7. OBJETOS Y CLASES

Este capítulo pretende brindarle al estudiante una introducción a los principales conceptos de la **POO** (Programación Orientada a Objetos), la cual presenta un enfoque muy diferente de la Programación Estructurada que se maneja en otros lenguajes de programación. Para aquellos que han venido trabajando en el paradigma de la programación estructurada, el cambio no será fácil, pero es importante que sepan que si quieren ser competitivos en el mundo de Java, es indispensable tener muy claros los conceptos de la **POO**.

7.1 CONCEPTOS

7.1.1 CLASE

Una clase es un conjunto de datos y métodos que operan sobre estos datos. "En su forma más sencilla, una clase se define con la palabra reservada ***class,*** seguida del nombre elegido para la clase y un bloque de definición, que se delimita haciendo uso de las llaves {}, estas llaves son muy importantes ya que dentro de ellas van los detalles descriptivos de la clase" [8]. Su definición se hace de la siguiente forma:

```
[public] class NombreClase
{
        tipodato nombreDato1;
        tipodato nombreDato2;
        tipodato nombreDaton;
        tipoRetorno nombreMetodo(argumento1, argumenton)
        {
                Instrucciones del método;
        }
}
```

La palabra ***public*** es opcional; si no se antepone a la palabra ***class***, de todas formas el compilador asume que la clase es pública y tendrá visibilidad para las demás clases del mismo paquete.

La palabra ***class*** es una palabra reservada que se utiliza siempre que se quiera definir una clase.

NombreClase corresponde al nombre bajo el cual agruparemos variables (datos) y métodos). Todas las variables y métodos deben declararse dentro del bloque de la clase { … }.

No siempre es necesario que una clase tenga incorporado uno o varios métodos. Los métodos contienen las instrucciones (operaciones) que se pueden realizar para una clase.

7.1.2 OBJETO

Luego de que una clase ha sido definida, un programa puede contener una ***instancia*** de la clase, denominada ***objeto de la clase.*** Un objeto se crea con el operador **new** aplicado a un constructor de la clase [9]. De esta manera un objeto o instancia es un ejemplo concreto de una clase. Por ejemplo si tenemos la clase Estudiante, un objeto o instancia de esta clase será Roberto, el cual estará conformado por las variables y métodos que componen la clase. La clase será un tipo de datos y el objeto una variable en particular.

```
NombreClase Objeto;
```

Por ejemplo:

Estudiante Roberto;

Cuando se habla de objetos es importante tener en cuenta tres características indispensables: identificación, funcionalidad y estado.

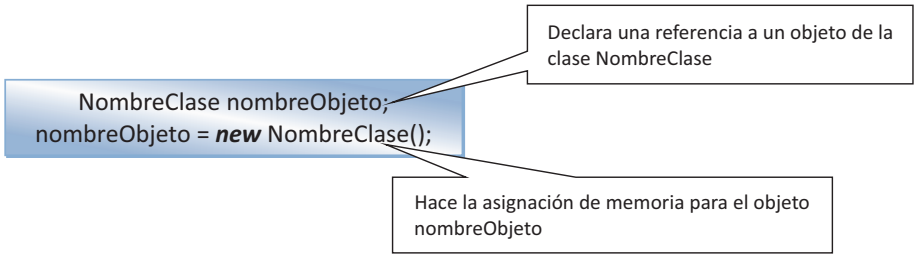
Identificación: característica que hace que un objeto sea único e identificable aunque se encuentre entre otros que tengan similar funcionalidad y estado.

Conducta: característica que determina cual es la misión del objeto y que métodos le pueden involucrar.

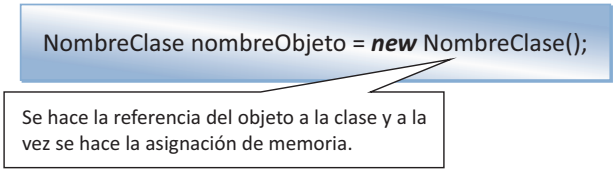
Estado: comportamiento que adquiere un método al aplicarle un método.

Para crear un objeto es necesario declarar una variable del tipo de la clase y posteriormente hacer la asignación de memoria para el objeto a través del operador new. Esta asignación de memoria se hace dinámicamente, es decir, se hace al momento de ejecutar el programa. Esta declaración se puede hacer de las siguientes formas:

Línea a línea



Combinada



A continuación se indica cómo crear la instancia apartamento1 de la clase Apartamento de las dos formas:

Línea a línea:

```
Apartamento apartamento1;
apartamento1 = new Apartamento();
```

Combinada:

```
Apartamento apartamento1 = new Apartamento();
```

7.1.3 MÉTODOS

Los métodos son las acciones que pueden ejecutar sobre una determinada clase y por ende en cada una de sus instancias (objetos). La estructura general de un método es la siguiente:

```
tipo_de_retorno nombreMetodo(parámetros)
        {
                Variables internas del método;
                Instrucciones del método;
                return valor;
        }
```

Algunos métodos no retornan ningún valor, por lo cual deben declararse de tipo ***void***. En cuanto al nombre del método, este puede ser cualquier expresión válida para java y, por buenas prácticas de programación se recomienda que este debe iniciar con una letra minúscula y si está formado por dos o más palabras, deben ir unidas y sus letras iniciales en mayúscula.

Los parámetros son variables que reciben los valores necesarios para que el método realice sus acciones. Estos valores son enviados por otras variables llamadas argumentos y que vienen desde el lugar del programa donde se haga el llamado al método. Tanto argumentos de llamada como parámetros del método deben ser del mismo tipo. La lista de parámetros está conformada por una serie de parejas separadas por comas, donde el primer miembro de cada pareja corresponde al tipo de datos que puede recibir el parámetro y el segundo es el nombre del parámetro.

```
tipoRetorno nombreMétodo(tipo nombreParámetro_1, tipo nombreParámetro_2,……..,
        tipo nombreParámetro_n){
        cuerpoDelMétodo;
}
```

Aunque algunos métodos no requieren parámetros para su funcionamiento, si queremos generalizar un método, es necesario que dentro de su estructura incluyamos parámetros que le permitan recibir datos desde el exterior. El siguiente ejemplo muestra de forma corriente como un método devuelve el valor promedio de 3 números:

```
double promedioNumeros()
        {
                return (5 + 3 + 6)/3;
        }
```

Observando de manera rápida es posible comprobar que el método anterior devuelve el valor promedio entre los números 5, 3 y 6. Pero también podemos ver que su uso está restringido a los valores que se han suministrado en el interior del método.

Si se modifica el método como se ilustra a continuación, éste se hace más útil, ya que estará en capacidad de recibir valores enviados desde diferentes puntos en los cuales sea llamado.

```
double calcularPromedio(double a, double b, double c)
        {
                return (a+b+c)/3;
        }
```

A continuación se ilustra un ejemplo de una *clase* en Java en el que se crea un arreglo para contener tres números digitados desde el teclado y se les calcula el promedio a través del método **calcularPromedio( )**, el cual contiene tres parámetros de tipo **double**.

```
import javax.swing.JOptionPane;

public class Promedio
        {
          double listaNumeros [] = new double[3];
          void leerNumeros()
              {
                  for(int i = 0;i<3;i++)
                     {
                        listaNumeros[i]=Double.valueOf(JOptionPane.showInput
                        (“Digite el “     +        (i+1) + “ número: “)).doubleValue();
                     }
              }
        double calcularPromedio(double a, double b, double c)
          {
             return (a+b+c)/3;
         }
        }
```

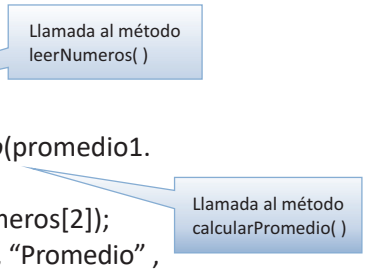
Los métodos de una clase deben estar contenidos dentro del cuerpo de la clase.

7.2 LLAMADA A UN MÉTODO

Una llamada a un método permite enviarle valores llamados argumentos para que el método pueda ejecutarse de manera generalizada. El siguiente ejemplo ilustra como enviar al método calcularPromedio( ) tres argumentos con valores de tipo doble contenidos en el arreglo listaNumeros[ ]].

```
import javax.swing.JOptionPane;

public class Main
  {
     public static void main(String[] args)
          {
             Promedio promedio1 = new Promedio();
             promedio1.leerNumeros();
             double resultado = promedio1.calcularPromedio(promedio1.
             listaNumeros[0],
             promedio1.listaNumeros[1], promedio1.listaNumeros[2]);
             JOptionPane.showMessageDialog(null, resultado , "Promedio" ,
             JOptionPane.INFORMATION_MESSAGE);
       }
}
```



7.3 CONSTRUCTORES

Los constructores son métodos especiales que tienen el mismo nombre que la clase a la que pertenecen y permiten inicializar un objeto una vez creado sin necesidad de hacer llamado al constructor o a cualquier otro método que permita asignar valores de inicialización al objeto. En el programa siguiente, el constructor Empleado( ) permite inicializar el objeto empleado1 con los valores: 0 para la ***cedula***, "Nombre empleado" para ***nombre*** y "Apellido empleado" para ***apellido***.

```
import javax.swing.JOptionPane;

public class Main
  {
    public static void main(String[] args)
          {
```

```
            Empleado empleado1 = new Empleado();
            JOptionPane.showMessageDialog(null, empleado1.cedula,
            "Empleados!",
            JOptionPane.INFORMATION_MESSAGE);
            JOptionPane.showMessageDialog(null, empleado1.apellido,
            "Empleados!",
            JOptionPane.INFORMATION_MESSAGE);
        }
}

public class Empleado
{
   String cedula;
   String nombre;
   String apellido;
   Empleado()
        {
                cedula = "0";
                nombre = "Nombre empleado";
                apellido = "Apellido empleado";
        }
}
```

Constructor Empleado( )

Si no hemos definido ningún constructor dentro de la clase, cuando se haga el llamado este a través de la instrucción new, se estará haciendo uso del constructor por defecto, que es aquel que aunque no esté definido explícitamente dentro del programa inicializa el objeto con los datos por defecto correspondientes a cada tipo.

El programa siguiente no define de forma explícita un constructor pero puede visualizar los datos por defecto para cada variable de la instancia creada.

7.4 SOBRECARGA DE MÉTODOS

Cuando dentro de una misma clase existen dos o más métodos con el mismo nombre aunque con diferentes tipos y números de parámetros se dice que existe sobre carga de métodos. Cuando se hace el llamado a un método sobrecargado, el compilador se vale del tipo y número de parámetros para saber a cuál de los métodos sobrecargados debe llamar.

```
public class Operacion {
   int numero1;
   int numero2;
   int numero3;
   char signo;
   int hacerOperacion(int a, int b, char s){
      numero1 = a;
      numero2 = b;
      signo = s;
      if(signo=='+') return a+b;
      if(signo=='-') return a-b;
      if(signo=='*') return a*b;
      if(signo=='/')return a/b;
      else
         System.out.println("Operación no válida!");
         return 0;
   }
   int hacerOperacion(int a,int b, int c){
      numero1 = a;
      numero2 = b;
      numero3 = c;
      return numero1+numero2+numero3;
   }
   int hacerOperacion(int a, int b){
      numero1 = a;
      numero2 = b;
      return numero1 * numero2;
   }
   int hacerOperacion(int base){
      numero1 = base;
      return numero1 * numero1;
   }
}
```

En este ejemplo el método hacerOperacion( ) está sobrecargado, de tal forma que el compilador puede hallar el cuadrado de un número, realizar una operación aritmética entre dos números, hacer la suma de tres números y hacer la multiplicación de dos números, haciendo la llamada de similar forma desde el método main( ) pero cambiando los argumentos que hacen parte de la llamada.

```
public class Main {

   public static void main(String[] args) {
     Operacion operacion1 = new Operacion();
     System.out.print("El cuadrado 10 es: ");
     System.out.println(operacion1.hacerOperacion(10));
     System.out.print("10 - 2 : ");
     System.out.println(operacion1.hacerOperacion(10, 2, '-'));
     System.out.print("1 + 2 + 3 : ");
     System.out.println(operacion1.hacerOperacion(1, 2, 3));

System.out.print("5 * 4 : ");
     System.out.println(operacion1.hacerOperacion(5, 4));
   }

}
```

Llama al método hacerOperacion(int

Llama al método hacerOperacion(int a, int b, char s)

Llama al método hacerOperacion(int a, int b, int c)

Llama al método hacerOperacion(int a, int b)

7.5 SOBRECARGA DE CONSTRUCTORES

De la misma forma cómo es posible tener en una clase sobrecarga de métodos, también es posible tener sobrecarga de constructores.

En el ejemplo siguiente, la clase **Operacion** tiene el constructor **Operacion( )** sobrecargado en tres versiones diferentes.

```
public class Operacion {
   int n1, n2, n3;

public Operacion(int a) {
      n1 = n2 = n3 = a;
   }

   public Operacion(int a, int b) {
      n1 = n2 = n3 = (a * b);
   }

   public Operacion(int a, int b, int c) {
      n1 = n2 = n3 = (a * b * c);
   }
   public  void visualizarDatos(){
      System.out.println(n1+" "+n2+" "+n3);
   }
}
```

Desde el método Main( ) se hace el llamado a las diferentes versiones del constructor cambiando el número de parámetros que se pasan como argumentos.

```
public static void main(String[] args) {
    Operacion operacion1, operacion2, operacion3;
    operacion1 = new Operacion(1);
    operacion1.visualizarDatos();
    operacion2 = new Operacion(1,2);
    operacion2.visualizarDatos();
    operacion3 = new Operacion(1,2,3);
    operacion3.visualizarDatos();
  }
}
```

Es muy importante recordar que una vez definido uno o varios constructores para una clase (sobre carga) el programa ya no hace llamado al constructor por defecto.

7.6 PASO DE OBJETOS COMO PARÁMETROS

De la misma forma cómo es posible pasar datos simples como parámetros a un método en java, también es posible pasar objetos. En el ejemplo siguiente la clase **Estudiante** contiene el método **compararNacimiento( )**, que tiene como parámetro el objeto **estudianteX** de tipo Estudiante y a través del cual son enviados al método los datos pertenecientes a un objeto de su clase. Este método retorna ***true*** si los dos objetos del tipo **Estudiante** (el objeto que hace el llamado y el objeto que hace de argumento) son iguales y ***false*** si son diferentes.

```
public class Estudiante {
  int yearNacimiento;
  int mesNacimiento;
  int diaNacimiento;
  //Constructor Estudiante
  Estudiante(int a,int b, int c){
     yearNacimiento = a;
     mesNacimiento = b;
     diaNacimiento = c;
  }
```

```
   /*Método que devuelve verdadero si el objeto recibido como parámetro
    * es igual al objeto con el que se le hace el llamado
    */
   boolean comparaNacimiento(Estudiante estudianteX){
      if(estudianteX.yearNacimiento==yearNacimiento &&
            estudianteX.mesNacimiento==mesNacimiento &&
            estudianteX.diaNacimiento==diaNacimiento){
         return true;
      }
      else
         return false;
   }
}

public class Main {
   public static void main(String[] args) {
      Estudiante estudiante1 = new Estudiante(1990,10,5);
      Estudiante estudiante2 = new Estudiante(1990,10,5);
      Estudiante estudiante3 = new Estudiante(1991,1,1);
      Estudiante estudiante4 = new Estudiante(1991,1,1);
      System.out.println("estudiante1 es igual a estudiante2 ? ");
       System.out.println("Respuesta: "+estudiante1.comparaNacimiento(estudi
ante2));
      System.out.println("estudiante3 es igual a estudiante4 ? ");
       System.out.println("Respuesta: "+estudiante3.comparaNacimiento(estudi
ante4));
   }
}
```

A continuación se presenta un programa en el que se hace la comparación de dos matrices de 3 x 3 en donde una es pasada como parámetro al método **compararMallas( )**. La clase Malla está compuesta por la variable **matrizA[ ][ ]** de tipo entero.

```
public class Malla {

   int matrizA[][] = new int[3][3];
```

Inicialmente, el constructor **Malla( )** asigna el valor -1 a cada uno de los elementos **i,j** de la matriz.

```
Malla() {
    for (int i = 0; i < 3; i++) {
       for (int j = 0; j < 3; j++) {
          matrizA[i][j] = -1;
       }
    }
 }
```

El método **imprimirMalla ( )** permite hacer la impresión de cada uno de los elementos **i,j** de la matriz.

```
 void imprimirMalla() {
    for (int i = 0; i < 3; i++) {
       for (int j = 0; j < 3; j++) {
          System.out.print(matrizA[i][j] + " ");
       }
       System.out.println();
    }
 }
```

El método **llenarMalla(int a)** recibe el parámetro a de tipo entero y lo asigna a cada uno de los elementos **i,j** de la matriz.

```
 void llenarMalla(int a) {
    for (int i = 0; i < 3; i++) {
       for (int j = 0; j < 3; j++) {
          matrizA[i][j] = a;
       }
    }
 }
```

El método **compararMallas(Malla mallaX)** recibe como parámetro el objeto **mallaX** del tipo Malla y lo compara elemento a elemento con el objeto que hace el llamado; en este caso la **matrizA[i][j]**. En este método se utiliza la variable bandera de tipo entero y que hace las veces de un acumulador para saber si todos los elementos de las dos matrices comparadas son iguales.

```
 boolean compararMallas(Malla mallaX){
    int bandera=0;
    for(int i=0;i<3;i++){
       for(int j=0;j<3;j++){
```

```
            if(matrizA[i][j]==mallaX.matrizA[i][j]){
               bandera+=1;
            }
         }
      }
      if(bandera==9)
         return true;
      else
         return false;
   }
}
```

El método **main( )** de este programa crea dos objetos **malla1** y **malla2** de tipo **Malla**, les asigna el valor -1 a través del constructor **Malla( )**, imprime el valor de las dos matrices, luego les asigna a cada uno de sus elementos **i,j** el valor 5 llamando al método **llenarMalla( )** y finalmente imprime si son o no iguales. Para este caso como las dos matrices tienen en cada uno de sus elementos **i,j** el valor 5 la salida impresa será ***true***.

```
public class Main {
   public static void main(String[] args) {
      Malla malla1 = new Malla();
      Malla malla2 = new Malla();
      malla1.imprimirMalla();
      malla1.llenarMalla(5);
      malla2.llenarMalla(5);
      malla1.imprimirMalla();
      System.out.println(malla1.compararMallas(malla2));
   }
}
```

Pruebe cambiando el valor 5 que se envía como argumento en la línea donde el objeto **malla2** hace el llamado al método **llenarMalla( )** y podrá ver que la salida pasará de ser true a false.

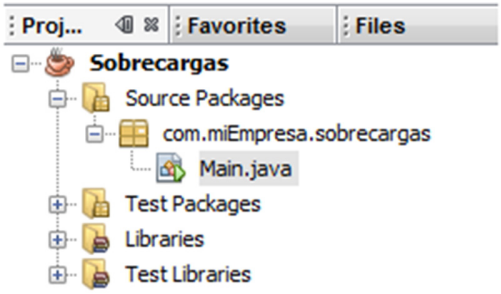
7.7 PAQUETES

Java le permite al programador agrupar en una colección llamada paquete las clases que éste considere, de tal forma que su trabajo sea más claro y organizado. Es muy similar a lo que se puede hacer con los archivos

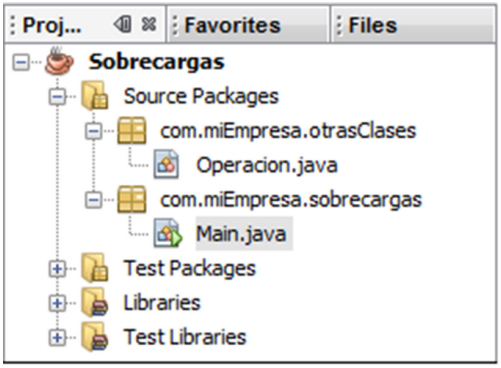
que guardamos en un computador; los organizamos por categorías y los agrupamos en carpetas según esta clasificación.

La compañía Sun Microsystems, creadora de Java, teniendo en cuenta que el nombre del dominio de todas las empresas es único sugiere que el nombre del paquete raíz donde almacenaremos nuestras clases corresponda con el nombre inverso de dicho dominio. Por ejemplo miEmpresa.com corresponde con com.miEmpresa. Además podemos crear subpaquetes según la necesidad que tengamos para un claro y manejable trabajo.

La figura siguiente, que ilustra la ventana de proyectos en NetBeans 6.8 muestra cómo queda inicialmente el árbol de paquetes para el proyecto Java de nombre Sobrecargas. En el paquete com.miEmpresa queda ubicada la clase Main de este proyecto.



De la misma forma como se tiene el paquete com.miEmpresa.sobrecargas, es posible crear otros paquetes donde es posible agrupar otras clases según la necesidad del programador.



Para poder hacer uso de las clases que se encuentran en otros paquetes es necesario importarlas desde la cabecera de la clase que las utilizará. Esto se hace a través de la instrucción **import,** con la cual podremos importar una clase específica del paquete o todas las clases que se encuentran dentro de este. Es importante recordar que el esquema general de la instrucción import es el siguiente:

```
Import nombrePaquete.nombreClase;
```

En la cabecera de la clase desde la que estamos haciendo la llamada a otras clases, también aparece la instrucción **package.nombrePaquete**;. Esta instrucción identifica el paquete al cual pertenece la clase actual.

```
package com.miEmpresa.sobrecargas;

import com.miEmpresa.otrasClases.*;
```

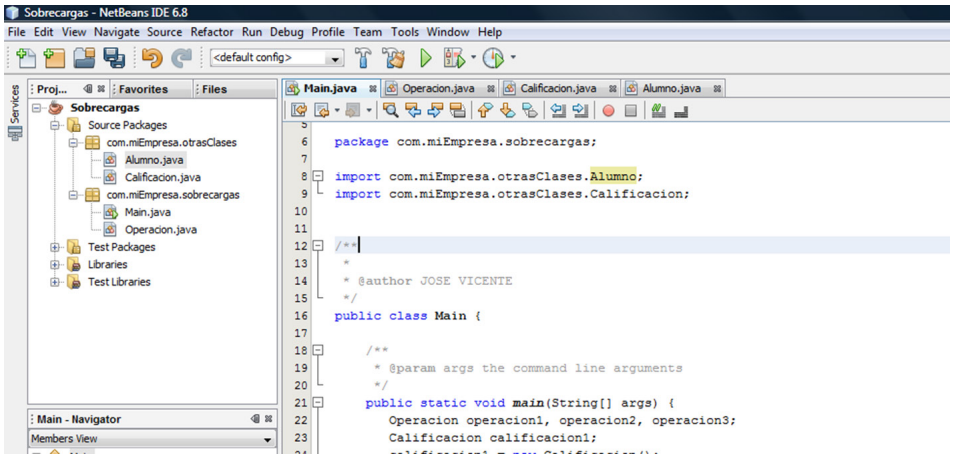
Importa todas las clases contenidas en el paquete com.miEmpresa.otrasClases

```
package com.miEmpresa.sobrecargas;

import com.miEmpresa.otrasClases.Alumno;
import com.miEmpresa.otrasClases.Calificacion;
```

Importan las clases Alumno y Calificacion contenidas en el paquete com.miEmpresa.otrasClases

La figura siguiente permite ver la estructura del proyecto **Sobrecargas**, en el que existen dos paquetes de clases y desde la clase **Main** se importan las clases **Alumno** y **Calificación**.



7.8 ACCESO A DATOS DE UNA CLASE DE OTRO PAQUETE

Para poder tener acceso a datos de una clase perteneciente a otro paquete, sus métodos deben ser declarados públicos, lo mismo que aquellos datos a los que deseemos acceder de forma directa. De lo contrario su acceso no será permitido, esto debido a la capacidad de encapsulamiento que tiene Java para proteger sus clases.