PRÁCTICA

CURSO	:	***Algorítmica II***
No.	:	*Práctica No.05*
TEMA	:	*Polimorfismo - Aseguradora*
DURACIÓN ESTIMADA	:	*01:40 horas.*

I. OBJETIVOS

La presente práctica tiene por objetivos:

- Utilizar oLoop para la especificación de pseudocódigo.
- Desarrollar una solución aplicando clases y herencia.
- Aplicar los conceptos de métodos de búsqueda por dispersión

II. RESUMEN

Debe desarrollar una aplicación en oLoop que se encargue de administrar la información de una Aseguradora. Considere que debe implementar la jerarquía de clases de la planilla.

III. CONSTRUCCIÓN DE LA APLICACIÓN

1. Crear la Clase CSeguro

Iniciamos la solución declarando la clase CSeguro que representa la información relevante del seguros.

```
Clase CSeguro
   Atributos
      poliza
      nombre
      prima
      planSeguro
      cobertura
   Metodos
      constructor(nPoliza, cNombre, nPrima, cPlan)
      obtenerPoliza()→
      obtenerNombre() →
      obtenerPrima() →
      obtenerPlanSeguro() →
      obtenerCobertura() →
      colocarPoliza( nPoliza )
      colocarNombre( cNombre )
      colocarPrima( nPrima )
      colocarPlanSeguro( cPlan )
      colocarCobertura( nCobertura )
      calcularCobertura()virtual abstracto
fClase
```

2. Especificar el cuerpo de los métodos de CSeguro

Una vez que hemos declarado la clase CSeguro, especificamos el cuerpo de cada método. A continuación ilustramos el uso de oLoop para algunos de los principales métodos.

```
Metodo CSeguro.constructor(nPoliza, cNombre, nPrima, cPlan)
   poliza ← nPoliza
```

```
   nombre ← cNombre
   prima ← nPrima
   planSeguro ← cPlan
   cobertura ← 0
fMetodo

Metodo CSeguro.colocarPoliza( nPoliza )
   poliza ← nPoliza
fMetodo

Metodo CSeguro.obtenerPoliza()→
   retornar poliza
fMetodo

Metodo CSeguro.colocarNombre( cNombre )
   nombre ← cNombre
fMetodo

Metodo CSeguro.obtenerNombre()→
   retornar nombre
fMetodo

Metodo CSeguro.colocarPrima( nPrima )
   prima ← nPrima
fMetodo

Metodo CSeguro.obtenerPrima ()→
   retornar prima
fMetodo

Metodo CSeguro.colocarPlanSeguro( cPlan )
   planSeguro ← cPlanSeguro
fMetodo

Metodo CSeguro.obtenerPlanSeguro() →
   retornar planSeguro
fMetodo

Metodo CSeguro.colocarCobertura( nCobertura )
   cobertura ← nCobertura
fMetodo

Metodo CSeguro.obtenerCobertura()→
   retornar cobertura
fMetodo
```

3. Crear mediante herencia la clase CVivienda

A continuación especificamos la clase CVivienda, es una subclase de la clase CSeguro.

```
//Clase CVivienda
Clase CVivienda viene-de CSeguro
   Atributos
      ubicacion
      precio
      porcentaje
   Metodos
      constructor( nPoliza, cNombre, nPrima, cPlan, cUbicacion, nPrecio,
   nPorcentaje)
      obtenerUbicacion()→
      colocarUbicacion( cUbicacion )
      obtenerPrecio()→
      colocarPrecio( nPrecio )
```

```
      obtenerPorcentaje()→
      colocarPorcentaje( nPorcentaje )
      calcularCobertura( )  sobrescribir
fClase
```

4. Especificar el cuerpo de los métodos de CVivienda

Declarado la clase CVivienda especificamos el cuerpo de cada método. A continuación ilustramos el uso de oLoop para algunos de los principales métodos.

```
Metodo CVivienda. constructor( nPoliza, cNombre, nPrima, cPlan, cUbicacion,
                                                          nPrecio, nPorcentaje)
        heredado constructor(nPoliza, cNombre, nPrima, cPlan)
        ubicación ← cUbicacion
        porcentaje ← nPorcentaje
        precio ← nPrecio
fMetodo

Metodo CVivienda.colocarUbicacion( cUbicacion )
        diasFalta ← cUbicacion
fMetodo

Metodo CVivienda.obtenerUbicacion() →
     retornar ubicacion
fMetodo

Metodo CVivienda.colocarPrecio( nPrecio )
        precio ← nPrecio
fMetodo

Metodo CVivienda.obtenerPrecio()→
     retornar precio
fMetodo

Metodo CVivienda.colocarPorcentaje( nPorcentaje )
        porcentajes ← nPorcentaje
fMetodo

Metodo CVivienda.obtenerPorcentaje() →
     retornar porcentaje
fMetodo

Metodo CVivienda.calcularCobertura()
     colocarCobertura( precio * porcentaje)
fMetodo
```

5. Crear mediante herencia la clase CMedico

A continuación especificamos la clase CMedico, es una subclase de la clase CSeguro.

```
//Clase CMedico
Clase CMedico viene-de CSeguro
   Atributos
      edad
      salud
   Metodos
      constructor( nPoliza, cNombre, nPrima, cPlan, nEdad, cSalud )
      colocarEdad( nEdad )→
      obtenerEdad() →
      colocarSalud( cSalud )→
      obtenerSalud() →
      calcularCobertura( ) sobrescribir
```

```
fClase
```

6. Especificar el cuerpo de los métodos de CMedico

Declarado la clase CMedico especificamos el cuerpo de cada método. A continuación ilustramos el uso de oLoop para algunos de los principales métodos.

```
Metodo CMedico.constructor(nPoliza, cNombre, nPrima, cPlan, nEdad, cSalud )
        heredado constructor(nPoliza, cNombre, nPrima, nPlan)
        edad ← nEdad
        salud ← cSalud
fMetodo

Metodo CMedico.colocarEdad( nEdad )
        edad ← nEdad
fMetodo

Metodo CMedico.obtenerEdad() →
        retornar edad
fMetodo

Metodo CMedico.colocarSalud( cSalud )
        salud ← cSalud
fMetodo

Metodo CMedico.obtenerSalud() →
        retornar salud
fMetodo

Metodo CMedico.calcularCobertura( )
   Si (edad > 80) entonces
      colocarCobertura( obtenerPrima() * 100)
   sino
      Si (edad > 60) entonces
         colocarCobertura( obtenerPrima() * 200)
      Sino
         Si (edad > 40 ) entonces
            colocarCobertura( obtenerPrima() * 300)
         Sino
            colocarCobertura( obtenerPrima() * 400)
         fSi
      fSi
   fSi
   Si (salud = "normal") entonces
      colocarCobertura( obtenerCobertura() * 2)
   Sino
      colocarCobertura( obtenerCobertura() / 2)
   fSi
fMetodo
```

7. Especificar la clase de la Interfaz

En las etapas anteriores, hemos concluido la especificación de las clases de la aplicación. En esta etapa de la especificación de la Interfaz del usuario, nos corresponde crear la clase de la Interfaz, la cual utilizará objetos de las clases de la jerarquía de clases de desarrollada.

```
Clase CAseguradora viene-de CFormulario
   Atributos
      // Arreglo de objetos
```

```
        arreglo seguros

        // Botones de Acción
        Objeto agregarBotón ejemplar-de CBotónAcción
        Objeto mostrarBotón ejemplar-de CBotónAcción
        Objeto buscarBotón ejemplar-de CBotónAcción

        // Campos de Texto
        Objeto polizaTexto ejemplar-de CTexto
        Objeto nombreTexto ejemplar-de CTexto
        Objeto primaTexto ejemplar-de CTexto
        Objeto planTexto ejemplar-de CTexto
        Objeto ubicacionTexto ejemplar-de CTexto
        Objeto precioTexto ejemplar-de CTexto
        Objeto porcentajeTexto ejemplar-de CTexto
        Objeto edadTexto ejemplar-de CTexto
        Objeto saludTexto ejemplar-de CTexto

        // Botones de Acción
        Objeto viviendaBotón ejemplar-de CBotónRadio
        Objeto medicoBotón ejemplar-de CBotónRadio

    Métodos
        Constructor( nTamano )
        // botones de acción para el arreglo
        agregarBotónAlClic()
        mostrarBotónAlClic()
        buscarBotónAlClic() →

fClase
```

8. Especificar los métodos de la clase Interfaz

En esta epata, especificamos el código de aquellos métodos que se ejecutarán en respuesta a los eventos de la interfaz del usuario. A continuación ilustramos algunos de ellos.

```
// En el constructor
Método CAseguradora.Constructor( nTamano )
     seguros dimension [1..nTamano] contiene CSeguro
     viviendaBoton.estado ← verdadero
fMétodo

// En el método de respuesta a la acción del botón Agregar
Método CAseguradora.agregarBotónAlClic() →
     poliza ← polizaTexto.obtenerTexto()
     nombre ← polizaTexto.obtenerTexto()
     prima ← polizaTexto.obtenerTexto()
     plan ← planTexto.obtenerTexto()

     pos ← (poliza mod Longitud(seguros)) + 1
     Si (poliza = Seguros[pos].obtenerPoliza()) entonces
        mensaje ← "Ya se encuentra ingresado"
     Sino
        Si (Seguros[pos].obtenerPoliza() = Vacio)  entonces
           almacenar( pos )
        Sino
           fil ← pos + 1
           Mientras (poliza = Seguros[fil].obtenerPoliza()) y
                   (Seguros[fil].obtenerPoliza() = Vacio) y
                   (fil <> pos)) hacer
              fil ← fil  + 1
              SI (fil = Longitud(seguros) + 1)  entonces
```

```
                fil ← 1
              fSi
            fMientras
            Si (poliza = Seguros[fil].obtenerPoliza()) entonces
              mensaje ← "Ya se encuentra ingresado"
            Sino
                Si (Seguros[fil].obtenerPoliza() = Vacio)  entonces
                  almacenar( fil )
                Sino
                  mensaje ← "No hay espacio donde adicionar elementos"
                fSi
            fSi
          fSi
      fSi
      retornar mensaje
fMétodo

Método CAseguradora.almacenar( pos )
   Si (viviendaBoton.estado() ) entonces

      ubicación ← ubicacionTexto.obtenerTexto()
      precio ←  precioTexto.obtenerTexto()
      porcentaje ← porcentajeTexto.obtenerTexto()

      seguros[ pos ] = CVivienda.Constructor( poliza, nombre, prima, plan,
            ubicacion, precio,porcentaje )

   sino
      edad ← edadTexto.obtenerTexto()
      salud ← saludTexto.obtenerTexto()

      seguros[ pos ] = CVivienda.Constructor( poliza, nombre, prima, plan,
            edad, salud )
   fSi
   seguros[ ind ].calcularCobertura()

fMetodo

// En el método de respuesta a la acción del botón buscar
Método CAseguradora.buscarBotónAlClic() →
      poliza ← polizaTexto.obtenerTexto()
      pos ← (poliza mod Longitud(seguros)) + 1
      encontrado ← falso
      Si (poliza = Seguros[pos].obtenerPoliza()) entonces
         encontrado ← verdadero
      Sino
         Si (Seguros[pos].obtenerPoliza() = Vacio)  entonces
            encontrado ← falso
         Sino
            fil ← pos + 1
            Mientras (poliza = Seguros[fil].obtenerPoliza()) y
                     (Seguros[fil].obtenerPoliza() = Vacio) y
                     (fil <> pos)) hacer
               fil ← fil  + 1
               SI (fil = Longitud(seguros) + 1)  entonces
                  fil ← 1
               fSi
            fMientras
            Si (poliza = Seguros[fil].obtenerPoliza()) entonces
               encontrado ← verdadero
               pos ← fil
            Sino
               Encontrado  ← falso
            fSi
```

```
         fSi
      fSi
      Si encontrado entonces
         mensaje ← "Poliza: " +  seguros[pos].obtenerPoliza() +
                  " Nombre: " + seguros[pos].obtenerNombre() +
                  " Prima: " + seguros[pos].obtenerPrima() +
                  " PlanSeguro: " + seguros[pos].obtenerPlanSeguro()+
                  "Cobertura: " + seguros[pos]).obtenerCobertura()
      Sino
         mensaje ← "No se encuentra"
      fSi
      retornar mensaje
fMétodo


// En el método de respuesta a la acción del botón Mostrar
Método CAseguradora.mostrarBotónAlClic()
      cadena ← ""
      Para  i  desde  1  hasta  Longitud(seguros)  hacer
         Si (seguros[i] <> nulo) entonces
            cadena ← cadena +
                  " Poliza: " +  seguros[i].obtenerPoliza() +
                  " Nombre: " + seguros[i].obtenerNombre() +
                  " Prima: " + seguros[i].obtenerPrima() +
                  " Plan: " + seguros[i].obtenerPlanSeguro()

            Si (seguros[i] esClase CVivienda) entonces
                  cadena ← cadena + " Ubicacion: " +
                     ((CVivienda) seguros[i]).obtenerUbicacion() +
                     " Precio: " +
                     ((CVivienda) seguros[i]).obtenerPrecio() +
                     " porcentaje: " +
                     ((CVivienda) seguros[i]).obtenerPorcentaje()
            Sino
                  cadena ← cadena + " Edad: " +
                              ((CMedico) seguros[i]).obtenerEdad() +
                              " Salud: " +
                              ((CMedico) seguros[i]).obtenerSalud()
            fSi
            cadena ← cadena + " Cobertura: " + seguros[i]).obtenerCobertura()
         fSi
      fPara
      escribir cadena
fMétodo
```