capítulo 8

Clases y objetos

objetivos

En este capítulo aprenderá a:

- Diseñar clases básicas.
- Diferenciar entre clase y objeto.
- Usar las clases más comunes en la construcción de programas sencillos.
- Distinguir el concepto de visibilidad: `public` y `private`.
- Usar la *recolección de basura*.
- Emplear los conceptos del objeto `this` y cómo utilizar los miembros `static` de una clase.
- Utilizar las bibliotecas de clases de Java y las clases fundamentales `System`, `Object` y `Math`.

introducción

Los lenguajes de programación soportan en sus compiladores tipos predefinidos de datos fundamentales o básicos, como `int`, `char`, `float` y `double`; además, Java tiene características que permiten amplitud al añadir sus propios tipos de datos.

Un tipo de dato definido por el programador se denomina TAD o tipo abstracto de dato (ADT, *abstract data type*). Si el lenguaje de programación soporta los tipos que desea el usuario y el conjunto de operaciones sobre cada tipo, se obtiene un TAD.

Una clase es un tipo de dato que contiene códigos o métodos, y datos; ademas permite encapsular todo el código y los datos necesarios para gestionar un tipo específico de un elemento de programa, como una ventana en la pantalla, un dispositivo conectado a una computadora, una figura de un programa de dibujo o una tarea realizada por una computadora; en este capítulo se aprenderá a crear, definir, especificar y utilizar clases individuales.

8.1 Clases y objetos

Las tecnologías orientadas a objetos combinan la descripción de los elementos en un entorno de proceso de datos con las acciones que dichos elementos ejecutan; las clases y los objetos como instancias o ejemplares de ellas son los elementos clave sobre los que se articula la orientación a objetos. Aunque ya se consideró el estudio de clases y objetos, se hará un recordatorio práctico antes de pasar al diseño y la construcción práctica.

8.1.1 ¿Qué son los objetos?

En el mundo real, las personas identifican los objetos como elementos que pueden ser percibidos por los cinco sentidos; cuentan con propiedades específicas que definen su estado como posición, tamaño, color, forma, textura, etcétera; además tienen ciertos comportamientos que los hacen diferentes de otros objetos.

Booch[1] define un objeto como "una entidad (algo) que tiene un estado, un comportamiento y una identidad". Supongamos que el estado de la máquina de una fábrica puede estar conformado por: encendida/apagada (*on/off*), su potencia, velocidad máxima, velocidad actual, temperatura, etcétera; su comportamiento puede incluir diversas acciones para encenderla y apagarla, obtener su temperatura, activar o desactivar otras máquinas, condiciones de señal de error o cambiar de velocidad; su identidad se basa en el hecho de que cada instancia de una máquina es única y está identificada por un número de serie; los que se eligen para enfatizar el estado y el comportamiento se apoyarán en cómo un objeto máquina se utilizará en una aplicación. En el diseño de un programa orientado a objetos, se crea una abstracción o modelo simplificado de la máquina basado en las propiedades y comportamiento que son útiles en el tiempo.

Martin y Odell definen un objeto como "cualquier cosa, real o abstracta, en la que se almacenan datos y aquellos métodos (operaciones) que manipulan los datos".[2] Para realizar esa actividad se añaden a cada objeto de la clase los propios datos y los asociados con sus propios métodos miembro que pertenecen a la clase.

Un mensaje es una instrucción que se envía a un objeto, el cual se ejecutará al recibirlo; incluye el identificador que contiene la acción a realizar por el objeto junto con los datos que éste necesita para efectuar su trabajo; los mensajes, por consiguiente, forman una ventana del objeto al mundo exterior.

El usuario se comunica con el objeto mediante su interfaz, un conjunto de operaciones definidas por la clase del objeto de modo que todas sean visibles al programa; por ejemplo: un dispositivo electrónico, como una máquina de fax, tiene una interfaz de usuario bien definida, incluye un mecanismo de avance del papel, botones de marcado, receptor y el botón "Enviar"; el usuario no tiene que conocer cómo está construida la máquina internamente, el protocolo de comunicaciones u otros detalles; de hecho, abrir la máquina durante el periodo de garantía puede anular la garantía.

8.1.2 ¿Qué son las clases?

En términos prácticos, una clase es un tipo definido por el usuario; son los bloques de construcción fundamentales de los programas orientados a objetos. Booch denomina clase como "un conjunto de objetos que comparten una estructura, comportamiento y semántica comunes".[3]

Una clase contiene la especificación de los datos que describen un objeto junto con la descripción de las acciones cuya ejecución conoce; dichas acciones se conocen como servicios o métodos. Una clase también incluye todos los datos necesarios para describir

---

[1] Booch, G., *Análisis y diseño orientado a objetos con aplicaciones*, Madrid, Díaz de Santos/Addison-Wesley, 1995, p. 78. Este libro fue traducido de su versión original (*Object-Oriented Analysis and Desing with Applications*, 2a. edición, Benjamin-Cummings, 1994) por los profesores españoles Luis Joyanes y Juan Manuel Cueva; en la tercera edición del libro en inglés (*Object-Oriented Analysis and Desing with Applications*, 3a. edición, Addison-Wesley/Pearson, 2007), Booch y sus coautores mantienen su definición original (p. 78) para definir la naturaleza de un objeto.

[2] Martin, J. y Odell, J., *Object-Oriented Analysis and Design*. Nueva Jersey, Prentice-Hall, 1992, p. 27.

[3] Booch, *op. cit*. p. 75. En la tercera edición Booch *et. al*. vuelven a definir la clase de igual forma dentro del contexto de análisis y diseño orientado a objetos.

los objetos creados a partir de la clase, los cuales se conocen como atributos, variables o variables instancia; el primer término se utilizará en análisis y diseño orientado a objetos, los otros, en programas orientados a objetos.

8.2 Declaración de una clase

Antes de que un programa pueda crear objetos de cualquier clase, ésta debe definirse, lo que implica darle un nombre a la clase y a los elementos que almacenan sus datos, así como describir los métodos que realizarán las acciones consideradas en los objetos.

Las definiciones o especificaciones no constituyen un código de programa ejecutable, sino que se utilizan para asignar almacenamiento a los valores de los atributos que usa el programa y reconocer los métodos que éste utilizará; normalmente se sitúan en archivos formando *packages*, utilizando un archivo para varias clases relacionadas.

sintaxis

Formato:

```
class NombreClase
{
  Lista_de_miembros
}
```

NombreClase	es definido por el usuario e identifica a la clase; puede incluir letras, números y subrayados como cualquier identificador válido.
Lista_de_miembros	son métodos y datos miembros de la clase.

EJEMPLO 8.1

Definición de una clase llamada `Punto` que contiene coordenadas x-y de un punto en un plano.

```
class Punto
{
  private int x;    // coordenada x
  private int y;    // coordenada y

  public Punto(int x_,int y_)  // constructor
  {
    x = x_;
    y = y_;
  }
  public Punto()               // constructor sin argumentos
  {
    x = y = 0;
  }

  public  int LeerX()          // devuelve el valor de x
  {
```

```
    return x;
  }
  public  int LeerY()          // devuelve el valor de y
  {
    return y;
  }
  void fijarX(int valorX)      // establece el valor de x
  {
    x = valorX;
  }
  void fijarY(int valorY)      // establece el valor de y
  {
    y = valorY;
  }

}
```

EJEMPLO 8.2

Declaración de la clase `Edad`.

```
class Edad
{
  private int edadHijo, edadMadre, edadPadre ;  ______ datos

  public Edad(){...}  ________    método especial: constructor

  public  void iniciar(int h,int e,int p){...}____    métodos
  public   int leerHijo(){...}
  public  int leerMadre(){...}
  public  int leerPadre(){...}
}
```

8.2.1 Creación de un objeto

Una vez que una clase fue definida, un programa puede contener una instancia de la clase, denominada objeto de la clase; éste se crea con el operador `new` aplicado a un constructor; por ejemplo, un objeto de la clase `Punto` inicializado a las coordenadas `(2,1)`:

```
new Punto(2,1);
```

El operador `new` crea el objeto y devuelve una referencia al objeto creado; esta referencia se asigna a una variable del tipo de la clase; el objeto permanecerá vivo siempre que esté referenciado por una variable de la clase que es instancia.

sintaxis

Formato para definir una referencia:

```
NombreClase   varReferencia;
```

Formato para crear un objeto:

```
varReferencia = new NombreClase(argmntos_constructor);
```

Toda clase tiene uno o más métodos, denominados constructores para inicializar el objeto cuando es creado, tienen el mismo nombre que el de la clase, no tienen tipo de retorno y pueden estar sobrecargados; en la clase `Edad` del ejemplo 8.2, el constructor no tiene argumentos, se puede crear un objeto:

```
Edad f = new Edad();
```

El operador de acceso selecciona un miembro individual de un objeto de la clase; por ejemplo:

```
Punto p;
p = new Punto();
P.fijarX (100);
System.out.println(" Coordenada   x es " + P.leerX());
```

NOTA

El operador punto (.) se utiliza con los nombres de los métodos y variables instancia para especificar que son miembros de un objeto; por ejemplo:

```
Clase DiaSemana, contiene un método visualizar()

DiaSemana hoy;          // hoy es una referencia
hoy = new DiaSemana();  // se ha creado un objeto
hoy.visualizar();       // llama al método visualizar()
```

8.2.2 Visibilidad de los miembros de la clase

Un principio fundamental en programación orientada a objetos es la ocultación de la información; esto significa que no se puede acceder por métodos externos a la clase a determinados datos internos; el mecanismo principal para ocultar datos es ponerlos en una clase y hacerlos privados; a éstos sólo se podrá acceder desde el interior de la clase. Por el contrario, los datos o métodos públicos son accesibles desde el exterior de la clase.

Para controlar el acceso a los miembros de la clase se utilizan tres diferentes especificadores de acceso: `public`, `protected` y `private`; cada miembro está precedido del especificador de acceso que le corresponde.

Formato:

```
class NombreClase
{
 private declaración miembro privado;  // miembros privados
```

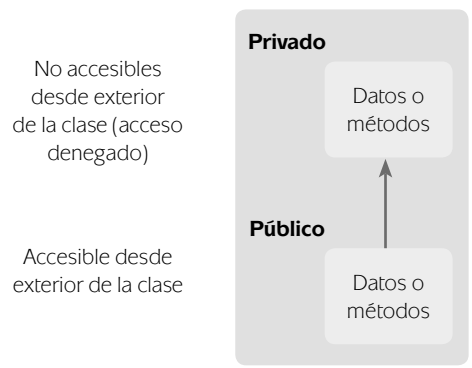

Figura 8.1 Secciones pública y privada de una clase.

```
 public declaración miembro público;   // miembros públicos
}
```

`public` define miembros públicos, aquellos a los que se puede acceder por cualquier método desde fuera de la clase; a los miembros de `private` sólo se puede acceder por métodos de la misma clase, mientras que a los de `protected` se puede ingresar por aquellos de la misma clase o de clases derivadas, así como por métodos de otras clases que se encuentran en el mismo paquete. Los especificadores `public, protected` y `private` pueden aparecer en cualquier orden; si no se especifica el acceso predeterminado a un miembro de una clase, a éste se puede ingresar desde los métodos de la clase y desde cualquier método de las clases del paquete en que se encuentra.

EJEMPLO 8.3

Declaración de las clases `Foto` y `Marco` con miembros declarados con distinta visibilidad; ambas forman parte del paquete `soporte`.

```
package soporte;

class Foto
{
  private int nt;
  private char opd;
  String q;
  public Foto(String r)   // constructor
  {
    nt = 0;
    opd = 'S';
    q = new String(r);
  }
  public double mtd(){...}
}

class Marco
{
  private double p;
  String t;
  public Marco()  {...}
  public void poner()
  {
    foto u = new Foto("Paloma");
    p = u.mtd();
    t = "**" + u.q + "**";
  }
}
```

Aunque las especificaciones públicas, privadas y protegidas pueden aparecer en cualquier orden, en Java los programadores suelen seguir una de las siguientes reglas en el diseño, y de las que usted puede elegir la que considere más eficiente.

1. Poner los miembros privados primero debido a que contiene los atributos o datos.
2. Poner los miembros públicos primero debido a que los métodos y los constructores son la interfaz del usuario de la clase.

En realidad, la labor más importante de los especificadores de acceso es implementar la ocultación de la información; este principio indica que toda la interacción con un

Tabla 8.1 Visibilidad, "x" indica que el acceso está permitido.

Tipo de miembro	Miembro de la misma clase	Miembro de una clase derivada	Miembro de clase del paquete	Miembro de clase de otro paquete
Private	x			
En blanco	x		X	
Public	x	x	X	x

objeto se debe restringir al uso de una interfaz bien definida que permita que los detalles de implementación de los objetos sean ignorados; por consiguiente, los datos y métodos públicos forman la interfaz externa del objeto, mientras que los elementos privados son los aspectos internos que no necesitan ser accesibles para su uso; los elementos de una clase sin especificador y los `protected` tienen las mismas propiedades que los públicos respecto a las clases del paquete.

NOTA

El principio de encapsulamiento significa que las estructuras de datos internas utilizadas en la implementación de una clase no pueden ser accesibles directamente al usuario de la clase.

8.2.3 Métodos de una clase

Los métodos en Java siempre son miembros de clases, no hay métodos o funciones fuera de ellas; su implementación se incluye dentro del cuerpo de la clase; la figura 8.2 muestra la declaración completa de una clase.

EJEMPLO 8.4

La clase `Racional` define el numerador y denominador característicos de un número; por cada dato, se proporciona una método miembro que devuelve su valor y un método que asigna numerador y denominador; tiene un constructor que inicializa un objeto a 0/1.

```
class Racional
{
  private int numerador;
  private int denominador;
  public Racional()
  {
```

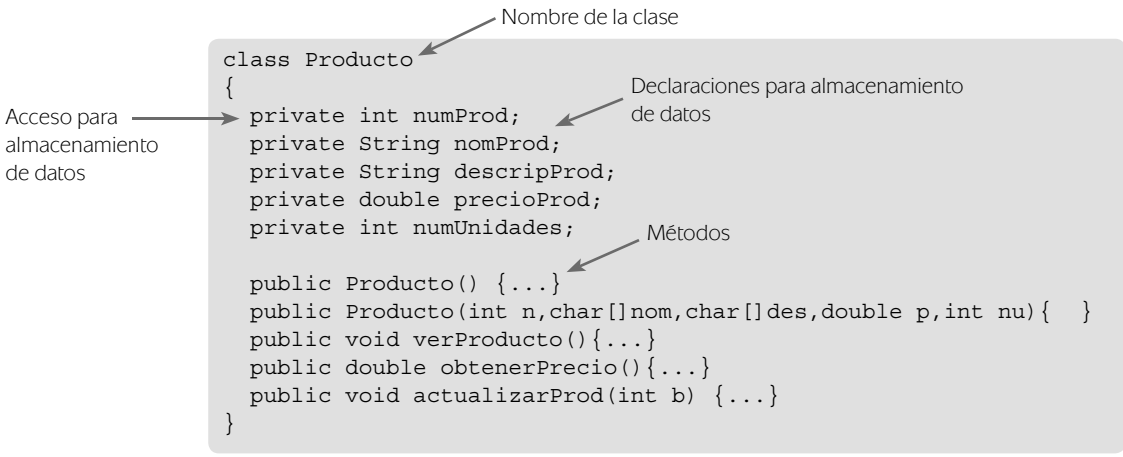

Figura 8.2 Definición típica de una clase.

```
    numerador = 0;
    denominador = 1;
  }
  public int leerN() { return numerador; }
  public int leerD() { return denominador; }
  public  void fijar (int n, int d)
  {
    numerador = n;
    denominador = d;
  }
}
```

ejercicio 8.1

Definir una clase `DiaAnyo` que contenga los atributos mes y día, los métodos `igual()` y `visualizar()`. Que el mes se registre como un valor entero: 1, Enero; 2, Febrero y así sucesivamente; el día del mes debe registrarse en la variable entera día. Escribir un programa que compruebe si una fecha es su cumpleaños.

El método `main()` de la clase principal, `Cumple`, crea un objeto `DiaAnyo` y llama al método `igual()` para determinar si la fecha del objeto coincide con la fecha de su cumpleaños, que se ha leído del dispositivo de entrada.

```
import java.io.*;
import java.util.*;

class DiaAnyo
{
  private int mes;
  private int dia;

  public DiaAnyo(int d, int m)
  {
    dia = d;
    mes = m;
  }
  public boolean igual(DiaAnyo d)
  {
    if ((dia == d.dia) && (mes == d.mes))
      return true;
    else
      return false;
  }

  public void visualizar()
  {
    System.out.println("mes = " + mes + " , dia = " + dia);
  }
}
    // clase principal, con método main
public class Cumple
{
  public static void main(String[] ar)throws IOException
  {
    DiaAnyo hoy;
```

```
    DiaAnyo cumpleanyos;
    int d, m;
    Scanner entrada = new Scanner(System.in));
    System.out.print("Introduzca fecha de hoy, dia: ");
    d = entrada.nextInt();
    System.out.print("Introduzca el número de mes: ");
    m = entrada.nextInt();
    hoy = new DiaAnyo(d,m);
    System.out.print("Introduzca su fecha de nacimiento, dia: ");
    d = entrada.nextInt();
    System.out.print("Introduzca el número de mes: ");
    m = entrada.nextInt();
    cumpleanyos = new DiaAnyo(d,m);
    System.out.print( " La fecha de hoy es ");
    hoy.visualizar();
    System.out.print( " Su fecha de nacimiento es ");
    cumpleanyos.visualizar();
    if (hoy.igual(cumpleanyos))
      System.out.println( "¡Feliz cumpleaños ! ");
    else
      System.out.println( "¡Feliz dia ! ");
  }
}
```

8.3 Implementación de las clases

El código fuente de la definición de una clase con todos sus métodos y variables instancia se almacenan en archivos de texto con extensión `.java` y con el nombre de la clase; por ejemplo: `Racional.java`; la implementación de cada clase normalmente se sitúa en un archivo independiente y éstas pueden proceder de diferentes fuentes:

- Declarar e implementar las propias; el código fuente siempre estará disponible y pueden organizarse por paquetes.
- Utilizar clases escritas por otras personas o compradas; en este caso, se puede disponer del código fuente o estar limitado a utilizar el bytecode de la implementación. Será necesario disponer del paquete donde se encuentran.
- Utilizar las de los diversos archivos o *packages* que acompañan el *software* de desarrollo Java.

8.4 Clases públicas

La declaración de una clase puede incluir el modificador `public` como prefijo en su cabecera; por ejemplo:

```
public class Examen
{
  // miembros de la clase
}
```

La clase `Examen` puede utilizarse por las clases que se encuentran en su mismo archivo o por clases externas; habitualmente se definen como `public`, a menos que se quiera restringir su uso; una clase declarada

ADVERTENCIA

El especificador de acceso `public` es el único que se puede establecer en la cabecera de una clase.

sin dicho prefijo establece una restricción importante y sólo podrá ser utilizada por las clases definidas en el mismo paquete.

8.5 Paquetes

Los paquetes es la forma que tiene Java de organizar los archivos con las clases necesarias para construir las aplicaciones. Java incorpora varios de ellos, por ejemplo: `java.lang`, `java.io`, o `java.util`, con las clases básicas para la construcción de programas: `System`, `String`, `Integer`, `Scanner`.

8.5.1 Sentencia `package`

¿Cómo se puede definir un paquete? Mediante la sentencia `package`; la cual, en primer lugar, se debe incluir como línea inicial del archivo fuente en cada clase del paquete; por ejemplo: si las clases `Lapiz`, `Boligrafo` y `Folio` se organizan para formar el paquete `escritorio`, el esquema que sigue es:

```
//  archivo fuente Lapiz.java

package escritorio;
public class Lapiz
{

  // miembros de clase Lapiz

}
//  archivo fuente Boligrafo.java
package escritorio;
public class Boligrafo
{

  // miembros de clase Boligrafo

}
//  archivo fuente Folio.java
package escritorio;
public class Folio
{

  // miembros de clase Folio

}
```

Formato:


```
package NombrePaquete;
```

En segundo lugar, una vez creado el archivo fuente de cada clase del paquete, éstas se deben ubicar en un subdirectorio con el mismo nombre que el del paquete; en el ejemplo

anterior `Lapiz.java`, `Boligrafo.java` y `Folio.java` se ubicarán en el path `escritorio`.

El uso de paquetes tiene dos beneficios importantes:

1. Las restricciones de visibilidad son menores entre las clases que están dentro del mismo paquete; desde cualquier clase, los miembros `protected` y los miembros sin modificador de visibilidad son accesibles; sin embargo no lo son desde las clases que se encuentran en otros paquetes.
2. La selección de las clases de un paquete se puede abreviar con la sentencia `import` del paquete.

8.5.2 Sentencia `import`

Ya se mencionó que las clases que se encuentran en los paquetes se identifican utilizando el nombre del paquete, el selector punto `(.)` y a continuación el nombre de la clase; un ejemplo es la declaración de la clase `Arte` con atributos de la clase `PrintStream` del paquete `java.io`, y `Lapiz` del paquete `escritorio`:

```
public class Arte
{
  private java.io.PrintStream salida;
  private escritorio.Lapiz p;
}
```

La sentencia `import` facilita la selección de una clase porque permite escribir únicamente su nombre, evitando el nombre del paquete; la declaración anterior se puede abreviar así:

```
import java.io.PrintStream;
import escritorio.*;
public class Arte
{
  private PrintStream salida;
  private Lapiz p;
}
```

La sentencia `import` debe aparecer antes de la declaración de las clases, a continuación de la sentencia `package`; existen dos formatos:

Formato:

sintaxis

```
import identificadorpaquete.nombreClase;

import identificadorpaquete.*;
```

El primer formato especifica una clase concreta; el segundo, que para todas las clases de un paquete no hace falta igualar el nombre de la clase con el del paquete.

Con frecuencia se utiliza el formato `.*`; el cual tiene la ventaja de simplificar cualquier clase del paquete, aunque se pueden señalar los siguientes problemas:

NOTA

Aunque aparezca la sentencia `import paquete.*`, el compilador genera bytecode sólo para las clases utilizadas.

- Se desconoce qué clases concretas del paquete se están utilizando; al contrario de la sentencia `import`.
- Puede haber colisiones entre nombres de clases declaradas en el archivo y nombres de clases del paquete.
- Mayor tiempo de compilación debido a que el compilador busca la existencia de cualquier clase en el paquete.

EJEMPLO 8.5

Se crea el paquete `numérico` con la clase `Random` y se utiliza en una aplicación.

```
package numerico;

public Random
{
  // ...
}
```

Al utilizar la clase en otro archivo:

```
import java.util.*
import numerico.*;
```

Como en el paquete `java.util` se encuentra la clase `Random`, se produce una ambigüedad con la del paquete `numérico`; es necesario cualificar completamente el nombre de la clase `Random`, en este caso de, `java.util`.

```
java.util.Random aleatorio;  // define una referencia
```

8.6 Constructores

REGLAS

1. El constructor tiene el mismo nombre que la clase.
2. Puede tener cero, o más argumentos.
3. No tiene tipo de retorno.

Un constructor es un método que se ejecuta automáticamente cuando se crea un objeto de una clase; sirve para inicializar los miembros de la misma.

El constructor tiene el mismo nombre que clase; cuando se define, no se puede especificar un valor de retorno porque nunca devuelve uno; sin embargo, puede tomar cualquier número de argumentos.

EJEMPLO 8.6

La clase `Rectangulo` tiene un constructor con cuatro parámetros.

```
public class Rectangulo
{
  private int izdo;
  private int superior;
  private int dcha;
  private int inferior;
    // constructor
  public Rectangulo(int iz, int sr, int d, int inf)
  {
```

```
    izdo = iz;
    superior = sr;
    dcha = d;
    inferior = inf;
  }
  // definiciones de otros métodos miembro
}
```

Al crear un objeto se transfieren los valores de los argumentos al constructor, con la misma sintaxis que la llamada a un método; por ejemplo:

```
Rectangulo Rect = new Rectangulo(25, 25, 75, 75);
```

Se creó una instancia de `Rectangulo`, pasando valores concretos al constructor de la clase, de esta forma queda inicializado.

8.6.1 Constructor por defecto

Un constructor que no tiene parámetros se llama *constructor por defecto*; el cual normalmente inicializa los miembros dato de la clase con valores predeterminados.

REGLA

Java crea automáticamente un constructor por defecto cuando no existen otros constructores; tal constructor inicializa las variables de tipo numérico, como `int` o `float` a cero, las variables de tipo `boolean` a `true` y las referencias a `null`.

EJEMPLO 8.7

El constructor por defecto inicializa `x` e `y` a cero.

```
public class Punto
{
  private  int x;
  private int y;

  public  Punto()          // constructor por defecto
  {
    x = 0;
    y = 0;
  }
}
```

Cuando se crea un objeto `Punto` sus miembros dato se inicializan a cero.

```
Punto P1 = new Punto() ;   // P1.x == 0, P1.y == 0
```

PRECAUCIÓN

Tenga cuidado con la escritura de una clase con un solo constructor con argumentos; si se omite un constructor sin argumento, no será posible utilizar el constructor por defecto; por ejemplo: la definición `NomClase c = new NomClase()` no será posible.

8.6.2 Constructores sobrecargados

Al igual que se puede sobrecargar un método de una clase, también se puede sobrecargar su constructor; de hecho, los constructores sobrecargados son bastante frecuentes porque proporcionan diferentes opciones de inicializar objetos.

REGLA

Para prevenir a los usuarios de la clase de crear un objeto sin parámetros, se puede: 1) omitir el constructor por defecto; o 2) hacer el constructor privado.

EJEMPLO 8.8

La clase `EquipoSonido` se define con tres constructores; uno por defecto, otro con un argumento de tipo cadena y el tercero con tres argumentos.

```
public class EquipoSonido
{
  private int potencia;
  private int voltios;
  private int numCd;
  private String marca;

  public  EquipoSonido()   // constructor por defecto
  {
    marca = "Sin marca";
    System.out.println("Constructor por defecto");
  }
  public EquipoSonido(String mt)
  {
    marca = mt;
    System.out.println("Constructor con argumento cadena ");
  }
  public EquipoSonido(String mt, int p, int v)
  {
    marca = mt;
    potencia = p;
    voltios = v;
    numCd = 20;
    System.out.println("Constructor con tres argumentos ");
  }
  public double factura(){...}

};
```

La instanciación de un objeto `EquipoSonido` puede hacerse llamando a cualquier constructor:

```
EquipoSonido rt, gt, ht;      // define tres referencias
rt = new EquipoSonido();      //  llamada al constructor por defecto
gt = new EquipoSonido("JULAT");
rt = new EquipoSonido("PARTOLA",35,220);
```

8.7 Recolección de basura (objetos)

En Java un objeto siempre debe estar referenciado por una variable, en el momento en que deja de estar referenciado se activa la rutina de recolección de memoria, el objeto es liberado y la memoria que ocupa puede reutilizarse; por ejemplo:

```
Punto p = new Punto(1,2);
```

la sentencia `p = null` provoca que `Punto` sea liberado automáticamente.

El propio sistema se encarga de la recolectar los objetos en desuso para aprovechar la memoria ocupada; para ello hay un proceso que se activa periódicamente y toma los objetos que no están referenciados por ninguna variable. El proceso lo realiza el método `system.gc,` que implementa la recolección de basura (*garbage collection*); el si-

guiente método, por ejemplo, crea objetos `Contador` y después se liberan al perder su referencia.

```
void objetos()
{
  Contador k, g, r, s;
    // se crean cuatro objetos
  k = new Contador();
   g = new Contador();
  r = new Contador();
  s = new Contador();
  /* la siguiente asignación hace que g referencie al mismo objeto que
      k, además el objeto original de g será automáticamente
      recolectado.  */
  g = k;
  /* ahora no se activa el recolector porque g sigue apuntando al
       objeto. */
  k = null;
  /* a continuación sí se activa el recolector para el objeto original
       de r. */
  r = new Contador();

}   // se liberan los objetos actuales apuntados por g, r, s
```

8.7.1 Método `finalize()`

El método `finalize()` es especial porque se llama automáticamente si fue definido en la clase, justo antes que la memoria del objeto recolectado vaya a ser devuelta al sistema; el método no es un destructor del objeto, ni libera memoria; en algunas aplicaciones se puede utilizar para liberar ciertos recursos del sistema.

REGLA

`finalize()` es un método especial con estas características:

- No devuelve valor, es de tipo `void`;
- No tiene argumentos;
- No puede sobrecargarse, y
- Su definición es opcional.

ejercicio 8.2

Se declaran dos clases, cada una con su método `finalize()`. El método `main()` crea objetos de ambas clases; las variables que referencian a los objetos se modifican para que cuando se active la recolección automática de objetos se libere la memoria de éstos; hay una llamada a `System.gc()` para no esperar la llamada interna del sistema.

```
class Demo
{
  private   int datos;  public  Demo(){datos = 0;}
  protected void finalize()
  {
    System.out.println("Fin de objeto Demo");
  }
}

class Prueba
{
  private double x;
  public  Prueba (){x = -1.0;}
  protected void finalize()
```

```
  {
    System.out.println("Fin de objeto Prueba");
  }
}

public class ProbarDemo
{
  public static void main(String[] ar)
  {
    Demo d1, d2;
    Prueba p1, p2;
    d1 = new Demo();
    p1 = new Prueba();
    System.gc();  // no se libera ningún objeto
    p2 = p1;
    p1 = new Prueba();
    System.gc();  // no se libera ningún objeto
    p1 = null;
    d1 = new Demo();
    System.gc();  // se liberan dos objetos
    d2 = new Demo();
  }  // se liberan los objetos restantes
}
```

8.8 Autorreferencia del objeto: `this`

Una referencia al objeto que envía un mensaje es `this`, o simplemente, una referencia al objeto que llama a un método, aunque éste no debe ser `static`; internamente se define:

```
final NombreClase this;
```

por lo que no puede modificarse; las variables y métodos de las clases se referencian implícitamente por `this`; en la siguiente clase, por ejemplo:

```
class Triangulo
{
  private double base;
  private double altura;
  public double area()
  {
    return base*altura/2.0 ;
  }
}
```

En el método `area()` se hace referencia a las variables instancia `base` y `altura`. ¿De qué objeto? El método es común para todos los objetos `Triangulo`; aparentemente no distingue entre un objeto y otro, sin embargo cada variable instancia implícitamente está cualificada por `this`; es como si estuviera escrito:

```
public double area()
{
  return this.base*this.altura/2.0 ;
}
```

Fundamentalmente this tiene dos usos:

- Seleccionar explícitamente un miembro de una clase con el fin de dar más claridad o de evitar colisión de identificadores; por ejemplo:

```
class Triangulo
{
  private double base;
  private double altura;
  public void datosTriangulo(double base, double altura)
  {
    this.base = base;
    this.altura = altura;
  }
  // ...
}
```

Con this se evita la colisión entre argumentos y variables instancia.

- Que un método devuelva el mismo objeto que le llamó; de esa manera se pueden hacer llamadas en cascada a métodos de la misma clase; nuevamente se define la clase Triangulo:

```
class Triangulo
{
  private double base;
  private double altura;
  public Triangulo datosTriangulo(double base, double altura)
  {
    this.base = base;
    this.altura = altura;
    return this;
  }
  public Triangulo visualizar()
  {
    System.out.println(" Base = " + base);
    System.out.println(" Altura = " + altura);
    return this;
  }
  public double area()
  {
    return base*altura/2.0 ;
  }
}
```

Ahora se pueden concatenar llamadas a métodos de un objetivo Triangulo:

```
Triangulo t = new Triangulo();
t.datosTriangulo(15.0,12.0).visualizar();
```

8.9 Miembros static de una clase

Cada instancia u objeto de una clase tiene su propia copia de las variables de la clase; si es necesario que haya miembros no ligados a los objetos sino a la clase, es decir, comunes a todos los objetos, éstos se declaran static.

8.9.1 Variables `static`

Las variables de clase `static` son compartidas por todos los objetos de la clase; se declaran de igual manera que otra variable, añadiendo como prefijo la palabra reservada `static`; por ejemplo:

```
public class Conjunto
{
  private static int k = 0;
  static Totem lista = null;
    // ...
}
```

Las variables miembro `static` no forman parte de los objetos de la clase sino de la propia clase; se accede a ellas de la manera habitual, simplemente con su nombre; desde el exterior se accede con el nombre de la clase, el selector y el nombre de la variable:

```
Conjunto.lista =  ...;
```

Aunque también se puede acceder a través de un objeto de la clase, no es recomendable ya que los miembros `static` no pertenecen a los objetos sino a las clases.

ejercicio 8.3

Dada una clase se quieren conocer en todo momento los objetos activos en la aplicación. Se declara la clase `Ejemplo` con un constructor por defecto y otro con un argumento; ambos incrementan la variable `static cuenta`, en uno; de esa manera cada nuevo objeto queda contabilizado. También se declara el método `finalize()`, de tal forma que al activarse `cuenta` disminuye en uno.

El método `main()` crea objetos de la clase `Ejemplo` y visualiza la variable que contabiliza el número de sus objetos.

```
class Ejemplo
{
  private   int datos;
  static int cuenta = 0;
  public  Ejemplo()
  {
    datos = 0;
    cuenta++;    // nuevo objeto
  }
  public  Ejemplo(int g)
  {
    datos = g;
    cuenta++;    // nuevo objeto
  }
    //
  protected void finalize()
  {
    System.out.println("Fin de objeto Ejemplo");
    cuenta--;
  }
}
```

```
public class ProbarEjemplo
{
  public static void main(String[] ar)
  {
    Ejemplo d1, d2;

    System.out.println("Objetos Ejemplo: " + Ejemplo.cuenta);
    d1 = new Ejemplo();
    d2 = new Ejemplo(11);
    System.out.println("Objetos Ejemplo: " + Ejemplo.cuenta);

    d2 = d1;
    System.gc();
    System.out.println("Objetos Ejemplo: " + Ejemplo.cuenta);

    d2 = d1 = null;
    System.gc();
    System.out.println("Objetos Ejemplo: " + Ejemplo.cuenta);
  }
}
```

Una variable `static` suele inicializarse directamente en la definición; sin embargo, existe una construcción de Java que permite inicializar miembros `static` en un bloque de código dentro de la clase; el bloque debe venir precedido de la palabra `static`; por ejemplo:

```
class DemoStatic
{
  private static int k;
  private static double r;
  private static String cmn;
  static
  {
    k = 1;
    r = 0.0;
    cmn = "Bloque";
  }
}
```

8.9.2 Métodos `static`

Los métodos de las clases se llaman a través de los objetos; en ocasiones interesa definir métodos que sean controlados por la clase, que no haga falta crear un objeto para llamarlos, son los métodos `static`. Muchos métodos de la biblioteca Java se definen como `static`; por ejemplo, los métodos matemáticos de la clase `Math: Math.sin()`, `Math.sqrt()`.

La llamada a los métodos `static` se realiza mediante la clase: `NombreClase.metodo()`, respetando las reglas de visibilidad; aunque también se pueden llamar con un objeto de la clase, no es recomendable debido a que son métodos dependientes de la clase y no de los objetos.

Los métodos definidos como `static` no tienen asignado la referencia `this`, por ello sólo pueden acceder a miembros `static` de la clase; es un error que un método `static` acceda a otros miembros de la clase; por ejemplo:

```
class Fiesta
{
  int precio;
  String cartel;
  public static void main(String[] a)
  {
    cartel = " Virgen de los pacientes";
    precio = 1;

    ...
```

al compilar resultan dos errores debido a que desde el método `main()`, definido como `static` se accede a miembros no `static`.

EJEMPLO 8.9

La clase `SumaSerie` define tres variables y un método `static`, éste calcula la suma cada vez que se llama.

```
class SumaSerie
{
  private static long n;
  private static long m;
  static
  {
    n = 0;
    m = 1;

  }
  public static long suma()
  {
    m += n;
    n = m - n;
    return m;
  }
}
```

8.10 Consideraciones prácticas de diseño de clases

En Java, como ya conoce el lector, las clases son los componentes principales en el diseño, construcción y ejecución de programa; por esta razón nos centraremos, en una primera aproximación, en las clases predefinidas; posteriormente, en las clases definidas por el usuario.

Java viene con gran cantidad de clases predefinidas; cada una a su vez contiene métodos predefinidos que realizan tareas útiles y detalladas previamente cuando se ejecutan; el apartado siguiente enseña a utilizar los métodos incluidos en las clases predefinidas y a escribir sus propios métodos, tema que, por otra parte, ya fue analizado.

Si desea ver las definiciones completas de las clases predefinidas de Java, tales como `String`, `Math` y `Scanner`, así como la jerarquía de clases, visite el sitio `web: http://java.sun.com/javase/6/docs/api`.

En el aparado 8.10.1 haremos un repaso de los conceptos básicos de clases y métodos definidos por el usuario, analizados anteriormente.

8.10.1 Métodos y clases predefinidos

Java incorpora gran cantidad de clases predefinidas; recordemos que éstas se organizan como una colección de paquetes denominadas bibliotecas de clases; un paquete específico puede contener varias clases y cada una contiene varios métodos; un método es una colección o conjunto de instrucciones que realiza una tarea concreta. El método `main` se opera automáticamente cuando se ejecuta un programa Java; otros se activan sólo cuando se invocan; las clases predefinidas tienen mucha importancia en sí mismas y su buen uso favorecerá el diseño y construcción de las clases propias del usuario y los principios de orientación a objetos. Esta sección analiza el uso de las clases predefinidas; cada una contiene muchos métodos predefinidos que realizan tareas importantes; para utilizarlas se necesita conocer el nombre del paquete, de la clase y del método, así como su modo de ejecución.

Existen dos tipos de métodos en una clase `static` y `non-static`; posteriormente veremos cómo funcionan. Los métodos de la clase se pueden ejecutar sin necesidad de conocer el nombre de los parámetros o argumentos (en caso de existir); éstas son algunas clases predefinidas:

`Math` —incluida en el paquete `java.lang`— contiene potentes y útiles métodos tales cómo `pow`, método *power* (potencia) que se utiliza para calcular *x* en un programa; el modo de invocar al método es:

```
nombreClase.nombreMetodo(x,y,...)

Math.pow(x,y)= x^y
Math.pow(4,3)= 4^3 = 64
Math.pow(16,0.5)= 16^0.5 = √16 = 4
```

`x` e `y` se denominan parámetros o argumentos reales de `pow`.

NOTA

Utilización de un método predefinido en un programa; es necesario conocer:

1. El nombre de la clase que contiene el método,
2. El nombre del paquete que contiene la clase para importarla,
3. El nombre del método y el número de parámetros o argumentos, así como su tipo y orden.

Por ejemplo:

```
Math.random
```

Otra clase predefinida ampliamente utilizada y que merece un capítulo completo, es `String`; la cual pertenece al paquete `java.lang` y se utiliza para manipular cadenas o secuencias de caracteres; también proporciona diferentes métodos que permiten procesar cadenas de diversas formas; por ejemplo: calcular la longitud de una cadena; extraer subcadena, es decir obtener una cadena de otra; o concatenar o unir dos o más cadenas. La clase `String` está disponible de modo automático en Java y, al contrario de otras clases, no necesita importarse y puede utilizar un método de `String` sólo con su nombre, parámetros y tarea hace el método.

Desde el punto de vista conceptual una cadena es una secuencia de 0 o más caracteres que se distinguen por su índice; en Java se deben utilizar entrecomilladas y constituyen una instancia de la clase `String`. El índice es la posición del carácter en la cadena donde 0 es el índice del primer carácter, 1 el del segundo y así sucesivamente; su longitud es el número de caracteres que contiene (índice + 1). Como se verá más adelante, en los programas de Java las cadenas se deben introducir en variables para poder representarlas y utilizarlas; por ejemplo:

```
"Juan sin nombre"              cadena longitudinal 15
Nombre= "Juan sin nombre"      nombre, variable de cadena
```

En los parámetros de Java, las variables de cadena deben ser declaradas como objetos de la clase `String` de las formas siguientes:

```
String nombre = "Juan sin miedo";
```

o

```
String nombre;
Nombre = "Juan sin miedo";
```

La sintaxis del método `length` (longitud) es `int length()` y devuelve el número de caracteres, en este caso 15, de la cadena cuando se invoca con la sentencia `nombre.length()`.

8.10.2 Clases definidas por el usuario

Ya hemos utilizado clases desde un punto de vista práctico y hasta ahora todas eran predefinidas o contenían el método `main` debido a que, como hemos mencionado, para construir un programa completo se deben combinar varias clases, una de las cuales obligatoriamente tiene un método `main`; el libro se centra en el diseño y construcción de clases definidas por el usuario y la utilización de clases predefinida de los paquetes de las bibliotecas de clases. La sintaxis de una clase es:


```
class NombreClase                          //cabecera de la clase
{
  MiembrosClase                            //cuerpo de la clase
}
```

Los miembros de la clase pueden ser constructores, métodos —contienen las operaciones, funciones o procedimientos de la clase— y los campos —contienen los datos o atributos de la clase— y entonces una sintaxis más completa sería:


```
class NombreClase
{
  constructor1  <---------- Definiciones de constructores
  constructor2

  ...

  método1       <---------- Definiciones de métodos
  método2

  ...

  campo1        <---------- Definiciones de campos
  campo2

  ...
}
```

Los constructores, como ya se mencionó antes, son métodos que se utilizan para crear una instancia del objeto de la clase; inicializan los campos de datos para cada una

de ellas o ejemplar del objeto; cuando se define una clase, también se pueden especificar los constructores de la clase y es posible que existan 1 o más constructores siempre con el mismo nombre de la clase. Si no se definen los constructores dentro de la clase, el compilador de Java crea un constructor por defecto; por esta razón y dada su importancia, nos centraremos en el diseño general de una clase; también debemos mencionar que el orden de definiciones de métodos y declaraciones de los datos pueden ser el señalado en la sintaxis o en sentido contrario; primero declaraciones de datos y luego definiciones de métodos.

```
class NombreClase
{
  declaración de datos
  definición de métodos
}
```

EJEMPLO 8.10

Declaración de la clase `Empleado` utilizada en los sistemas de gestión de personal de un programa de contratación, gestión de nóminas, seguridad social, etcétera.

```
class Empleado
{
  private String nombre;       //visibilidad privada
  private double salario;      //salario en bruto
  //más datos
  public String lerrNombre ()  //nombre empleado
  {
    ...
  }

  //más métodos
}
```

Clase `Tiempo` para conocer la hora de un suceso

```
class Tiempo
{
  private int hora, minuto, seg; // datos
  public Tiempo() {...}          // constructor
  public void iniciar(int h,int m,int s){...}
  public int leerHora () {...}
  public int leerMin  () {...}
}
```

8.11 Biblioteca de clases de Java

Java incorpora una amplia biblioteca de clases e interfaces denominado Java API; sus clases se pueden utilizar para formar otras nuevas, crear objetos, utilizar sus métodos; la biblioteca se organiza por paquetes que contienen colecciones de clases; para emplear estas últimas sin tener que hacerlas preceder del nombre del paquete se utiliza la sentencia `import`; algunos de los paquetes más utilizados son:

```
java.applet     java.awt          java.awt.image       java.awt.peer

java.io         java.lang         java.net             java.util
```

El paquete `java.lang` es el más importante; como se considera el estándar, todos los programas lo importan automáticamente; contiene a las clases que encapsulan los tipos de datos primitivos: `Double`, `Float`, `Long`, etcétera; `String` y `StringBuffer` para el tratamiento de cadenas también se encuentran en este paquete, al igual que `Math` con las funciones matemáticas mas importantes; pero las clases más importantes de `java.lang` son `Object`, `System` , `Thread` y `Throwable`.

El paquete `java.util` define un conjunto de clases para distintos cometidos: `Date` sirve para manejo de fechas en distintos formatos, `Random` genera números aleatorios, `Scanner` es para entrada de datos al programa; este paquete sí debe importarse si se utiliza alguna de sus clases o interfaz; por ejemplo:

```
import java.util.*;          para cualquier clase del paquete
import java.util.Date;       para utilizar sólo la clase Date
import java.util.Scanner;    para emplear sólo la clase Scanner
```

8.11.1 Clase `System`

Esta clase se utiliza con frecuencia, ya que es un depósito de objetos asociados con entrada y salida estándar, y de métodos útiles; no se pueden crear objetos de esta clase; sus miembros se definen como `static`, por lo que para referirse a ellos se antepone el nombre de la clase: `System`.

`System.in` es un objeto definido: `static final InputStream in`; normalmente corresponde con la entrada por teclado y se utiliza como argumento para el constructor que crea un objeto de entrada `Scanner`:

```
Scanner entrada = new Scanner(System.in);
```

El objeto `System.out` queda definido: `static final PrintStream out`; normalmente se asocia con la salida por pantalla; es habitual la llamada a los métodos

```
System.out.print()   y
System.out.println()
```

para la salida de datos previamente convertidos en cadena; el segundo se diferencia del primero en que, una vez mandada la cadena a la pantalla, salta a la línea siguiente.

Entre los métodos de la clase se encuentran `exit()` y `gc()`. El primero termina la ejecución de una aplicación, está declarado: `static void exit(int status)`. Se acostumbra que el argumento `status` sea 0 si la terminación es sin errores, o un valor distinto de 0 para indicar una anomalía.

El método `gc()` (*garbage collector*) está declarado: `staic void gc()`; una llamada a `System.gc()`, hace que se active la liberación de objetos creados pero no referenciados por ninguna variable.

8.11.2 Clase `Object`

`Object` es la superclase base de todas las clases de Java; todas heredan de ella y, en consecuencia, toda variable referencia a una clase se convierte, automáticamente, al tipo `Object`; por ejemplo:

```
Object g;
String cd = new String("Barranco la Parra");
Integer y = new Integer(72);  // objeto inicializado a 72

g = cd;  // g referencia al mismo objeto que cd
g = y;   // g ahora referencia a un objeto Integer
```

La clase `Object` tiene dos métodos importantes: `equals()` y `toString()`; generalmente se redefinen en las clases para especializarlos.

`equals()`

Compara el objeto que hace la llamada con el que se pasa como argumento, devuelve `true` si son iguales.

```
boolean equals(Object k);
```

El siguiente ejemplo compara dos objetos; la comparación es `true` si contienen la misma cadena.

```
String ar = new String("Iglesia románica");
String a = "Vida sana";
if (ar.equals(a))             //...no se cumple, devuelve false
```

`toString()`

Este método construye una cadena que representa el objeto, devuelve la cadena; normalmente se redefine en las clases para dar detalles explícitos de los objetos de la clase.

```
String toString()
```

El siguiente ejemplo de un objeto `Double` llama al método `toString()` y asigna la cadena a una variable.

```
Double r = new Double(2.5);
String rp;
rp = r.toString();
```

8.11.3 Operador `instanceof`

Con frecuencia se necesita conocer la clase de la que un objeto es instancia; se debe considerar que en las jerarquías de clases se dan conversiones automáticas entre clases derivadas y su base, en particular, cualquier referencia se puede convertir a una variable de tipo `Object`.

Con el operador `instanceof` se determina la clase a la que pertenece un objeto, tiene dos operandos; el primero es un objeto y el segundo, una clase. Evalúa la expresión a `true` si el primer operando es una instancia del segundo; la siguiente función tiene un argumento de tipo `Object`, así puede recibir cualquier referencia y seleccionar la clase a la que pertenece el objeto transmitido; por ejemplo: `String`, `Vector`:

```
public void hacer (Object g)
{
  if (g instanceof String)
  ...
```

NOTA

El operador `instanceof` se puede considerar relacional, su evaluación da como resultado un valor de tipo `boolean`.

```
else if (g instanceof Vector)
...
```

8.11.4 Clase `Math`, funciones matemáticas

`Math` es una clase diseñada para utilidades matemáticas; contiene una colección de funciones que se pueden necesitar dependiendo del tipo de programación a realizar; por ejemplo: para calcular x^n

```
Math.pow(x,n)
```

Sin embargo, para calcular x^2 puede ser más eficiente calcular `x*x`.

Para extraer la raíz cuadrada de un número se puede utilizar el método `sqrt`:

```
double q = 9.0;
double y = Math.sqrt(x);
System.out.println(y = " + y); // se visualiza 3
```

EJEMPLO 8.12

Calcular el valor de $(-b + \sqrt{(b^2 - 4ac)})/2a$

```
(-b + Math.sqrt(b*b -4*a*c))/(2*a)
```

La tabla 8.2 muestra los métodos de `Math`.

Tabla 8.2 Métodos matemáticos.

Función	Devuelve
`Math.sqrt(x)`	raíz cuadrada de x (x ≥ 0)
`Math.pow(x,y)`	x^y (x>0, x = 0 y >0, x<0 e y entero)
`Math.exp(x)`	e^x
`Math.log(x)`	logaritmo natural (ln(x), x > 0)
`Math.round(x)`	entero más próximo a x (long)
`Math.ceil(x)`	entero más pequeño ≥ x (double)
`Math.floor(x)`	entero más largo ≥ x (double)
`Math.abs(x)`	valor absoluto de x
`Math.max(x,y)`	valor mayor de x e y
`Math.min(x,y)`	valor menor de x e y
`Math.sin(x)`	seno de x (x en radianes)
`Math.cos(x)`	coseno de x (x en radianes)
`Math.tan(x)`	tangente de x (x en radianes)
`Math.asin(x)`	arco seno de x
`Math.acos(x)`	arco coseno de x
`Math.atan(x)`	arco tangente de x
`Math.atan2(y,x)`	arco cuya tangente es y/x
`Math.toRadianes(x)`	convierte x grados a radianes
`Math.toDegrees(x)`	convierte x radianes a grados

resumen

- Los TAD o tipos abstractos de datos describen un conjunto de objetos con la misma representación y comportamiento; por consiguiente, se pueden utilizar implementaciones alternativas para el mismo tipo abstracto de dato sin cambiar su interfaz; en Java se implementan mediante clases.
- Una clase es un tipo de dato definido por el programador que sirve para representar objetos del mundo real; un objeto de una clase tiene dos componentes: un conjunto de atributos o variables instancia y un conjunto de comportamientos o métodos. Los atributos también se llaman *variables instancia* o *miembros dato* y los comportamientos se llaman *métodos miembro*.

```
class Circulo
{
  private double centroX;
  private double centroY;
  private double radio;
  public double superficie(){}
}
```

- Un objeto es una instancia de una clase y una variable cuyo tipo sea la clase es una referencia a un objeto de la misma.

```
Circulo unCirculo;   //  variable del tipo clase
```

- Una clase, en cuanto a visibilidad de sus miembros, tiene tres secciones: pública, privada y protegida.
- La sección pública contiene declaraciones de los atributos y el comportamiento del objeto al que son accesibles los usuarios; se recomienda declarar los constructores en esta sección.
- La sección privada contiene los métodos y los datos que son ocultos o inaccesibles a los usuarios del objeto; éstos son accesibles sólo para los miembros del objeto.
- Los miembros de la sección protegida son accesibles para cualquier usuario de la clase que se encuentre en el mismo paquete; también son accesibles para las clases derivadas; el acceso por defecto, sin modificador, tiene las mismas propiedades.
- Un constructor es un método miembro con el mismo nombre que su clase; no puede devolver un tipo, pero puede ser sobrecargado.

```
class Complejo
{
  public Complejo (double x, double y){}
  public Complejo(complejo z){}
}
```

- El constructor es un método especial que se invoca cuando se crea un objeto; normalmente se utiliza para inicializar los atributos de un objeto. Por lo general, al menos se define un constructor sin argumentos, llamado constructor por defecto; en caso de no concretarse, implícitamente queda definido un constructor sin argumentos que inicializa cada miembro numérico a 0, los miembros `boolean` a `true` y las referencias a `null`.
- El proceso de crear un objeto se llama *instanciación* (creación de instancia); en Java se crea un objeto con el operador `new` y un constructor de la clase.

```
Circulo C = new Circulo();
```

- En Java, la liberación del objeto es automática cuando deja de estar referenciado por una variable es candidato a que la memoria que ocupa sea liberada y posteriormente reutilizada; dicho proceso se denomina *garbage collection*, el método `System.gc()` lo realiza.
- Los paquetes son agrupaciones de clases relativas a un tema: el sistema suministra paquetes con clases que facilitan la programación, como el caso de *java.lang* que es donde se encuentran las clases más utilizadas, por eso se incorpora automáticamente a los programas.
- Los miembros de una clase definidos como `static` no están ligados a los objetos de ésta sino que son comunes a todos ellos; se cualifican con el nombre de la clase, por ejemplo: `Math.sqrt(x)`.

conceptos clave

- Abstracción
- Componentes
- Constructores
- Encapsulación
- Especificadores de acceso: `public`, `protected`, `private`
- Interfaz
- Ocultación de la información
- Reutilización
- Tipos de datos y variables

ejercicios

8.1 ¿Qué está mal en la siguiente definición de la clase?

```
import java.io.*;
class Buffer
{
  private char datos[];
  private int cursor ;
  private Buffer(int n)
  {
    datos = new char[n]
  };
  public static int Long( return cursor;);
  public String contenido(){}
}
```

8.2 Dado el siguiente programa, ¿es legal la sentencia de `main()`?

```
class Punto
{
  public   int x, int y;
  public Punto(int x1, int y1) {x = x1 ; y = y1;}
}
class CreaPunto
{
  public static void main(String [] a)
  {
    new Punto(25, 15);        //¿es legal esta sentencia ?
    Punto p = new Punto();    //¿es legal esta sentencia ?
  }
}
```

8.3 Con la respuesta del ejercicio anterior, ¿cuál será la salida del siguiente programa?

```
class CreaPunto
{
  public static void main(String [] a)
  {
    Punto q;
    q = new Punto(2, 1);
    System.out.println("x = " + p.x + "\ty = " + p.y);
```

8.4 Dada la siguiente clase, escribir el método `finalize()` y un programa que cree objetosy después se pierdan las referencias a los objetos creados y se active el método `finalize()`.

```
class Operador
{
  public float memoria;
  public Operador()
  {
    System.out.println("Activar maquina operador");
    memoria = 0.0F;
  }

  public float sumar(float f)
  {
    memoria += f;
    return memoria;
  }
}
```

8.5 Realizar una clase `Vector3d` que permita manipular vectores de tres componentes en coordenadas `x`, `y`, `z` de acuerdo con las siguientes normas:

- Usar sólo un método constructor.
- Usar un método miembro `equals()` para saber si dos vectores tienen sus componentes o coordenadas iguales.

8.6 Incluir en la clase del ejercicio anterior el método `normamax` que permita obtener la norma de dos vectores; considere que la norma de un vector $v = x, y, z$ es $\sqrt{x^2+y^2+z^2}$).

8.7 Realizar la clase `Complejo` que permita la gestión de números complejos (un número complejo es igual a dos números reales `double`: una parte real más una parte imaginaria). Las operaciones a implementar son las siguientes:

- `establecer()` permite inicializar un objeto de tipo `Complejo` a partir de dos componentes `double`.
- `imprimir()` realiza la visualización formateada de `Complejo`.
- `agregar()` sobrecargado añade respectivamente `Complejo` a otro y dos componentes `double` a un `Complejo`.

8.8 Añadir a la clase `Complejo` del ejercicio anterior las siguientes operaciones:

- suma: `a + c = (A+C, (B+D)i)`.
- resta: `a - c = (A-C, (B-D)i)`.
- multiplicación: `a*c=(A*C-B*D, (A*D+B*C)i)`
- multiplicación: `x*c = (x*C, x*Di)`, donde *x* es real.
- conjugado: `~a = (A ,-Bi)`.

Siendo `a = A+Bi; c = C+Di`

8.9 Implementar la clase `Hora`; donde cada objeto represente una hora específica del día, almacenando horas, minutos y segundos como enteros; incluir un constructor, métodos de acceso, un método `adelantar(int h, int m, int s)` para actualizar la hora de un objeto existente, un método `reiniciar(int h, int m, int s)` que reinicie la hora actual de un objeto existente y un método `imprimir()`.

problemas

8.1 Implementar la clase `Fecha` con miembros dato para mes, día y año; sus clases deben representar una fecha que almacene día, mes y año como enteros; incluir un constructor por defecto, métodos de acceso, un método `reiniciar (int d, int m, int a)` para reiniciar la fecha de un objeto existente, un método `adelantar(int d, int m, int a)` para cambiar una fecha existente (día, `d`, mes, `m`, y año `a`) y un método `imprimir()`. Escribir un método de utilidad, `normalizar()`, para asegurar que los miembros dato están en el rango correcto 1 ≤ año, 1 ≤ mes ≤ 12, día ≤ días (Mes), donde `días(Mes)` es otro método que devuelve el número de días de cada mes.

8.2 Ampliar el programa anterior de modo que pueda aceptar años bisiestos. Un año es bisiesto si es divisible entre 400, o si es divisible entre 4 pero no entre 100; por ejemplo, 1992 y 2000 fueron años bisiestos, 1997 y 1900 no.