PRÁCTICA

CURSO	:	***Algorítmica II***
No.	:	*Práctica Dirigida No.04*
TEMA	:	*Clases, Herencia y Búsqueda – Planilla de personal*
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.

II. RESUMEN

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

III. CONSTRUCCIÓN DE LA APLICACIÓN

1. Crear la Clase CTrabajador

Iniciamos la solución declarando la clase CTrabajador que representa la información relevante del personal.

```
Clase CTrabajador
   Atributos
      codigo
      nombre
      basico
      retenciones
      neto
   Metodos
      constructor(nCodigo, sNombre, basico)
      obtenerCodigo()→
      obtenerNombre() →
      obtenerBasico() →
      obtenerRetenciones() →
      obtenerNeto() →
      colocarCodigo( nCodigo )
      colocarNombre( sNombre )
      colocarBasico( nBasico )
      colocarRetenciones( nRetenciones )
      colocarNeto( nNeto )
      calcularRetenciones( nTasa )
      calcularNeto()virtual abstracto
fClase
```

2. Especificar el cuerpo de los métodos de CTrabajador

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

```
Metodo CTrabajador.constructor(nCodigo, sNombre, nBasico)
   codigo ← nCodigo
   nombre ← sNombre
   basico ← nBasico
```

```
   retenciones ← 0
   neto ← 0
fMetodo

Metodo CTrabajador.colocarCodigo( nCodigo )
   codigo ← nCodigo
fMetodo

Metodo CTrabajador.obtenerCodigo()→
   retornar codigo
fMetodo

Metodo CTrabajador.colocarNombre( sNombre )
   nombre ← sNombre
fMetodo

Metodo CTrabajador.obtenerNombre()→
   retornar nombre
fMetodo

Metodo CTrabajador.colocarBasico( nBasico )
   basico ← nBasico
fMetodo

Metodo CTrabajador.obtenerBasico ()→
   retornar basico
fMetodo

Metodo CTrabajador.colocarRetenciones( nRetenciones )
   retenciones ← nRetenciones
fMetodo

Metodo CTrabajador.obtenerRetenciones() →
   retornar retenciones
fMetodo

Metodo CTrabajador.colocarNeto( nNeto )
   neto ← nNeto
fMetodo

Metodo CTrabajador.obtenerNeto()→
   retornar neto
fMetodo

Metodo CTrabajador.calcularRetenciones( nTasa )
   retenciones ← basico * nTasa
fMetodo
```

3. Crear mediante herencia la clase CEmpleado

A continuación especificamos la clase CEmpleado, es una subclase de la clase CTrabajador.

```
//Clase CEmpleado
Clase CEmpleado viene-de CTrabajador
   Atributos
      diasFaltas
      descuentos
      horasExtras
   Metodos
      constructor( nCodigo, sNombre, nBasico, nDias, nHoras)
      obtenerDiasFalta()→
      colocarDiasFalta( nDias )
      obtenerDescuentos()→
      colocarDescuentos( nDescuento )
```

```
      obtenerHorasExtras()→
      colocarHorasExtras( nHoras )
      calcularDescuento()
      calcularSueldoHoraExtra() →
      calcularNeto( )  sobrescribir
fClase
```

4. Especificar el cuerpo de los métodos de CEmpleado

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

```
Metodo CEmpleado. constructor( nCodigo, sNombre, nBasico, nDias, nHoras)
        heredado constructor(nCodigo, sNombre, nBasico)
        diasFaltas ← nDias
        horasExtras ← nHoras
        descuentos ← 0
fMetodo

Metodo CEmpleado.colocarDiasFalta( nDias )
        diasFalta ← nDias
fMetodo

Metodo CEmpleado.obtenerDiasFalta() →
     retornar diasFalta
fMetodo

Metodo CEmpleado.colocarDescuentos( nDescuentos )
        descuentos ← nDescuentos
fMetodo

Metodo CEmpleado.obtenerDescuentos()→
     retornar descuentos
fMetodo

Metodo CEmpleado.colocarHorasExtras( nHoras )
        horasExtras ← nHoras
fMetodo

Metodo CEmpleado.obtenerHorasExtras() →
     retornar horasExtras
fMetodo

Metodo CEmpleado.calcularDescuento()
    descuentos ← (obtenerBasico()/30)*diasFalta
fMetodo

Metodo CEmpleado.calcularSueldoHoraExtra() →
    retornar ((obtenerBasico() / 30) / 8 ) * 1.5
fMetodo

Metodo CEmpleado.calcularNeto()
    colocarNeto( obtenerBasico() – obtenerRetenciones() – descuentos +
                                  horasExtras * calcularSueldoHoraExtra() )
fMetodo
```

5. Crear mediante herencia la clase CConsultor

A continuación especificamos la clase CConsultor, es una subclase de la clase CTrabajador.

```
//Clase CConsultor
Clase CConsultor viene-de CTrabajador
```

```
   Atributos
      bonificación
   Metodos
      constructor( nCodigo, sNombre, nBasico, nBonificacion )
      obtenerBonificacion()→
      obtenerBonificacion() →
      calcularNeto( ) sobrescribir
fClase
```

6. Especificar el cuerpo de los métodos de CConsultor

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

```
Metodo CConsultor.constructor(nCodigo, sNombre, nBasico, nBonificacion )
        heredado constructor(nCodigo, sNombre, nBasico)
        benificacion ← nBonificacion
fMetodo

Metodo CConsultor.colocarBonificacion( nBonficacion )
        bonificacion ← nBonificacion
fMetodo

Metodo CConsultor.obtenerBonificacion() →
        retornar bonificacion
fMetodo

Metodo CConsultor.calcularNeto( )
        colocarNeto(obtenerBasico()- obtenerRetenciones()+ bonificacion)
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 CPlanilla viene-de CFormulario
   Atributos
      // Arreglo de objetos
      arreglo personal dimension 1..10  contiene CTrabajador
      // Contador de elementos del arreglo
      ind

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

      // Campos de Texto
      Objeto codigoTexto ejemplar-de CTexto
      Objeto nombreTexto ejemplar-de CTexto
      Objeto basicoTexto ejemplar-de CTexto
      Objeto diasFaltaTexto ejemplar-de CTexto
      Objeto horasExtraTexto ejemplar-de CTexto
      Objeto bonificacionTexto ejemplar-de CTexto

      // Botones de Acción
      Objeto empleadoBotón ejemplar-de CBotónRadio
      Objeto consultorBotón ejemplar-de CBotónRadio
```

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

      // Botones de radio
      empleadoBotónAlClic()
      consultorBotónAlClic()
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 CPlanilla.Constructor()
      num = 0
      empleadoBoton.estado ← verdadero
fMétodo

// En los radio buton y casillas de verificación
Método CPlanilla.empleadoBotonAlclic()
      Si ( consultorBoton.estado ) entonces
         consultorBoton.estado ← falso
         empleadoBoton.estado ← verdadero
      fSi
fMétodo

Método CPlanilla.consultorBotonAlclic()
      Si ( empleadoBoton.estado ) entonces
         empleadoBoton.estado ← falso
         consultorBoton.estado ← verdadero
      fSi
fMétodo

// En el método de respuesta a la acción del botón Agregar
Método CPlanilla.agregarBotónAlClic()
      num = num + 1
      Proteger
            Si ( num > Longitud( personal) ) entonces
               provocar CRangoExcepcion.constructor("Arreglo lleno")
            Sino
                  Si ( empleadoBoton.estado() ) entonces
                     personal[ num ] = CEmpleado.Constructor(
                                    codigoTexto.obtenerTexto(),
                                    nombreTexto.obtenerTexto(),
                                    basicoTexto.obtenerTexto(),
                                    diasFaltaTexto.obtenerTexto(),
                                    horasExtraTexto.obtenerTexto()
                                    )
                  sino
                     personal[ num ] = CEmpleado.Constructor(
                                    codigoTexto.obtenerTexto(),
                                    nombreTexto.obtenerTexto(),
                                    basicoTexto.obtenerTexto(),
                                    bonificacionTexto.obtenerTexto()
                                    )
                  fSi
                  personal[ num ].calcularRetenciones()
                  personal[ num ].calcularNeto()
         fSi
```

```
      Excepcion
         Cuando rangoExcepcion ejemplar-de CRangoExcepcion
            rangoExcepcion.Mostrar()
         fCuando
      fProteger
fMétodo

// En el método de respuesta a la acción del botón Mostrar
Método CPlanilla.mostrarBotónAlClic()
      cadena ← ""
      Para  i  desde  1  hasta  Longitud(personal)  hacer
         Si (personal[i] <> nulo) entonces
            cadena ← cadena +
                  "Codigo: " +  personal[i].obtenerCodigo() +
                  " Nombre: " + personal[i].obtenerNombre() +
                  " Basico: " + personal[i].obtenerBasico() +
                  " Retenciones: " + personal[i].obtenerRetenciones()

            Si (personal[i] esClase CEmpleado) entonces
                  cadena ← cadena +
                        " Dias: " + ((CEmpleado) personal[i]).obtenerDiasFalta()
            Sino
                  cadena ← cadena + " Bonificacion: " +
                              ((CConsultor) personal[i]).obtenerBonificacion()
            fSi
            cadena ← cadena + "Neto: " + personal[i]).obtenerNeto()
         fSi
      fPara
      escribir cadena
fMétodo

// En el método de respuesta a la acción del botón Cerrar
Método CPlanilla.buscarBotónAlClic() →
      codigo ← codigoTexto.obtenerTexto()
      i ← 1
      encontrado ← falso
      Mientras (i <= Longitud(personal) y No encontrado) hacer
         Si (codigo <> personal[i].obtenerCodigo() ) entonces
            i ← i + 1
         Sino
            encontrado ← verdad
         fSi
      fpara
      Si encontrado entonces
         mensaje ← "Codigo: " +  personal[i].obtenerCodigo() +
                  " Nombre: " + personal[i].obtenerNombre() +
                  " Basico: " + personal[i].obtenerBasico() +
                  " Retenciones: " + personal[i].obtenerRetenciones()+
                  "Neto: " + personal[i]).obtenerNeto()
      Sino
         mensaje ← "No se encuentra"
      fSi
      retornar mensaje
fMétodo
```

9. Especificar la clase de Excepción

Debemos especificar la subclase que hereda de la clase Excepción.

```
// Declaración de la Clase CRangoException

Clase CRangoException viene-de CException
```

```
   Atributos
        mensaje
   Métodos
      Constructor( sMensaje )
      mostrar() →
fClase
```

10. Especificar los métodos de la clase Exception

En esta epata, especificamos el código del método que se ejecutarán en respuesta a la creación de los objetos.

```
// En el constructor

Método CRangoException.Constructor( sMensaje )
     mensaje  ← sMensaje
fMétodo

// En el método mostrar

Método CRangoException.Mostrar() →
     retornar  mensaje
fMétodo
```