Section2:Task

1.a **package** AagyaUprety;

**import** java.util.Scanner;

**public** **class** Even {

**public** **static** **void** main(String[] args){

Scanner Input = **new** Scanner(System.***in***);//collecting the user input

System.***out***.print("Please Enter a number: ");//number has been set

**int** value = Input.nextInt();

**if**(value==0){

System.***out***.println(value+ " is neither odd nor even");

}

**else**{

**boolean** condition = *isEven*(value);// using boolean expression

**if** (condition){

System.***out***.println(value+" is even");//value is set as even

}

**else** {

System.***out***.println(value+" is odd");//value has been set as odd

}

}

}

**public** **static** **boolean** isEven(**int** value){//returning boolean worth addressed by predetermined

**if** (value%2==0){ //substantial boolean articulation that checks whether number %2 is comparable to 0

**return** **true**;// the expression is true

}

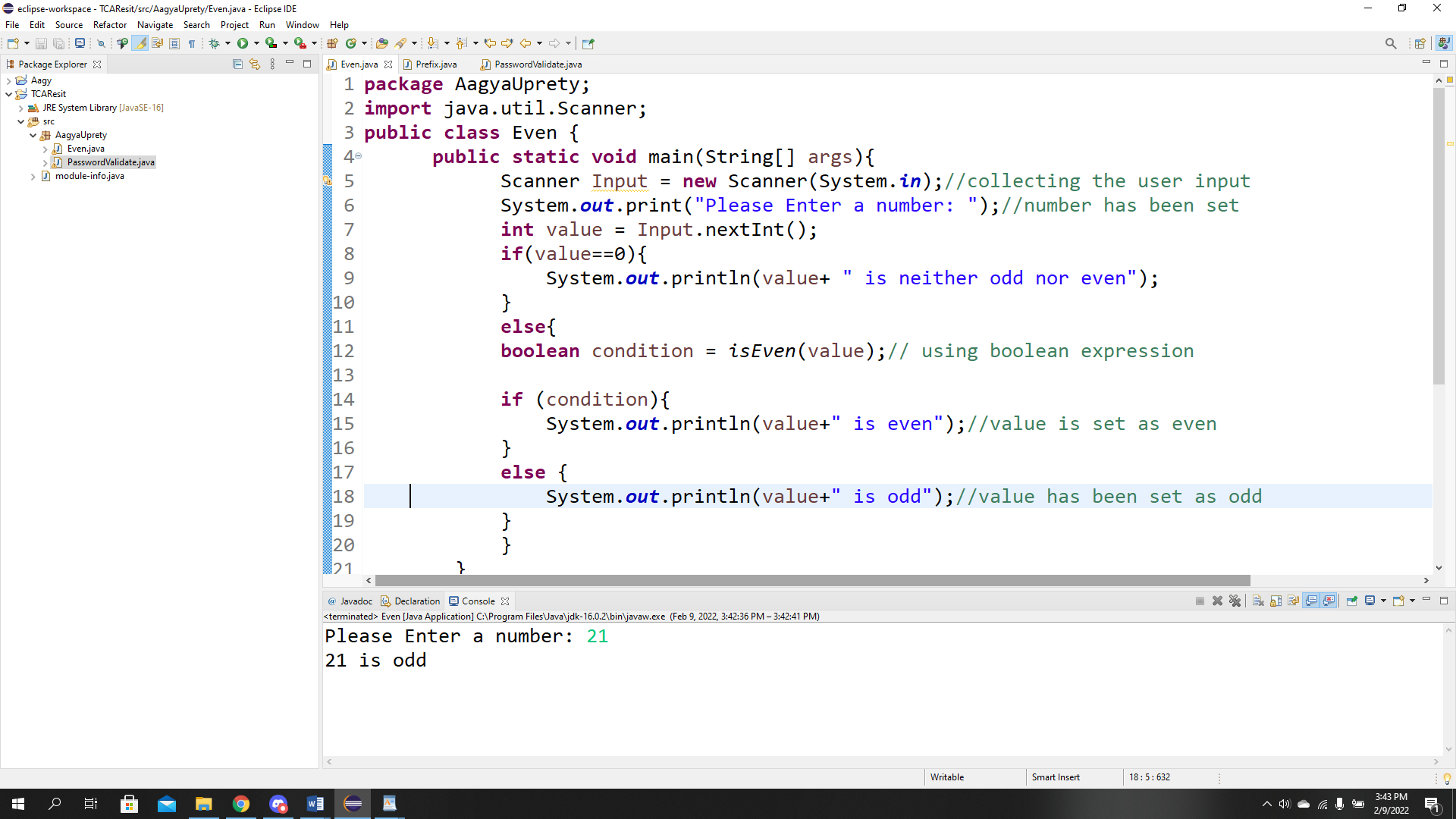
**else** {

**return** **false**;// expression is false

}

}

}



1.b. Differences:

|  |  |
| --- | --- |
| Prefix increment | Postfix increment |
| 1. The prefix structure initially plays out the addition activity and afterward returns the worth of the augmentation activity. | 1. The postfix form first returns the current value of the expression and then performs the increment operation on that value. |
| 2. The augmentation administrator ++ can be place before a variable whenever it is done, it is prefix | 2. The addition administrator ++ can be put behind a variable when it is done, that is postfix operators. |
| 1. Example :   Int a= 99;  Int b= ++a;  After the execution of these two assertions, both an a and b will have the worth of 100. | 1. Example   int a= 99;  int b=a++;  After the execution of these two assertions, a will have the worth of 100 and b will have the worth of 99. |
| 4. The pre-increase administrator is addressed as the twofold in addition to (++a) image, attached before the variable's name. | 4. Post-augmentation is an addition administrator, addressed as the twofold in addition to (a++) image followed by an administrator 'a'. |

Example of prefix increment:

Public class PrefixIncrement {

Public static void main(String[] args){

int a= 5;

int b =0;

System.out.println(“a value is “ + a);

System.out.println(“b value is “ + b);

b= ++a;

System.out.println(“a value is ” +a);

System.out.println(“b value is “ + b);

}

}

Example of postfix increment:

Public class PostfixIncrement {

Public static void main(String[] args){

int a= 5;

int b =0;

System.out.println(“a value is “ + a);

System.out.println(“b value is “ + b);

b= a++;

System.out.println(“a value is ” +a);

System.out.println(“b value is “ + b);

}

}

1.c.

Differences:

|  |  |
| --- | --- |
| Reference types | Primitive types |
| 1. It can be null. | It can’t be void. It generally has esteem. |
| 1. It holds protests that have information and strategy. | It holds a straightforward, indecomposable worth as single numbers and single characters. |
| 1. It starts with Uppercase letter . | It starts with lowercase letter. |
| 1. Java virtual machine designates 8 bytes for each reference variable, as a matter of course. | It’s size relies upon information type. |
| 1. Examples of it :   Class, Arrays, String etc. | Examples of primitive datatypes are int,float,double,Boolean etc. |

Examples of primitive datatype for Boolean expression:

public class Primitive{

public static void main(String args[])

{

boolean b= false

if (b == false)

system.out.println(“Hello PK”);

}

}

Example of reference type for string :

public String{

public static void main(String args[])

{

String s1=”AAU”;

String s2=”AAU”;

String s3=new String(“AAU”);

String s4=new String(”AAU”);

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

System.out.println(s4);

}

}

Task2

1. The output of the fragments is 26. In the beginning we can observe that println characterizes that the worth is written in all various lines and afterward in section we can see val1++ which is postfix administrators which show just the information prior to adding whatever other numbers which is 2 and too we can likewise see val2 where we val2++ is postfix so the worth is 6 and the option of those 8 and we can val1 where information is addition so the information is 3 and - val2 characterizes that information stays same which is 6 and in conclusive we can see (2+6+3\*6) which is complete of 26 as a result.

2.b. differences :

|  |  |
| --- | --- |
| While loop | Do while loop |
| 1. In while circle condition can be assessed first and afterward circle body is executed. | In do while circle body is executed first and afterward after the given condition is checked. |
| 2. No semicolon after the condition in the linguistic structure. | There is semicolon after the condition in the grammar. |
| 3. It permits introduction of counter factor prior to entering circle body. | It permits instatement of counter factor when entering circle body. |
| 4.Entry controlled loop is known for while loop | Exit controlled loop is known as do while loop. |
| 5. The circle body would be executed, provided that the given condition is valid. | The circle body would be executed once, regardless of whether the given condition is assessed as misleading. |

Example of while loop:

public class whileloop{

public static void main(String args[])

{

Int i= 1;

While(i< 8) {

System.out.println(“Aagya Uprety”)

i++;

}

}

}

Example of do while loop:

public class DoWhileLoop{

public static void main(String args[]args){

int arr[]={2,4,8,10,24};

int i=0;

do{

system.out.println(arr[i]);

i++;

}while(i<5);

}

}

2.c.

The circle body would be executed once, whether or not the given condition is evaluated as deceiving. A constructor instates an item when it is made. It calls a default constructor in the event that there is no constructor accessible in the class. It is called at whatever point you make an article utilizing new catchphrase. Constrictor empowers an item to introduce itself at the hour of its creation without the need to settle on a different decision to the occurrence strategy. It is on the grounds that java compiler makes a default constructor in the event that your class doesn't have any. In class article arranged programming, a constructor is a unique kind of subroutine called to make an item. It readies the new item for use, regularly tolerating contentions that the constructor uses to set required part factors.

Example of class constructor:

public class Main{

int a;

public main(int b) {

a=b;

}

Public static void main(String[] args) {

Main myObj = new Main(8);

System.out.println(myObj.x);

}

}

Task3 :

**package** AagyaUprety;

**import** java.util.Scanner;

**public** **class** PasswordValidate {

**static** String *error*= "";

**public** **static** **boolean** isValid(String password)//show that if the password is valid or not

{

**if** (!((password.length() >= 8)//to check the length of password

&& (password.length() <= 12))) {

*error* = "The length is not valid";

**return** **false**;

}

**if** (password.contains(" ")) { //for the gap check

*error* = "The password contains empty character";

**return** **false**;

}

**if** (**true**) {

**int** count = 0;

**for** (**int** i = 0; i <= 9; i++) { // whether the digits are from the 0 to 9

String str1 = Integer.*toString*(i);

**if** (password.contains(str1)) {

count = 1;

}

}

**if** (count == 0) {

*error* = "The password doesn't contain any numeric value";

**return** **false**;

}

}

// using of special character

**if** ((password.contains("@") || password.contains("#")

|| password.contains("!") || password.contains("~")

|| password.contains("$") || password.contains("%")

|| password.contains("^") || password.contains("&")

|| password.contains("\*") || password.contains("(")

|| password.contains(")") || password.contains("-")

|| password.contains("+") || password.contains("/")

|| password.contains(":") || password.contains(".")

|| password.contains(", ") || password.contains("<")

|| password.contains(">") || password.contains("?")

|| password.contains("|"))) {

*error* = "The password contains non alpha numeric characters";

**return** **false**;

}

**if** (**true**) {

**int** count = 0;

**for** (**int** i = 65; i <= 90; i++) {

**char** c = (**char**)i;

String str1 = Character.*toString*(c);

**if** (password.contains(str1)) {

count = 1;

}

}

**if** (count == 0) {

*error* = "The password doesn't contain any uppercase value";

**return** **false**;

}

}

**if** (**true**) {

**int** count = 0;

**for** (**int** i = 90; i <= 122; i++) {

**char** c = (**char**)i;

String str1 = Character.*toString*(c);

**if** (password.contains(str1)) {

count = 1;

}

}

**if** (count == 0) {

*error* = "The password doesn't contain any lowercase value";

**return** **false**;

}

}

**return** **true**;

}

**public** **static** **void** main(String[] args)

{

Scanner getValuesAsInput = **new** Scanner(System.***in***);

System.***out***.print("Please provide a password: ");

String password = getValuesAsInput.nextLine();

**if** (*isValid*(password)) {

System.***out***.println("Valid Password");

}

**else** {

System.***out***.println("Invalid Password!!!" + *error*);

}

}

}

