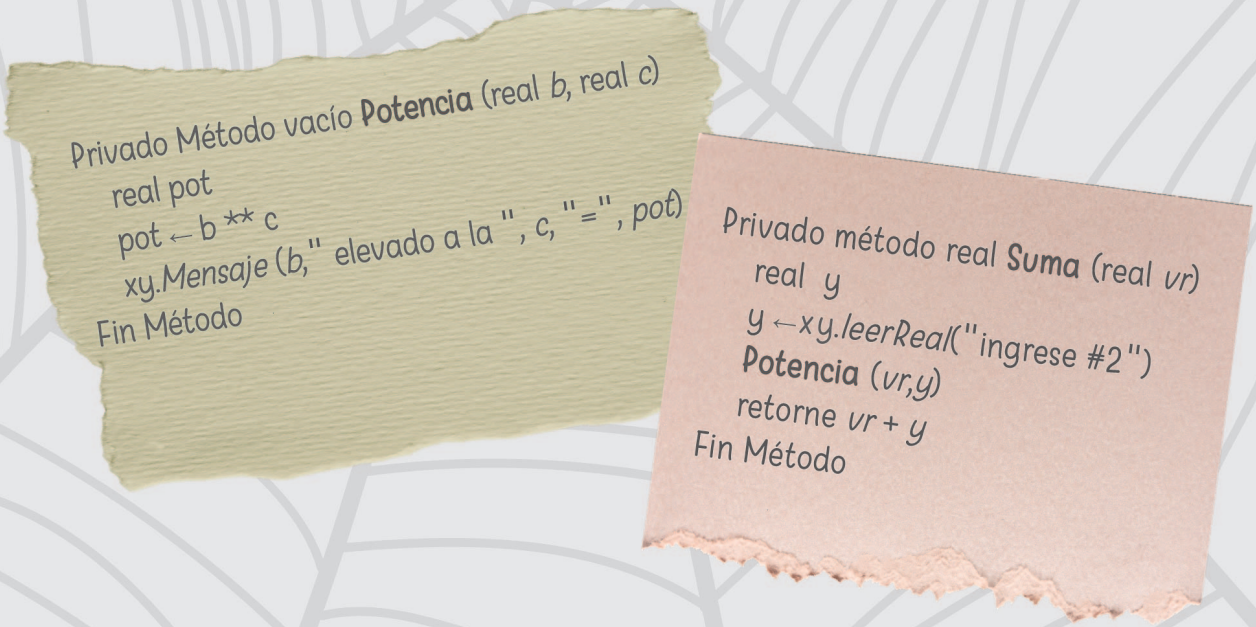
Institución Universitaria
Reacreditada en Alta Calidad

APÉNDICES

Lógica de programación básica orientada a objetos con ejercicios resueltos



Delio A. Aristizábal Martínez

Sandra M. Quiceno Metaute

APÉNDICES

APÉNDICE 1. Clase clsGenerales – Base

La clase *clsGenerales* inicial en UML es:

clsGenerales	
+	**Mensaje** (Cadena *Texto*) : vacío
+	**leerEntero** (Cadena *Texto*) : entero
+	**leerReal** (Cadena *Texto*) : real
+	**leerCadena** (Cadena *Texto*) : cadena
Constructor:	
+	**clsGenerales** ( )

La implementación de los anteriores métodos en la clase queda de la siguiente manera:

```
Público Clase clsGenerales
  Público clsGenerales ( )
      // Es el constructor, por el momento sin implementación alguna.
  Fin Método

  Público Método vacío Mensaje ( Cadena Texto )
    Muestre Texto
  Fin Método

  Público Método entero leerEntero ( Cadena Texto )
    Entero Dato
    Muestre Texto
    Lea Dato
    retorne Dato
  Fin Método
```

```
  Público Método real leerReal ( Cadena Texto )
    Real Dato
    Muestre Texto
    Lea Dato
    retorne Dato
  Fin Método

  Público Método cadena leerCadena ( Cadena Texto )
    Cadena Dato
    Muestre Texto
    Lea Dato
    retorne Dato
  Fin Método
Fin Clase
```

APÉNDICE 2. Clase clsGenerales – Reformada

La clase *clsGenerales* en su segunda presentación con UML es:

clsGenerales	
+	**Mensaje** (Cadena *Texto*) : vacío
+	**leerEntero** (Cadena *Texto*) : entero
+	**leerReal** (Cadena *Texto*) : real
+	**leerCadena** (Cadena *Texto*) : cadena
+	**leerEnteroPos** (Cadena *Texto*) : entero
+	**leerEnteroPosMy0** (Cadena *Texto*) : entero
+	**leerEnteroNeg** (Cadena *Texto*) : entero
+	**leerRealPos** (Cadena *Texto*) : real
+	**leerRealPosMy0** (Cadena *Texto*) : real
+	**leerRealNeg** (Cadena *Texto*) : real
+	**leerCadena2** (Cadena *Texto*) : cadena
+	**leerNota_05** (Cadena *Texto*) : real
Constructor:	
+	**clsGenerales** ( )

En esta reforma, se agregan los siguientes métodos utilizando el concepto de algoritmo recursivo:

- ✓ leerEnteroPos
- ✓ leerEnteroPosMy0
- ✓ leerEnteroNeg
- ✓ leerRealPos
- ✓ leerRealPosMy0
- ✓ leerRealNeg
- ✓ leerCadena2
- ✓ leerNota_05

El desarrollo de los anteriores métodos en la clase queda de la siguiente manera:

Los métodos: **Mensaje, leerEntero, leerReal, leerCadena,** están implementados en el apéndice 1.

```
Público Clase clsGenerales
  Público clsGenerales ( )  // Constructor
  Público Método vacío    Mensaje ( Cadena Texto )
  Público Método entero   leerEntero ( Cadena Texto )
  Público Método real     leerReal ( Cadena Texto )
  Público Método cadena   leerCadena ( Cadena Texto )

  Público Método entero leerEnteroPos ( Cadena Texto )
    Entero Dato
    Muestre Texto
    Lea Dato
    Si ( Dato >= 0 ) Entonces
      retorne Dato
    Sino
      Mensaje ("Valor no válido, debe ser entero mayor o igual que cero")
      retorne leerEnteroPos ( Texto )
    Fin Si
  Fin Método

  Público Método entero leerEnteroPosMy0 ( Cadena Texto )
    Entero Dato
    Muestre Texto
    Lea Dato
    Si ( Dato > 0 ) Entonces
      retorne Dato
    Sino
      Mensaje ("Valor no válido, debe ser Entero mayor que cero")
      retorne leerEnteroPosMy0 ( Texto )
    Fin Si
  Fin Método

  Público Método entero leerEnteroNeg ( Cadena Texto )
    Entero Dato
    Muestre Texto
    Lea Dato
    Si ( Dato < 0 ) Entonces
      retorne Dato
    Sino
      Mensaje ("Valor no válido, debe ser Entero menor que cero")
      retorne leerEnteroNeg ( Texto )
    Fin Si
  Fin Método
```

```
Público Método real leerRealPos ( Cadena Texto )
  Real Dato
  Muestre Texto
  Lea Dato
  Si ( Dato >= 0 ) Entonces
    retorne Dato
  Sino
    Mensaje ( "Valor no válido, debe ser Real mayor o igual que cero" )
    retorne leerRealPos ( Texto )
  Fin Si
Fin Método

Público Método real leerRealPosMy0  ( Cadena Texto )
  Real Dato
  Muestre Texto
  Lea Dato
  Si ( Dato > 0 ) Entonces
    retorne Dato
  Sino
    Mensaje ( "Valor no válido, debe ser Real mayor que cero" )
    retorne leerRealPosMy0 ( Texto )
  Fin Si
Fin Método

Público Método real leerRealNeg  ( Cadena Texto )
  Real Dato
  Muestre Texto
  Lea Dato
  Si ( Dato < 0 ) Entonces
    retorne Dato
  Sino
    Mensaje ( "Valor no válido, debe ser Real menor que cero" )
    retorne leerRealNeg ( Texto )
  Fin Si
Fin Método
```

```
  Público Método cadena leerCadena2  ( Cadena Texto )
    Cadena Dato
    Muestre Texto
    Lea Dato
    Si ( Dato ≠ "" ) Entonces
      retorne Dato
    Sino
      Mensaje ("Valor no válido, debe ser Cadena diferente de vacío o espacios")
      retorne leerCadena2  ( Texto )
    Fin Si
  Fin Método

  Público Método real leerNota_05  ( Cadena Texto )
    Real Dato
    Muestre Texto
    Lea Dato
    Si ( Dato  >=  0  Λ  Dato  <=  5 ) Entonces
      retorne Dato
    Sino
      Mensaje ("Valor no válido, debe ser Real mayor o igual que cero y menor o igual que 5")
      retorne leerNota_05 ( Texto )
    Fin Si
  Fin Método
Fin Clase
```

Ejemplo del código en lenguaje de programación JAVA:

```
  public static int leerEnteroPos ( String Texto )
  {
    int Dato = Integer.parseInt( JOptionPane.showInputDialog ( null, Texto ) );
    if ( Dato >= 0 )
      return Dato;
    else
    {
      Mensaje ("Valor no válido, debe ser mayor o igual a cero, reintente por favor");
      return leerEnteroPos ( Texto );
    }
  }
```

APÉNDICE 3. Clase clsVectorGral

Para la construcción de la clase *clsVectorGral* se utiliza la clase *clsGenerales* para poder utilizar sus diferentes métodos; además, se utiliza el ciclo tipo *Para*/*Hacer* para la captura de la información y la respectiva asignación al arreglo tipo unidimensional/vector o para recuperar la información contenida en él.

La clase *clsVectorGral* en UML es:

clsVectorGral	
+	**llenarEnteros** (Entero [ ] *Vect*, Entero *Tam*, Cadena *Texto*) : entero [ ]
+	**llenarEnterosPos** (Entero [ ] *Vect*, Entero *Tam*, Cadena *Texto*) : entero [ ]
+	**llenarEnterosPosMy0** (Entero [ ] *Vect*, Entero *Tam*, Cadena *Texto*) : entero [ ]
+	**llenarEnterosNeg** (Entero [ ] *Vect*, Entero *Tam*, Cadena *Texto*) : entero [ ]
+	**llenarReales** (Real [ ] *Vect*, Entero *Tam*, Cadena *Texto*) : real [ ]
+	**llenarRealesPos** (Real [ ] *Vect*, Entero *Tam*, Cadena *Texto*) : real [ ]
+	**llenarRealesPosMy0** (Real [ ] *Vect*, Entero *Tam*, Cadena *Texto*) : real [ ]
+	**llenarRealesNeg** (Real [ ] *Vect*, Entero *Tam*, Cadena *Texto*) : real [ ]
+	**llenarCadenas** (Cadena [ ] *Vect*, Entero *Tam*, Cadena *Texto*) : cadena [ ]
+	**llenarCadenas2** (Cadena [ ] *Vect*, Entero *Tam*, Cadena *Texto*) : cadena [ ]
+	**mostrarEnteros** (Entero [ ] *Vect*, Entero *Tam*, Cadena *Texto*) : vacío
+	**mostrarReales** (Real [ ] *Vect*, Entero *Tam*, Cadena *Texto*) : vacío
+	**mostrarCadenas** (Cadena [ ] *Vect*, Entero *Tam*, Cadena *Texto*) : vacío
+	**clsVectorGral** ( ) // *Constructor: (Constructor vacío)*

El desarrollo de los anteriores métodos en la clase queda de la siguiente manera:

```
Importar clsGenerales
Público Clase clsVectorGral
  // Variables y objetos globales
  Entero i
  Cadena Dato
  clsGenerales objG = nuevo clsGenerales ( )

  Público clsVectorGral ( )
      // Es el constructor, por el momento sin implementación alguna.
  Fin Método
```

```
Público Método entero [ ] llenarEnteros ( Entero [ ] Vect, Entero Tam, Cadena Texto )
  Para ( i ← 0 Hasta Tam – 1, 1 ) Hacer
    Dato ← "[", i, "] ", Texto, ": "
    Vect [ i ] ← objG.leerEntero ( Dato )
  Fin Para
  retorne Vect
Fin Método

Público Método entero [ ] llenarEnterosPos ( Entero [ ] Vect, Entero Tam, Cadena Texto )
  Para ( i ← 0 Hasta Tam – 1, 1 ) Hacer
    Dato ← "[", i, "] ", Texto, ": "
    Vect [ i ] ← objG.leerEnteroPos ( Dato )
  Fin Para
  retorne Vect
Fin Método

Público Método entero [ ] llenarEnterosPosMy0 ( Entero [ ] Vect, Entero Tam, Cadena Texto )
  Para ( i ← 0 Hasta Tam – 1, 1 ) Hacer
    Dato ← "[", i, "] ", Texto, ": "
    Vect [ i ] ← objG.leerEnteroPosMy0 ( Dato )
  Fin Para
  retorne Vect
Fin Método

Público Método entero [ ] llenarEnterosNeg ( Entero [ ] Vect, Entero Tam, Cadena Texto )
  Para ( i ← 0 Hasta Tam – 1, 1 ) Hacer
    Dato ← "[", i, "] ", Texto, ": "
    Vect [ i ] ← objG.leerEnteroNeg ( Dato )
  Fin Para
  retorne Vect
Fin Método

Público Método real [ ] llenarReales ( Real [ ] Vect, Entero Tam, Cadena Texto )
  Para ( i ← 0 Hasta Tam – 1, 1 ) Hacer
    Dato ← "[", i, "] ", Texto, ": "
    Vect [ i ] ← objG.leerReal ( Dato )
  Fin Para
  retorne Vect
Fin Método
```

```
Público Método real [ ] llenarRealesPos  ( Real [ ] Vect, Entero Tam, Cadena Texto )
  Para ( i ← 0 Hasta Tam – 1, 1 ) Hacer
    Dato ← "[", i, "] ", Texto, ": "
    Vect [ i ] ← objG.leerRealPos ( Dato )
  Fin Para
  retorne Vect
Fin Método

Público Método real [ ] llenarRealesPosMy0  ( Real [ ] Vect, Entero Tam, Cadena Texto )
  Para ( i ← 0 Hasta Tam – 1, 1 ) Hacer
    Dato ← "[", i, "] ", Texto, ": "
    Vect [ i ] ← objG.leerRealPosMy0 ( Dato )
  Fin Para
  retorne Vect
Fin Método

Público Método real [ ] llenarRealesNeg  ( Real [ ] Vect, Entero Tam, Cadena Texto )
  Para ( i ← 0 Hasta Tam – 1, 1 ) Hacer
    Dato ← "[", i, "] ", Texto, ": "
    Vect [ i ] ← objG.leerRealNeg ( Dato )
  Fin Para
  retorne Vect
Fin Método

Público Método cadena [ ] llenarCadenas ( Cadena [ ] Vect, Entero Tam, Cadena Texto )
  Para ( i ← 0 Hasta Tam – 1, 1 ) Hacer
    Dato ← "[", i, "] ", Texto, ": "
    Vect [ i ] ← objG.leerCadena ( Dato )
  Fin Para
  retorne Vect
Fin Método

Público Método cadena [ ] llenarCadenas2 ( Cadena [ ] Vect, Entero Tam, Cadena Texto )
  Para ( i ← 0 Hasta Tam – 1, 1 ) Hacer
    Dato ← "[", i, "] ", Texto, ": "
    Vect [ i ] ← objG.leerCadena2 ( Dato )
  Fin Para
  retorne Vect
Fin Método
```

```
  Público Método vacío mostrarEnteros ( Entero [ ] Vect, Entero Tam, Cadena Texto )
    Dato ← Texto
    Para ( i ← 0 Hasta Tam – 1, 1 ) Hacer
      Dato ← Dato, Vect [ i ], ", "
    Fin Para
    objG.Mensaje ( Dato )
  Fin Método

  Público Método vacío mostrarReales ( Real [ ] Vect, Entero Tam, Cadena Texto )
    Dato ← Texto
    Para ( i ← 0 Hasta Tam – 1, 1 ) Hacer
      Dato ← Dato, Vect [ i ], ", "
    Fin Para
    objG.Mensaje ( Dato )
  Fin Método

  Público Método vacío mostrarCadenas ( Cadena [ ] Vect, Entero Tam, Cadena Texto )
    Dato ← Texto
    Para ( i ← 0 Hasta Tam – 1, 1 ) Hacer
      Dato ← Dato, Vect [ i ], ", "
    Fin Para
    objG.Mensaje ( Dato )
  Fin Método
Fin Clase
```

APÉNDICE 4. Clase clsMatrizGral

Para la construcción de la clase *clsMatrizGral* se utilizan la clase *clsGenerales* para poder usar sus diferentes métodos; además, el ciclo tipo *Para*/*Hacer* para la captura de la información y la respectiva asignación al arreglo tipo bidimensional/matriz o para recuperar la información contenida en él.

La clase *clsMatrizGral* en UML es:

clsMatrizGral	
+	**llenarEnteros** (Entero [ ] [ ] *Mat*, Entero *kFil*, Entero *kCol*, Cadena *Texto*) : entero [ ] [ ]
+	**llenarEnterosPos** (Entero [ ] [ ] *Mat*, Entero *kFil*, Entero *kCol*, Cadena *Texto*) : entero [ ] [ ]
+	**llenarEnterosPosMy0** (Entero [ ] [ ] *Mat*, Entero *kFil*, Entero *kCol*, Cadena *Texto*) : entero [ ] [ ]
+	**llenarEnterosNeg** (Entero [ ] [ ] *Mat*, Entero *kFil*, Entero *kCol*, Cadena *Texto*) : entero [ ] [ ]
+	**llenarReales** (Real [ ] [ ] *Mat*, Entero *kFil*, Entero *kCol*, Cadena *Texto*) : real [ ] [ ]
+	**llenarRealesPos** (Real [ ] [ ] *Mat*, Entero *kFil*, Entero *kCol*, Cadena *Texto*) : real [ ] [ ]
+	**llenarRealesPosMy0** (Real [ ] [ ] *Mat*, Entero *kFil*, Entero *kCol*, Cadena *Texto*) : real [ ] [ ]
+	**llenarRealesNeg** (Real [ ] [ ] *Mat*, Entero *kFil*, Entero *kCol*, Cadena *Texto*) : real [ ] [ ]
+	**llenarCadenas** (Cadena [ ] [ ] *Mat*, Entero *kFil*, Entero *kCol*, Cadena *Texto*) : cadena [ ] [ ]
+	**llenarCadenas2** (Cadena [ ] [ ] *Mat*, Entero *kFil*, Entero *kCol*, Cadena *Texto*) : cadena [ ] [ ]
+	**mostrarEnteros** (Entero [ ] [ ] *Mat*, Entero *kFil*, Entero *kCol*, Cadena *Texto*) : vacío
+	**mostrarReales** (Real [ ] [ ] *Mat*, Entero *kFil*, Entero *kCol*, Cadena *Texto*) : vacío
+	**mostrarCadenas** (Cadena [ ] [ ] *Mat*, Entero *kFil*, Entero *kCol*, Cadena *Texto*) : vacío
+	*Constructor:* **clsMatrizGral** ( ) (Constructor vacío)

La implementación de los anteriores métodos en la clase queda de la siguiente manera:

```
Importar clsGenerales
Público Clase clsMatrizGral
  // Variables y objetos globales
  Entero i, j
  clsGenerales objG = nuevo clsGenerales ( )

  Público clsMatrizGral ( )
      // Es el constructor, por el momento sin implementación alguna.
  Fin Método
```

```
Público Método entero [ ][ ] llenarEnteros ( Entero [ ][ ] Mat, Entero kF, Entero kC, Cadena Texto )
  Para ( i ← 0 Hasta kF – 1, 1 ) Hacer
    Para ( j ← 0 Hasta kC – 1, 1 ) Hacer
      Mat [ i ][ j ] ← objG.leerEntero ( "[", i, "][", j, "] :", Texto )
    Fin Para
  Fin Para
  retorne Mat
Fin Método

Público Método entero [ ][ ] llenarEnterosPos ( Entero [ ][ ] Mat, Entero kF, Entero kC, Cadena Texto )
  Para ( i ← 0 Hasta kF – 1, 1 ) Hacer
    Para ( j ← 0 Hasta kC – 1, 1 ) Hacer
      Mat [ i ][ j ] ← objG.leerEnteroPos ( "[", i, "][", j, "] :", Texto )
    Fin Para
  Fin Para
  retorne Mat
Fin Método

Público Método entero [ ][ ] llenarEnterosPosMy0 ( Entero [ ][ ] Mat, Entero kF, Entero kC, Cadena Texto )
  Para ( i ← 0 Hasta kF – 1, 1 ) Hacer
    Para ( j ← 0 Hasta kC – 1, 1 ) Hacer
      Mat [ i ][ j ] ← objG.leerEnteroPosMy0 ( "[", i, "][", j, "] :", Texto )
    Fin Para
  Fin Para
  retorne Mat
Fin Método

Público Método entero [ ][ ] llenarEnterosNeg ( Entero [ ][ ] Mat, Entero kF, Entero kC, Cadena Texto )
  Para ( i ← 0 Hasta kF – 1, 1 ) Hacer
    Para ( j ← 0 Hasta kC – 1, 1 ) Hacer
      Mat [ i ][ j ] ← objG.leerEnteroNeg ( "[", i, "][", j, "] :", Texto )
    Fin Para
  Fin Para
  retorne Mat
Fin Método

Público Método real [ ][ ] llenarReales ( Real [ ][ ] Mat, Entero kF, Entero kC, Cadena Texto )
  Para ( i ← 0 Hasta kF – 1, 1 ) Hacer
    Para ( j ← 0 Hasta kC – 1, 1 ) Hacer
      Mat [ i ][ j ] ← objG.leerReal ( "[", i, "][", j, "] :", Texto )
    Fin Para
  Fin Para
  retorne Mat
Fin Método
```

```
Público Método real [ ][ ] llenarRealesPos  ( Real [ ][ ] Mat, Entero kF, Entero kC, Cadena Texto )
  Para ( i ← 0 Hasta kF – 1, 1 ) Hacer
    Para ( j ← 0 Hasta kC – 1, 1 ) Hacer
      Mat [ i ][ j ] ← objG.leerRealPos ( "[", i, "][", j, "] :", Texto )
    Fin Para
  Fin Para
  retorne Mat
Fin Método

Público Método real [ ][ ] llenarRealesPosMy0  ( Real [ ][ ] Mat, Entero kF, Entero kC, Cadena Texto )
  Para ( i ← 0 Hasta kF – 1, 1 ) Hacer
    Para ( j ← 0 Hasta kC – 1, 1 ) Hacer
      Mat [ i ][ j ] ← objG.leerRealPosMy0 ( "[", i, "][", j, "] :", Texto )
    Fin Para
  Fin Para
  retorne Mat
Fin Método

Público Método real [ ][ ] llenarRealesNeg  ( Real [ ][ ] Mat, Entero kF, Entero kC, Cadena Texto )
  Para ( i ← 0 Hasta kF – 1, 1 ) Hacer
    Para ( j ← 0 Hasta kC – 1, 1 ) Hacer
      Mat [ i ][ j ] ← objG.leerRealNeg ( "[", i, "][", j, "] :", Texto )
    Fin Para
  Fin Para
  retorne Mat
Fin Método

Público Método cadena [ ][ ] llenarCadenas ( Cadena [ ][ ] Mat, Entero kF, Entero kC, Cadena Texto )
  Para ( i ← 0 Hasta kF – 1, 1 ) Hacer
    Para ( j ← 0 Hasta kC – 1, 1 ) Hacer
      Mat [ i ][ j ] ← objG.leerCadena ( "[", i, "][", j, "] :", Texto )
    Fin Para
  Fin Para
  retorne Mat
Fin Método

Público Método cadena [ ][ ] llenarCadenas2 ( Cadena [ ][ ] Mat, Entero kF, Entero kC, Cadena Texto )
  Para ( i ← 0 Hasta kF – 1, 1 ) Hacer
    Para ( j ← 0 Hasta kC – 1, 1 ) Hacer
      Mat [ i ][ j ] ← objG.leerCadena2 ( "[", i, "][", j, "] :", Texto )
    Fin Para
  Fin Para
  retorne Mat
Fin Método
```

```
  Público Método vacío mostrarEnteros ( Entero [ ][ ] Mat, Entero kF, Entero kC, Cadena Texto )
    Cadena totMat ← Texto, Fila
    Para ( i ← 0 Hasta kF – 1, 1 ) Hacer
      Fila ← ""
      Para ( j ← 0 Hasta kC – 1,1 ) Hacer
        Fila ← Fila, Mat [ i ][ j ] , ",    "
      Fin Para
      totMat ← totMat, Fila
    Fin Para
    objG.Mensaje ( totMat )
  Fin Método

  Público Método vacío mostrarReales ( Real [ ][ ] Mat, Entero kF, Entero kC, Cadena Texto)
    Cadena totMat ← Texto, Fila
    Para ( i ← 0 Hasta kF – 1, 1 ) Hacer
      Fila ← ""
      Para ( j ← 0 Hasta kC – 1, 1 ) Hacer
        Fila ← Fila, Mat [ i ][ j ] , ",    "
      Fin Para
      totMat ← totMat, Fila
    Fin Para
    objG.Mensaje ( totMat )
  Fin Método

  Público Método vacío mostrarCadenas ( Cadena [ ][ ] Mat, Entero kF, Entero kC, Cadena Texto )
    Cadena totMat ← Texto, Fila
    Para ( i ← 0 Hasta kF – 1, 1 ) Hacer
      Fila ← ""
      Para ( j ← 0 Hasta kC – 1, 1 ) Hacer
        Fila ← Fila, Mat [ i ][ j ] , ",    "
      Fin Para
      totMat ← totMat, Fila
    Fin Para
    objG.Mensaje ( totMat )
  Fin Método
Fin Clase
```

APÉNDICE 5. Respuestas a repasos

Las respuestas a los diferentes repasos son:

Repaso 2.1.

1). Una posible solución de orden sería:

Público Clase *repaso2_1*
  Público Método vacío **Principal** ( )
    10. Cenar a las 10:40 p. m.
    6. Salir del trabajo a las 5:00 p. m.
    8. Repasar estudio hasta las 6:00 p. m.
    3. Dirigirse al trabajo a las 7:15 a. m.
    4. Entrar al trabajo a las 8:00 a. m.
    12. Dormir hasta las 6:00 a. m.
    5. Almorzar de 12:30 p. m. a 1:30 p. m. y lavarse los dientes.
    7. Dirigirse a la universidad.
    2. Bañarse, vestirse, desayunar y lavarse dientes.
    9. Regresar a casa a las10:15 p. m.
    1. Levantarse a las 6:00 a. m.
    11. Repasar estudio hasta las 12:00 noche.
  Fin Método

Repaso 2.2.

1). Valores en prueba de escritorio:

x ? 6.25

y ? 0

z ? 144

m ? Verdadero

2). Leer una velocidad en kilómetros por hora; convertirla y mostrarla en metros por segundo.

3). Una posible solución de orden sería:

```
Público Clase repaso2_2C
  Público Método vacío Principal ( )
   6. Lea H
   9. Muestre "Valor del Área: ", A,  ", Valor del volumen: ", V
   2. Real R ← 0, H ← 0, V ← 0, A ← 0
   3. Escriba "Ingrese el valor del Radio: "
   8. V ← PI * R**2 * H
   5. Escriba "Ingrese el valor de la Altura: "
   1. Const Real PI ← 3.1415926
   7. A ← PI * R * H
   4. Lea R
  Fin Método
Fin Clase
```

Repaso 3.1.

1). Errores en el algoritmo

Línea	Error

2. Falta el parámetro de tipo Entero con nombre *Cod*, según línea 6 y la línea 24.

5. Expresión no válida por jerarquía de operadores para realizar el proceso de hallar la nota definitiva, la correcta es: *Nd* ← (*N1* + *N2* + *N3*) / 3

23. No captura *Nota3*, lo hace sobre la *Not1*; ocasionando el sobreescribir *Not1* y *Not3* queda vacío.

2). Un posible enunciado sería:

Cree un algoritmo para la empresa de servicios públicos que, cobra el metro cú bico a $X pesos y el kilovatio/hora a $Y. Para determinar el valor total por pagar y por cada concepto en una vivienda, se realiza con base en la diferencia entre la lectura actual y la lectura anterior de cada uno de los servicios y una tarifa fija por cada servicio. Cree la clase repaso3_1B con el método pagoVivienda sin argumentos de entrada que realice todo el proceso para obtener lo solicitado

Repaso 3.2.

1). No existe error alguno.

2). Una posible solución de orden sería:

20. Lea *porcIva*
4. Fin Método
9. **Facturar** ( )
11. Privado Método vacío **Facturar** ( )
14. Lea *Cant*
22. *vrDscto* ← *vrSubTotal* * *porcDscto* / *100*
1. Público Clase *repaso3_2B*
15. Muestre "Ingrese el valor unitario: "
10. Fin Método
3. retorne *subTotal* * *porcImpto* / 100
16. Lea *vrUnit*
24. *vrAPag* ← **aPagar** (*vrSubTotal*, *vrDscto*, *vrIva*)
7. Fin Método
28. " Valor a Pagar = ", *vrAPag*
5. Privado Método Real **aPagar** (Real *subTotal*, Real *vrDscto*, Real *impIva*)
13. Muestre "Ingrese la cantidad: "
6. retorne *subTotal* – *vrDscto* + *impIva*
25. Muestre "Valor a pagar sin descuento: ", *vrSubTotal*,
29. Fin Método
17. Muestre "Ingrese el porcentaje del descuento: "
27. "Valor IVA: ", *vrIva*,
2. Privado Método Real **hallarIva** (Real *subTotal*, Real *porcImpto*)
19. Muestre "Ingrese el porcentaje del IVA: "
30. Fin Clase
18. Lea *porcDscto*
21. *vrSubTotal* ← *Cant* * *vrUnit*
8. Público Método vacío **Principal** ( )
23. *vrIva* ← **hallarIva** (*vrSubTotal* – *vrDscto*, *porcIva*)
12. Real *Cant*, *vrUnit*, *porcDscto* ← 0, *porcIva* ← 0, *vrDscto*, *vrSubTotal*, *vrIva*, *vrAPag*
26. "Valor descuento: ", *vrDscto*,

Repaso 4.1.

1). Que al evaluar la expresión de *Condición* la respuesta sea *verdadero.*

Repaso 4.2.

1). Dados tres números enteros *n1*, *n2*, *n3*, determine si la suma de los dos primeros (*n1*, *n2*) es igual al tercer número (*n3*). Si se cumple esta condición, escriba "Los dos primeros suman el tercero". Cree la clase *repaso4_2A* para el algoritmo con la solución.

2). Las condiciones faltantes serían:

C1? *Hora* >= 18

C2? *Hora* == 12

C3? *Hora* < 0 v *Hora* > 23

Repaso 4.3.

1). Que al evaluar la expresión de *Condición* la respuesta sea *verdadero.*

2). Que al evaluar la expresión de *Condición* la respuesta sea *falso.*

Repaso 4.4.

1). Una posible solución de orden sería:

```
 8 .    Unid ← Copia Mod 10
15 .      objG.Mensaje ( "No es # capicúa" )
 2 . Público Clase repaso4_4A
11 .    Cent ← ( Copia – Dec ) / 10
 7 .    Copia ← Nro
17 .  Fin Método
14 .    sino
10 .    Dec ← Copia Mod 10
 1 . Importar clsGenerales
 4 .  Público Método vacío Principal ( )
 9 .    Copia ← ( Copia – Unid ) / 10
 3 .  clsGenerales objG = nuevo clsGenerales ( )
18 . Fin Clase
 5 .    Entero Nro, Copia, Unid, Dec, Cent
13 .      objG.Mensaje ( "Es # capicúa" )
 6 .    Nro ← objG.leerEntero ( " Ingrese un número > 0 y < 1000: " )
16 .    Fin Si
12 .    Si ( ( Unid * 100 + Dec * 10 + Cent ) == Nro ) Entonces
```

2). Línea(s) Error

2.	Falta la palabra reservada Clase
3.	Falta el signo de = antes de la palabra reservada Nuevo
6, 7, 8 y 9.	El nombre del método invocado desde el objeto objGral.*leer_Real* no es correcto, debe ser objGral.*leerReal*.
10.	La expresión correcta es, Si (($x1 \geq 0$ $y1 \geq 0$ ($x2 \geq 0$ $y2 \geq 0$)) Entonces, todos los valores deben ser positivos si pertenecen al primer cuadrante del plano cartesiano.
11.	Falta incluir la conversión del radical, por lo tanto, la instrucción correcta de asignación es $D \leftarrow$ (((*x2 – x1*) ^ 2 + ((*y2 – y1*) ^ 2) ^ 2) ^ (1/2).
14 -15.	Falta la instrucción *Fin Si* para cerrar la estructura *Si*, entre ambas líneas.

Repaso 4.5.

1). Que al ser evaluada *C1* retorne *verdadero*.

2). Que al ser evaluada *C1* sea *falso* y al ser evaluada *C2* sea *verdadero*.

3). Que al ser evaluada *C1* sea *falso* y al ser evaluada *C2* sea *falso*.

Repaso 4.6.

1). Que al evaluar la *Vble/Expresión* sea igual a *vr1*.

2). Que al evaluar la *Variable/Expresión* sea igual a *vr3*.

3). Que al evaluar la *Variable/Expresión* no cumpla ninguno de los casos.

Repaso 5.1.

1). Que al evaluar la expresión *Condición* que acompaña la instrucción *Mientras*, al principio de la estructura, sea un valor de verdadero.

2). Que al evaluar la expresión *Condición* que acompaña la instrucción *Mientras*, al principio de la estructura, sea un valor de falso.

Repaso 5.2.

1). Con solo entrar al ciclo, después de la instrucción *Hacer*, ya se está ejecutando *P1* la primera vez; ahora, para que se ejecute en otras ocasiones, el resultado de

la evaluación de la condición que acompaña la instrucción *Mientras* al final de la estructura debe ser un valor de verdadero.

2). Que al evaluar la expresión *Condición* que acompaña la instrucción *Mientras* al final de la estructura, sea un valor de falso.

3). Que el resultado de la expresión o el valor de la *Condición* que acompaña la instrucción *Mientras* al final de la estructura sea un valor de falso. Por eso, es un validador de datos por excelencia.

Repaso 5.3.

1). Para ejecutar el bloque de instrucciones *P1* se requiere ingresar al ciclo y para ingresar al ciclo, el valor de la variable de control (*Vc*) debe ser menor o igual al valor final (*Vf*), si el comportamiento de la variable de control es aumentar; o la variable de control (*Vc*) debe ser mayor o igual al valor final (*Vf*), si el comportamiento de la variable de control es disminuir.

2). Para dejar de ejecutar el bloque de instrucciones *P1* se requiere que el valor de la variable de control (*Vc*) sobrepase el valor de la variable final (*Vf*).

Repaso 6.1.

1). El índice especifica la posición de cada elemento en el arreglo y brinda la posibilidad de recuperar o adicionar un elemento al vector.

2). Si la variable de control es superior al tamaño del arreglo, se produce un error de ejecución por apuntar a una posición no existente del arreglo.

3). La definición de un arreglo establece que contiene valores homogéneos, o sea, del mismo tipo de datos; si se desean almacenar valores de diferente tipo, se debe convertir o tratar cada valor como de un mismo tipo. El único tipo que lo permite es el de tipo Cadena; por lo tanto, sí se puede almacenar en un vector valores de diferentes tipos de datos, pero tratando cada valor como tipo de dato Cadena.

Repaso 6.2.

1). El manejo de la memoria es muy importante para el buen funcionamiento del programa; usar varios valores bajo un mismo nombre hace muy eficiente dicho manejo, aun sabiendo que es el sistema operativo el que realiza dicho proceso; además, para el programador minimiza la cantidad de variables a utilizar.

2). El uso de 2 índices para el manejo de una matriz es vital, por cuanto permite la administración de los elementos que contiene, bajo el concepto de fila y columna.

Aunque para ciertos procedimientos no se requiere de ambos, por ejemplo; en el mostrar los elementos de la diagonal principal de una matriz cuadrada.

3). La definición de un arreglo establece que contiene valores homogéneos, o sea del mismo tipo, si se desea almacenar valores de diferente tipo, se deben convertir o tratar cada valor como de un mismo tipo, el único tipo que lo permite es el de tipo Cadena; por lo tanto, sí se puede almacenar en una matriz valores de diferentes tipos, pero tratando cada valor como de tipo de dato Cadena.