Programación orientada a objetos

Resumen de Temas

Unidad 5: Herencia

5.1 Introducción a la Herencia

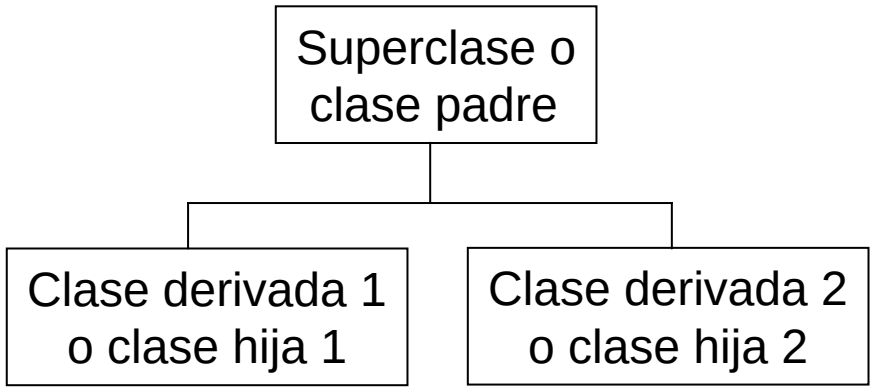
- La herencia es el mecanismo fundamental de relación entre clases en la orientación a objetos. Relaciona las clases de manera jerárquica; una clase **padre** o **superclase** sobre otras clases hijas o subclases derivadas.
- Los descendientes de una clase heredan todas las variables y métodos que sus ascendientes hayan especificado como heredables, además de crear los suyos propios.
- La característica de herencia, nos permite definir nuevas clases derivadas de otra ya existente, que la especializan de alguna manera. Así logramos definir una jerarquía de clases, que se puede mostrar mediante un árbol de herencia.
- En todo lenguaje orientado a objetos existe una jerarquía, mediante la que las clases se relacionan en términos de herencia. En Java, el punto más alto de la jerarquía es la clase **Object** de la cual derivan todas las demás clases.

5.2 Herencia simple y múltiple

En la orientación a objetos, se consideran dos tipos de herencia:

- Herencia simple: una clase sólo puede derivar de una única superclase.
- Herencia múltiple: una clase puede descender de varias superclases.

En Java sólo se dispone de herencia simple, para una mayor sencillez del lenguaje, sin embargo se compensa de cierta manera la inexistencia de herencia múltiple con un concepto denominado **inteface.**



5.3 Clase base y clase derivada

- Cuando una clase deriva de otra, hereda todas sus variables y métodos. Estas funciones y variables miembro pueden ser redefinidas (overridden) en la clase derivada, que puede también definir o añadir nuevas variables y métodos. En cierta forma es como si la sub-clase (la clase derivada) “contuviera” un objeto de la super-clase (clase base); en realidad lo “amplía” con nuevas variables y métodos.
- Java permite múltiples niveles de herencia, pero no permite que una clase derive de varias (no es posible la herencia múltiple). Se pueden crear tantas clases derivadas de una misma clase como se quiera.
- Todas las clases de Java creadas por el programador tienen una super-clase. Cuando no se indica explícitamente una super-clase con la palabra extends, la clase deriva de java.lang.Object, que es la clase raíz de toda la jerarquía de clases de Java. Como consecuencia, todas las clases tienen algunos métodos que han heredado de Object.
- La composición (el que una clase contenga un objeto de otra clase como variable miembro) se diferencia de la herencia en que incorpora los datos del objeto miembro, pero no sus métodos o interface (si dicha variable miembro se hace private).

5.3.1 Declaración

- Para indicar que una clase deriva de otra, heredando sus propiedades (métodos y atributos), se usa el término **extends**, como en el siguiente ejemplo:

```
public class SubClase extends SuperClase {
    // Contenido de la clase
}
```

- Los atributos privados de la clase base (Superclase) existen en la Subclase pero no son accesibles directamente por ningún método de la subclase.

- Lo atributos static de la clase base se heredan significando que la subclase y clase base comparten una copia de los atributos static.

5.4 Parte protegida

Todos los campos y métodos de una clase son siempre accesibles para el código de la misma clase. Para controlar el acceso desde otras clases, y para controlar la herencia por las subclases, los miembros (atributos y métodos) de las clases tienen tres modificadores posibles de control de acceso:

- ***public***: Los miembros declarados *public* son accesibles en cualquier lugar en que sea accesible la clase, y son heredados por las subclases.
- ***private***: Los miembros declarados *private* son accesibles sólo en la propia clase y aunque son heredados a las subclases no son accesibles directamente por éstas. Las sub-clases heredan los miembros private de su super-clase, pero sólo pueden acceder a ellos a través de métodos public, protected o package de la super-clase.
- **protected**: Los miembros declarados *protected* son accesibles sólo para la clase base y sus subclases.

5.4.1 Ejemplo de accesibilidad:

```
class Padre { // Hereda de Object
// Atributos
private int numeroFavorito, nacidoHace, dineroDisponible;
// Métodos
public int getApuesta() { return numeroFavorito;}
protected int getEdad() { return nacidoHace;}
private int getSaldo() {return dineroDisponible;}
}

class Hija extends Padre {
// Definición
}

class Visita {
// Definición
}
```

La clase Hija, hereda los tres atributos y los tres métodos de la clase Padre, y podrá invocarlos. Sin embargo, un objeto de la clase Hija, no podrá invocar al método getSaldo() de un objeto de la clase Padre.

La clase Visita, solo podrá acceder al método getApuesta(), para averiguar el número favorito de un Padre, pero de ninguna manera podrá conocer ni su saldo, ni su edad.

5.5 Redefinición de los miembros de las clases derivadas[1]

- Una clase puede redefinir (volver a definir) cualquiera de los métodos heredados de su super-clase que no sean final. El nuevo método sustituye al heredado para todos los efectos en la clase que lo ha redefinido.
- Se puede redefinir cualquier método de la clase base declarando el método en su subclase con la misma firma o signatura (mismo tipo de retorno, nombre del método y lista de parámetros o argumentos)
- Las métodos de la super-clase que han sido redefinidos pueden ser todavía accedidos por medio de la palabra super desde los métodos de la clase derivada, aunque con este sistema sólo se puede subir un nivel en la jerarquía de clases.
- Los métodos redefinidos pueden ampliar los derechos de acceso de la super-clase (por ejemplo ser public, en vez de protected o package), pero nunca restringirlos.
- Los métodos de clase o static no pueden ser redefinidos en las clases derivadas.

5.5 Redefinición de los miembros de las clases derivadas$_2$

- Si se redefine un método de la clase base y se quiere acceder a la versión de la clase base o superclase, se puede utilizar la palabra ***super,*** por ejemplo:

```
class Subclase extends Clasebase {
…
public String MetodoBase () {
     return super.MetodoBase()+"\n"; }
       }
```

- Una clase declarada final no puede tener clases derivadas. Esto se puede hacer por motivos de seguridad y también por motivos de eficiencia, porque cuando el compilador sabe que los métodos no van a ser redefinidos puede hacer optimizaciones adicionales.
- Análogamente, un método declarado como final no puede ser redefinido por una clase que derive de su propia clase.

5.6 Constructores y destructores en clases derivadas$_1$

- Un constructor de una clase puede llamar por medio de la palabra this a otro constructor previamente definido en la misma clase. En este contexto, la palabra this sólo puede aparecer en la primera sentencia de un constructor.
- De forma análoga el constructor de una clase derivada puede llamar al constructor de su super-clase por medio de la palabra super(), seguida entre paréntesis de los argumentos apropiados para uno de los constructores de la super-clase. De esta forma, un constructor sólo tiene que inicializar directamente las variables no heredadas.
- La llamada al constructor de la super-clase debe ser la primera sentencia del constructor, excepto si se llama a otro constructor de la misma clase con this(). Si el programador no la incluye, Java incluye automáticamente una llamada al constructor por defecto de la super-clase, super(). Esta llamada en cadena a los constructores de las super-clases llega hasta el origen de la jerarquía de clases, esto es al constructor de Object.

5.6 Constructores y destructores en clases derivadas$_2$

- Si el constructor de una clase es private, sólo un método static de la propia clase puede crear objetos.
- Si el programador no prepara un constructor por defecto, el compilador crea uno, inicializando las variables de los tipos primitivos a sus valores por defecto, y los Strings y demás referencias a objetos a null. Antes, incluirá una llamada al constructor de la super-clase.
- En el proceso de finalización o de liberación de recursos (diferentes de la memoria reservada con new, de la que se encarga el garbage collector), es importante llamar a los finalizadores de las distintas clases, normalmente en orden inverso al de llamada de los constructores. Esto hace que el finalizador de la sub-clase deba realizar todas sus tareas primero y luego llamar al finalizador de la super-clase en la forma super.finalize().
- Los métodos finalize() deben ser al menos protected, ya que el método finalize() de Object lo es, y no está permitido reducir los permisos de acceso en la herencia.

5.6.1 Constructores de Subclase

- Los constructores de una subclase deben siempre primero llamar a uno de los constructores de la clase base
- Esto se hace usando la palabra reservada **super**

```
class SubClase extends ClaseBase{
    SubClase() {
        super(..argumentos..);
    }
}
```

- Si no se llama al constructor de la clase base, Java automáticamente llamará al constructor por default de la clase base al inicio de su constructor
- Si la clase base no tiene un constructor por default, se generará un error

5.7 Clase Object

- La clase Object es la raíz de toda la jerarquía de clases de Java. Todas las clases y jerarquías de clases, incluyendo las creadas por el programador, dependerán (directa o indirectamente) de la clase Object.
- Si no se especifica una clase base, la clase automaticamente extiende (hereda) de la clase Object.
- Cada clase en Java contiene los métodos de funcionalidad básica definidos en la clase Object.
- La clase Object se encuentra incluida en el paquete java.lang .

5.7.1 Métodos que se pueden redefinir

- clone() Crea un objeto a partir de otro objeto de la misma clase. El método original heredado de Object lanza una CloneNotSupportedException. Si se desea poder clonar una clase hay que implementar la interface Cloneable y redefinir el método clone(). Este método debe hacer una copia miembro a miembro del objeto original. No debería llamar al operador new ni a los constructores.
- equals() Indica si dos objetos son o no iguales. Devuelve true si son iguales, tanto si son referencias al mismo objeto como si son objetos distintos con iguales valores de las variables miembro.
- toString() Devuelve un String que contiene una representación del objeto como cadena de caracteres, por ejemplo para imprimirlo o exportarlo.
- finalize() Este método es para definir finalizadores.

5.7.2 Métodos que no pueden ser redefinidos

La clase Object tiene métodos final, por lo que no se pueden redefinir:

- getClass() Devuelve un objeto de la clase Class, al cual se le pueden aplicar métodos para determinar el nombre de la clase, su super-clase, las interfaces implementadas, etc. Se puede crear un objeto de la misma clase que otro sin saber de qué clase es.
- notify(), notifyAll() y wait() Son métodos relacionados con las threads (hilos).

5.8 Aplicaciones y ejercicios

- Hacer un clase CuentaUsuario para representar la información de un usuario de correo, los atributos necesarios son el Nombre, Apellido Paterno, Materno y Edad, Departamento e ID. Cuando se cree una cuenta de Usuario se le debe asignar automaticamente un ID que se deriva de la siguiente manera:
  - Si es del departamento de Sistemas(ISC), el ID es dado por las primeras cinco letras del apellido paterno seguido por la primera letra del nombre.
  - Si es del departamento de Informatica(INF), el ID es dado por las primeras cuatro letras del apellido paterno seguido por un subrayado y la primera letra del nombre.
  - Si es de Administracion(ADM), el ID es dado por la cadena AD seguido de un subrayado, las primeras cuatro letras del apellido paterno, un subrayado y las primeras dos letras del nombre.

Continuación de ejercicio

- Una vez creado, se deben poder hacer las siguientes operaciones:
  - **obtenerID()** Regresa el ID asignado al usuario
  - **cambiaNombre(nombre,app, apm)**
  - **cambiaEdad(edad)**
  - **obtenNombreCompleto()** – Regresa el nombre completo del usuario
  - **obtenEdad()** - Regresa la edad del usuario
  - **cambiaDep(departamento)** – cambia el departamento asignado al usuario
  - **obtenDep()** – Regresa el departamento al que esta adscrito el usuario
- NOTA: Nuevamente, los atributos deben ser **private**.

Continuación de ejercicio

Se requiere almacenar la informacion de un usuario de correo, los usuarios pueden ser de dos tipos: docentes y alumnos. La información requerida para cada uno es la siguiente:

Docente	Alumno
Nombre	Nombre
Apellido Paterno	Apellido Paterno
Apellido Materno	Apellido Materno
Edad	Edad
Departamento	Departamento
ID	ID
Grado Academico	Promedio
Antigüedad	Semestre

Generar las clases necesarias para representar esta informacion sabiendo que los campos deben ser privados, y para ambos tipos de usuarios se debe poder cambiar el nombre, edad, departamento así como obtener su ID, nombre completo, edad y departamento. Para los docentes también se debe poder cambiar y obtener su grado academico y antigüedad, para los alumnos se debe poder cambiar y obtener su promedio o semestre. El ID se debe generar automaticamente (con las reglas utilizadas en la clase CuentaUsuario)

Continuación de ejercicio

Suponga que cuando se obtenga el nombre completo si es alumno se preceda con la palabra “Alumno” o si es docente indique cual es su grado academico:

d.obtenNombreCompleto()

M.C. Roberto Solis Robles

a.obtenNombreCompleto()

Alumno Andres Ultreras Correa

Haga los cambios en las clases correspondientes para obtener este resultado