Programación 2

Lenguaje Java - Conceptos claves (*3era parte*)

Rodrigo Olivares
Mg. en Ingeniería Informática
rodrigo.olivares@uv.cl

25 de abril de 2017

Contenido

Conceptos

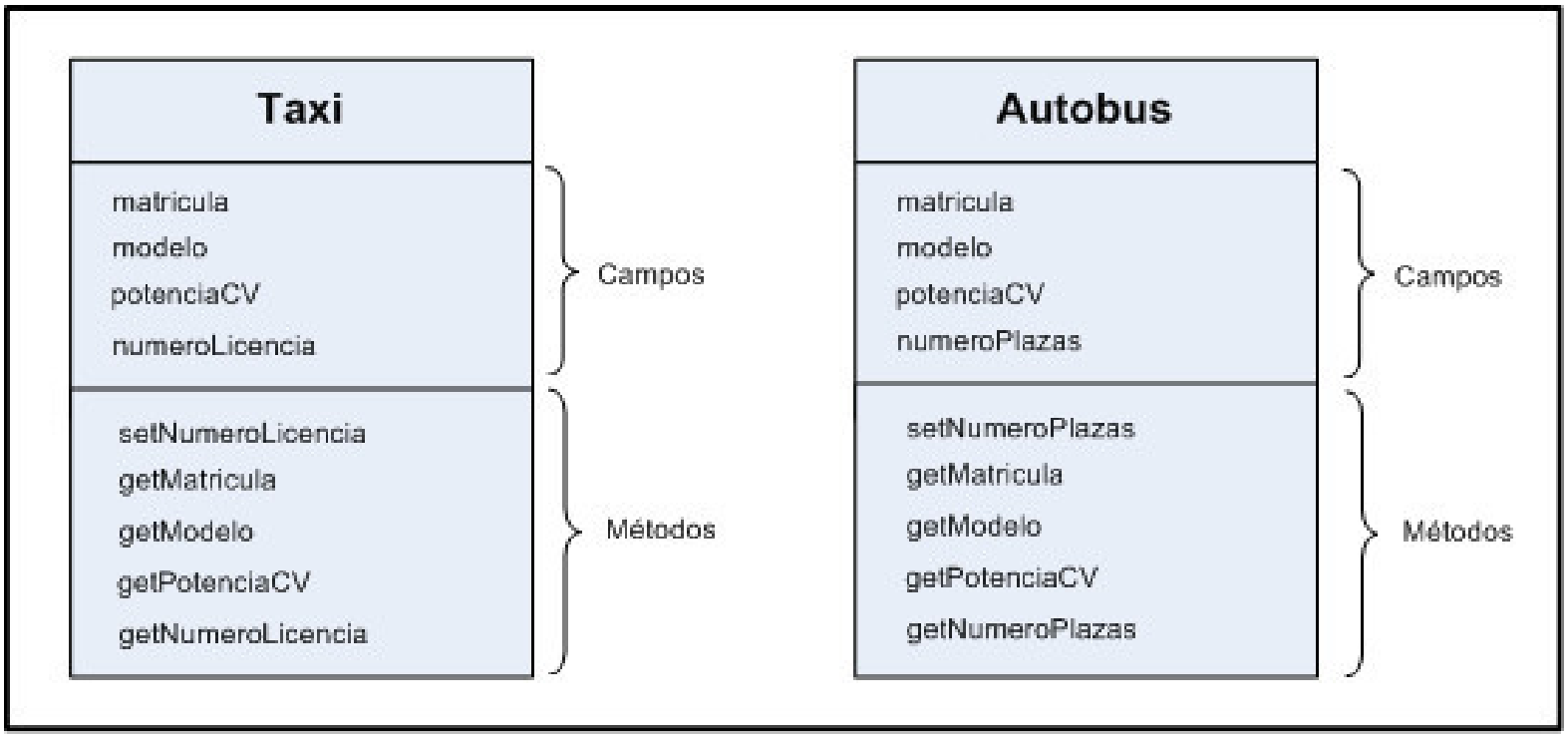
Herencia

Introducción

- Se puede construir una clase a partir de otra, mediante el mecanismo de la **herencia**.
- Para indicar que una clase deriva de otra, se utiliza la palabra reservada **extends**. Ejemplo:
  - **public** ClaseHijo **extends** ClasePadre { ...
- Cabe señalar que sólo es posible extender una única clase padre.

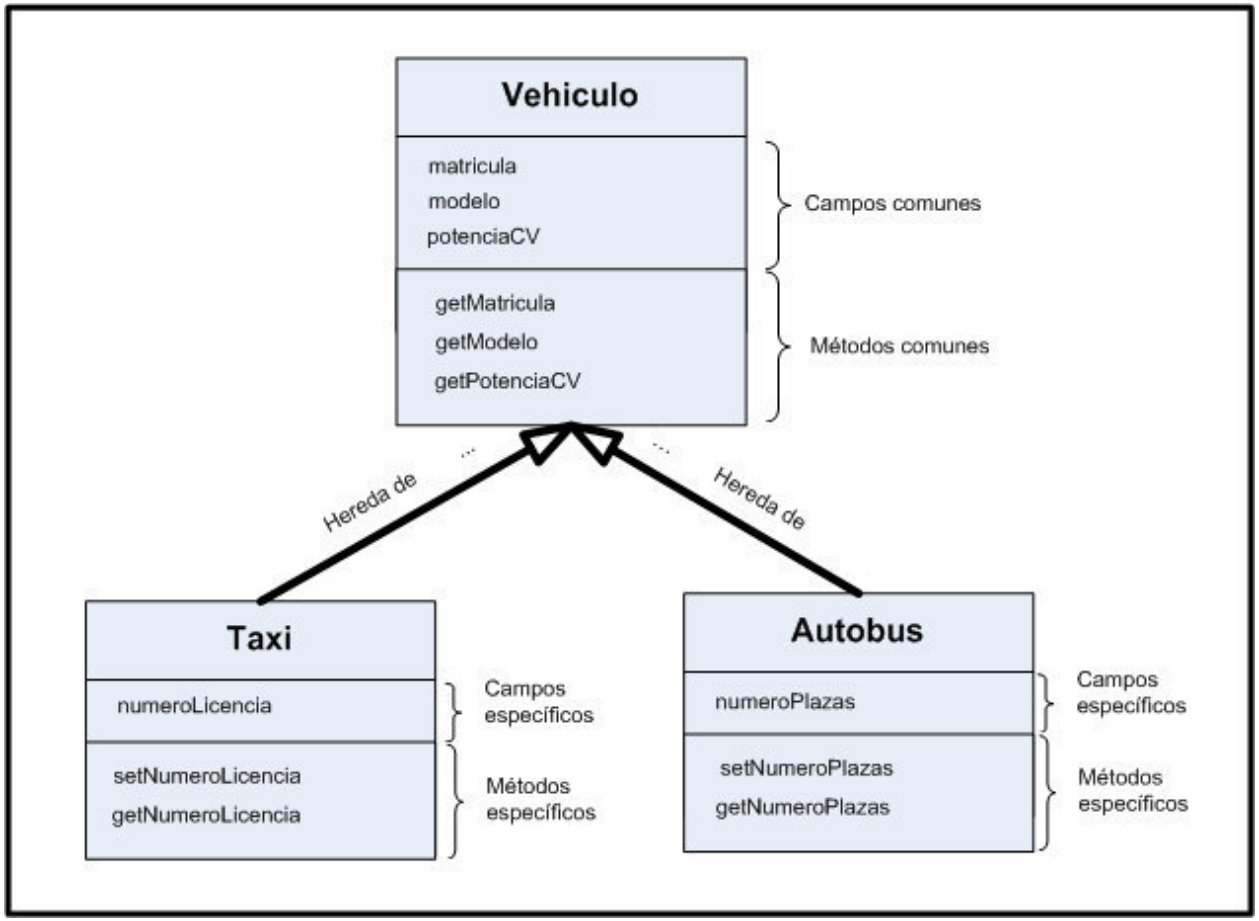
Conceptos

Herencia



Conceptos

Herencia



Conceptos

Herencia

Introducción

- Cuando una clase deriva de otra, hereda todos los atributos y métodos (siempre y cuando sean definidos como **public** o **protected**).
- Estos métodos y atributos, pueden ser redefinidos (**overridden**), en la clase derivada (sub-clase).
- Además, la clase derivada o sub-clase, puede agregar nuevos métodos.
- En cierto modo puede confundir la idea de que la sub-clase **contenga** un objeto de la super-clase, pero la verdad es que la sub-case **amplía** la super-clase.

Conceptos

Herencia

Introducción

- 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 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 métodos que han heredado de **Object**.

Conceptos

Herencia

Introducción

- 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.
- 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.

Conceptos

Clases abstractas

Introducción

- Los lenguajes de programación permiten expresar la solución de un problema de forma comprensible simultáneamente por la máquina y el humano.
- Constituyen un puente entre la abstracción de la mente y una serie de instrucciones ejecutables.
- En consecuencia, la capacidad de abstracción es una característica deseable de los lenguajes de programación, pues cuanto mayor sea, mayor será su aproximación al lado humano.

Conceptos

Clases abstractas

Una clase abstracta

- No permite instanciar o crear objetos de si misma.
- Su utilidad es permitir que otras clases deriven de ella, proporcionando un marco o modelo a seguir y métodos de utilidad general.
- Las clases abstractas se declaran anteponiendo la palabra reservada **abstract**. Ejemplo:
  - **public abstract class** className { ...

Conceptos

Clases abstractas

Una clase abstracta

- Una clase abstracta puede tener métodos declarados como **abstract**, en cuyo caso no se da la definición del método, más si si declaración.
- Si una clase tiene al menos un método de tipo **abstract**, la clase debe ser declarada como **abstract**.
- En toda sub-clase este método deberá ser:
  - Redefinido (volver a declararlo como **abstract**, al igual que la sub-clase).
  - Implementarlo.
- Una clase abstracta puede tener métodos declarados como no abstracto. En este caso, la sub-clase hereda el método completamente para ser utilizado.

Conceptos

Interfaz

Concepto de interfaz

- Una interface es un conjunto de declaraciones de métodos (sin definición).
- También puede definir constantes, que son implícitamente **public**, **static** y **final**, y deben siempre inicializarse en la declaración.
- Estos métodos definen un tipo de *comportamiento*.
- Todas las clases que implementan una determinada interface están obligadas a proporcionar una definición de los métodos de la interface, y en ese sentido adquieren una conducta o modo de funcionamiento.

Conceptos

Interfaz

Concepto de interfaz

- Una clase puede implementar una o varias interfaces.
- Para indicar que una clase implementa una o más interfaces se ponen los nombres de las interfaces, separados por comas, detrás de la palabra reservada **implements**, que a su vez va siempre a la derecha del nombre de la clase o del nombre de la super-clase en el caso de herencia.
- Por ejemplo,
  - public class className **implements** interfaceName1, interfaceName2, ... { ...

Conceptos

Interfaz

Interfaz v/s clase abstract

- Ambas pueden contener varias declaraciones de métodos (la clase abstract puede además definir otros).
- Una clase no puede heredar de dos clases abstract, pero sí puede heredar de una clase abstract e implementar una interface, o bien implementar dos o más interfaces.
- Las interfaces permiten mucha más flexibilidad para conseguir que dos clases tengan el mismo comportamiento, inpendientemente de su situación en la jerarquía de clases de JAVA.
- Las interfaces permiten "publicar" el comportamiento de una clase desvelando un mínimo de información.

Conceptos

Ejercicio

Ejercicio

Se plantea desarrollar un programa Java que permita la gestión de una empresa de alimentos que trabaja con tres tipos de productos: productos frescos, productos refrigerados y productos congelados. Todos los productos llevan esta información común: fecha de caducidad y número de lote. A su vez, cada tipo de producto lleva alguna información específica. Los productos frescos deben llevar la fecha de envasado y el país de origen. Los productos refrigerados deben llevar el código del organismo de supervisión alimentaria. Los productos congelados deben llevar la temperatura de congelación recomendada. Crear el código de las clases Java implementando una relación de herencia desde la superclase **Producto** hasta las subclases **ProductoFresco**, **ProductoRefrigerado** y **ProductoCongelado**. Cada clase debe disponer de constructor, permitir establecer y recuperar el valor de sus atributos y tener un método que permita mostrar la información del objeto *(toString)*. Crear una clase principal con el método *main* donde se cree un objeto de cada tipo y se ingresen y muestren los datos de cada uno de los objetos creados.

Conceptos

Ejercicio

Ejercicio

Se requiere desarrollar un sistema de apoyo a la docencia de geometría matemáticas. Para ello, se debe definir una categorizaciń de figuras geométricas, por ahora sólo Cuadrados, Triángulos y Circunferencias. Para cada uno de ellos se debe calcular el perímetro y el área. Es relevante para el sistema mostrar la información de la figura geométrica a través de algún método. Desarrolle este sistema en JAVA utilizando el conceto de herencia, implementado una interfaz para las figuras geométricas.

Tipos de datos abstractos

Definición

Definición

Es un modelo compuesto por una colección de operaciones definidas sobre un conjunto de datos para el modelo.

Ejemplos de usos de TDA

Conjuntos: Implementación de conjuntos con sus operaciones básicas (unión, intersección y diferencia), operaciones de inserción, borrado, búsqueda...

Árboles Binarios de Búsqueda: Implementación de árboles de elementos, utilizados para la representación interna de datos complejos. Aunque siempre se los toma como un TDA separado son parte de la familia de los grafos.

Pilas y Colas: Implementación de los algoritmos FIFO y LIFO.

Grafos: Vértices unidos mediante arcos o aristas.

Tipos de datos abstractos

Beans

Definición

Un Bean es un componente software que tiene la particularidad de ser reutilizable y así evitar la tediosa tarea de reescribir los distintos componentes uno a uno.

Criterios que debe cumplir un Bean

- Un Bean debe cumplir los siguientes criterios:
  - Tener todos sus atributos privados (private).
  - Tener métodos set() y get() públicos de los atributos privados.
  - Tener un constructor público por defecto (opcional).

Tipos de datos abstractos

Beans

Ejemplo

Considere los datos de una persona como atributos de una clase. Cree una clase Bean denominada **Persona**, que implemente los métodos get() y set() y el constructor más apropiado.

Tipos de datos abstractos

Listas

Definición Nodos

- Toda lista está compuesta por nodos:
  - Se denomina nodo, elemento o ítem, a la unidad de informacián más elemental o indivisible dentro de una lista.
- Una lista lineal es un conjunto de **n** nodos con $n \geq 0$, cuyas propiedades estructurales esenciales incluyen sólo las posiciones lineales relativas de los nodos.

Tipos de datos abstractos

Listas

Algunas operaciones sobre las listas

- Tener acceso a un nodo.
- Insertar y eliminar un nodo en la lista.
- Combinar dos o más listas en una.
- Dividir una lista en dos o más listas.
- Determinar la cantidad de nodos de la lista.
- Ordenar la lista de acuerdo a un criterio.
- Buscar un elemento bajo una condición.
- Etc.

Tipos de datos abstractos

Listas

Consideraciones sobre las listas

- $n = 0$ denota a la lista vacía, o sea, una lista que no tiene elementos.
- Si $n > 0$, l_0 es el primer nodo.
- Si $1 > k > n$, l_k es precedido por el nodo l_{k-1} y seguido por el nodo l_{k+1}.
- Si $n > 0$, l_{n-1} es el último nodo.

Tipos de datos abstractos

Listas

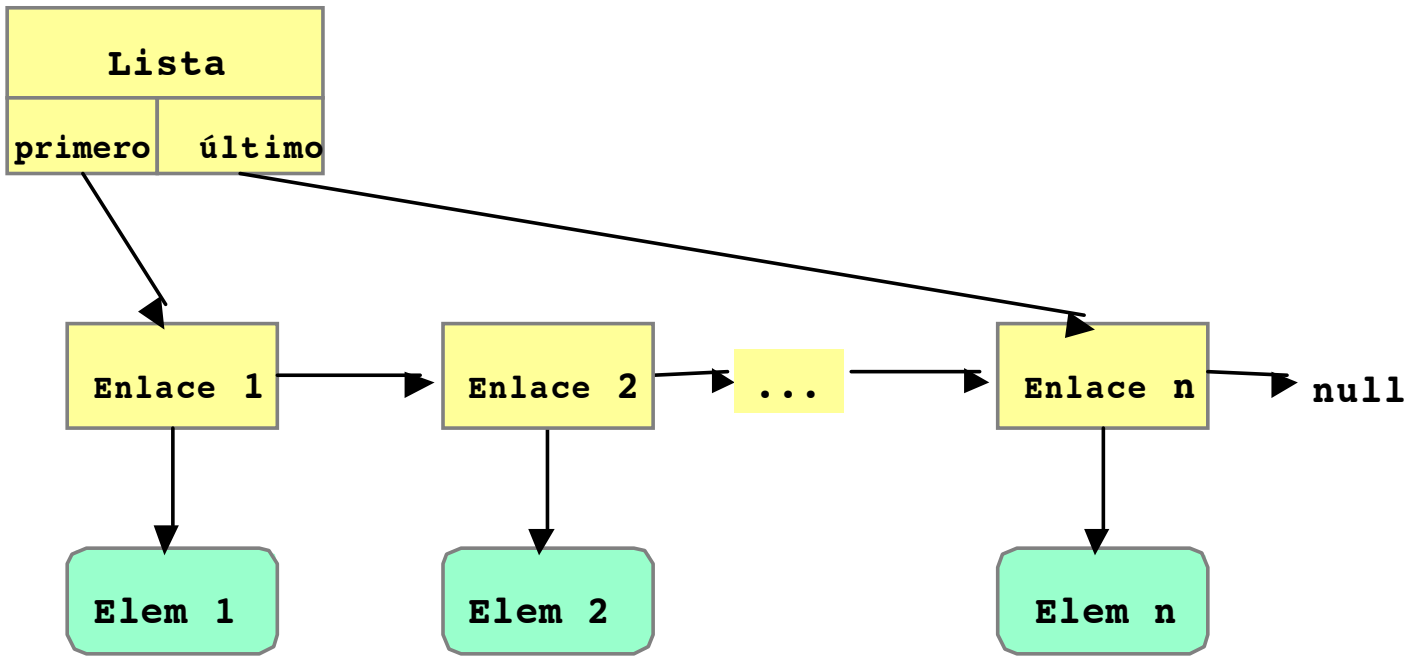
Consideraciones sobre las listas

- En una lista se asigna memoria para el almacenar los elementos, conforme se va necesitando, es decir a medida que se añaden o insertan los elementos, y se conectan los elementos de la lista con punteros.
- La memoria es liberada cuando ya no se necesita más un elemento en la lista.
- Esquemáticamente una lista enlazada se representa por una secuencia de nodos conectados por enlaces.

Tipos de datos abstractos

Listas

Esquema gáfico de las listas



Tipos de datos abstractos

Listas

Interface List

```
public interface List {
    public abstract boolean isEmpty();
    public abstract void add(Object o);
    public abstract boolean contains(Object o);
    public abstract boolean remove(Object o);
    public abstract void clear();
    public abstract int size();
    public abstract List subList(int firstIndex, int lastIndex);
    ...
}
```

Tipos de datos abstractos

Listas

Implementación de la interface List

Vector v/s ArrayList

Sincronización: La clase **Vector** es sincronizada (synchronized) y, por tanto, su contenido está protegido de otros hilos, es decir, es thread-safe. Por al contrario, los **ArrayList** no son sincronizados y por tanto no son thread-safe.

Se debe tener en cuanta esto porque los **Vectores** tienen un costo en tiempo de ejecución que no tienen los **ArrayList**. Si no se necesita thead-safe, se utiliza **ArrayList**.

Tipos de datos abstractos

Listas

Implementación de la interface List

- Implementación con **Vector**
  - **List** lista = **new Vector ()**;
  - **List** <**Class**> lista = **new Vector** <**Class**> **()**;
- Implementación con **ArrayList**
  - **List** lista = **new ArrayList ()**;
  - **List** <**Class**> lista = **new ArrayList** <**Class**> **()**;
- Como nueva norma o estándar se adoptó especificar el tipo de objetos que se almacenará en las listas a diferencia de las listas genéricas que antiguamente se creaban.

Fuente de datos

Archivos - Definición

Definición

- Un archivo es un conjunto de bits almacenado en un dispositivo.
- Un archivo es identificado por un nombre y la descripción del directorio que lo contiene.
- Los archivos proporcionan una manera de organizar los recursos usados para almacenar permanentemente datos en un sistema informático.

Fuente de datos

Lectura de archivos en JAVA

Consideraciones

- El proceso de lectura de un archivo en JAVA, es similar a la lectura desde el dispositivo estándar (teclado).
- Creamos un objeto entrada de la clase **FileReader** en vez de **InputStreamReader**.
- El final del archivo viene dado cuando la función **readLine** retorna **null**.
- El resto del código es similar.

Fuente de datos

Ejemplo

Ejemplo

```
public void readFile() {
    File archivo = null;
    FileReader fileReader = null;
    try {
        archivo = new File ("archivo.txt");
        String linea;
        fileReader = new FileReader (archivo);
        BufferedReader br = new BufferedReader (fileReader);
        while((linea = br.readLine()) != null) {
            System.out.println(linea);
         }
    }
```

Fuente de datos

Ejemplo - Continuación

Ejemplo - Continuación

```
    catch (IOException e) {
        System.out.println(e);
    }
    finally {
        try {
            if (fileReader != null) {
                fileReader.close();
            }
        }
        catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

Fuente de datos

Escritura de archivos en JAVA

Consideraciones

- Creamos un objeto de entrada de la clase **FileWriter**.
- Si queremos añadir al final de un fichero ya existente, simplemente debemos agregar un flag a **true** como segundo parámetro del constructor de **FileWriter**.

Fuente de datos

Ejemplo

Ejemplo

```
public void writeFile() {
    FileWriter archivo = null;
    PrintWriter printWriter = null;
    try {
        archivo = new FileWriter ("archivo.txt");
        printWriter = new PrintWriter (archivo);
        for(int count = 0; count < 10; count++) {
            printWriter.println ('Linea: " + count);
         }
    }
```

Fuente de datos

Ejemplo - Continuación

Ejemplo - Continuación

```
    catch (IOException e) {
        System.out.println(e);
    }
    finally {
        try {
            printWriter.close();
        }
        catch (IOException e) {
            System.out.println(e);
        }
    }
}
```