Page :
Date : / /

08/11/19

Q→ WAP to print total number of unique characters in your name

Soln

```
class UnqChar
{
    public static void main (String[] args)
    {
        n = "Kardam";
        String[] arr = new String(n);
```

1) create a HashSet to store characters.
2) Add first charac. to HashSet.
3) Repeat step 3 for length of name.
4) Print the size of the set.

```
import java.util.HashSet;

class UnqChar {

    psvm (s→) 0
    {
        Hashset <String>

import java.util.HashSet;

class UnqChar
{
    public static void main (String[] args)
    { n = "Kardam";
      HashSet <Character> set = new HashSet<>();
```

Page :
Date : / /

```
StringBuilder str = new StringBuilder(n);

for(int i=0; i<str.length(); i++)
{
    set.add(charAt(i));
}

System.out.println(set.size());

}
```

↳ Ignore the case of character

* If you want to create the ~~trash~~ Hash Set of your custom class then it will be your responsibility to provide methods to check for equality.

* override equals method. It returns boolean value.
* It accepts object of class.
* Set ke underlying mein map hota hai. It stores the values as key - value pair.
* If the objects are same then key must be same.

* If two objects are equal then their hash code must be equal.
* You can get hash code of any object by calling hashCode method of Object class.

Page :
Date : / /

* If you are overriding equals you must override hashCode method.

* hashCode method will return an integer and will not accept any argument.

* To generate an hashCode of any object you can use hash() method defined in the Objects class. This method will accept variable length argument and you can pass all the fields that you used for testing equality in your equals method.

* To apply sorting we have to implement Comparable or Comparator and override compareTo or compare.

Ex.
```
class Name
{
    private String firstName;
    private String lastName;

    Name(String fistName, String lastName)
    {
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```

Page :
Date : / /

* instanceOf is an operator which will return true if the given reference contains the object of given type.

```
class xyz
{
    psvm()
    {
        Name obj = new Name("Ramesh", "Sharma");
        Name obj2 = new Name("Ramesh", "Sharma");

        Set<Name> s = new HashSet<>();

        s.add(obj);
        s.add(obj2);
    }
}
```

St will add both the objects into the set s because

* Comparison of objects is done on the basis of hashCode of the object and their values stored in that obj.

```
class Name
{
    private String firstName;
    private String lastName;

    Name (String firstName, String lastName)
    {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Override
    public boolean equals (Object o)
    {
        if (o instanceOf Name)
        {
            Name obj = (Name) o;

        if (firstName.equals (obj.firstName)
            && (lastName.equals (obj.lastName)))
        {
            return true;
        }
    }
    public int hashCode()
    {
        return hashCode (firstName, lastName);
                    or
        return firstName.hashCode()
                + lastName.hashCode();
    }
}
```

Arrays.toString(arr);

Thursday
14/11/19

Page :
Date : / /

Regular Expression

A regular expression defines a search pattern for strings. The search pattern can be anything from a single character to a complex expression containing special characters defining the pattern.

Regular expressions are used to manipulate text or to search ~~and added~~ or edit any string.
You can also use regular expressions for analysing any given string or text.

For eg.
You can use regular expressions to perform some of the operations on the string by using the methods defined in the String library.
Like, you can use split() function which accepts a regular expression in form of string and will split the given text on the basis of supplied regular expression.
i.e., `split(String regex);`

This function will return an array of string containing seperated strings on the basis of supplied expression.

Ex.1

```
String text = "Hello World";
String regex = " ";
String[] arr = text.split(regex);
```

Output → [Hello, World] ← arr.

Ex. 2 String text = "abcd abcd abcd"

String regex = "b";

String[] arr = text.split(regex)

output → [a, cd a, cd a, cd] e arr.

Q→ WAP to print total no. of words contained in a given text.

*Complex regex.

* In java you can use regex package for implementing regular expression.
This regex package contains two classes, —

(i) regex.Pattern ; } under regex.
(ii) regex.Matcher ; } → engine to check regular expression

* where Pattern class is used to compile any regular expressions as an object of Pattern class.

* Matcher class is used to perform search operation on any given text on the basis of compile Pattern.

* regex is a sub-package of java.util introduced in JDK 1.4 and updated in JDK 1.8 and so on.
i.e., java.util.regex.

Example

→ You can match any regular expression or a pattern in the given string by using matches() method of String class which also accepts a regular expression and

returns a boolean value, i.e, syntax

boolean matches (String regex);

```
String text = "http//:www.example.com";
String regex = "http";
Sopln (text.matches(regex));
```

output → false

matches → it uses exact match, i.e, both the strings & regex must be same.

There are many simple ways to write a regular expression:-

1) By supplying a string literal. You can supply a single character or a sequence of characters/multiple characters as a regular expression but it will match exactly the same sequence of characters.

2) You can create complex expressions by using or supplying meta characters which have special meaning while performing the search in the given text.

meta character → used to create Character class.

Ex: "[http]"

→ P-T-O

Page :
Date : / /

String text = "http//:www.example.com";
String regex = "[http]+.*"; ← complex expression
Sopln (text.matches(regex));

Output → True.

This expression will match anything followed by "http".
For eg:
If your text contain "my web link is http//:www.example.com" then it will also match.

For eg.
If you want to check whether the given name is valid or invalid.

valid name → anything which starts with a character & it does not contain spl. character

All the valid name starts with a character
To check whether a given text is starting with a character, ~~Carat (^)~~ we will ~~you~~ use another meta character with our regular expression.

Carat (^)

Ex:

→ P-T-O.

Page :
Date : / /

String text = "Vaibhav";
String regex = "^[a-z A-Z]" ← valid name.
group
System.out.println(text.matches(regex));

Character class → [abc]

complex expression → ^[abc] • ← contains. (text)

negation → [^abc] ← does not contain. (text)

5/11/19

* You can create complex regular expressions by defining set of characters. To define set of characters you can use [abc] and can place any no. of characters within the square bracket.

[abc], this expression can match a, b or c in the given text.

Ex. [abc]
String text = "Hello how are you";

* You can also define range of characters by providing the starting point and the end point of the given range seperated by '-' within the square brackets.

Ex.
[a-z].

Page :
Date : / /

Ex:

```
String text = "a";
String regex = "[abc]";
boolean result = text.matches(regex);
```

output → result will store true.

~~You can~~

Ex:

```
String text = "B";
String regex = "[a-zA-Z]";
boolean result = text.matches(regex);
```

* You can define a set for accepting the given characters by using ~~char~~ carat (^) symbol within the square brackets.

Ex:

```
String text = "1";
String regex = "[^a-zA-Z]";
boolean result = text.matches(regex);

// result → true.
```

* You can define a complex expression by using alterations. for defining alternations you can use pipeline symbol inbetween the characters

Ex: (a|b)

* Alternations must be enclosed within the

parenthesis.

* You can choose any one of the given alternations. The alternations can be a single character or multiple characters.

Ex.

"(a|b)";

String text = "abc";
String regex = "(abc|xyz|pqr)";
boolean result = text.matches(regex);
result ← true.

Short hands for commonly used regex patterns

You can use predefined short hands :-

(i) \\d :- to match any digit. → [0-9]

(ii) \\D :- to match any non-digit → [^0-9]

(iii) \\s :- to match any white space character including newline and tab

(iv) \\S :- for a non-white space character

(v) \\w :- for a alpha-numeric character
For matching a alpha-numeric character which containing alphabet or number.

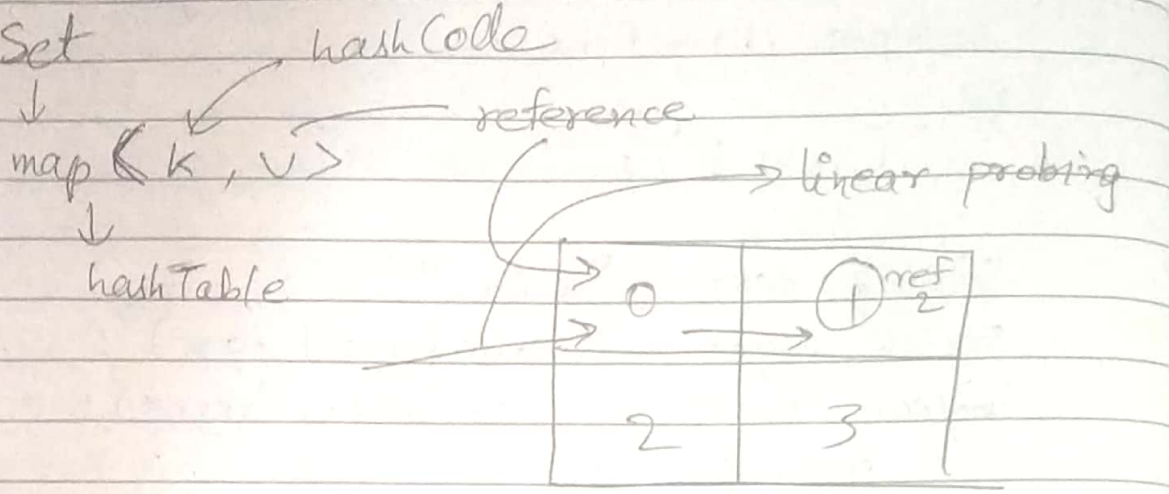
(vi) \\W :- to match a non-alpha numeric character.

OOPS LAB

16/11/19

Page :
Date : / /

Q> WAP to show implementation of a set using your own custom class.



Collection ← interface
Collections ← class

Objects class → hash method ← static.
equals method to check equality of two objects

```
public boolean equals (Object o)
{
    if (this == o) return true;
    if (!(o instanceof Student)) return false;
    Student s = (Student) o;
```

Ex:

Variable length argument → ... (ellipses)

Page :
Date : / /

```
class Book {
    private long isbn;
    private String bookName;
    private String authorName;

    // setter getter
    // constructor          // hashcode
    // to string            // equals.

int hash(Object ... args)
```

public ← it must be public, because in parent class it is defined as public, we can't narrow down.

(I) generate

```
int hashCode()
{
    return Objects.hash(isbn, bookName);
}
```

variable length arguments are used to remove function overloading.

Ex of use :

```
public static void main(String ... args)
{
    System.out
```

* It must be the last argument of a function.

Ex:

```
int add(String name, int ...args);   ✓

int add(int ...args, String name);   ✗
```

* Linear probing
object.getClass(); or
instance of operator

when two objects are in same memory location then they are equal.

Date : / /

equals method :

current object

(II) public boolean equals(Object o)
{
if(this == o) return true;

if(o instanceof Book)
{Book obj = (Book) o;} can be accessed in same class

now the fields of Book class can be accessed.

if(isbn == obj.isbn && bookName.equals(obj.bookName)
&& authorName.equals(obj.authorName))
{
return true
}
else
{
return false;
}
}
else {
return false;
}
}

{fall through the order ';' operated values

7/11/19 wamp server
local host

Page :
Date : / /

8/11/19 OOPS Lecture

Quantifiers

? → ~~one or~~

Quantifiers allow you to check ~~whether~~ one or more occurence of ~~a~~ given expression.

? → one or not at all
* → zero or more
+ → once or more times
{n} → exact n times
{n,} → atleast n times
{n, m} → at least n, atmost m

Valid name

* Any name starting with character followed by space.

Ex. $([a\text{-}zA\text{-}Z]+[\ ]?)+$ ~~at least once~~

(space)

To process the regular expression in java we will use Pattern class & the Matcher class. These classes are defined in package → java.util.regex;

Pattern class

Pattern class is used to compile any given regular expression as an object of Pattern class.
To create an object we will use —

ClassName.methodName → calling of Matcher method.

Page :
Date : / /

- static compile method of Pattern class which accepts character sequence.
  - i.e., static Pattern compile (CharSequence string) ← or regex
    - string → variable name

* Some methods of Pattern class

(i) Matcher matcher (CharSequence text);

Returns object of Matcher class & accepts text on which you want to match the given pattern.

(ii) String[] split (CharSequence text);

It accepts one argument of type String and returns an array of string.
This method will split the given text on the basis of compiled pattern.

(iii) static boolean matches (String regex, String text);

It would be called by object of Pattern class.
For eg.
If you have a regular expression
([a-zA-Z]+[ ]?)+ and input text your name → Kardam Pandey
then,

Patter.matches (regex, input);

this will always return true.

If regex = "A" & input = "A" then it will also return true (i.e., exact match)

Matcher class

We will use Matcher class to find all the matched sub string. Matcher class does not provide any public constructor so we will use matcher method of Pattern class to create object of Matcher class.

The generated Matcher object will be used to process the match of the pattern and the given (matched) text. Matcher object can give you all the substring by the method →

→ find which will return true if any next subsequence is find.

```
boolean find();
```

to print the matched subsequence you can use another method → group() method which will return String,

```
String group();
```

group() method is overloaded it can also accept number of groups to be printed, i.e., an int value.

Ex.

```
String group(5);
```

Page :
Date : / /

Steps to process regular expression in java

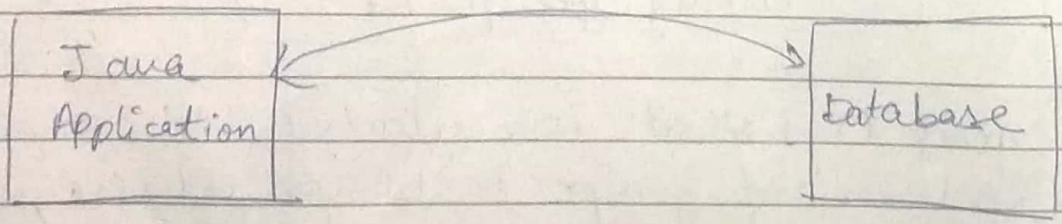
```
import java.util.regex.*;

class RegexDemo
{
    public static void main(String[] args)
    {
        String regex = "([a-zA-Z]+[ ]?)+";
        Pattern p = Pattern.compile(regex);
        Matcher m = p.matcher("Vaibhav Diwan");
        while(m.find())
        {
            System.out.println(m.group());
        }
    }
}
```

OOPS LAB

25/11/19 JDBC → Java Database Connectivity

JDBC API provides you to connect to a data source. Data source can be any relational database management system or a spread sheet or to a flat file system.



Step 1 → Load the Driver

we use pure java driver

Page :
Date : / /

To connect to a data source you must load the driver for the data source. Drivers are provided by the data source vendor.
Drivers are of different type

(i) Type 1 : Drivers that implements the JDBC API as a mapping to another data access API such as ODBC (Open Database Connectivity). These drivers are also known as JDBC-ODBC bridge. These drivers are generally dependent on native libraries

(ii) Type 2 : Drivers that are written partially in java programming language and partially in native code. These driver use a native client library specific to the data source to which they connect.
part java part native.

(iii) Type 3 :- Drivers that use a pure java client and a middleware to communicate with the database. The middleware can be server that communicates the java client request to the data source.

(iv) Type 4 :- pure java drivers
Drivers that are pure java and implement the network protocol for a specific data source. The java client connects directly to the data source

Note:- Check which of the type are depricated.

Please download MySQL J connector version 8.

Step 2:- Establish a connection.

To Establish a connection with the data source JDBC API typically provides you two classes

→ DriverManager :- A class that is a fully implemented class connects an application to a data source which is specified by database URL. When this class first attempts to establish a connection it automatically loads any "JDBC 4.0 drivers" found within the class path. It is your responsibility to add the driver jar file in the class path of your application.

→ You will use an static getConnection method which accepts three arguments → first is database URL, second is the username and third is the password All the arguments are string argument.
URLs can be different for different data source as well as URL is also depend on the type of driver you are using.

MySQL → localhost:3306/databaseName
oracle uses 1521
ip address will be provided of the database server in space of URL

* getConnection, method an interface returns, object of Connection

Connection is an interface with some concrete implementation like OracleConnection.
it is in java.sql package.

connector 5 and connector 8 i.e 1.8 or above

Step 3: Prepare the statement

Page :
Date : / /

(Create, retrieve, update, delete)

You can prepare a statement by using Statement, PreparedStatement, CallableStatement
stored procedure

* Statement is defined under java.sql package.
You can create object of any of these by using the Connection object.

Once the Statement object is ready you can either execute the query or you can execute the update
executeUpdate
executeQuery

executeUpdate will return no. of rows affected and executeQuery will return object of ResultSet which contains one or more rows of data.

Step 4 Process the ResultSet

ResultSet is an heavy - object.

Step 5 Close the connection.
It is always better to close the connection so other users can use the connection.

connection Object. close.

Task Create a Student class which contains 3 fields. Create a database corresponding to the given student class.

Page:
Date: / /

Insert detail of 5 student in the database and display the complete database. Create a menu driven program which provides two operations → add a new data and display all.

util ← this contains static method.

- Connection
- Data Access Object ← dao
  (DAO)
- main ← main
- Helper ← helper
- Student ← databean.

~~JDBC~~ jdbcTask1

download: for MySQL ← j connector → 8
ojdbc.jar ←

~~import java.sql.Connection~~ import java.sql.Statement;
import java.sql.DriverManager;
import java.sql.Connection;

Class Task1
{
  ps vm
  {
    // Step 1 load driver.
try { Class.forName ("com.mysql.jdbc.Driver");
("com.mysql.cj.jdbc.Driver");
("oracle.jdbc.driver.OracleDriver");

mysql + connector
/ JS
/ JS

for oracle database connectivity.

→ Class.forName will throw ClassNotFoundException
→ getConnection method will throw an exception called SQLException defined in java.sql package.

Class.forName method is used for loading and registering the driver. If we are not using this then we have to use DriverManager's class static method registerDriver to register the driver of database.

```
//Step 2 establish Connection.
Connection con = DriverManager.getConnection
("jdbc:mysql://localhost:3306/demostudent", "root", "");

("jdbc:oracle:thin:@localhost:1521:xe", "un", psswd)

//step3 prepare statement

Statement stmt = con.createStatement();

// To create a statement createStatement method
// of Connection class that will throw SqlException.

String query = "select * from student";

// we will use executeQuery method to run
// the select query. ← retrieval
// we will use executeUpdate method to run
// insert or update query. ← insertion

// Both these method accept String query as
// an argument, and will throw SQLException.
// executeQuery will return ResultSet object.
```

// executeUpdate will return integer value, and
// throw SQLException.

ResultSet rs = stmt.executeQuery(query);

mysql 5 connector.
if cj not → j5
connector / j8 (Download)

```
import java.sql.*;
class DemoJDBC {
    psvm() {
        try {
            Class.forName("com.mysql.cj.jdbc.Drivers");
            Connection con = DriverManager.getConnection
            ("jdbc:mysql://localhost:3306/demo", "root", "");

            Statement stmt = con.createStatement();
            String sql = stmt
            String sql = "INSERT INTO Student (ROLLNO, NAME, CPI)
                          VALUES (12, 'Ramesh', 5-73)";

            int rows = stmt.executeUpdate(sql);

            Sopln("no. of rows affected" + rows);
        }

        catch (ClassNotFoundException e)
            {
            }

        catch (SQLException e)
            {
            }
}
```

Ctrl + Alt + Shift + S ← Project Structure
↓
Library
↓
blank window (+)
↓
java
↓
apply and close

java.lang

Page: / Date: / javac -classpath jarName executableClassName

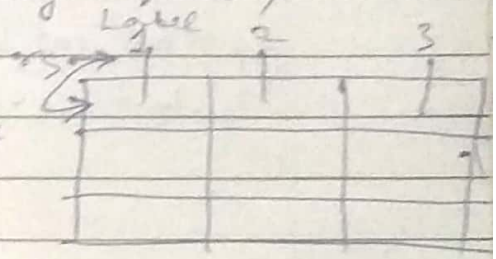
java -classpath jarName executableClassName

→ Modify the given program to display all the records from the student table.

☆ Modified: (a) Above Program

```
String sql1 = "SELECT * FROM Student";

ResultSet rs = stmt.executeQuery(sql1);
```



// boolean next() → (i) returns true if next row is present
(ii) points to that row.

```
while (rs.next())
{
    int rollno = rs.getInt(1);
    String name = rs.getString(2);
    double cpi = rs.getDouble(3);
    System.out.println(rollno + "\t\t" + name + "\t\t" + cpi);
}
```

← index or ("ROLLNO");

a

Q→ WAP to demonstrate, connection with mysql database. Print true if you are successful in creating a connection. Modify the above program to create connection with oracle, database.
Assumption should be mentioned.

Prepared Statement

Prepared Statement is used to build query with the changeable values.

Page :
Date : / /

For eg. If you want to insert 10 records in Student table and all the values are coming from the user input then you can build the query with a prepared statement.

Eg.

String query = "INSERT INTO Student (ROLLNO, NAME, CPI) VALUES (?, ?, ?)" ← parameter no.

Parameters.

PreparedStatement pstmt = con.prepareStatement(query);

pstmt.setInt(parameter index, value);

pstmt.setInt(1, sc.nextInt(S));

or

rollNo

or

student.getRollNo();

pstmt.setString(2, name);

pstmt.setDouble(3, CPI);