Hafiz Ayuba

Final Project

import java.util.Scanner;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

class queBlockNode

{

public String pswd;// holds the password

public String secret; // holds the secret word

public int en\_Word; // holds the encrypted secret word

public queBlockNode link; // link to connect the next node

}

public class Password\_Class {

static Scanner scanner = new Scanner(System.in);

static queBlockNode head[] = new queBlockNode[4];

static queBlockNode current[] = new queBlockNode[4];

static queBlockNode hold[] = new queBlockNode[4];

public static void main(String[] args) {

for(int i = 0 ; i < 4; i++)

{

head[i] = null;

current[i] = null;

hold[i] = null;

}

while(true)

{

System.out.println("1. Create Password");

System.out.println("2. Reset my password");

System.out.println("3. Print the information in particular queBlock");

System.out.println("4. Print the information in all queBlocks");

System.out.println("5. Stop");

String choice = scanner.nextLine();

switch(choice)

{

case "1":

createPassword();

break;

case "2":

System.out.println("Enter secret word: ");

String secret = scanner.nextLine();

int hash = getHash(secret);

changePassword(hash % 4, secret);

break;

case "3":

System.out.println("Enter block number: ");

int temp = Integer.parseInt(scanner.nextLine());

print(temp);

break;

case "4":

print();

break;

case "5":

System.exit(0);

break;

default:

System.out.println("Wrong choice!");

}

}

}

public static void changePassword(int number, String secret)

{

queBlockNode temp = head[number];

boolean found = false;

while(temp != null)

{

if(temp.secret.equals(secret))

{

found = true;

int i = 0;

System.out.println("Enter password");

String password = scanner.nextLine();

if(!validPassword(password))

{

while(!validPassword(password) && i < 2)

{

System.out.println(head[number].pswd);

System.out.println("Enter password");

password = scanner.nextLine();

temp.pswd = password;

i++;

}

if(i >= 2)

{

System.out.println("Password can't be changed!");

found = false;

}

}

else

{

temp.pswd = password;

}

}

temp = temp.link;

}

if(!found)

{

System.out.println("Secret word not found\n\n");

}

}

public static void print(int number)

{

if(number > 3)

{

System.out.println("Number should be 0 - 3");

return;

}

queBlockNode temp = head[number];

while(temp != null)

{

System.out.println("Password: " + temp.pswd);

System.out.println("Secret word: " + temp.secret);

System.out.println("Encrypted secret word: " + temp.en\_Word);

System.out.println();

temp = temp.link;

}

System.out.println();

}

public static void print()

{

for(int i = 0; i < 4; i++)

{

System.out.println("QueBlock " + i + "\n");

queBlockNode temp = head[i];

while(temp != null)

{

System.out.println("Password: " + temp.pswd);

System.out.println("Secret word: " + temp.secret);

System.out.println("Encrypted secret word: " + temp.en\_Word);

System.out.println();

temp = temp.link;

}

}

}

public static int getHash(String secret)

{

int hash = 0;

for(int i = 0 ; i < secret.length(); i++)

{

switch(secret.charAt(i))

{

case 'a':

case 'A':

hash += 1;

break;

case 'b':

case 'B':

hash += 2;

break;

case 'c':

case 'C':

hash += 3;

break;

case 'd':

case 'D':

hash += 4;

break;

case 'e':

case 'E':

hash += 5;

break;

case 'f':

case 'F':

hash += 6;

break;

case 'g':

case 'G':

hash += 7;

break;

case 'h':

case 'H':

hash += 8;

break;

case 'i':

case 'I':

hash += 9;

break;

case 'j':

case 'J':

hash += 10;

break;

case 'k':

case 'K':

hash += 11;

break;

case 'l':

case 'L':

hash += 12;

break;

case 'm':

case 'M':

hash += 13;

break;

case 'n':

case 'N':

hash += 14;

break;

case 'o':

case 'O':

hash += 15;

break;

case 'p':

case 'P':

hash += 16;

break;

case 'q':

case 'Q':

hash += 17;

break;

case 'r':

case 'R':

hash += 18;

break;

case 's':

case 'S':

hash += 19;

break;

case 't':

case 'T':

hash += 20;

break;

case 'u':

case 'U':

hash += 21;

break;

case 'v':

case 'V':

hash += 22;

break;

case 'w':

case 'W':

hash += 23;

break;

case 'x':

case 'X':

hash += 24;

break;

case 'y':

case 'Y':

hash += 25;

break;

case 'z':

case 'Z':

hash += 26;

break;

}

}

return hash;

}

public static void createPassword()

{

System.out.println("Enter password");

String password;

password = scanner.nextLine();

int i = 0;

while(!validPassword(password) && i < 2)

{

System.out.println("Enter password");

password = scanner.nextLine();

i++;

}

if(i >= 2)

{

return;

}

System.out.println("Enter secret word");

String secret = scanner.nextLine();

int hash = getHash(secret);

int node\_number = hash % 4;

queBlockNode queblock = new queBlockNode();

queblock.pswd = password;

queblock.secret = secret;

queblock.en\_Word = hash;

queblock.link = null;

if(head[node\_number] == null)

{

head[node\_number] = queblock;

hold[node\_number] = head[node\_number];

}

else

{

current[node\_number] = queblock;

hold[node\_number].link = current[node\_number];

hold[node\_number] = current[node\_number];

}

}

public static boolean validPassword(String password)

{

if(password.length() < 10)

{

System.out.println("The password must be atleast of lenght 10!");

return false;

}

else

{

Pattern inputRegexes[] = new Pattern[3];

inputRegexes[0] = Pattern.compile(".\*[A-Z].\*");

inputRegexes[1] = Pattern.compile(".\*\\d.\*");

inputRegexes[2] = Pattern.compile(".\*[`~!@#$%^&\*()\\-\_=+\\\\|\\[{\\]};:'\",<.>/?].\*");

Matcher matcher[] = new Matcher[3];

matcher[0]= inputRegexes[0].matcher(password);

matcher[1]= inputRegexes[1].matcher(password);

matcher[2]= inputRegexes[2].matcher(password);

if(!matcher[0].matches())

{

System.out.println("The password must contain Upper case alphabet!");

return false;

}

if(!matcher[1].matches())

{

System.out.println("The password must contain a digit!");

return false;

}

if(!matcher[2].matches())

{

System.out.println("The password must contain a special character!");

return false;

}

}

return true;

}

}