Unidad III

Clases y objetos.

3.1 Definición de una clase.

Vamos a poner un ejemplo del segundo tipo, que simule la utilización de liberías de clases para crear un interfaz gráfico de usuario como Windows 3.1 o Windows 95.
Supongamos que tenemos una clase que describe la conducta de una ventana muy simple, aquella que no dispone de título en la parte superior, por tanto no puede desplazarse, pero si cambiar de tamaño actuando con el ratón en los bordes derecho e inferior.
La clase Ventana tendrá los siguientes miembros dato: la posición x e y de la ventana, de su esquina superior izquierda y las dimensiones de la ventana: ancho y alto.

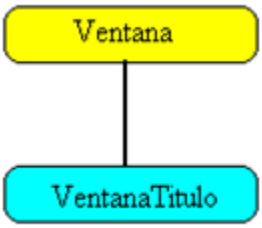
```
package ventana;


public class Ventana {
protected int x;
protected int y;
protected int ancho;
protected int alto;


public Ventana(int x, int y, int ancho, int alto) {
this.x=x;
this.y=y;
this.ancho=ancho;
this.alto=alto;
}
public void mostrar(){
System.out.println("posición : x="+x+", y="+y);
System.out.println("dimensiones : w="+ancho+", h="+alto);
}
public void cambiarDimensiones(int dw, int dh){
ancho+=dw;
alto+=dh;
}
}
```

La clase derivada

Incrementamos la funcionalidad de la clase Ventana definiendo una clase derivada denominada VentanaTitulo. Los objetos de dicha clase tendrán todas las características de los objetos de la clase base, pero además tendrán un título, y se podrán desplazar (se simula el desplazamiento de una ventana con el ratón).



La clase derivada heredará los miembros dato de la clase base y las funciones miembro, y tendrá un miembro dato más, el título de la ventana.

```
public class VentanaTitulo extends Ventana{

protected String titulo;

public VentanaTitulo(int x, int y, int w, int h, String nombre)
{
super(x, y, w, h);
titulo=nombre;
}
```

extends es la palabra reservada que indica que la clase VentanaTitulo deriva, o es una subclase, de la clase Ventana. La primera sentencia del constructor de la clase derivada es una llamada al constructor de la clase base mediante la palabra reservada super. La llamada super (x, y, w, h);
En la clase derivada se define una función que tiene el mismo nombre y los mismos parámetros que la de la clase base. Se dice que redefinimos la función mostrar en la clase derivada. La función miembro mostrar de la clase derivada VentanaTitulo hace una llamada a la función mostrar de la clase base Ventana, mediante super.mostrar();

3.2 Declaración de clases.

La declaración de una clase define la estructura de la misma. Dicho de otra forma, la declaración de una clase informa de los elementos que la conforman. Posteriormente a ser declarada, una clase debe ser implementada convenientemente, es decir, se debe escribir el código correspondiente a los procedimientos y funciones que determinan el funcionamiento de esa clase.
Las clases se declaran en la sección TIPO del script pues las clases son, al fin y al cabo, tipos de datos.
La programación orientada a objetos se basa en la programación de clases; a diferencia de la programación estructurada, que está centrada en las funciones.
Una clase es un molde del que luego se pueden crear múltiples objetos, con similares características.
Una clase es una plantilla (molde), que define atributos (variables) y métodos (funciones)
La clase define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.
Debemos crear una clase antes de poder crear objetos (instancias) de esa clase. Al crear un objeto de una clase, se dice que se crea una instancia de la clase o un objeto propiamente dicho.

Tipos de atributos.
Objetivos:
a) Profundizar en el concepto de atributo de una clase e indicar los tipos de atributos en Java
b) Interpretar el código fuente de una aplicación Java donde aparecen distintos tipos de atributos
c) Construir una aplicación Java sencilla, convenientemente especificada, que emplee clases con diferentes tipos de atributos.

Los atributos, también llamados datos o variables miembro son porciones de información que un objeto posee o conoce de sí mismo. Una clase puede tener cualquier número de atributos o no Tener ninguno. Se declaran con un identificador y el tipo de dato correspondiente.

Modificador	Visibilidad
public	Pública (+)
protectec	Protegida / en la herencia(#)
private	Privada(-)
package	De paquete (~)

Métodos.

Java como todo lenguaje de programación orientado a objetos utiliza los llamados métodos. A continuación veremos cómo se crea un método y como se utilizan.
Se podría decir que existen 2 grandes tipos de métodos, el primer tipo de método son métodos que realizan procesos, puedes realizar cualquier operación con ellos, sin embargo el propósito es manipular variables existentes. El segundo tipo de métodos son los que realizan un proceso o cálculo, y calculan una variable específica, un ejemplo podría ser un método para obtener el valor de una multiplicación.
Los métodos en java pueden tener parámetros, es decir, que un método puede utilizar variables predefinidas para ser utilizadas en sus procesos.

Encapsulamiento.
Como se puede observar de los diagramas, las variables del objeto se localizan en el centro o núcleo del objeto. Los métodos rodean y esconden el núcleo del objeto de otros objetos en el programa. Al empaquetamiento de las variables de un objeto con la protección de sus métodos se le llama encapsulamiento. Típicamente, el encapsulamiento es utilizado para esconder detalles de la puesta en práctica no importantes de otros objetos. Entonces, los detalles de la puesta en práctica pueden cambiar en cualquier tiempo sin afectar otras partes del programa.

El encapsulamiento de variables y métodos en un componente de software ordenado es, todavía, una simple idea poderosa que provee dos principales beneficios a los desarrolladores de software.

Modularidad.
esto es, el código fuente de un objeto puede ser escrito, así como darle mantenimiento, independientemente del código fuente de otros objetos. Así mismo, un objeto puede ser transferido alrededor del sistema sin alterar su estado y conducta.
Ocultamiento de la información, es decir, un objeto tiene una "interfaz publica" que otros objetos pueden utilizar para comunicarse con él. Pero el objeto puede mantener información y métodos privados que pueden ser cambiados en cualquier tiempo sin afectar a los otros objetos que dependan de ello.
Los objetos proveen el beneficio de la modularidad y el ocultamiento de la información. Las clases proveen el beneficio de la reutilización. Los programadores de software utilizan la misma clase, y por lo tanto el mismo código, una y otra vez para crear muchos objetos.
En las implantaciones orientadas a objetos se percibe un objeto como un paquete de datos y procedimientos que se pueden llevar a cabo con estos datos. Esto encapsula los datos y los procedimientos. La realidad es diferente: los atributos se relacionan al objeto o instancia y los métodos a la clase. ¿Por qué se hace así? Los atributos son variables comunes en cada objeto de una clase y cada uno de ellos puede tener un valor asociado, para cada variable, diferente al que tienen para esa misma variable los demás objetos. Los métodos, por su parte, pertenecen a la clase y no se almacenan

en cada objeto, puesto que sería un desperdicio almacenar el mismo procedimiento varias veces y ello va contra el principio de reutilización de código.

3.3 Miembros de una clase.

Campos de datos dentro de una clase: *variables miembros o propiedades de la clase*.

Operaciones con los datos de una clase: *funciones miembros o métodos de la clase*.

Los miembros de una clase (propiedades y métodos) pueden ser:

- *Públicos*; se puede acceder a ellos desde fuera de la clase
- *Privados*; sólo se puede acceder a ellos desde dentro de la clase

Para acceder a propiedades privadas de la clase se definen *métodos de acceso*.

Un miembro puede ser *estático*, si pertenece a la clase en sí y no a los objetos de la clase**.**

Definición en Java de una clase con sus propiedades y métodos:

```
<modificador> class <nombre_de_la_clase> {
   private <tipo_de_dato> <nombre_del_dato>;
   public <nombre_de_la_clase> (<lista_de_argumentos>) {
      <codigo_del_metodo_constructor>
   }
   public <tipo_de_dato_devuelto> <nombre_del_metodo>
     (<lista_de_argumentos>) {
      <codigo_del_metodo>
   }
```

3.4 Ámbito referente a una clase.

El ámbito de una variable es la región de un programa dentro del cual la variable puede ser referenciada con su nombre simple.

Ámbito es diferente de visibilidad, el cual aplica sólo a variables miembro y determina si la variable puede ser usada desde fuera de la clase dentro de la cual es declarada.

La visibilidad se establece con un modificador de acceso.

La localización de la declaración de la variable dentro del programa establece su ámbito y la coloca dentro de una de estas cuatro categorías:

- Variable miembro

(pertenece a una clase).

- Variable local
- Parámetro de método.
- Parámetro de manejador

de excepciones.

3.5 Especificadores de acceso.

Hemos comentado que en el fichero .h los atributos son privados y los métodos son públicos. Está claro por qué son públicos los métodos: los hemos precedido con la palabra **public**. Así, desde cualquier otro objeto o función se podrá acceder a ellos.

Los atributos son privados (sólo podrán ser leídos o modificados por los métodos del propio objeto), y esto lo podemos conseguir precediéndolos con la palabra **private**. No lo hemos hecho así porque ese es el valor por defecto en C++: si no indicamos lo contrario, se asume que los atributos y métodos son privados.

Existe un tercer modo de acceso, que se denota con la palabra **protected**. Los atributos y/o métodos que definamos como protegidos serán accesibles sólo por los métodos de este objeto y por sus descendientes (dentro de poco veremos la herencia).

Hay una excepción a todo esto: podremos declarar funciones "amigas" (**friend**), que sí tendrán acceso a los miembros privados y/o protegidos, pero es algo que no trataremos... al menos por ahora.

3.6 Creación de objetos.

A usa B envía mensajes a B

A tiene-un B A contiene atributo clase B
También llamada Agregación

A es-un Bherencia de B

Diagrama de clases muestras estas asociaciones.
Por ejemplo Jgrasp puede generar estos diagramas a partir del código Java.
Otras editores también.
Ej: Rational Rose y Together (pagadas)
ArgoUML, GebtleWare Open source.

3.7 Puntero this.

El this es un puntero (C++) o referencia (Java) al objeto.

El programador no lo crea ni lo inicializa. Cuando se llama a una función miembro, el código oculto genera el this con la dirección del objeto y lo pasa como parámetro.

En el cuerpo de método se puede referir al objeto actual, desde donde fue llamado el método, utilizando el ***this***.

En Java, sólo se puede usar ***this*** dentro del cuerpo de la definición de un método de instancia, nunca en el cuerpo de un método de clase.

3.8 Constructores y destructores.

- Método que inicializa el objeto en su creación.
- Se llama automáticamente cuando se crea un objeto.

- ❖ Su nombre es igual que el de la clase y no tiene tipo de retorno.
- ❖ Java proporciona un constructor sin parámetros por defecto que deja de estar disponible cuando se añade algún constructor.

■ Destructor es un:

✓ Es un método perteneciente a una clase que es ejecutado de forma automática cuando un objeto es destruido. Java no soporta los destructores.

3.9 Clases Predefinidas.

Las clases contenedor/colección (como java.util.Vector y java.util.Hashtable) son clases predefinidas de las bibliotecas y no se acostumbra mostrarlas de forma explícita en el diagrama de clases porque aportan muy poca información nueva e incorporan ruido.

- Se utiliza de la forma
- Import elemento.elemento.*
- Ejemplo
- Import java.lang.*
- Recuerda que todo recurso comienza en java para la utilización de clases predefinidas

3.10 Definición, creación y reutilización de paquetes/librerías.

El Java API (Application Program Interfase) es un conjunto de librerías que permiten el desarrollo de aplicaciones en Java, brinda funciones de uso común para el programador como por ejemplo:

Creación y manejo de elementos de GUI

Manejo de archivos

Funciones de red

Comunicación entre programas

Existen dentro de su librería clases gráficas (awt y swing), las cuales permiten crear objetos gráficos comunes altamente configurables y con una arquitectura independiente de la plataforma.

Hay gran cantidad de herramientas para generar interfaces gráficas como:

- JBuilder
- NetBeans
- Fote4J
- Jdeveloper
- Eclipse

Se puede acceder a bases de datos fácilmente con *JDBC*, independientemente de la plataforma utilizada.

Existen clases JDBC para las Bases de Datos más comunes, entre ellas:

- Oracle
- PostgreSQL
- MySQL

3.11 Manejo de excepciones

Condición anormal que se produce en una porción de código durante su ejecución.

- Cuando aparece una condición excepcional se crea un objeto *Throwable* que se envía al método que la ha generado.

Permite la detección y corrección de errores en ejecución.

- Se diferencia el código normal del código de tratamiento de errores.
- Se usa cuando no se puede resolver la situación anómala directamente en el contexto.

Excepciones:

- El programador proporciona el código que las trate.

Impiden completar la ejecución del código.

Errores:

Situaciones de error normalmente no recuperables.

El programador no tiene porque proporcionar el código que las gestione.