PRO

CAPITULO 10

SUBCLASES E INTERFACES

Las caracteristicas fundamentales de la POO son abstracción,encapsulamiento,herencia y polimorfismo.
La herencia provee el mecanismo más simple para especificar una forma alternativa de acceso a una clase existente,o bien para definir una nueva clase que añada nuevas características a una clase existente.Esta nueva clase se denomina subclase o clase derivada y la claase existente,superclase o clase base.

Con la herencia todas las clases están clasificadas en una jerarquía escrita.Cada clase tiene su superclase(la clase superior en la jerarquía),y cada clase puede tener una o más subclases.Las clases que están en la parte inferior en la jerarquía.

Una jerarquía de clases muestra cómo los objetos se derivan de otros objetos más simples heredando su comportamiento.

CLASES Y METODOS ABSTRACTOS

En una jerarquía de clases,una clase es tanto más especializada cuanto más alejada esté de la raíz,entendiendo por clase raíz aquiella de la cual herezan directa o indirectamente el resto de las clases de la jerarquía;y es tanto más genérica cuanto más cerca esté de la raíz.

Cuando una clase se diseña para ser genérica,es casi seguiro que no necesitaremos crear objetos de ella;la razón de su existencia es proporcionar los atributos y comportamientos que serán compartidos por todas sus subclases.Una clase que se comporte de la forma descrita se denomina clase abstracta y se define como tal calificándola explícitamente abstracta(abstract).

Una clase abstracta puede contener el mismo tipo de miembros que una clase que no lo sea,y ademas pueden contener metodos abstractos que una clase no abstracta no puede contener.
¿Que es un metodo abstracto?Es un metodo calificado abstract con la particularidad de que no tiene cuerpo.

¿Por qué no tiene cuerpo?Porque la idea es proporcionar métodos que deban ser redefinidos en las subclases de la clase abstracta con la intencion de adaptarlos a las necesidades particulares de éstas.

SUBCLASES Y HERENCIA

La herencia es la solución para reutilizar código perteneciente a otras clases.

DEFINIR UNA SUBCLASE

Una subclase es un nuevo tipo de objetos definido por el usuario que tiene la propiedad de heredar los atributos y métodos de otra clase definida previamente ,denominada superclase.
La palabra clave extends significa que se está definiendo una clase denominada nombre_subclase que es una extensión de otra denominada nombre_superclase,también se puede decir que nombre_subclase es una clase derivada de nombre_superclase.

Si no se especifica la cláusula extends con el nombre de la superclase,se entiende que la superclase es la clase Object.
Una subclase puede serlo d euna sola superclase,lo que se denomina herencia simple o derivacion simple.Java,a diferencia de otros lenguajes orientados a objetos,no permite la herencia múltiple o derivación múltiple,esto es,que una subclase se derive de dos o más clases.

Una subclase puede , a su vez,ser una superclase de otra clase,dando lugar así a una jerarquía de clases.Por lo tanto,una clase puede ser una superclase directa de una subclase,si figura explícitamente en la definicion de la subclase,o una superclase indirecta si está varios niveles arriba en la jerarquía de clases,y por lo tanto no figura explícitamente en el encabezado de la subclases

CONTROL DE ACCESO A LOS MIEMBROS DE LAS CLASES

Para controlar el acceso a los mie,bros de una clase,Java provee las palabras clave private,protected y public,o bien pueden omitirse (acceso predeterminado).

QUE MIEMBROS HEREDA UNA SUBCLASES

Una subclase hereda todos los miembros de su superclase,excepto los constructores lo que no significaque tenga acceso directo a todos los miembros.Una consecuencia inmediata de esto es que la estructura interna de datos de un objeto de una subclase estara formada por los atributos que ella define y por los heredados de su superclase.
Una subclase no tiene acceso directo a los miembros privados(private) de su superclase.
Una subclase sí puede acceder directamente a los miembros públicos y protegidos d esu supreclase;y en el caso de que pertenezca al mismo paquete de su superclase,tambien puede acceder a los miembros predeterminados.

Una subclase puede añadir sus propios atributos y métodos.Si el nombre de alguno de estos miembros coincide con el de un miembro heredadomeste ultimo queda oculto para la subclase,que se traduce en que la subclase ya no puede acceder directamente a ese miembro.Lógicamente,lo expuestro tiene sentido siempre que nos refiramos a los miembros de la superclase a los que ña subclase podía acceder ,segun el control de acceso aplicado.

Los miembros heredados por una subclase pueden,a su vez,ser heredados por más subclases de ella.A esto se le llama propagación de herencia.
Es posible referirse a un miembro oculto utilizando la palabra reservada super de Java:super.miembro_oculto.

Los “métodos” de una subclase no tiene acceso alos miembros privados de su superclase,pero sí lo tienen a sus miembros protegidos y publicos;y si la subclase pertenece al mismo paquete que la superclase también tiene acceso a sus miembros predeterminados.
Esta restriccion puede sorprender,pero es así para imponer la encapsulación.Si una subclase tuviera acceso a los miembros privados de su superclase entonces cualquiera podría acceder a los miembros privados de una clase,simplemento derivando una clase d eella.Consecuentemente,si una subclase quiere acceder a los miembros privados de su superclase,debe hacerlo através de la interfaz pública,protegida,o predeterminada en su caso,de dicha superclase.

ATRIBUTOS CON EL MISMO NOMBRE

Una subclase puede acceder directamente aun atributo público,protegido,o predeterminado en su casomde su superclase.Qué sucede si definidos en la subclas euno de estos atributos,con el mismo nombre que tiene en la superclase

La tecnica de realizar una conversión explícita u obligada es la que tendremos que utilizar si necesitamos referirnos a un miembro oculto perteneciente a una clase por encima de la superclase(una clase base indirecta).

REDEFINIR METODOS DE LA SUPERCLASE

Cuando se invoca a un método en respuesta a un mensaje recibido por un objeto,Java busca su definición en la clase del objeto.La definicion del metodo que alli se encuentr apuede pertenecer ala propia clase o puede haber sido heredada de alguna de sus superclases.

Sin embargo,puede haber ocasione sen que deseemos que un objeto de una subclase responda al mismo método heredado de su superclase pero con un comportamiento diferente.Esto implica redefinir en la subclase el método heredado de su superclase.

Redifinir un método heredado significa volverlo a escribir en la subclase con el mismo nombre,la misma lista d eparámetros y el mismo tipo del valor retornado que tenía en la superclase;su cuerpo será adaptado a las necesidades de la subclase.

Cuando en una subclase se refedine un método de una superclase,se oculta le método de la superclase,pero no las sobrecargas que existan del mismo en dicha superclase.Si el método se redefine en la subclas econ distinto tipo o número de parámetros,el método de la superclase no se oculta,sino que s ecomporta como una sobrecarga de ese metodo.

En el caso de que el método heredado por la subclase sea abstracto,es obligatorio redefinirlo,de lo contrario la subclase debería ser declarada también abstracta.

A diferencia de lo que ocurría con los atributos redefinidos,el control de acceso de un método que se redefine no puede modificarse en cualquier sentido.Como regla,no se puede redefinir un método en una subclase y hacer que su control de acceso sea más restrictivo que el original.El orden de los tipos de control de acceso de más a menos restrictivo e sasí:private,predeterminado,protected y public.Aplicando esta regla,un metodo que en la superclase sea protected,en la subclase podrá ser redefinido como protected o public;si es public sólo podrá ser redefinido como public.Si es private no tiene sentido hablar de refefinicion,por que no puede ser accedido nada más que desde su propia clase.

Para acceder a un método de la superclase que ha sido redefinido en la subclase,igual que se expuso para los atributos,tendremos que utilizar la palabra reservada super.Es importante que super puede ser utilizado sólo desde dentro de la clase que proporciona los miembros redefinidos.

CONSTRUCTORES DE LAS SUBCLASES

Sabemos que cuandos e crea un objeto de una clase se invoca a su constructor.También sabemos que los constructores de la superclase no son heredados por sus subclases.En cambio,cuando se crea un objeto de una subclase,se invoca a su constructo,que a su bez invoca al contructor sin parámetros de la superclase,que a su vez invoca al contructor de su superclase, y asi sucesivamente.

Lo expuesto se traduce en que primero se ejecutan los constructores de las superclases de arriba a abajo en la jerarquía de clases y finalmente el de la subclase.Esto sucede así,por que una subclase

contiene todos los atributos de su superclase,y todos tienen que ser iniciados,razón por la que el constructor de la subclase tiene que llamar implícita o explícitamente al de la superclase.

Sin embargo,cuando se hayan definido constructores con parámetros tanto en las subclases con en las superclases,tal vez se desee contruir un objeto de la subclase iniciándolo ccon unos valores determinados.En este caso,la definicion ya conocida para los constructores de una clase cualquiera se extienda ahora par apermitir al constructor de la subclase invocar explícitamente al constructor de la superclase.Esto se hace utilizando la palabra reservada super:

Cuando desde un constructor de una subclase s einvoque al contructor de su superclase esta línea tiene que se rla primera.Se puede observar que la sintaxis y los requerimientos son análogos a los utilizados con this cuando se llama a otro constructor d ella misma clase.

Si la superclase,no tiene un constructor d eforma explícita o tiene uno que no requiere parámetros,no se necesita invocarlo explícitamente,ya que Java lo invocará automáticamente mediane super() sin argumentos.Por el contrario,sí e snecesario invocarlo cuando se trate d eun constructor con parametros,par apoder así pasar los argumentos necesrios en la llamada.

Evidentemente,si sólo se desea inciar algunos de los atributos de un objeto,hay que escribir los constructores adecuados tanto en la subclase como en la superclase.

Según lo expuesto,cuando se crea un objeto de una subclase,primero se construye la porción del objeto correspondiente a su superclase y a continuación la porcion del objeto correspondiente a su subclase.Esto es una forma lógica de operar,ya que permite al constructor de la subclase hacer referencia a los atributos de su superclase que ya han sido iniciados.

Según lo expuesto,los objetos d euna subclase son construidos de abajo hacia arriba:esto es,la pila de llamadas relativas a los constructores de las clases involucradas crece hasta llegar a la raíz en la jerarquía de clases;en este instante,comienza a ejecutarse el constructor de este superclase;pprimero se construyen sus atributos ejecutando,cuando sea necesario,los constructores de los mismos,y despues,se pasa a ejecutar el cuerpo del constructor de dicha superclase: y a continuacion se ejecuta el cuerpo del constructor de la subclase.Este orden se aplica recursivamente para cada constructor de cad auna de las clases

DESTRUCTORES DE LAS SUBCLASES

Acabamos de ver que en Java,el constructor de una subclase invoca automáticamente al constructor sin parámetros de su superclase.En cambio,con los destructores(metodos finalize) no ocurre los mismo.

Si definimos en una subclase un método finalize para liberar los recursos asignados por dicha clase,debemos redefinir el método finalize en la superclase para liberar también los recursos asignados por ella.Pero si el método finalize de la subclase no invoca explícitamente al método finalize de la superclase,este último nunca será ejecutado y los recursos asignados por la superclase no serán liberados¿Cuándo debemos invocar al método finalize de la superclase?El mejor lugar para hacerlo es en la última línea del método finalize de la subclase,porque como la parte del objeto de la subclase se ha construido una vez que estaba construida la parte del objeto de la superclase,en más de una ocasión los vínculos existentes entre una y otra parte exigirán deshacerlo construido,justo en el orden inverso.

Para ver cómo son invocados los destructores,el método main de la clase Test crea un objeto de la claseB referenciado por objClaseB y cuando finaliza el trabajo con el mismo asigna a la variable objClaseB el valor null con la intención de enviar el objeto referenciado por ella a la basura.Finalmente fuerza al recolector de basura a que recoja la basura,lo que provocará la ejecución de los destructores_primero el de la ClaseB y después el de la ClaseA.

Puesto que Java proporciona para cada clase que definamos un método finalize heredado de la clase Object,una programacion segura aconseja redefinir este método en cada una de las subclases que escribamos,aunque no haga nada;simplemente con la intencion de invocar al método finalize de la superclase,por si alguna versión futura de la misma incuye un método finalize

JERARQUIA DE CLASES

Una subclase puede asimismo ser una superclase de otra clase, y asi sucesivamente.En la siguiente figura se puede ver esto con claridad:

El conjunto de clases así definido da lugar a una jerarquía de clases.Cuando cada subclase lo es de una sola superclase,como ocurre en Java,la estructura jerárquica recibe el nombre de árbol de clases.

La raíz del árbol es la clase que representa el tipo más general,y las clases terminales en el árbol representan los tipos más especializados.

El cuerpo del constructor consta de la llamada al constructor de su superclase y de las llamadas a los métodos de la propia clase que permiten iniciar de forma segura los atributos de la misma.También se ha implementado un constructor sin parámetros.

Mientras que una subclase tiene una única superclase directa,puede tener varias superclases indirectas:todas las que haya en el camino para llegar desde su superclase hasta la clase raíz;esto es importante porque lo que una subclase hereda de su superclase,será heredado a su vez por una subclase de ella,y así sucesivamente.

Una subclase que redefina un método heredado sólo tiene acceso a su propia versión y a la publicada por su superclase directa. Entonces la subclase ademas se du propia version solo puede acceder a la version de su superclase directa por medio de la palabra super(en este caso ambas versiones son la misma),pero no puede acceder a la version de su superclase indirecta(super.super no es una expresion admitida por el compilador Java).

Finalmente,indicar que aunque en ninguna clase de nuestra jerarquia han intervenido miembros static,su comportamiento en cuanto a la herencia se refiere ,es el mismo que el de los otros miembros,pero teniendo presente que son miembros de la clase;y si es necesario,cuando se trate de métodod,tambien pueden ser redefinidos,aunque , en este caso,el nombre de la clase indicará la versión del metodo que se invocara.Una advertencia ,si definiera en Ccuenta, el atributo tipoDeInteres static,logicamente se mantendría una única copia que utilizarían tanto los objetos de Ccuenta como los de sus subclases.

REFERENCIAS A OBJETOS DE UNA SUBCLASES

Las referencias a objetos de una subclase pueden ser declaradas y manipuladas de la misma forma que las referencias a objetos de una clase cualquiera.

CONVERSIONES IMPLICITAS

Que pasaria si a la variable cliente01 le asignamos la referencia a un objeto de la subclase.
Si ejecutamos este ejemplo seguira bien.

Pero si cuando accedemos aun objeto por medio de una variable no del tipo del objeto,sino del tipo de alguna de sus superclases(directa o indirectas)según muestra el ejemplo anterior,es el tipo de la variable el que determina que mensajes puede recibir el objeto referenciado;dicho de otra forma,es este tipo el que determina qué métodos pueden ser invocados por el objeto referenciado.¿Cuales son esos metodos?Pues los correspondientes al tipo de la variable que utilizamos para hacer referencia al objeto,nolo s de la clase del objeto.

Resumiento:cuando accedemos a un objeto de una subclase por medio de una referencia a su superclase,ese objeto solo puede se rmanipulado por los metodos de su superclase.

Si ocasiona un error es por que el tipo de la variable que es “normal” determian que el objeto referenciado por ella solo puede recibir mensajes de dicha variable;dicho de otra forma,solo puede ser manipulado por metodos de la clase “normal”(propias y heredadas).
En cambio,cuando se invoca a un metodo que esta definido en la superclase y redefinido en sus subclases,la version que se ejecuta depende de la clase del objeto referenciado,no del tipo de la variablo que lo referencia.

CONVERSIONES EXPLICITAS

La conversion contraria,de una referencia a un objeto de la superclase a una referencia a su subclase,no se puede hacer,aunque se fuerce a ello utilizando una construccion cast,excepto cuando el objeto al que se tiene acceso atrabes de la referencia a la superclase es un objeto de la subclase.

POLIMORFISMO

La utilizacion de subclases y de metodos definidos en una clase y redefinidos en sus clases derivadas es frecuentemente denominada programacion oriaentada a objeots.En cmabio,la facultad de llamar a una variedad de metodos utilizando exavtamente el mismo medio de accesomproporcionada por los metodos redefinidos en las subclasesmes aveces denomianda polomorfismo.

La palabra “polimorfismo” significa “la faculta de asumir muchas formas”,referiendose a la facultad de llamar a muchos metodos diferentes utilizando a una unica sentencia.

Recuerde que cuando se invoca a un metodo que esta definido en la superclase y redefninido en sus subclases,la version que se ejecuta depende de la clase del objeto referenciado,no del tipo de la variable que lo referencia.

Asimismo,sabemos que una referencia a una subclase puede ser convertida implicitamente por Java en una referencia a su superclase directa o indirecta.Esto significa que es posible referirse a un onjeto de una subclase utilizando una variable del tipo de su superclase.

Segun lo expuesto,y en un intento de buscar una codificacion más genérica,pensemos en una matriz de referencias en la que cada elemento señale a un onjeto de alguna de la ssubclases de la jerarquia construida anteriormente.¿De que tipo deben ser los elementos de la matriz?Segun el parrafo anterior deben de ser de la clase Ccuenta ;de esta forma ellos podran almacenar indistintamente referencias a objetos de cualquiera d ellas subclases.

En este ejemplo define una matriz cliente de tipo Ccuenta con 100 elementos que Java inicia con el valor null.Despues crea un objeto de una de las subsclases y almacena su referencia en el primer elemento de la matrz;aqui Java realizara una conversion implicita del tipo de la referencia devuelta por new al tipo Ccuenta.Este proceso se repetira para cada objeto nuevo que deseemos crear.

Sabiendo que cualquier referencia a un objeto de una subclase puede convertirs eimplicitamente en una referencia a un objeto de su superclase,la estructura idonea es una matriz de rederencia a un objeto de su superclase,la estructura idonea e suna matriz de referencias ala superclase Ccuenta.Esta matriz sera dinamica ;esto es, aumentara en un elemento cuando se añada un objeto de alguna de las subclases y disminuira en uno cuandose elimine;inicialmente tendra 0 elementos.

Analizando la clase Cbanco,observamos que su constructor incia la matriz clientes con 0 elementos;que añadir un objeto(un cliente) a la matriz se hace ne dos pasos:uno,incremenentar la matriz en un elemento,y dos,asignar la referencia al objeto al nuevo elemento de la matriz;que eliminar un objeto de la matriz tambien se hace en dos pasos:uno ,poner a null el elemento de la matriz que referencia al objeto que se desea eliminar(el objeto se envia a la basura para que sea recogido por el recolector de basura),y dos, quitar ese elemento de la matriz decrementando su tamaño en 1.

METODOS EN LINEA

Cuando el compilador Java conoce con exactitud que metodo tiene que llamar para responder al mensaje que se ha programado que un objeto reciba en un instante determinado,puede tomar la iniciativa de reemplazar la llamada al metodo por el cuerpo del mismo.Se dice entonces que el metodo esta en linea,El que se produzca esta circunstancia,por que el metodo es corto,redundara en tiempos de ejecucion mas bajos ya que se evita que el interprete Java tenga que llamar al metodo.En principio,en Java ,todos los metodos de una clase pueden ser metodos en linea.

¿Cuando un metodo no podra pasar a ser un metodo en linea?Cuando el compilador no sepa con exactitud a que version del metodo tiene que invocar.

La consulta dinamica acerca del metodo que hay que invocar es rapida,pero no tan rapida como invocar a un metodo directamente.Afortunadamente no hay muchos casos en los que Java necesite usar consulta dinamica.Los metodos finales,static y private pueden ser invocados directamente;y si son cortos son candidatos a ser metodos en linea.Si un metodo es final ,el compilador sabe que ese metodo no puede ser redefinido,por lo tanto existe una sola version;si es estatico es invocado anteponiendo el nombre de su clase;y si es privado no puede ser invocado por un metodo que no sea de su clase.Por lo tanto en ninguno de los tres casos habra que tomar una decision acerca de a que metodo hay que llamar.