentPattern: expressions) {
      p = Pattern.compile(currentPattern);
      m = p.matcher("a7b@z#9");
      while (m.find()) {
        System.out.println(m.start() + "-------" + m.group());
      }
      System.out.println("-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");
    }
  }
}
```

Output

```
The ab is from 0 to 1
The ab is from 4 to 5
The ab is from 7 to 8
The no of occurrences of is 3
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
Hello
how
are
```

```
you
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
0-------a
2-------b
4-------z
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
1-------7
6-------9
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
0-------a
2-------b
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
1-------7
3-------@
4-------z
5-------#
6-------9
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
3-------@
5-------#
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
```

An Example

```
package com.masai.sprint3;
import java.util.regex.*;

class ValidationregExpDemo{
  public static void main(String[] args) {
    Pattern p;
    Matcher m;

    //code to check if a 10 digit indian mobile number of valid
    //Pattern contains the Regular Expression
    p = Pattern.compile("^[6-9]\\d{9}");

    //Matcher is created from pattern such that it contains the input
    m = p.matcher("9865326958");

    //use matches function to perform the matching
    System.out.println("Is 9865326958 a valid indian phone number " +
m.matches());

    //is same as System.out.println(Pattern.matches("^[6-
9]\\d{9}","9865326958"));
    //for validating the name such that it will validate name containing
letters and spaces only
    p = Pattern.compile("^[A-Za-z\\s]+$");
    m = p.matcher("Kapil Agrawal");
    System.out.println("Is Kapil Agrawal a valid name? " + m.matches());
    //for validating the username such that it will validate name
containing letters, digits and spaces only
    p = Pattern.compile("^[A-Za-z0-9_.]+$");
    m = p.matcher("Kapil_Agrawal");
    System.out.println("Is Kapil_Agrawal a valid name? " + m.matches());

    //code to check if an email is valid in format or not
    p = Pattern.compile("^[a-zA-Z0-9_.]+@[a-zA-Z0-9.-]+$");
    m = p.matcher("abc.123@gmail.com");
    System.out.println("Is abc.123@gmail.com a valid email? " +
m.matches());
  }
}
```

Output

```
Is 9865326958 a valid indian phone number true
Is Kapil Agrawal a valid name? true
Is Kapil_Agrawal a valid name? true
Is abc.123@gmail.com a valid email? true
```

Exception Handling

Types of Error

Compile Time Error: Due to incorrect syntaxes; Program will not compile (and definitely will not run)

```
Int a;    //int not Int
system.out.println("abc");    //System not system
```

Logical Error: Due to inocrrect logic; Program will compile and run also (but will produce incorrect output)

```
int age = sc.nextInt();

if(age > 18){
  System.out.println("Adult");
}else{
  System.out.println("Minor");
}
```

Output

```
age = 18
Minor
```

Run Time Error: When program try to break rules of JVM; Program will compile and run also (but it will crash if error not handled)

```
int arr[] = {25, 24, 78, 96, 110}; //Total elements: 5 index: 0 - 4
System.out.println(arr[10]);  //Run Time Error, Program will crash here
System.out.println("bye bye");
```

Exception

- It is Run Time Error
- It disrupt normal flow of program execution
- If not handled then leads to the crashing of the program

An Example

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

public class ExceptionDemoOne{
  public static void main(String args[]){
    //Create an array of five elements
    int arr[] = {10, 20, 54, 86, 9};
    Scanner sc = new Scanner(System.in);

    //code to take input
    System.out.print("Enter the index of element to access ");
    int index = sc.nextInt();

    System.out.println("The element at " + index + " is " + arr[index]);
    System.out.println("Bye Bye");

    //close the object of Scanner class
    sc.close();
  }
}
```

Run-1

```
Enter the index of element to access 2
The element at 2 is 54
Bye Bye
```

Run-2

```
Enter the index of element to access 6
Exception  in  thread  "main"  java.lang.ArrayIndexOutOfBoundsException:
Index 6 out of bounds for length 5
  at package
com.masai.sprint3.ExceptionDemoOne.main(ExceptionDemoOne.java:16)
```

In case of Run-02 when value of variable index is 6 then for the line

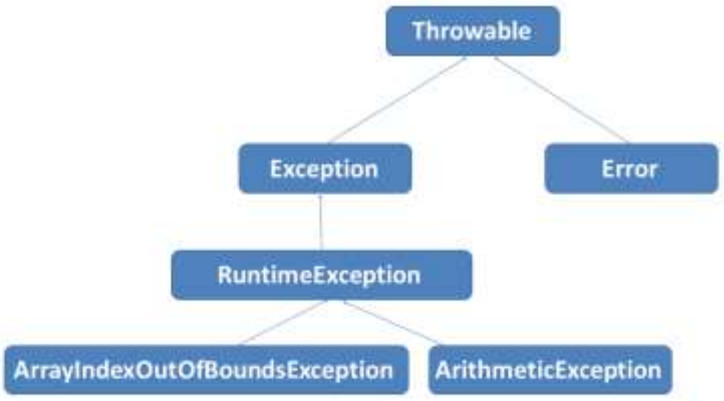
```
  System.out.println("The element at " + index + " is " + arr[index]);
```

JVM will find that the program is trying to access arr[6] that actually doesn't exists because valid range for array index is from 0 to 4. Java finds it against the law decided by JVM so JVM will generate a run time error (exception) on the same line.

For this run time error, JVM will create an object that contains the information about the exception like ExceptionType, line number, method name, class-name, file name etc and that it will throw the same object to the program. The process of creating the object and handing over it to the program is referred to as throwing an exception.

If program is not able to handle the exception then it will be handled by the default handler which is provided by JVM. The default handler terminate the program in the same line where exception was generated and display the stack trace (kind of error log) so that programmer can make correction in program.

Exception Hierarchy



Checked Exception

- These are the exceptions that a program is expected to handle i.e. programmer has to write the code to handle such kind of exception.
- If programmer fails to write code to handle such exception then compiler will not let program compile. i.e. one can say that compiler is checking if programmer has written the code to handle exception or not.
- Exception class and its subclasses of exception class are representing the checked exception e.g. FileNotFoundException, SQLException, ClassNotFoundException

Unchecked Exception

- These are the exceptions that a program is not expected to handle i.e. programmer is not bound to write the code to handle such kind of exception.
- If programmer not writing code to handle such exception then compiler not report any error. i.e. one can say that compiler is not checking if programmer has written the code to handle exception or not.
- subclasses of RuntimeException & Error class represents the unchecked exception e.g. ArrayIndexOutOfBoundsException, ArithmeticException, NullPointerException etc.

The try-catch block

```
try{
  statement-1
  statement-2 (code that may generate exception)
  statement-3
}catch(ExceptionType ex){
  code to handle the exception
}
statement-x
```

Exception-pattern-1 (When no exception)

```
statement-1 -> statement-2 -> statement-3 -> statement-x [No catch block because no exception]
```

Exception-pattern-2 (When exception occurs)

```
statement-1 -> statement-2 (Exception occurs) -> catch: code to handle the exception -> statement-x
```

An Example

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

public class ExceptionDemoTwo{
  public static void main(String args[]){
    //Create an array of five elements
    int arr[] = {10, 20, 54, 86, 9};
    Scanner sc = new Scanner(System.in);

    try{
      System.out.println("Entered in try block...");
      System.out.print("Enter the index of element to access ");
      int index = sc.nextInt();
      System.out.println("The element at " + index + " is " +
arr[index]);
      System.out.println("Leaving the try block...");
    }catch(ArrayIndexOutOfBoundsException ex){
      System.out.println("Inside catch block");
      System.out.println("Index must be from 0 to " + (arr.length - 1));
    }
    System.out.println("Bye Bye");

    //close the object of Scanner class
    sc.close();
  }
}
```

Run-1

```
Entered in try block...
Enter the index of element to access 3
The element at 3 is 86
Leaving the try block...
Bye Bye
```

Run-2

```
Entered in try block...
Enter the index of element to access 8
Inside catch block
Index must be from 0 to 4
Bye Bye
```

Multiple catch blocks

A single line of code may generate multiple type of Exceptions; To handle them we can use multiple catch blocks with a single try block. In case of exception, the catch blocks will be checked in the order of writing, whichever first catch block is compatible will be executed and remaining will be bypassed.

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

public class ExceptionDemoThree{
  public static void main(String args[]){
    //Create an array of five elements
    int arr[] = {10, 20, 54, 86, 9};
    Scanner sc = new Scanner(System.in);

    try{
      System.out.println("Entered in try block...");
      System.out.print("Enter the index of element to access ");
      int index = sc.nextInt();
      System.out.println("arr[" + index + "] / " + (index - 1) + " = " +
(arr[index]/(index - 1)));
      System.out.println("Leaving the try block...");
    }catch(ArrayIndexOutOfBoundsException ex){
      System.out.println("Inside catch block");
      System.out.println("Index must be from 0 to " + (arr.length - 1));
    }catch(ArithmeticException ex){
      System.out.println("Inside catch block");
      System.out.println("The divisor must not be zero");
    }
    System.out.println("Bye Bye");

    //close the object of Scanner class
    sc.close();
  }
}
```

Run-1

```
Entered in try block...
Enter the index of element to access 3
arr[3] / 2 = 43
Leaving the try block...
Bye Bye
```

Run-2

```
Entered in try block...
Enter the index of element to access 8
Inside catch block
```

```
Index must be from 0 to 4
Bye Bye
```

Run-3
```
Entered in try block...
Enter the index of element to access 1
Inside catch block
The divisor must not be zero
Bye Bye
```

Note
```
catch(Throwable ex) {
  ...
}
```

The above catch block can handle any type of exception because Throwable class is super class of all Exception classes and a reference variable of super class can point to the object of child class.

One important Point
If more than one catch block is there with the try block such that the catch blocks contain the exception-type of super-type and sub-type then make sure to write sub-type catch block first followed by catch block of super type otherwise sub-type catch block will be unreachable.

This implementation is correct
```
catch(ArrayIndexOutOfBoundsException ex) {
  System.out.println("Inside first catch block");
  System.out.println("The index must be from 0 to " + (arr.length - 1));
  System.out.println("Leaving first catch block");
}catch(Exception ex) {
  System.out.println("Inside second catch block");
  System.out.println("The divisor must not be zero");
  System.out.println("Leaving second catch block");
}
```

This implementation is incorrect
```
catch(Exception ex) {
  System.out.println("Inside second catch block");
  System.out.println("The divisor must not be zero");
  System.out.println("Leaving second catch block");
}catch(ArrayIndexOutOfBoundsException ex) {
  System.out.println("Inside first catch block");
  System.out.println("The index must be from 0 to " + (arr.length - 1));
  System.out.println("Leaving first catch block");
}
```

Since java 7.0, you can use one catch block to handle multiple types of exception such that you want to perform one common action for multiple types of exceptions.

```
try{
  some code that may generate exception
}catch(ArrayIndexOutOfBoundsException | ArithmeticException ex){
  System.out.println("Something went wrong!");
}
```

You Activity

Mr. Ankesh is a CA and he often need PAN & adhaar number of tax payer to file and update the ITR related work. Often it happens that his clients provide wrong PAN/adhaar number (Like sometime skipping a character or writing characters for two times). You need to write a java code for him such that both details must be validated for the format
(i) PAN Number is a sequence of 10 alpha-numerical characters such that the first five characters are letters followed by four digits followed by a single letter e.g. ABCDE0123Z
Some invalid values for PAN numbers are "_BCDE1234Z" "ABCDE12345" "ABCD 1234Z" "ABCDE1234ZZ" (Check you code for these values also)
(ii) Adhaar is 12 digit number only (No space it has).
Some invalid values for adhaar numbers "12655241789" "1265524178998" "1234 5263 7485", "123652 857496" (Check you code for these values also)

Now his assistance Nagarjuna comes and says that the combination of adhaar and PAN is unique for every tax payer and ion their office they maintains name, adhaar and PAN of their clients. Nagarjuna maintains the details in the array such that no client should be added twice i.e. the combination of adhaar and PAN must be unique for tax payer.

Help ankesh and nagarjuna by completing this code

```
package com.masai.sprint3;
import java.util.regex.Pattern;
import java.util.regex.Matcher;
import java.util.Scanner;

class TaxPayer{
  private String PAN;
  private long adhaarNumber;
  private String name;

  //generate paremterized constructors
  //generate getter & setter method

  //override equals method that must return true if PAN and adhaar is
same for calling and parameter object
  //ovrride the hashCode method to generate same hashcode for two
taxpayers whose adhaar and PAN is same
  //override the toString() method to returb String representation of
TaxPayer as follow-
  // Name: <name>, PAN: <pan-number>, Adhaar Number: <adhaar-number>
[See output for details]
```

```
}

class Demo{
  /**
   * Method to check if PAN is valid or Not
   * @param PAN: String, The PAN of User
   *  @return  true  if  PAN  number  is  valid  according  to  format,  false
otherwise
   */
  static boolean checkPAN(String PAN) {
    //write code to validate the PAN number using regular expression
  }

  /**
   * Method to check if adhaar number is valid or Not
   * @param adhaarNumber: long, The adhaar number of User
   *  @return  true  if  adhaar  number  is  valid  according  to  format,  false
otherwise
   */
  static boolean checkAdhaar(long adhaarNumber) {
    //write  code  to  validate  the  adhaarNumber  number  using  regular
expression
  }

  /**
   * Method to add new tax payer to array of tax payers iff new tax has
unique combination of PAN and adhaar
   * @param tpArr: TaxPayer[] The array of TaxPayers in which tax payers
to be added
   * @param newTP: TaxPayer The taxpayer to be added to the array
   * @return: true if the taxpayer added to the array, false otherwise
   */
  static boolean addTaxPayer(TaxPayer tpArr[], TaxPayer newTP) {
    //Write code to add newTP to tpArr if taxpayers with newTP's PAN and
adhaar is not here
  }

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    //Create an array of objects
    TaxPayer tpArr[] = new TaxPayer[5];
    //
    int totalTaxPayers = 0;
    while(true) {
      System.out.print("Please enter name, PAN and adhaar Number ");
      String name = sc.next();
      String panNumber = sc.next();
      long adhaarNumber = sc.nextLong();
      if(checkPAN(panNumber) && checkAdhaar(adhaarNumber)) {
        //you are here means PAN and adhaar are in correct format
        if(addTaxPayer(tpArr,   new   TaxPayer(panNumber,   adhaarNumber,
name))) {
          //you are here means tax payer has added to the array
          System.out.println("Yahoo! Tax payer added\n");
          if(++totalTaxPayers == tpArr.length)
            //the array is full so better to break the loop
            break;
        }else {
          System.out.println("Tax Payer with this PAN & adhaar already
exists\n");
        }
      }else {
        System.out.println("Enter correct PAN & adhaar number\n");
      }
    }

    System.out.println("Details of tax payers are as follow-");
    for(TaxPayer tp: tpArr)
      System.out.println(tp);

    sc.close();
  }
}
```

Output

```
Please enter name, PAN and adhaar Number ABC ABCDE1234Z 123456789012
Yahoo! Tax payer added

Please enter name, PAN and adhaar Number XYZ ABCDE1234Z 123456789012
Tax Payer with this PAN & adhaar already exists

Please enter name, PAN and adhaar Number BCD ABCDE1234Y 123456789013
Yahoo! Tax payer added

Please enter name, PAN and adhaar Number CDE ABCDE1234X 123456789014
Yahoo! Tax payer added

Please enter name, PAN and adhaar Number DEF ABCDE1234W 123456789015
Yahoo! Tax payer added

Please enter name, PAN and adhaar Number DEF _BCDE1234Z 123456789016
Enter correct PAN & adhaar number

Please enter name, PAN and adhaar Number DEF ABCDE1234W 12655241789
Enter correct PAN & adhaar number

Please enter name, PAN and adhaar Number EFG ABCDE1234V 123456789017
Yahoo! Tax payer added
```

Details of tax payers are as follow-
Name: ABC, PAN: ABCDE1234Z, Adhaar Number: 123456789012
Name: BCD, PAN: ABCDE1234Y, Adhaar Number: 123456789013
Name: CDE, PAN: ABCDE1234X, Adhaar Number: 123456789014
Name: DEF, PAN: ABCDE1234W, Adhaar Number: 123456789015
Name: EFG, PAN: ABCDE1234V, Adhaar Number: 123456789017