Clases y Objetos

Como en otras áreas de la vida humana, no tiene sentido reinventar o reconstruir todo a partir de los elementos básicos. Hay puertas y ventanas estándares para construir casas. Los neumáticos la misma cosa y son reusables en cualquier auto.
En software se quiere hacer lo mismo. Construir soluciones a problemas bien definidos y luego reusar estas componentes o clases.
Al usuario de algo empaquetado no le debería interesar ni requerir la información sobre cómo algo fue hecho, sólo le interesan los aspectos de uso externo. Es el caso de componentes electrónica, donde se usa la hoja de datos para el diseño sin saber los detalles internos (cuántos transistores usó, en qué configuración,...)
Las clases intentan ser el empaquetado básico descrito.

Herencia: Se refiere a la facultad de definir una clase a partir de otra. Esto tiene sentido cuando ambas clases cumplen la **relación "es-un"**. Se usó el término herencia como en genética donde se pasa información de padres a hijos. En el caso de los lenguajes el concepto se asemeja porque se desea indicar que la nueva clase comparte todo lo que tiene la clase de la cual **hereda** (o **deriva**); sin embargo, no se semeja en el hecho que la clase derivada incorpora elementos propios que la hace más específica. Por ejemplo un perro es un mamífero. El perro por **ser un** mamífero, cumple con la descripción de mamífero; sin embargo, es más específico por cuanto el perro ladra y no todos los mamíferos ladran. Luego si tenemos una clase mamífero, podremos construir la clase perro a partir de la otra, aprovechar todo lo ya definido en mamífero y agregar sólo los elementos específicos que caracterizan la nueva clase perro. Igualmente, si ya disponemos la clase ComponenteElectrónicaBásica, como una resistencia **es una** componente electrónica básica, puedo definir la clase Resistencia como ***sub-clase*** o ***clase derivada*** de ComponenteElectrónicaBásica.

Hay tres tipos de **relaciones entre clases**:

1. Dependencia corresponde a relación "usa-un": Esta relación generalmente se refleja en el uso de instancias de otras clases para cumplir los servicios o responsabilidades de una clase. Eje.: Un amplificador **usa** un parlante para emitir el sonido. Un Computador **usa** la red para enviar sus datos. Es decir si en nuestro diseño el Computador es el responsable de enviar los datos, entonces éste deberá acceder a una instancia de Red para enviar los datos.
2. Agregación corresponde a la relación "tiene-un": Esta relación describe el tipo de dependencia que generalmente se refleja por la presencia de atributos en una clase que son instancias de la otra clase. Eje.: Un circuito RC **tiene un** condensador y una resistencia. Un auto **tiene** ruedas, motor, y puertas.
3. Herencia corresponde a la relación "**es-un**": Eje.: un condensador **es una** Componente Básica. Un Auto **es un** medio de transporte. Un círculo **es una** forma geométrica.

Estas dependencias reflejan el tipo de relación o asociación de las clases que dan forma a

una solución.

Estudio de ejemplos: CalendarTest.java Employee.java y EmployeeTest.java

Observar:

1. Estructura para definir una clase: constructor, métodos, datos
2. Definiciones:

   Un **método** se dice que es un **mutador** si su invocación permite cambiar el estado del objeto (cambia alguno de sus atributos -miembros dato)

   Un **método** se dice "accesor" si su invocación sólo permite acceder a un objeto pero no lo modifica.

3. **Calificadores de acceso o visibilidad**: Corresponde a las palabras reservadas usadas para definir la visibilidad de clases, sus atributos y métodos.
   1. **public**: El acceso es abierto para cualquier código que tiene acceso a la clase. Una clase puede ser publica o existir sólo dentro del paquete.
   2. **protected**: miembro visible a cualquier clase derivada o del mismo paquete.
   3. **package**: visible al paquete. Es el valor por defecto si lo omitimos el calificador de acceso.
   4. **private**: miembros accesibles sólo a miembros de la misma clase. Es decir, sólo en las implementaciones de la clase.

4. **Datos estáticos**: Corresponden a datos de los cuales hay sólo uno para toda la clase (no uno por objeto). Puede ser entendido como un atributo de la clase y no de un objeto. Almacena el estado de una clase, el cual es compartido (común) con todos los objetos de esa clase.
   Por ejemplo, si creamos objetos de la clase Chileno y dentro de ésta tenemos el método nacionalidad, sería un poco absurdo asociar a cada objeto un estado cuando es mejor asociarlo a la clase, dado que es común para todas instancias de la clase, basta con dar la opción a que todos accedan a él.
   Ciertas constantes se manejan de esta forma. Por ejemplo, en la clase matemática nos conviene definir la constante PI. Si deseamos conocer el número de instancias de una clase, podemos incrementar un contador cada vez que se crea una nueva instancia y lo decrementamos cuando ésta concluye. Resulta conveniente que este contador sea un atributo de la clase al cual todos puedan acceder.

5. **Métodos estáticos**: también usan el calificador static y sólo pueden manipular datos estáticos. Se invocan anteponiendo el nombre de la clase, pueden ser invocados desde una instancia pero se recomienda su acceso desde el nombre de la clase para ser explícitos en que se trata de un miembro estático. Como en el caso de los atributos estáticos, los métodos estáticos surgen en forma similar, por ejemplo dónde ponemos la funciones trigonométricas sin, cos, etc. Éstas están en la clase matemática de Java (Math). Ver StaticTest.java

6. **Invocación de métodos**: Todas las invocaciones en Java son por valor (no por referencia). Los tipos primitivos copian su valor en el parámetro, luego el parámetro actual (aquel usado en el llamado) nunca es modificado. Los nombre de objetos son referencias a éstos, luego cuando tenemos un objeto como parámetro, el objeto referenciado sí puede ser modificado. Ver ParamTest.java.
7. **Sobrecarga**: Se refiere a que un nombre de método puede poseer varias implementaciones siempre que se diferencien en sus argumentos. En Java un método queda entonces definido en forma única al considerar su nombre más el conjunto de parámetros. Éste es un tipo de polimorfismo. Ejemplo, en una clase podemos tener los métodos
   int numeroCaracteres(String s) {.....}
   int numeroCaracteres(int i) {.....}

8. **Inicialización** de miembros dato de la clase: Se puede hacer en el constructor o en su definición. Primero se ejecuta la definición y luego lo del constructor.
9. Llamado entre constructores. Para ello se puede usar this(/* parámetros del constructor a invocar*/); Ver ConstructorTest.java

10. La palabra reservada **this**: también podemos usar la palabra reservada this como referencia al objeto responsable de la ejecución de las instrucciones donde nos encontramos. Por ejemplo cuando un método tiene un argumento cuyo nombre coincide con el nombre de un atributo, éste no es visible directamente a través del nombre -que se referiría al argumento; pero podemos acceder al atributo usando this. Ejemplo:

```
Class  Visibilidad {
  int a;   /* atributo de la clase*/
  int metodo( int a) {  /* argumento que apantalla al atributo */
    this.a+=a;    /* podemos acceder a atributo */
    a++;          /* éste es el argumento*/
    return a;   /* se retorna el argumento */
  }
}
```

11. **Packages**: Son colecciones de clases. Son buenos para separar los módulos y proyectos. Su principal razón es garantizar unicidad de nombres. Para poner una clase en un paquete se debe poner el nombre del paquete antes de la definición de la clase:

```
package cl.utfsm.elo.elo330;
public class Employee
{
 ...
}
```

    ver: PackageTest.java
12. **Documentación**: Java incluye un esquema estándar para incorporar la documentación como parte del código.
    1. Comentarios Generales:

1. @author nombre
2. @version texto
3. @since texto
4. @see link
   En este caso link puede ser: package.class#característica
   como en @see cl.utfsm.elo.Employee#raiseSalary(double)

2. Para una clase debe ir inmediatamente antes de la clase y ser encerrada entre /** y */ Notar los dos ** iniciales.
3. Para los métodos: usar los rótulos
   1. @param variable descripción
   2. @return descripción
   3. @throws descripción

1. Para los datos públicos: /** ...*/
2. Se puede usar todo tipo de rótulos html incrustados.
3. ¿Cómo extraer la documentación a un formato web?
   $ javadoc -d *docDirectory nameOfPackage*
   Para extraer la documentación de un archivo específico en el paquete por defecto: javadoc -d *docDirectory *.java*
4. Más sobre javadoc sitio sun

13. La ruta para la búsqueda de todas las clases: **CLASSPATH**
    El compilador y el interprete java buscan los archivos en el directorio actual. Si el proyecto está compuesto por varias clases en diferentes directorios, javac y java buscan las clases en los directorios indicados en la variable de ambiente CLASSPATH. En Linux ésta se configura con
    export CLASSPATH=/home/user/classdir1:/home/user/classdir2:.
    Estudiar la forma de configurar esta variable de ambiente en Windows.