11/04/11

Regular Expressions

→ Any group of Strings according to a particular pattern is Called "regular Expression".

ex:- ① We Can write a regular Expression to represent all valid mail-ids & By using that regular Expression we Can validate whether the given mail-id is valid or not.

② we Can write a regular Expression to represent all valid Java identifiers.

→ The main Application areas of regular Expressions are

1. We Can implement validation mechanism.
2. we Can develope pattern matching applications.
3. we Can develope translators like Compilers interpreters e.t.c.
4. we Can use for designing digital Circuits
5. we Can use to develope Communication protocols like TCP by IP, UDP e.t.c

Ex:
```
import java.util.regex.*;
class RegExDemo
{
 p.s.v.m(String[] args)
 {
   pattern p = Pattern.compile("ab");
   Master m = p.matcher("abbbabbcbdab");
while(m.find())
{
S.o.pln(m.start() +"---" +m.end() + "---" +m.group());
}
}
}
```

O/P:-
```
0 ---2 ---ab
4 ---6 ---ab
10 ---12 ---ab
```

Pattern class:-

→ A pattern object represents Compiled Version of regular Expression
We Can Create a pattern object by using Compile() of Pattern class.

Pattern p = Pattern.compile(String regularExpression);

Matcher Class:-

→ A matcher object Can be used to match character Sequence against a regular Expression. We Can Create a Matcher Object by using Matcher() of pattern class

Matcher M = p.matcher(String target);

Important methods of matcher class:-

(1) boolean find();

→ It attempts to find next match & if it is available returns True otherwise returns false.

(ii) int start();-

→ returns start index of the match

(iii) int end();

→ returns end index of the match

(iv) String group();

→ returns the matched pattern

Character Classes :-

① [a-z] → Any lower case alphabet Symbol
② [A-Z] → Any upper " "
③ [a-zA-Z] → Any alphabet Symbol
④ [0-9] → Any digit from 0 to 9
⑤ [abc] → either a or b or c
⑥ [^abc] → Except a or b or c.
⑦ [0-9a-zA-Z] → Any alpha numeric character.

Ex:-

```
Pattern p = Pattern.compile("x");
Matcher m = p.matcher("a3b@c4z#");
while(m.find())
{
S.o.pln(m.start()+"---"+m.group());
}
```

x = [ab]
```
0----a
2----b
```

x = [a-z]
```
0---a
2---b
4---c
6---z
```

x = [0-9]
```
1---3
5---4
```

x = [0-9a-z]
```
0---a
1---3
2---b
4---c
5---4
6---z
```

Predefined-character class:-

Space character	$\longrightarrow$	\s
[0-9]	$\longrightarrow$	\d
[0-9a-zA-Z]	$\longrightarrow$	\w
Anycharacter	$\longrightarrow$	.

Ex:.

```
Pattern p = Pattern.compile("x");
Matcher m = p.matcher("a3z4@ k7#");
                       012345678
while(m.find())
{
 S.o.pln(m.start() + "----" + m.group());
}
```

```
x = \\d
1 ----- 3
3 ----- 4
7 ----- 7
```

```
x = \\w
0 --- a
1 --- 3
2 --- z
3 --- 4
6 --- k
7 --- 7
```

```
x = \\s
5 -----
```

```
x = .
0 — a
1 — 3
2 — z
3 — 4
4 — @
5 —
6 — k
7 — 7
8 — #
```

Quantifiers:-

→ We can use Quantifiers to specify no. of characters to match

Ex:-

1) a $\longrightarrow$ Exactly one a
2) a^+ $\longrightarrow$ atleast one a
3) a^* $\longrightarrow$ Any no. of a's
4) a? $\longrightarrow$ atmost one a.

ex:-
```
Pattern p = Pattern.compile("x");
Matcher m = p.matcher("abaabaaab");
while(m.find())
{
  S.o.pln(m.start()+"-----"+m.group());
}
```

x = a
```
0 ----a
2 ---a
3 ---a
5 ---a
6 ---a
7 ---a
```

x = a+
```
0 --- a
2 --- aa
5 ---aaa
```

x = a*
```
0 ---- a
1------
2 ----aa
4-----
5---- aaa
8 ----
9----
```

x = a?
```
0 ---- a
1 ----
2 ----a
3 --- a
4 ----
5 ----a
6 ----a
7 ----a
8 ----
9-----
```

Split method (8) :-

Pattern class Contains Split() method to Split given String according to a regular Expression.

ex:-
```
Pattern p = pattern.compile("\\s");
String[] s = p.split("Durga software Solutions");
for(String s1 : s)
 {
   S.o.pln(s1); // Durga
 }              Software
                Solutions.
```

Ex(2):-

```
Pattern p = pattern.compile("\\.");        "[.]"
String[] s = p.split("www.durgaJobs.com");
for (String s1 : s)
{
  S.o.pln(s1);          o/p:- www
}                             durgajobs
                              Com
```

String class split() method :-

→ String class also Contains spilt() to Spilt the given String against a regular Expression

Ex:-

```
String s = " www.durgaJobs.com";
String[] s1 = s.spit("\\.");
 for ( String s2 : s)
  {
   S.o.pln( s2);        www
  }                     durgajobs
                        com
```

Note:-

Pattern class Split() Can take target String as argument where as String class Split() Can take regular Expression as argument.

StringTokenizer :-

→ We can use StringTokenizer to divide the target String into Stream of Tokens according to the

→ StringTokenizer class presenting in java.util package.

Ex!-

①
```
StringTokenizer st = new StringTokenizer("Durga Software Solu-tions");
while (st.hasMoreTokens())
{
 S.o.pln (st.nextToken());
}
```
o/p!-
```
Durga
Software
Solutions
```

Note!- The default regular Expression is Space

②
```
StringTokenizer st = new StringTokenizer("1,00,000", ",");
while(st.hasMoreTokens())
{
 S.o.pln(st.nextToken());
}
```
o/p!-
```
1
00
000
```

o/p !-
```
1
00
000
```

Ex(a) :-

Write a regular Expression to represent the Set of all valid identifiers in Java language.

Rules: (1) The length of each identifier is atleast 2

(2) The allowed characters are a to z
A to Z
0 to 9
.
_

(3) The first character should not digit

R.E:- $[a\text{-}zA\text{-}Z.\_]\underbrace{[a\text{-}zA\text{-}Z0\text{-}9.\_]}_{x}\underbrace{[a\text{-}zA\text{-}Z0\text{-}9.\_]^{*}}_{x^{*}}$ (or) $x.x^{*} = x^{+}$

$[a\text{-}zA\text{-}Z.\_][a\text{-}zA\text{-}Z0\text{-}9.\_]^{+}$

```
import java.util.regex.*;
class RegExDemo2
{
 p.s.v.m(String[] args)
 {
  Pattern p = Pattern.compile("[a-zA-Z._][a-zA-Z0-9._]+");
  Matcher m = p.matcher(args[0]);
  if(m.find()&& m.group().equals(args[0]))
  {
   S.o.pln("valid Identifier");
  }
  else
  {
   S.o.pln("Invalid Identifier");
  }
 }
}
```

② w.a. RE to represent all valid mobile numbers

rule:- (1) mobile No contains 10 digits

(2) the first digit should be 7 to 9

RegEx:- [7-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]

(or)

[7-9]{\d{9}

③ W.a. regular Expression to represent all valid mail_ids?

rules:

(1) The set of allowed characters in mail-id are 0 to 9, a-z, A-Z .,_

(2) should starts with alphabet symbol

(3) should contain atleast one symbol.

RegExp:- [a-zA-Z][a-zA-Z0-9._]* @ [a-zA-Z0-9]+ ([.][a-zA-Z]+)+

RegExp:

" @gmail[.]com

" @(gmail|yahoo|hotmail)[.]com

Ex:-

```
import java.io.*;
import java.util.regex.*;
class MobileExtractor
{
  P.S.v.m(String[] args)throws IOException
  {
    PrintWriter pw = new PrintWrite("mobile.txt");
    BufferedReader br = new BufferedReader(new FileReader("input.txt"));
String line = br.readLine();
Pattern p = Pattern.compile("[7-9][0-9]{9}");
while (line != null)
{
  Matcher m = p.matcher(line);
  while(m.find())
  {
    pw.println(m.group());
  }
  line = br.readLine();
 }
 pw.flush();
 }
}
```

P) W.a.p to Extract mail-ids from the given file where mail-ids are mixed with some raw data ?

→ In the above Example replace regular Expression with the following mail-id regular Expression.

```
[a-zA-Z][a-zA-Z0-9._]*@[a-zA-Z0-9]+([.][a-zA-Z]+)+
```

P) W.a.p to display all text files present in the given directory ?

```
import java.io.*;
import java.util.regex.*;
class FileNameExtractor
{
```

```
public Static void main (String[] args) throws IOException
 {
   int Count = 0;
   Pattern p = pattern.Compile("[a-zA-z0-9_]+[.]txt");
   File f = new File ("D:\\durga_classes");
   String[] S = f.list();
   for (String Si : S)
    {
      Matcher m = p.matcher(Si);
      if(m.find() && m.group().equals(Si))
       {
         Count ++;
         S.o.pln(Si);
       }
    }
    S.o.pln (Count);
   }
 }
```

P) w.a.p to delete all .bak files present in D:\\durga-class

```
   import java.io.*;
   import java.util.regex.*;
   class FileNamesDeleter
    {
     p.s.v.m(String[] args) throws IOException
      {
       int Count = 0;
      Pattern p = pattern.compile("[a-zA-Z0-9_.]+[.]bak");
```

```
File f =new File("D:\\durga_classes");
String[] s = f.list();
for(String s1:s)
{
    Matcher m = p.matcher(s1);
    if(m.find() && m.group().equals(s1))
    {
        Count++;
        S.o.pln(s1);
        File f1 =new file(f, s1);
        f1.delete();
    }
}
S.o.pln(Count);
}
}
```

=×=

Enumeration (enum)

15/5/11

→ we can use enum to define a group of named constants

Ex:- ①
```
enum month
{
    JAN, FEB, MAR ----- DEC(;) → optional.
}
```

②
```
enum Bear
{
    KF, KO, RC, FO(;) → optional
}
```

→ By using enum we can define our own datatypes

→ enum concept introduced in 1.5v.

→ when compared with old languages enum Java's enum is more powerfull.

Internal implementation of enum :-

→ enum concept internally implemented by using class concept.

→ every enum constant is a reference variable to enum type object.

→ every enum constant is always public static final by default.

Ex:-
```
enum Month
{
  JAN, FEB, ---- DEC;
}
```

```
class month
{
    public static final month JAN = new Month();
    public static final Month FEB = new Month();
    |
    public static final month DEC = new Month();
}
```

Month.JAN

Declaration and usage of enum :-

Ex:-

```
enum Beer
{
 KF, KO, RC, FO;
}
Class Test
{
 p.s.v.m(String[] args)
 {
  Beer b1 = Beer.KF;
  S.o.pln(b1); // KF
 }
}
```

→ we can declare enum either with in the class or outside of the class but not inside a method.

→ if we are trying to declare enum with in a method we will get Compiletime Error.

Ex:-

```
enum X
{
}
Class Y
{
}
✓
```

```
Class Y
{
 enum X
 {
 }
}
✓
```

```
Class Y
{
 public void m1()
 {
  enum X
  {
  }
 }
}
✗
```

C.E.- enum types must not be local

→ if we declare enum outside the class the applicable modifiers are public, default, strictfp.

→ if we declare enum with in a class the applicable modifiers are public, default, strictfp, private, protected, static.

enum Vs Switch Statement :-

→ until 1.4v the allowed datatypes for switch arguement are byte, short, char, int.

→ But from 1.5v onwards in addition to above the corresponding wrapper classes ~~I~~ Byte, short, character, Integer, + enum type also allowed

```
Switch ( )
{
}
```

1.4 V	1.5 V	1.7 V
byte short char int	Byte Short character Integer + enum	String

→ Hence from 1.5 version onwards we can use enum as arguement to switch statement.

Ep:-

```
enum Beer
{
  KF, KO, RC, FO;
}
class Test
{
```

```
p.s.v.m(——)
{
 Beer b1 = Beer.Rc;
 Switch (b1)
 {
   Case KF:
        S.o.pln(" It is childrens brand);
        break;
   Case KO:
        S.o.pln(" It is too lite");
        break;
   Case Rc:
        S.o.pln(" It is challengers brand");
        break;
   Case FO:
        S.o.pln(" Buy one get one");
        break;
   default:
        S.o.pln(" other brands not recommended to
                                        take");
   }
  }
 }
```

o/p:- It is challengers brand.

→ If we are passing enum type as arguement to Switch Statement every Case label should be a valid enum Constant.

eg:-
```
enum Beer
{
 KF, KO, RC, FO;
}
Beer b1 = Beer.KF;
Switch(b1)
{
 Case KF: ✓
 Case KO: ✓
 Case RC: ✓
 Case KALYANI: ✗ C.E: unQualified Enumeration Constant name
                       required
}
}}
```

enum Vs Inheritance :-

→ every enum in Java is direct child class of java.lang.Enum

→ As every enum is always extending java.lang.Enum there is no chance of extending any other enum(because java won't provide Support for multiple inheritance).

→ As every enum is always final implicitly we can't create child enum for our enums.

→ because of above reasons we can conclude inheritance concept is not applicable for enums Explicitly.

→ But enum can implement any no. of interfaces at a time.

Ex!- ①

```
enum X
{
}
enum Y extends X
{
}
```

X

C.E:.

Cannot inherit from final X

enum types are not extensible

②

```
enum X extends java.lang.Enum
{

}
```

X

C.E:-

③

```
enum X
{

}

Class Y extends X
{
}
```

X

C.E1:- Can not inherit from final X

C.E2:- enum types are not extensible

④

```
Class X
{
}
enum Y extends X
{
}
```

X

C.E!-

⑤

```
interface ~~X~~
{
}
enum Y implements X
{
}
```

✓

java.lang.Enum :-

↦ every enum in java ~~should~~ is always direct child class of java.lang.Enum class.

↦ The power of enum is inheriting from this class only to our enum classes.

(Javap java.lang.Enum)

↦ It is an abstract class & direct child class of Object class.

↦ This class implements Comparable & Serializable interfaces. hence every enum in java is bydefault Serializable and Comparable.

values() method :-

→ we can use values() method to list out all values of enum.

```
eg. Beer[ ] b = Beer.values();
```

ordinal() method :-

→ with in the enum the order of constants is important we can specify its order by using ordinal value.

→ we can find ordinal value of enum constant by using ordinal method.

```
        public int ordinal();
```

→ Ordinal value is zero-based.

Ex:-

```
        enum Beer
        {
          KF, KO, RC, FO;
        }
```

```
class Test
{
  p.s.v.m(String[] args)
  {
    Beer[] b = Beer.values();
    for(Beer b1 : b)
    {
      S.o.pln(b1+"----"+b1.ordinal());
    }
  }
}
```

O/P :-
```
KF ---- 0
KO ---- 1
RC ---- 2
FO ---- 3
```

Enum class Constructors & Speciality of Java enum :-

→ when Compared with old languages enum, Java enum is more Powerful because in addition to Constants we Can take variables, methods, Constructors e.t.c... which may not possible in old languages. This extra facility is due to internal implementation of enum concept which is class based.

→ Inside enum we can declare main() method & hence we Can invoke enum class directly from Command prompt.

ex:-
```
enum Fish
{
  STAR, GOLD, GUPPY, APOLLO, KILLER;  ↳ mandatory.
  p.s.v.m(String[] args)
  {
    S.o.pln("ENUM MAIN METHOD");
  }
}
```

```
> Javac Fish.java
> Java Fish
```

o/p:- Enum main method.

→ In addition to constant if we want to take any extra members compulsory list of constants should be in the 1st line & should ends with ";"

ex:-

①
```
enum Color
{
RED, GREEN, BLUE
Public void m1()
{
}
}
```
✗

②
```
enum Color
{
Public void m1()
{
}
RED, GREEN, BLUE;
}
```
✗

③
```
enum Color
{
RED, GREEN, BLUE;
public void m1()
{
}
}
```
✓

④
```
enum Color
{
Public void m1()
{
}
}
```
✗

⑤
```
enum Color
{
}
```
✓

→ Inside enum with out having constant we can't to take any extra members, but Empty enum is always valid.

ex:

```
enum Color
{
public void m1()
{
}
```
✗

```
enum Color
{
}
```
✓

Enum class Constructors :-

→ with-in Enum we can take Constructors also.

→ Enum class Constructors will be executed automatically at the time of Enum class loading. ~~Here~~ because Enum Constants will be Created at the time of Class loading only.

→ We Can't invoke Enum Constructors explicitly

Ex:-

```
enum Beer
{
  KF, KO, RC, FO;
  Beer()
  {
    S.o.pln("Constructor");
  }
}
Class Test
{
  Public static void main(String[] args)
  {
    Beer b1 = Beer.KF;
    S.o.pln(b1);
  }
}
```

o/p:-

```
Constructor
Constructor
Constructor
Constructor
KF
```

→ We Can't Create Objects of Enum explicitly & hence we Can't Call Constructors directly.

```
Beer b = new Beer() ;    X
```

C.E:-
Enum types may not be instantiated.

Ex:-

```
enum Beer
{
KF(75), KO(90), RC(90), FO;
int price;
Beer(int price)
{
 this.price = price;
}
Beer()
{
 this.price = 65;
}
Public int getPrice()
{
 return price;
}}
class Test
{
p.s.v.m(String[] args)
 {
 Beer[] b = Beer.values();
 for(Beer b1 : b)
 {
 S.o.pln(b1+"......"+b1.getPrice());
 }
}}
```

KF ⇒ p.s.f. Beer KF = new Beer();
KF(100) ⇒ p.s.f. Beer KF = new Beer(100);
KF(100, "Gold", "Bitter")
⇒ p.s.f. Beer KF = new Beer(100, "Gold", "Bitter")

o/p:-

```
KF ---- 75
KO ---- 90
RC ---- 70
FO ---- 65
```

→ Within the enum we Can take instance & Static methods but we Can't to take abstract methods

→ every enum Constant represents an object hence what ever The methods we Can apply on ~~Enum~~ normal Java object we Can apply those on enum Constants also.

ex:-

Q) which of the following Expressions are valid

✓ ① Beer.KF.equals(Beer.RC) // False

✓ ② Beer.KF.hashCode() // ✓

✓ ③ Beer.KF~~.Beer~~ == Beer.RC ⟶ false

✗ ④ Beer.KF > Beer.RC

✓ ⑤ Beer.KF.ordinal > Beer.R~~C~~.ordinal

Case(1):-

```
Package Pack1;
public enum Fish;
{
 STAR, GUPPY, APOLLO;
}
```

```
Package Pack2;
Class Test1
{
 p.s.v.m(___)
 {
  S.o.pln(STAR);
 }
}
```

import static pack1.Fish.STAR;
(or)
import static pack1.Fish.*;

```
Package pack3;
---
Class Test2
{
 p.s.v.m(---)
 {
  Fish f = Fish.STAR;
  S.o.pln(f);
 }
}

import pack1.Fish;
   (or)
import pack1.*;
```

```
package Pack4;
---
class Test3
{
 p.s.v.m(------)
 {
  Fish f = Fish.STAR;
  S.o.pln(Guppy);
 }
}

import pack1.Fish (or)
     import pack1.*;

import static pack1.Fish.Guppy;
         (or)
import static pack1.Fish.*;
```

Case2 :-

```
enum Color
{
  BLUE, RED
  {
    public void info()
    {
      S.o.pln(" Dangerous Color");
    }
  }, GREEN;
  public void info()
  {
   S.o.pln(" universal Color");
  }
}
```

```
class Test
{
  p.s.v.m(———)
  {
    Color[] c = ~~new~~ Color.values();
    for (Color c1 : c)
    {
      c1.info();
    }
  }
}
```

O/P :-
```
universal Color
Dangerous Color
universal Color.
```

enum Vs Enum Vs Enumeration :-

enum :-

→ It is a Keyword which Can be used to define a group of named Constants.

Enum :-

→ It is a class present in java.lang package which acts as a base class for all Java enums

Enumeration :-

→ It is an Interface present in java.util package, which Can be used for retrieving objects from Collection one by one.