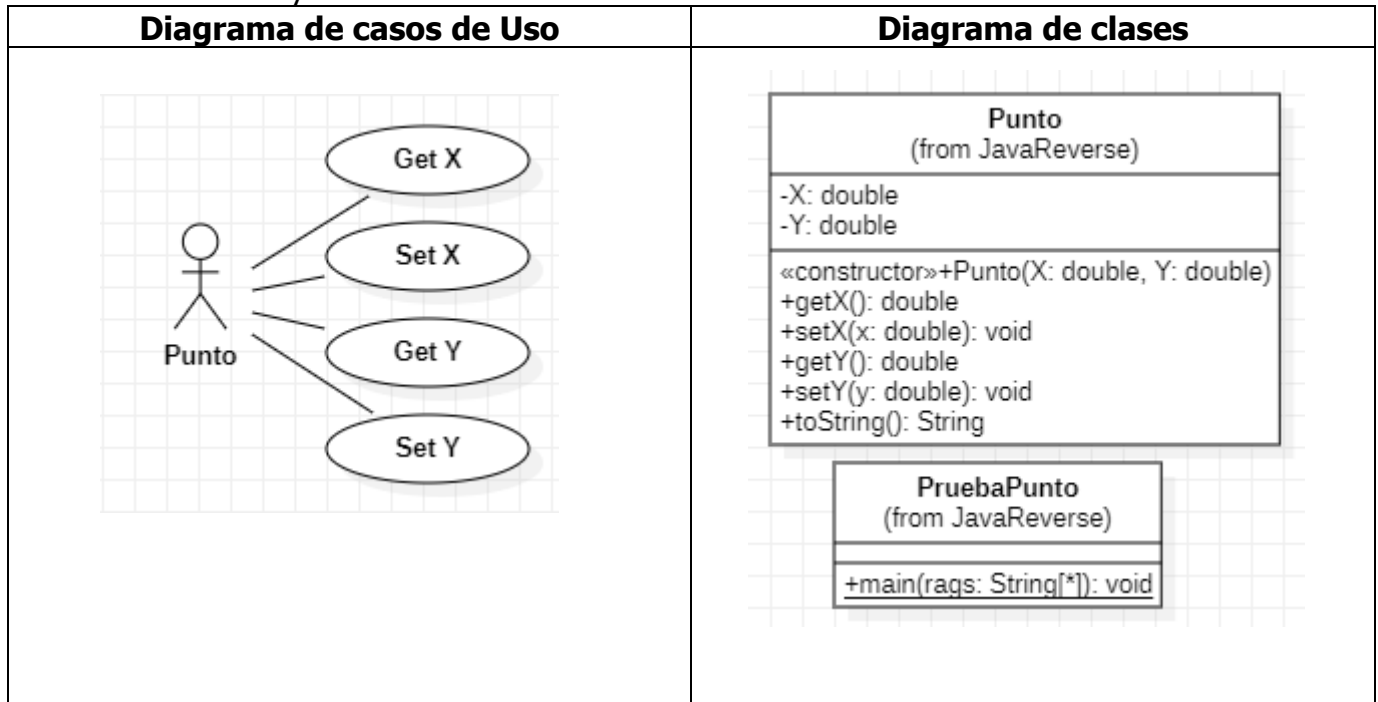
 UNIVERSIDAD CENTRAL	Facultad de Ingeniería y Ciencias Básicas Programa de Ingeniería Electrónica	POO – G04
		Portafolio
Cód:43390843	Est: Santiago Steven Reyes Naranjo	Segundo corte 2022-2S

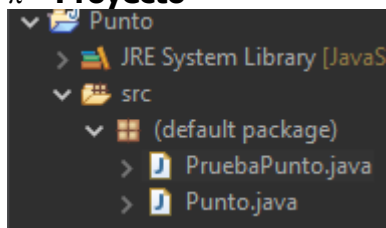
Método Get y Set

π Enunciado #1

Generar un programa haciendo uso de las buenas prácticas de la POO que genere un punto y vaya cambiando sus coordenadas en X y Y cambiando así su cuadrante de ubicación haciendo uso de los métodos Get y Set.



π Proyecto



π Código Fuente

```
Punto.java X PruebaPunto.java
1 // Santiago Reyes
2
3 public class Punto {
4     private double X;
5     private double Y;
6
7     public Punto(double X, double Y){
8         this.X=X;
9         this.Y=Y;
10    }
11
12    public double getX() {
13        return X;
14    }
15
16    public void setX(double x) {
17        X = x;
18    }
19
20    public double getY() {
21        return Y;
22    }
23
24    public void setY(double y) {
25        Y = y;
26    }
27
28    @Override
29    public String toString() {
30        return "Punto [X=" + X + ", Y=" + Y + ", getX()=" + getX() + ", getY()=" + getY() + "]";
31    }
32
33 }
```

```
Punto.java PruebaPunto.java X
1 // Santiago Reyes
2
3 public class PruebaPunto {
4     public static void main(String[] args) {
5         Punto unPunto = new Punto(0.0, 0.0);
6         System.out.println("inicializando -> " + unPunto);
7         Punto miPunto = new Punto (1.0, 2.0);
8         System.out.println("estoy en el cuadrante I -> " + miPunto);
9
10        miPunto.setX(-2.0);
11        System.out.println("estoy en el cuadrante II -> " + miPunto);
12        System.out.println("-> se actualizó la coordenada X = " + miPunto.getX());
13
14
15        miPunto.setY(-2.5);
16        System.out.println("estoy en el cuadrante III -> " + miPunto);
17        System.out.println("-> se actualizó la coordenada X = " + miPunto.getY());
18
19
20        miPunto.setX(1.0);
21        System.out.println("estoy en el cuadrante IV -> " + miPunto);
22        System.out.println("-> se actualizó la coordenada X = " + miPunto.getX());
23
24    }
25
26
27 }
28
```

π Pruebas

```

Console X
<terminated> PruebaPunto [Java Application] C:\Program Files\OpenJDK\openjdk-11.0.13_8\bin\javaw.exe (11/
inicializando -> Punto [X=0.0, Y=0.0, getX()=0.0, getY()=0.0]
estoy en el cuadrante I -> Punto [X=1.0, Y=2.0, getX()=1.0, getY()=2.0]
estoy en el cuadrante II -> Punto [X=-2.0, Y=2.0, getX()=-2.0, getY()=2.0]
-> se actualizó la coordenada X = -2.0
estoy en el cuadrante III -> Punto [X=-2.0, Y=-2.5, getX()=-2.0, getY()=-2.5]
-> se actualizó la coordenada X = -2.5
estoy en el cuadrante IV -> Punto [X=1.0, Y=-2.5, getX()=1.0, getY()=-2.5]
-> se actualizó la coordenada X = 1.0

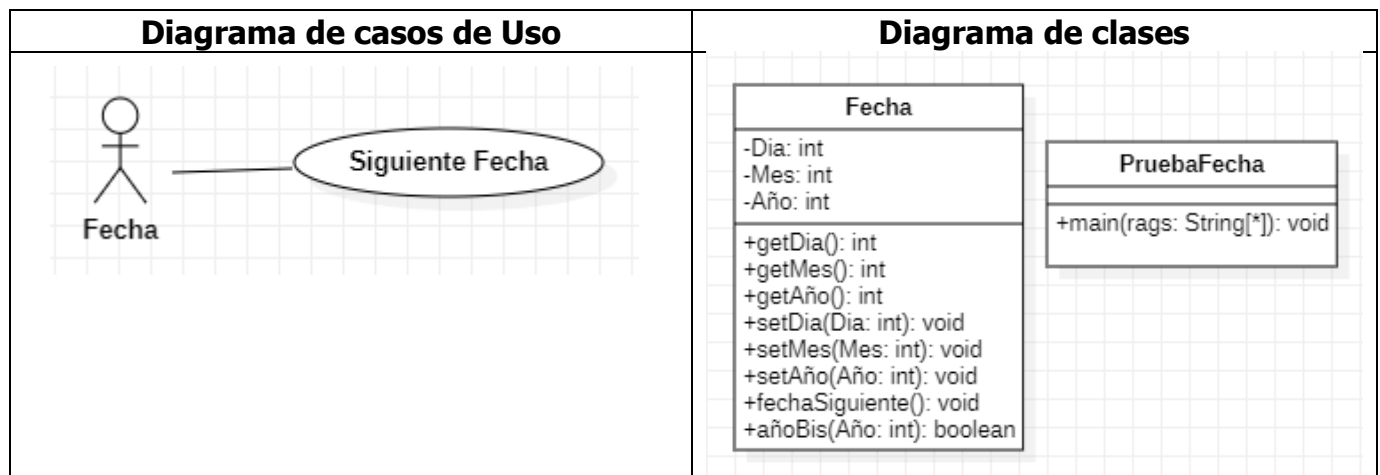
```

π Análisis de resultados

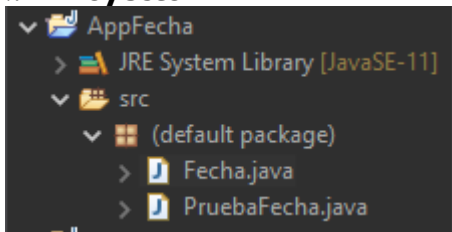
Se Creó un proyecto en Java llamado **Punto** con las clases de **Punto** y **PruebaPunto**, en la clase punto se generaron los atributos de X y Y para la ubicación del punto en los ejes del plano cartesiano, con su respectivo método constructor, y haciendo uso de los métodos **Get** y **Set**, los cuales obtienen el valor de la variable y le asignan el valor a la variable (respectivamente) para que haciendo uso de ellos podamos cambiar el valor en las coordenadas X y Y, para así poder desplazar el punto a través de los 4 cuadrantes del plano cartesiano, en la clase de **PruebaPunto** se llamaron los métodos de **Set** para cambiar el valor del punto en las respectivas coordenadas y se utilizó el método **Get** para obtener el valor de las coordenadas del punto y así poder imprimir o mostrar las nuevas coordenadas del punto.

π Enunciado #2

Generar un programa haciendo uso de las buenas prácticas de la POO que dada una fecha este nos diga cual es la fecha siguiente teniendo en cuenta los meses de 30, 31 y 28 días y también los años bisiestos.



π Proyecto



π Código Fuente

```
Fecha.java X PruebaFecha.java
1 //Santiago Reyes
2 public class Fecha {
3     //Atributos
4     private int Dia;
5     private int Mes;
6     private int Año;
7
8     //Metodo inicializador
9
10    public Fecha () {
11        this.Dia = 0;
12        this.Mes = 0;
13        this.Año = 0;
14    }
15    //Metodo constructor
16    public Fecha (int Dia ,int Mes, int Año) {
17        this.Dia = Dia;
18        this.Mes = Mes;
19        this.Año = Año;
20    }
21
22    //Metodos get y set de los atributos
23    public int getDia() {
24        return Dia;
25    }
26    public void setDia(int dia) {
27        Dia = dia;
28    }
29    public int getMes() {
30        return Mes;
31    }
32    public void setMes(int mes) {
33        Mes = mes;
34    }
35    public int getAño() {
36        return Año;
37    }
38    public void setAño(int año) {
39        Año = año;
40    }
41
42    public void FechaSiguiente() {
43        if (this.Mes == 1 || this.Mes == 3 || this.Mes == 5 || this.Mes == 7 ||
44            this.Mes == 8 || this.Mes == 10 || this.Mes == 12){
45            if (this.Dia > 0 && this.Dia <= 31 ) {
46                if (this.Dia == 31 && this.Mes == 12) {
47                    this.Dia = 1;
48                    this.Mes = 1;
49                    this.Año = this.Año + 1 ;
50                }else if (this.Dia == 31) {
51                    this.Dia = 1;
52                    this.Mes = this.Mes + 1;
53                }else {
54                    this.Dia = this.Dia +1;
55                }
56            }else {
57                System.out.println("Dia ingresado invalido");
58            }
59        }else if (this.Mes == 2) {
60            if(añoBis (this.Año)) {
61                if (this.Dia > 0 && this.Dia <= 29) {
62                    if(this.Dia == 29) {
63                        this.Dia = 1;
64                        this.Mes = this.Mes + 1;
65                    }else {
66                        this.Dia = this.Dia +1;
67                    }
68                }
69            }else if (this.Dia > 0 && this.Dia <= 28) {
70                if (this.Dia == 28) {
71                    this.Dia = 1;
72                    this.Mes = this.Mes + 1 ;
73                }else {
74                    this.Dia = this.Dia +1;
75                }
76            }else {
77                System.out.println("Dia ingresado invalido");
78            }
79        }
80    }
81}
```

```

79         }else if (this.Mes == 4 || this.Mes == 6 || this.Mes == 9 || this.Mes == 11) {
80             if(this.Dia > 0 && this.Dia <= 30 ) {
81                 if (this.Dia == 30) {
82                     this.Dia = 1;
83                     this.Mes = this.Mes + 1;
84                 }else {
85                     this.Dia = this.Dia + 1;
86                 }
87             }else {
88                 System.out.println("Dia ingresado invalido");
89             }
90         }
91         System.out.println("la fecha siguiente es :"+ this.Dia + "/" +
92             this.Mes + "/" + this.Año);
93     }
94 }
95
96 public boolean añoBis(int Año) {
97     if (Año <= 1852) {
98         return false;
99     }else {
100         if(Año % 4 == 0) {
101             if (Año % 100 == 0) {
102                 if (Año % 400 == 0) {
103                     return true;
104                 }else {
105                     return false;
106                 }
107             }else {
108                 return true;
109             }
110         }else {
111             return false;
112         }
113     }
114 }
115
116 @Override
117 public String toString() {
118     return "Fecha {" +
119         "dia=" + Dia +
120         ", mes =" + Mes +
121         ", Año= " + Año +
122         '}';
123 }
124
125 }
126 }

```

```

Fecha.java  PruebaFecha.java
1 //Santiago Reyes
2 import java.util.Scanner;
3 public class PruebaFecha {
4
5
6     public static void main (String[] args) {
7         try (Scanner scan = new Scanner (System.in)) {
8             Fecha UnaFecha = new Fecha();
9             System.out.println( "inicializando el objeto" + UnaFecha.toString());
10
11
12
13             Fecha MiFecha = new Fecha();
14             System.out.println( "Ingrese dia actual");
15             MiFecha.setDia(scan.nextInt());
16             System.out.println( "Ingrese la mes actual");
17             MiFecha.setMes(scan.nextInt());
18             System.out.println( "Ingrese el año actual ");
19             MiFecha.setAño(scan.nextInt());
20             if(MiFecha.getDia() > 0 && MiFecha.getDia() <= 31 && MiFecha.getMes() > 0 &&
21                 MiFecha.getMes() <= 12 && MiFecha.getAño() > 0) {
22                 System.out.println("Fecha actual : " + MiFecha.getDia() + "/" + MiFecha.getMes()+"/"+
23                     + MiFecha.getAño() );
24                 MiFecha.FechaSiguiente();
25             }else {
26                 System.out.println( "los Valores ingresados son incorrectos");
27             }
28         }
29     }
30 }
31 }
32 }

```

π Pruebas

```
Console X
<terminated> PruebaFecha [Java Application] C:\Program Files\OpenJDK\openjd
inicializando el objetoFecha {dia=0,mes =0,Año= 0}
Ingrese dia  actual
15
Ingrese la mes actual
06
Ingrese el año actual
2022
Fecha actual :15/6/2022
la fecha siguiente es :16/6/2022
```

Prueba con mes de 30 días

```
Console X
<terminated> PruebaFecha [Java Application] C:\Program Files\OpenJDK\open
inicializando el objetoFecha {dia=0,mes =0,Año= 0}
Ingrese dia  actual
30
Ingrese la mes actual
04
Ingrese el año actual
2022
Fecha actual :30/4/2022
la fecha siguiente es :1/5/2022
```

Prueba con mes de 31 días

```
Console X
<terminated> PruebaFecha [Java Application] C:\Program Files\OpenJDK\o
inicializando el objetoFecha {dia=0,mes =0,Año= 0}
Ingrese dia  actual
31
Ingrese la mes actual
03
Ingrese el año actual
2022
Fecha actual :31/3/2022
la fecha siguiente es :1/4/2022
```

Prueba Febrero

```
Console X
<terminated> PruebaFecha [Java Application] C:\Program Files\OpenJDK\o
inicializando el objetoFecha {dia=0,mes =0,Año= 0}
Ingrese dia  actual
28
Ingrese la mes actual
02
Ingrese el año actual
2022
Fecha actual :28/2/2022
la fecha siguiente es :1/3/2022
```

Prueba Febrero año bisiesto

```
Console X
<terminated> PruebaFecha [Java Application] C:\Program Files\OpenJDK\
inicializando el objetoFecha {dia=0,mes =0,Año= 0}
Ingrese dia actual
28
Ingrese la mes actual
02
Ingrese el año actual
2020
Fecha actual :28/2/2020
la fecha siguiente es :29/2/2020
```

Prueba fin de año

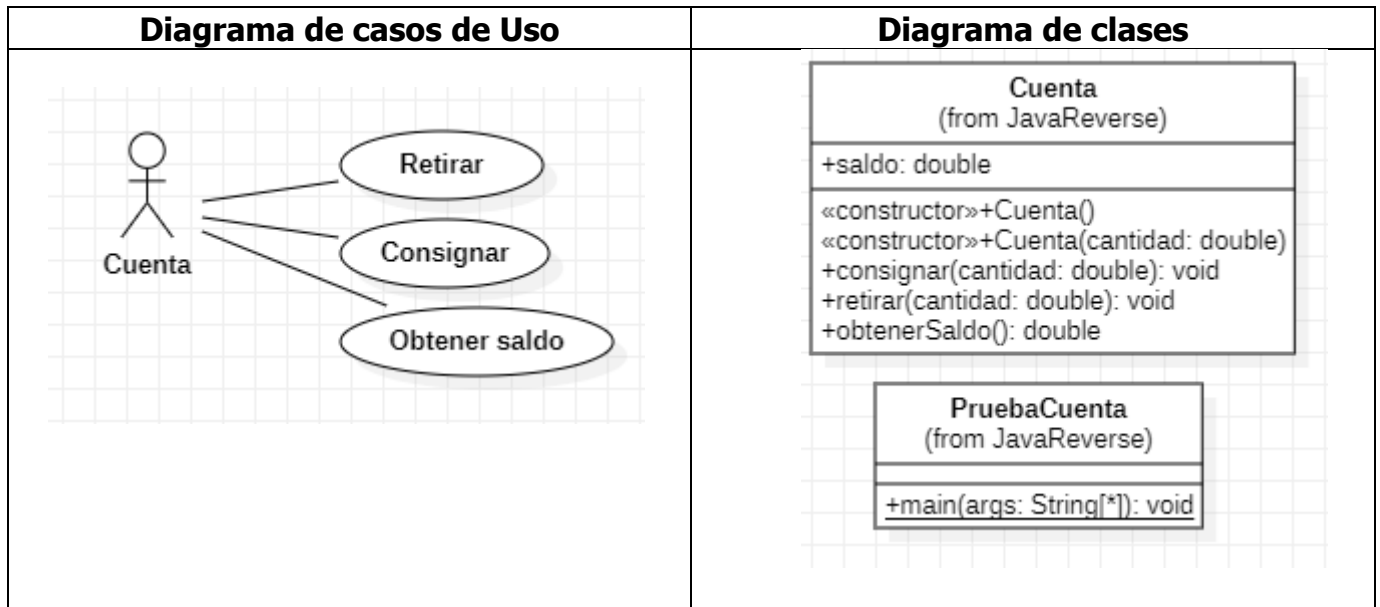
```
Console X
<terminated> PruebaFecha [Java Application] C:\Program Files\OpenJ
inicializando el objetoFecha {dia=0,mes =0,Año= 0}
Ingrese dia actual
31
Ingrese la mes actual
12
Ingrese el año actual
2021
Fecha actual :31/12/2021
la fecha siguiente es :1/1/2022
```

π **Análisis de resultados**

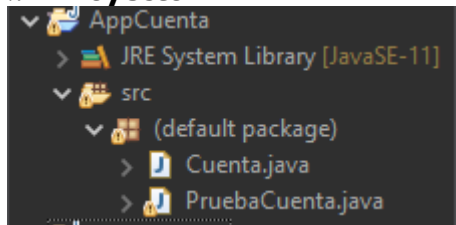
Se Creó un proyecto en Java llamado **AppFecha** con las clases de **Fecha** y **PruebaFecha**, en la clase fecha se generaron los atributos de Día, Mes y año para definir una fecha inicial la cual luego va a ser cambiada, con su respectivo método constructor, y haciendo uso de los métodos **Get** y **Set**, los cuales obtienen el valor de la variable y le asignan el valor a la variable (respectivamente) para que haciendo uso de ellos podamos cambiar el valor de los días, meses y años, para así poder cambiar los valores del día, mes y año para dar el día siguiente dependiendo de la fecha ingresada en la consola, en la clase de **PruebaFecha** se llamaron los métodos de **Set** para cambiar el valor de la fecha y se utilizó el método **Get** para obtener el valor de la fecha para así imprimir o mostrar cual es el día siguiente al día que fue ingresado por consola.

π **Enunciado #3**

Generar un programa haciendo uso de las buenas prácticas de la POO que cree una cuenta de banco con un saldo y con éste saldo se puedan hacer las diferentes operaciones que se podrían hacer en un cajero de banco.



π Proyecto



π Código Fuente

```

Cuenta.java x PruebaCuenta.java
1 // Santiago Reyes
2
3 public class Cuenta {
4
5     public double saldo;
6
7     public Cuenta() {}
8
9     public Cuenta(double cantidad) {
10         if (cantidad > 0) {
11             this.saldo = cantidad;
12         }
13     }
14     public void consignar (double cantidad) {
15         if (cantidad > 0){
16             saldo = saldo + cantidad;
17             // mensaje para advertir que no es aceptado
18         }
19         else {
20             System.out.println("El valor no es aceptado...");
21         }
22     }
23     public void retirar (double cantidad) {
24         if (cantidad > 0 && cantidad < saldo){
25             saldo = saldo - cantidad;
26             // mensaje para advertir que no es aceptado
27         }
28         if (cantidad > saldo){
29             System.out.println("No hay dinero suficiente para retirar");
30         }
31         else {
32             System.out.println("No hay dinero para retirar");
33         }
34     }
35     public double obtenerSaldo() {
36
37         return saldo;
38     }
39

```



```
Cuenta.java PruebaCuenta.java X
1 // Santiago Reyes
2 import java.util.Scanner;
3
4 public class PruebaCuenta {
5     public static void main(String[] args){
6         double valor; // var local
7         Scanner lea = new Scanner (System.in);
8
9         //Inicializar la cuenta
10
11         Cuenta unaCuenta = new Cuenta();
12         System.out.println("Creando la cuenta...$ " + unaCuenta.obtenerSaldo());
13
14         //Apertura con saldo inicial
15
16         System.out.print("cantidad inicial: ");
17         valor = lea.nextDouble();
18         Cuenta miCuenta = new Cuenta(valor);
19         System.out.println("Apertura de la cuenta con $:" + miCuenta.obtenerSaldo());
20
21         //realizar una consignacion
22
23         System.out.print("Cantidad a consignar: ");
24         valor = lea.nextDouble();
25         miCuenta.consignar(valor);
26         System.out.println("Su saldo es de $ " + miCuenta.obtenerSaldo());
27
28         // realizar un retiro
29
30         System.out.print("Cantidad a rerirar: ");
31         valor = lea.nextDouble();
32         miCuenta.retirar(valor);
33         System.out.println("su nuevo saldo es de $:" + miCuenta.obtenerSaldo());
34
35         //consultar el saldo
36
37         System.out.println("su saldo a la fecha es de $:" + miCuenta.obtenerSaldo());
38     }
39 }
```

π Pruebas

```
Console X
<terminated> PruebaCuenta [Java Application] C:\Program F
Creando la cuenta...$ 0.0
cantidad inicial: 200
Apertura de la cuenta con $:200.0
Cantidad a consignar: 100
Su saldo es de $ 300.0
Cantidad a rerirar: 30
No hay dinero para retirar
su nuevo saldo es de $:270.0
su saldo a la fecha es de $:270.0
```

Prueba con retiro mayor al valor en cuenta.

```
Console X
<terminated> PruebaCuenta [Java Application] C:\Progran
Creando la cuenta...$ 0.0
cantidad inicial: 100
Apertura de la cuenta con $:100.0
Cantidad a consignar: 20
Su saldo es de $ 120.0
Cantidad a rerirar: 150
No hay dinero suficiente para retirar
su nuevo saldo es de $:120.0
su saldo a la fecha es de $:120.0
```

Retiro con valor negativo.

```

Console X
<terminated> PruebaCuenta [Java Application] C:\
Creando la cuenta...$ 0.0
cantidad inicial: 20
Apertura de la cuenta con $:20.0
Cantidad a consignar: 50
Su saldo es de $ 70.0
Cantidad a retirar: -20
Valor incorrecto.
su nuevo saldo es de $:70.0
su saldo a la fecha es de $:70.0

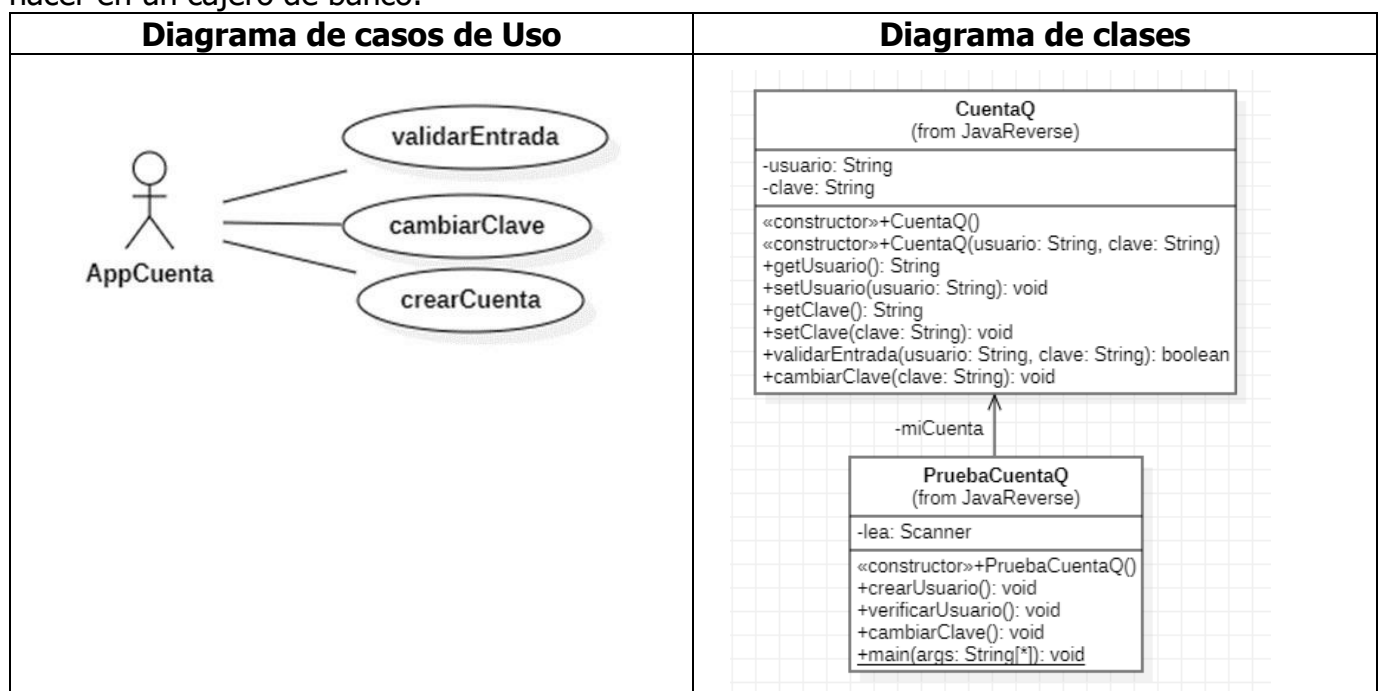
```

π Análisis de resultados

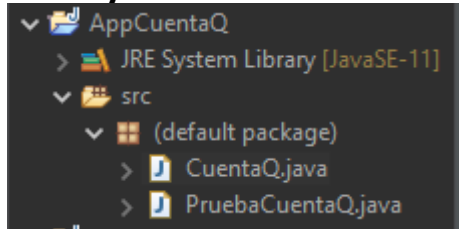
Se Creó un proyecto en Java llamado **AppCuenta**, dentro del cual se crearon las clases de **Cuenta** y **PruebaCuenta**, en la clase de **Cuenta**, se definió el atributo tipo double llamado saldo que es el valor con el que se van a hacer las operaciones de banco, luego de esto, se hizo una validación para que cuando se cree la cuenta, no pueda tener un saldo negativo al momento de ser creada, después de esto se crearon los métodos para consignar en el cual también se hace una validación de que el valor a consignar debe ser un valor superior a 0, retirar en el cual se realiza la validación de que el valor a retirar debe ser superior a 0 y menor que el saldo actual en la cuenta, ya que no puede quedar un saldo en negativo, y obtener saldo el cual nos permite obtener el saldo que se tiene actualmente en la cuenta; en la clase **PruebaCuenta** se hacen los llamados de los métodos creados en la clase **Cuenta** primeramente generando la cuenta con el saldo respectivo, y luego de esto ahí si se realizan las operaciones que se requieren.

π Enunciado #4

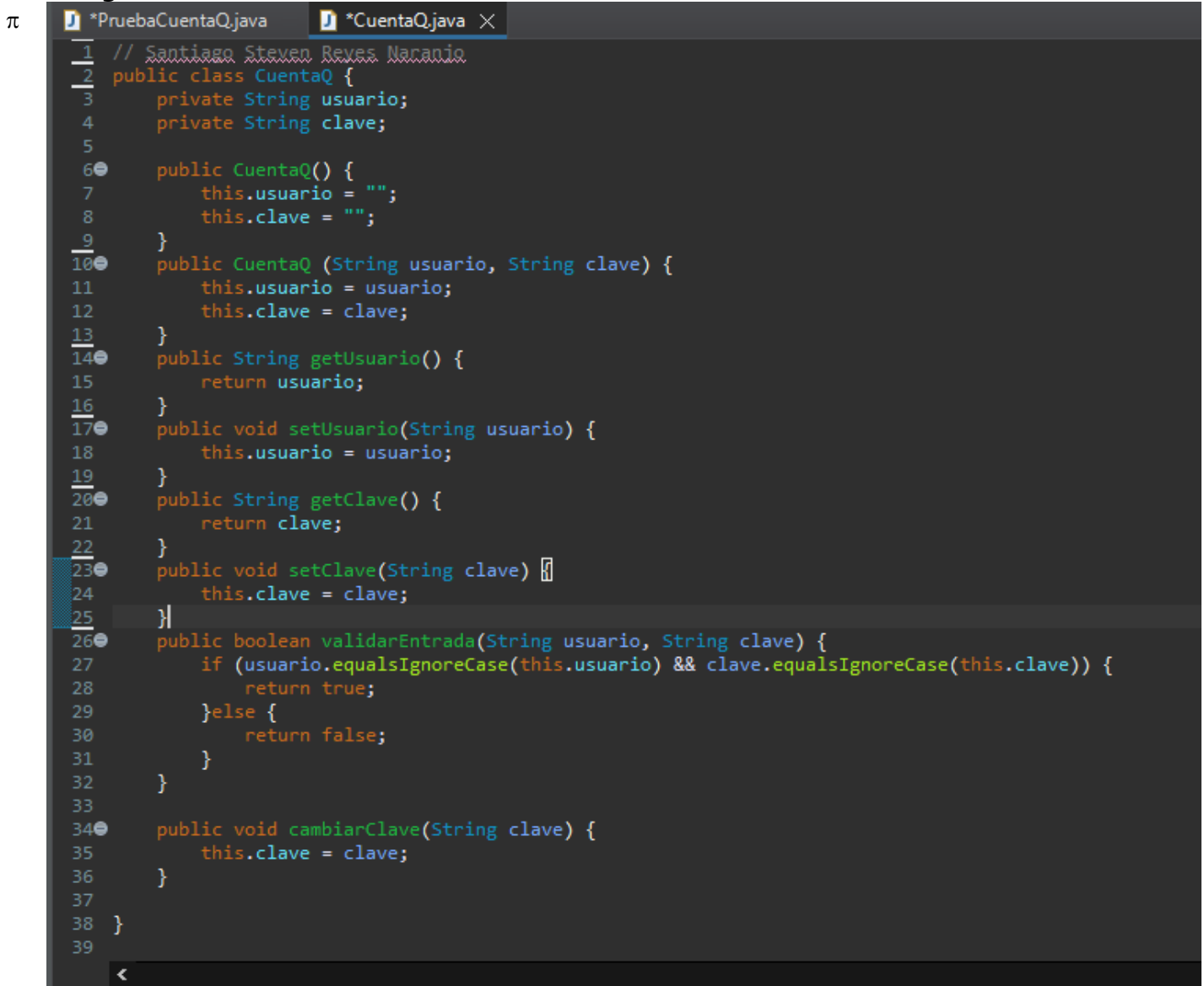
Generar un programa haciendo uso de las buenas prácticas de la POO que cree una cuenta de banco con un saldo y con éste saldo se puedan hacer las diferentes operaciones que se podrían hacer en un cajero de banco.



π Proyecto



π Código Fuente



π

```
PruebaCuentaQ.java X CuentaQ.java
1 import java.util.Scanner;
2 // Santiago Steven Reyes Naranjo
3 public class PruebaCuentaQ {
4
5     private CuentaQ miCuenta;
6     private Scanner lea;
7
8     public PruebaCuentaQ() {
9         this.miCuenta = new CuentaQ();
10        this.lea = new Scanner(System.in);
11    }
12    public void crearUsuario() {
13        String usuario = "";
14        String clave = "";
15        System.out.println("Ingrese usuario...: ");
16        usuario = lea.next();
17        miCuenta.setUsuario(usuario);
18        System.out.println("Ingrese clave.... ");
19        clave = lea.next();
20        miCuenta.setClave(clave);
21    }
22
23    public void verificarUsuario() {
24        String usuario = "";
25        String clave = "";
26        System.out.println("\n Verificacion de usuario y clave\n");
27        System.out.println("Ingrese usuario...:");
28        usuario = lea.next();
29        System.out.println("Ingrese clave...:");
30        clave = lea.next();
31        if (miCuenta.validarEntrada(usuario,clave)) {
32            System.out.println("Ingreso exitoso..." + miCuenta.getUsuario());
33        }else {
34            System.out.println("Ingreso No Exitoso..." + miCuenta.getUsuario());
35        }
36    }
37    public void cambiarClave() {
38        String clave = "";
39        String clave2 = "";
40        String clave3 = "";
41        System.out.println("Ingrese Clave actual: ");
42        clave = lea.next();
43        if (clave.equalsIgnoreCase(miCuenta.getClave())) {
44            System.out.println("Ingrese Nueva Clave ");
45            clave2 = lea.next();
46            System.out.println("Ingrese nuevamente su Clave ");
47            clave3=lea.next();
48            if (clave2.equalsIgnoreCase(clave3)) {
49                miCuenta.cambiarClave(clave2);
50                System.out.println("Cambio realizado satisfactoriamente ");
51            } else {
52                System.out.println("Claves incorrectas... ");
53            }
54        }else {
55
56            System.out.println("Ingrese Actual incorrecta...");
57        }
58    }
59 }
60
61 public static void main(String[] args) {
62     PruebaCuentaQ prueba= new PruebaCuentaQ();
63     prueba.crearUsuario();
64     prueba.verificarUsuario();
65     prueba.cambiarClave();
66     prueba.verificarUsuario();
67 }
68 }
69
```

π

π Pruebas

```
Console X
<terminated> PruebaCuentaQ [Java Application] C:\Program File
Ingrese usuario...:
santiago
Ingrese clave....
reyes

  Verificacion de usuario y clave

Ingrese usuario...:
santiago
Ingrese clave...:
reyes
Ingreso exitoso...santiago
Ingrese Clave actual:
reyes
Ingrese Nueva Clave
naranja
Ingrese nuevamente su Clave
naranja
Cambio realizado satisfactoriamente

  Verificacion de usuario y clave

Ingrese usuario...:
santiago
Ingrese clave...:
naranja
Ingreso exitoso...santiago
<
```

Prueba con clave incorrecta.

```
Console X
PruebaCuentaQ [Java Application] C:\Program Files\OpenJDK\openjdk-11.0.13
Ingrese usuario...:
santiago
Ingrese clave....
reyes

  Verificacion de usuario y clave

Ingrese usuario...:
santiago
Ingrese clave...:
naranja
Ingreso No Exitoso...santiago
```

Prueba con usuario incorrecto.

```
Console X
PruebaCuentaQ [Java Application] C:\Program Files\OpenJDK
Ingreso usuario...:
santiago
Ingreso clave....
reues

Verificacion de usuario y clave

Ingreso usuario...:
santiago
Ingreso clave...:
reues
Ingreso No Exitoso...santiago
```

Clave incorrecta al querer cambiar de clave

```
Console X
PruebaCuentaQ [Java Application] C:\Program Files\OpenJDK
Ingreso usuario...:
santiago
Ingreso clave....
reyes

Verificacion de usuario y clave

Ingreso usuario...:
santiago
Ingreso clave...:
reyes
Ingreso exitoso...santiago
Ingreso Clave actual:
naranja
Ingreso Actual incorrecta...
```

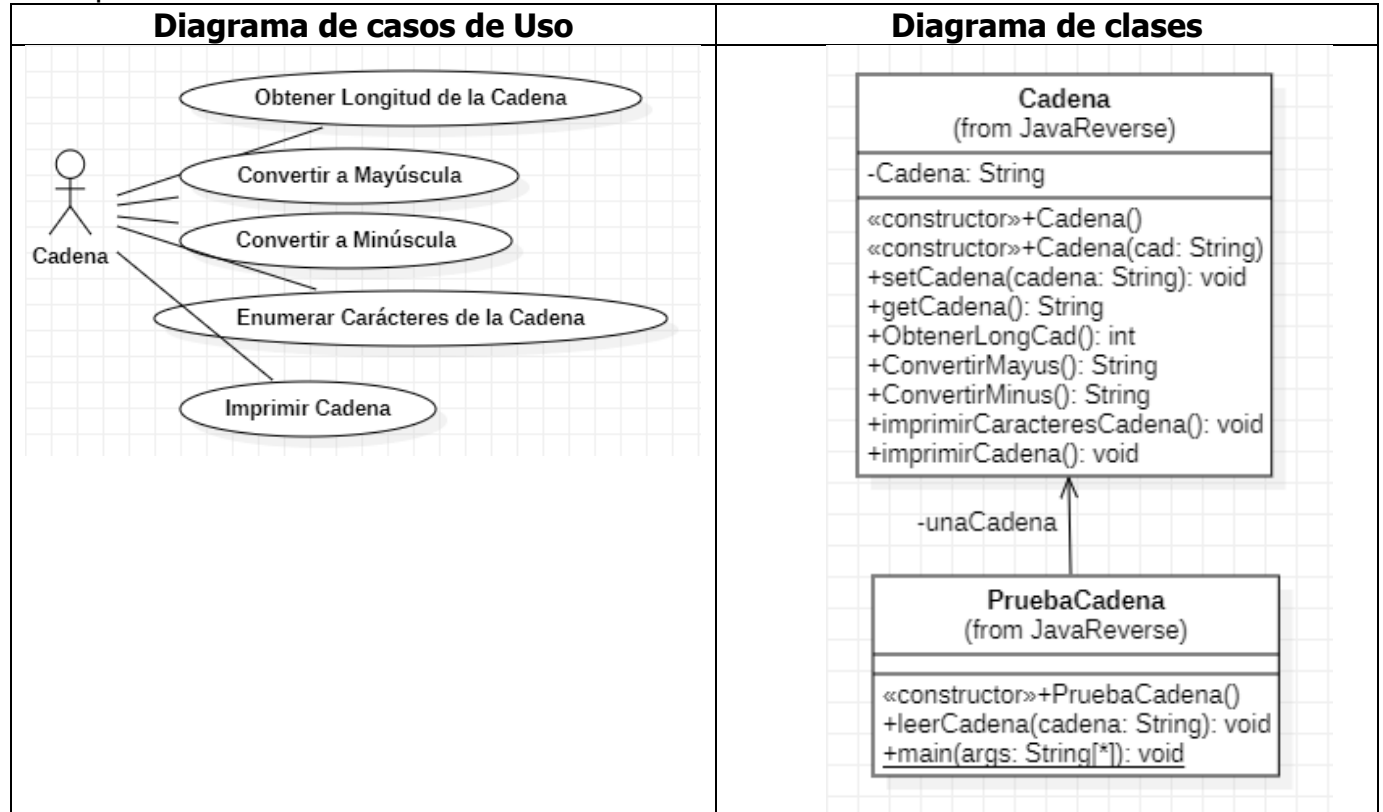
π **Análisis de resultados**

Se Creó un proyecto en Java llamado **AppCuentaQ**, dentro del cual se crearon las clases de **CuentaQ** y **PruebaCuentaQ**, en la clase de **CuentaQ**, se definieron los atributos tipo string llamados usuario y clave, luego de esto, se generó el método inicializador y constructor con los respectivos atributos, después de esto se generaron los métodos **Get** y **Set** de los atributos de **usuario y contraseña** el método **get** se usa para realizar a validación de que la cuenta y la contraseña sean iguales y el método **set** se utiliza para cambiar la clave del respectivo usuario; en la clase **PruebaCuentaQ** se crean unas variables locales para indexar los valores a los respectivos atributos a través del método **Set** y así se genera el nuevo usuario con su respectiva contraseña, luego para realizar la verificación del usuario, se genera un método que tiene unas variables locales que le permiten comparar el valor de **usuario y contraseña** para validar el acceso del usuario, haciendo uso del método **Get** para realizar la comparación, Para el método cambiarClave se generaron 3 variables locales, una para comparar lo indexado por el cliente con la clave actual, los otros 2 se generan para primero, almacenar el valor de la clave en el atributo clave a través del método **Set** y la siguiente, para validar que la verificación de la clave cambiada ha sido correcta (la clave cambiada se obtiene 2 veces para validar.), al final de la clase, se llaman los métodos en su respectivo orden.

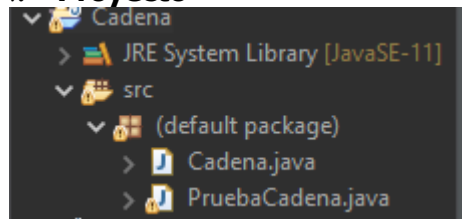
Cadenas de Texto.

π Enunciado #5

Generar un programa haciendo uso de las buenas prácticas de la POO que tome una cadena, cuente sus caracteres, la transforme en mayúscula y minúscula, y que devuelva cada carácter con su respectivo índice.



π Proyecto



Código Fuente

```
Cadena.java X PruebaCadena.java
1 // Santiago Steven Reyes Naranjo
2 public class Cadena {
3
4     private String Cadena ;
5     public Cadena () {
6         this.Cadena = "";
7     }
8
9     public Cadena (String cad) {
10         this.Cadena = cad;
11     }
12
13     public void setCadena(String cadena) {
14         Cadena = cadena;
15     }
16     public String getCadena() {
17         return Cadena;
18     }
19     public int  ObtenerLongCad(){
20         return Cadena.length();
21     }
22     public String ConvertirMayus(){
23         return Cadena.toUpperCase();
24     }
25
26     public String ConvertirMinus(){
27         return Cadena.toLowerCase();
28     }
29     public void imprimirCaracteresCadena () {
30         char c = ' ';
31         System.out.print("\nImpresion de la cadena por caracteres ");
32         for (int i = 0; i < Cadena.length(); i++) {
33             c = Cadena.charAt(i);
34             System.out.print(i + " " + c + " / ");
35         }
36         System.out.print(" ");
37     }
38     public void imprimirCadena() {
39         System.out.print(" " + Cadena);
40     }
41 }
```


π

```
Cadena.java PruebaCadena.java X
1 // Santiago Reyes
2
3 import java.util.Scanner;
4
5 public class PruebaCadena {
6
7     private Cadena unaCadena;
8
9     public PruebaCadena() {
10         this.unaCadena = new Cadena();
11     }
12
13     public void leerCadena(String cadena) {
14         unaCadena.setCadena(cadena);
15         System.out.println("\n La longitud de la cadena : " + unaCadena.ObtenerLongCad());
16         System.out.println("\n La cadena texto en mayúsculas : " + unaCadena.ConvertirMayus());
17         System.out.println("\n La cadena texto en minúsculas : " + unaCadena.ConvertirMinus());
18         unaCadena.imprimirCaracteresCadena();
19         unaCadena.imprimirCadena();
20     }
21
22
23     public static void main(String[] args) {
24         Scanner lea = new Scanner(System.in);
25         PruebaCadena prueba = new PruebaCadena();
26         String cad;
27         // se lee la cadena por teclado
28         System.out.println("Digite Cadena: ");
29         cad = lea.nextLine();
30         prueba.leerCadena(cad);
31     }
32
33 }
34
```

π

Pruebas

```
Console X
<terminated> PruebaCadena [Java Application] C:\Program Files\OpenJDK\openjdk-11.0.13_8\bin\javaw.exe (11/10/2022, 11:55:48 p. m. - 11:56:01 p. m.) [pid: 26392]
Digite Cadena:
merequetengue

La longitud de la cadena : 13

La cadena texto en mayúsculas : MEREQUETENGUE

La cadena texto en minúsculas : merequetengue

Impresion de la cadena por caracteres 0 m / 1 e / 2 r / 3 e / 4 q / 5 u / 6 e / 7 t / 8 e / 9 n / 10 g / 11 u / 12 e / merequetengue
```

π

Análisis de resultados

Se Creó un proyecto en Java llamado **Cadena**, dentro del cual se crearon las clases de **Cadena** y **PruebaCadena**, en la clase de **Cadena**, se definió el atributo tipo string llamado cadena, luego de esto, se generó el método inicializador y constructor con el atributo, después de esto se generaron los métodos **Get** y **Set** del atributo de **Cadena** el método **set** se utiliza para asignar la cadena que digita el usuario en el atributo de **Cadena** del objeto **unaCadena** creado o generado a través del uso de la clase de **Cadena**, luego se genera el método **obtenerLongCad** que nos retorna el largo o la cantidad de caracteres que tiene unaCadena, haciendo uso del método **length** que sirve para contar la totalidad de los caracteres de una variable o atributo, luego de este se generó el método **ConvertirMayus**, método que su

función es convertir toda la cadena diligenciada en una cadena de caracteres en mayúscula, haciendo uso del método de toUpperCase, que cumple ésta función (convertir los caracteres de una cadena en mayúscula), luego se generó el método convertirMinus que cumple con una función similar a la de ConvertirMayus, solo que, en vez de mayúsculas se transforman todas las letras y/o caracteres en minúscula y el método imprimirCaracteresCadena a través de un for, nos retorna la posición o índice de la letra donde está ubicado; en la clase **PruebaCadena** se genera el objeto **unaCadena**, se obtiene la cadena que fue escrita por la persona en la consola y ésta se transforma llamando los métodos correspondientes de la clase **Cadena**.

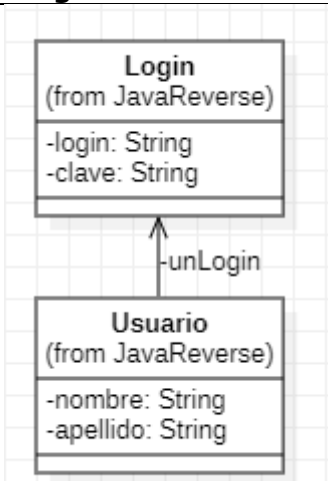
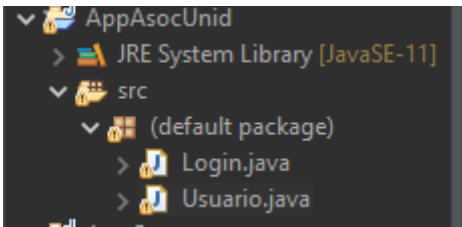
Asociaciones.

Enunciado #6

Generar un programa haciendo uso de las buenas prácticas de la POO que nos permita analizar y representar qué es una asociación unidireccional.

Diagramas de caso de uso

N/A.

Diagramas de clases.	Proyecto.
	

Código Fuente

```

Login.java
1 // Santiago Reyes
2
3 public class Login {
4     private String login;
5     private String clave;
6 }
7

Usuario.java
1 // Santiago Reyes
2
3 public class Usuario {
4     private String nombre;
5     private String apellido;
6     private Login unLogin;
7
8 }
```

Pruebas

N/A.

Análisis de resultados

Se creo un proyecto java haciendo uso de las buenas prácticas de la POO llamado **AppAsocUnid**, en la cual se crearon las clases **Usuario** y **Login**, en la clase **Usuario** se declararon tres atributos privados, dos atributos de tipo String llamados nombre y apellido y el otro atributo cumple con la función de llamar a la otra clase creada llamada **Login** con el

nombre unlogin. Y de esa manera es como se crea una asociación de clases, en la clase **Login** se declaran dos atributos privados, los cuales son tipo String llamados login y clave.

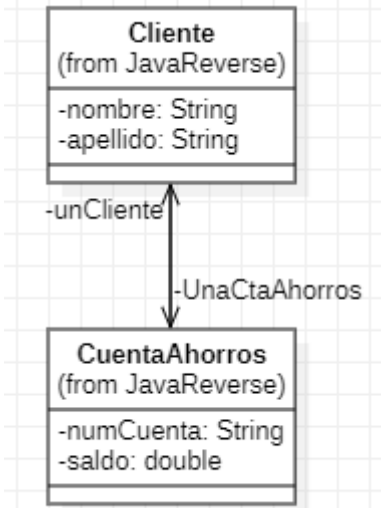
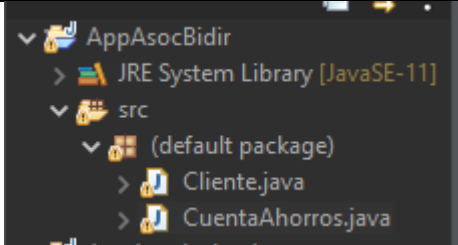
π De esta manera se realiza una asociación 0-1 cero a uno.

π Enunciado #7

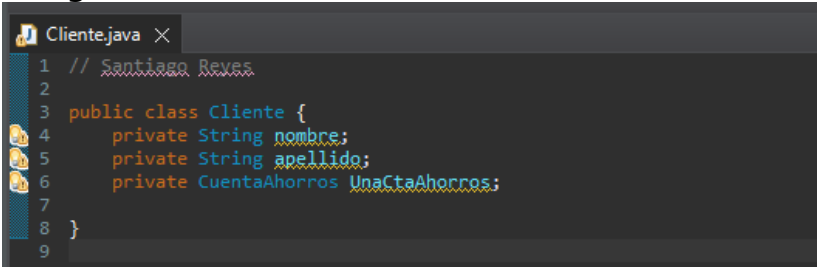
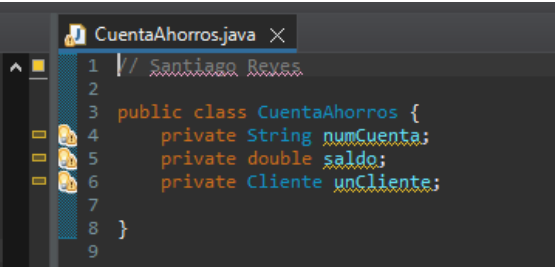
Generar un programa haciendo uso de las buenas prácticas de la POO que nos permita analizar y representar qué es una asociación bidireccional.

π Diagramas de caso de uso

π N/A.

π Diagramas de clases.	π Proyecto.
<p>π</p>  <pre> classDiagram class Cliente { -nombre: String -apellido: String } class CuentaAhorros { -numCuenta: String -saldo: double } Cliente "0" -- "1" CuentaAhorros : -unCliente / -UnaCtaAhorros </pre>	<p>π</p> 

π Código Fuente

<p>π</p>  <pre> 1 // Santiago Reyes 2 3 public class Cliente { 4 private String nombre; 5 private String apellido; 6 private CuentaAhorros unCtaAhorros; 7 } 8 9 </pre>	<p>π</p>  <pre> 1 // Santiago Reyes 2 3 public class CuentaAhorros { 4 private String numCuenta; 5 private double saldo; 6 private Cliente unCliente; 7 } 8 9 </pre>
---	---

π Pruebas

N/A.

π Análisis de resultados

π Se creo un proyecto java llamado **AppAsocBidir**, en la cual se crearon dos clases una llamada **Cliente** y otra clase llamada **CuentaAhorros**, en la clase **Cliente** se declararon tres atributos privados, dos atributos son tipo **String** llamados nombre y apellido y el otro atributo cumple con la función de llamar a la otra clase creada llamada **CuentaAhorros** con el nombre **unaCuentaAhorros**. Y de esa manera es como se crea una asociación de clases, en la clase **CuentaAhorros** se declaran tres atributos privados, un atributo es de tipo String llamado numCuenta. El segundo atributo es de tipo double llamado saldo. Y el ultimo atributo cumple con a función de llamar a la clase llamada **Cliente** la cual tiene como nombre unCliente.

π De esta manera se realiza una asociación de 1....1 uno a uno.

π **Enunciado #8**

Generar un programa haciendo uso de las buenas prácticas de la POO que nos permita analizar y representar los tipos de relaciones unidireccional y bidireccional.

π **Diagramas de caso de uso**

π N/A.

π Diagramas de clases.	π Proyecto.
<p>π</p> <pre>classDiagram class Cliente { -nombre: String -apellido: String } class CuentaAhorros { -numCuenta: String -saldo: double } class Login { -login: String -clave: String } Cliente --> CuentaAhorros : -unCliente Cliente --> Login : -segunLogin CuentaAhorros --> Login : -segunCuenta</pre>	<p>π</p> <pre>graph TD Asociacion --> JRE["JRE System Library [JavaSE-11]"] Asociacion --> src src --> Cliente["(default package)"] Cliente --> Cliente_java["Cliente.java"] Cliente --> CuentaAhorros_java["CuentaAhorros.java"] Cliente --> Login_java["Login.java"]</pre>

π **Código Fuente**

<p>π</p> <pre>1 // Santiago Reyes 2 3 public class CuentaAhorros { 4 private String numCuenta; 5 private double saldo; 6 private Cliente unCliente; 7 } 8 9</pre>	<p>π</p> <pre>1 // Santiago Reyes 2 3 public class Cliente { 4 private String nombre; 5 private String apellido; 6 private CuentaAhorros segunCuenta; 7 private Login segunLogin; 8 } 9</pre>	<p>π</p> <pre>1 // Santiago Reyes 2 3 public class Login { 4 private String login; 5 private String clave; 6 } 7</pre>
---	---	--

π **Pruebas**

π N/A.

π **Análisis de resultados**

π Se creo un proyecto java llamado **Asociacion**, en la cual se crearon tres clases una llamada **Cliente**, otra clase llamada **CuentaAhorros** y otra llamada **Login**.

π En la clase **Cliente** se declararon cuatro atributos privados, dos atributos son tipo String llamados nombre y apellido, el otro atributo cumple con la función de llamar a otra clase creada llamada **CuentaAhorros** con el nombre **unaCuentaAhorros**. Y el ultimo atributo también cumple con la función de llamar a otra clase llamada **Login** con el nombre unLogin. de esa manera es como se crea una asociación de clases, en la clase **CuentaAhorros** se declaran tres atributos privados, un atributo es de tipo String llamado numCuenta. El segundo atributo es de tipo double llamado saldo. Y el ultimo atributo cumple con a función de llamar a la clase llamada Cliente la cual tiene como nombre unCliente, en la clase **Login** se declaran dos atributos privados, los cuales son tipo String llamados login y clave.

π De esta menara se realiza una asociación 0-1 cero a uno y de 1-1 uno a uno.

π **Enunciado #9**

π Generar un programa haciendo uso de las buenas prácticas de la POO que nos permita analizar y representar los tipos de relaciones unidireccional y bidireccional de uno a muchos.

π **Diagramas de caso de uso**

π N/A.

<p>π Diagramas de clases.</p> <p>π</p>	<p>π Proyecto.</p> <p>π</p>
---	--

Código Fuente

<p>π Persona.java</p> <pre> 1 //Santiago Reyes 2 3 import java.util.TreeSet; 4 5 public class Persona { 6 7 private String nombre; 8 TreeSet<Perro> mascotas = new TreeSet<>(); 9 } </pre>	<p>π Perro.java</p> <pre> 1 // Santiago Reyes 2 3 public class Perro { 4 5 private String name; 6 public Persona propietario; 7 8 } 9 </pre>
--	--

Pruebas

N/A.

Análisis de resultados

Se creo un proyecto java llamado **AppAsociacion4**, en la cual se crearon dos clases una llamada **Persona**, y otra clase llamada **Perro**, en la clase Persona se declaró un atributo privado, el atributo es de tipo String llamado nombre, y el otro atributo cumple con la función de multiplicidad de tipo **Treeset** la cual cumple con llamar a la otra clase creada llamada Perro con el nombre mascotas. Se cea un método de importación llamada java.util.Treeset, y con ello poder generar la multiplicidad correctamente a esa clase, en la clase Perro se declaró un atributo privado, el atributo es de tipo String llamado nombre. Y otro atributo es público que cumple con la función de llamar a la otra clase creada llamada Persona con el nombre de propietario.

De esta menara se realiza una asociación 1-* uno a muchos y de *-1 muchos a uno.

Enunciado #10

Generar un programa haciendo uso de las buenas prácticas de la POO que nos permita analizar y representarlos tipos de relaciones unidireccional y bidireccional de uno a muchos.

Diagramas de caso de uso

N/A.

<p>π Diagramas de clases.</p> <p>π</p>	<p>π Proyecto.</p> <p>π</p>
---	--

Código Fuente

π

Ley.java ×

```

1 // Santiago Revas
2
3 import java.util.TreeSet;
4
5 public class Ley {
6     private String numLey;
7     public TreeSet<Congresista> voto = new TreeSet<>();
8
9 }
10

```

Congresista.java ×

```

1 // Santiago Revas
2
3 import java.util.TreeSet;
4
5 public class Congresista {
6
7     private String nombre;
8     TreeSet<Ley> fueVotadoPor = new TreeSet<>();
9
10 }
11

```

π

Pruebas

N/A.

π

Análisis de resultados

π

Se creo un proyecto java llamado **AppAsociacion5**, en la cual se crearon dos clases una llamada **Ley**, y otra clase llamada **Congresista**, en la clase **Congresista** se declaró un atributo privado, el atributo es de tipo String llamado nombre, y el otro atributo cumple con la función de multiplicidad de tipo **Treeset** la cual cumple con llamar a la otra clase creada llamada **Ley** con el nombre fueVotadoPor. Se crea un método de importación llamada java.util.Treeset, y con ello poder generar la multiplicidad correctamente a esa clase, en la clase **Ley** se declaró un atributo privado, el atributo es de tipo String llamado numLey. Y otro atributo es público que cumple con la función de llamar a la otra clase creada llamada **Congresista** con el nombre de propietario.

De esta menara se realiza una asociación *-* muchos a muchos.

π

Enunciado #11

π

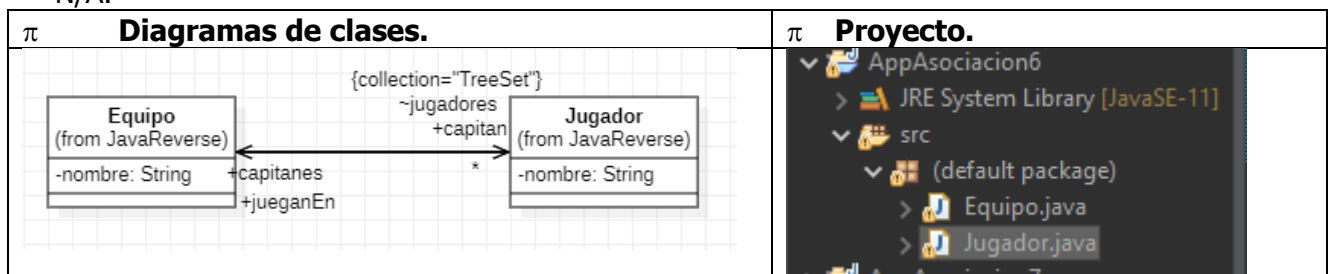
Generar un programa haciendo uso de las buenas prácticas de la POO que nos permita analizar y representar qué es una asociación unidireccional.

π

Diagramas de caso de uso

π

N/A.



Código Fuente

Equipo.java

```

1 // Santiago Reyes
2
3 import java.util.TreeSet;
4
5 public class Equipo {
6
7     private String nombre;
8     public Jugador capitan;
9     TreeSet<Jugador> jugadores = new TreeSet<>();
10
11 }
12

```

Jugador.java

```

1 //Santiago Reyes
2
3 public class Jugador {
4
5     private String nombre;
6     public Equipo capitanes;
7     public Equipo jueganEn;
8
9 }
10

```

Pruebas

N/A.

Análisis de resultados

Se creo un proyecto java llamado **AppAsociacion6**, en la cual se crearon dos clases una llamada **Equipo**, y otra clase llamada **Jugador**.

En la clase **Equipo** se declaró un atributo privado, el atributo es de tipo String llamado nombre, otro atributo es público cumple con a función de llamar a la clase llamada **Jugador** la cual tiene como nombre capitan, y el otro atributo cumple con la función de multiplicidad de tipo Treeset la cual cumple con llamar a la otra clase creada llamada Jugador con el nombre jugadores. Se crea un método de importación llamada java.util.Treeset, y con ello poder generar la multiplicidad correctamente a esa clase.

En la clase Jugador se declaró un atributo privado, el atributo es de tipo String llamado nombre. Y los otros dos atributos es públicos que cumplen con la función de llamar a la otra clase creada llamada Equipo con el nombre de capitanes y jueganEn.

De esta menara se realiza una asociación 1-* uno a muchos y de *-1 muchos a N.

Enunciado #12

Generar un programa haciendo uso de las buenas prácticas de la POO que nos permita analizar y representar qué es una asociación unidireccional.

Diagramas de caso de uso

N/A.

Diagramas de clases.	Proyecto.
<p> <pre> classDiagram class Gente { +String nombre } class Equipo { +String nombre +Jugador capitan +TreeSet Jugador jugadores } class Jugador { +String nombre +Equipo capitanes +Equipo jueganEn } Equipo "1" -- "*" Jugador : ~esQueridaPor Equipo "1" -- "*" Jugador : ~quiereA </pre> </p>	

Código Fuente

```
Gente.java X
1 // Santiago Reyes.
2
3 import java.util.TreeSet;
4
5 public class Gente {
6
7     private String nombre;
8     TreeSet<Gente> esQueridaPor = new TreeSet<>();
9     TreeSet<Gente> quiereA = new TreeSet<>();
10
11 }
```

Pruebas

N/A.

Análisis de resultados

Se creo un proyecto java llamado **AppAsociacion7**, en la cual se creó una clase una llamada **Gente**.

En la clase **Gente** se declaró un atributo privado, el atributo es de tipo String llamado nombre, y los otros atributos cumple con la función de multiplicidad de tipo TreeSet la cual cumple con llamar a la misma clase Gente con los nombres esQueridaPor y quiereA. Se crea un método de importación llamada java.util.Treeset, y con ello poder generar la multiplicidad correctamente a esa clase.

De esta menara se realiza una asociación 1-* uno a muchos y de *-1 muchos a N.