Tema 6- Estructuras de Datos: Jerarquía de una EDA

Germán Moltó
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Tema 6- Estructuras de Datos: Jerarquía de una EDA

Índice general:

Objetivos

- Conocer el Concepto de Estructura de Datos (EDA).
- Comprender la Separación entre Especificación e Implementación en el Diseño de una EDA.
- Conocer y Utilizar el Mecanismo de Interfaces para la Declaración de Modelos en Java.
- Presentar las Estructuras de Datos de Uso más Frecuente en Programación.

Bibliografía

- Capítulo 6 del libro de M.A. Weiss "Estructuras de Datos en Java" (Adisson-Wesley, 2000).
- Capítulos 9-14 del libro de R. Wiener, L.J. Pinson "Fundamentals of OOP and Data Structures in Java" (Cambridge University Press)
- Capítulos 9-16 del libro S. Sahni "Data Structures, Algorithms and Applications in Java" (McGraw-Hill Higher Education, 2000)

Motivación

- Las aplicaciones informáticas suelen requerir la manipulación de colecciones de datos.
  - *lasFiguras* : Gestión de un grupo de Figuras.
  - *Listín telefónico* : Gestión de un grupo de abonados.
  - *Venta de Entradas* : Gestión del sistema de ventas.
- Sobre la colección de datos se pretenden realizar una serie de operaciones:
  - Insertar un elemento en la colección.
  - Borrar un elemento de la colección.
  - Recuperar un elemento que satisfaga alguna propiedad.

Introducción

Estructura de Datos (EDA) = Modelo o Especificación + Implementación

- Estructura de Datos (EDA): Organización concreta de una colección de datos cuyo comportamiento se especifica de forma independiente a su implementación.
  - El *Modelo* especifica el conjunto de operaciones que admite la estructura de datos. (¿Qué?).
  - La *Implementación* detalla cómo se implementan las operaciones especificadas por el Modelo. (¿Cómo?).
- El Modelo debe ser independiente de la Implementación.
  - Debe ser posible realizar múltiples implementaciones del Modelo.

Especificación vs Implementación

- Desarrollar una aplicación que transforme un conjunto de ficheros MP3 (.mp3) a formato OGG (.ogg).
  - Utilizar una supuesta clase Java MP3ToOGG que realiza la transformación.

```
public class MP3ToOGG {
/**
 * Converts a .mp3 file to an .ogg file within the same directory.
 * @throws UnableToConvert If the underlying decoder had troubles while converting.
 */
  public static void convert(String filePath) throws UnableToConvert{ … }
}
```


Ejemplo de Especificación vs Implementación

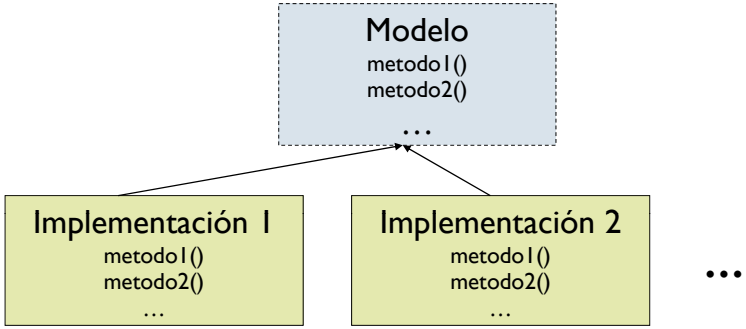
- Práctica 3: Gestión de un Grupo de Figuras en Java

GrupoDeFiguras

```
int talla();
void insertar(Figura f);
boolean eliminar(Figura fig);
Figura recuperar(int pos) throws ElementoNoEncontrado;
…
```

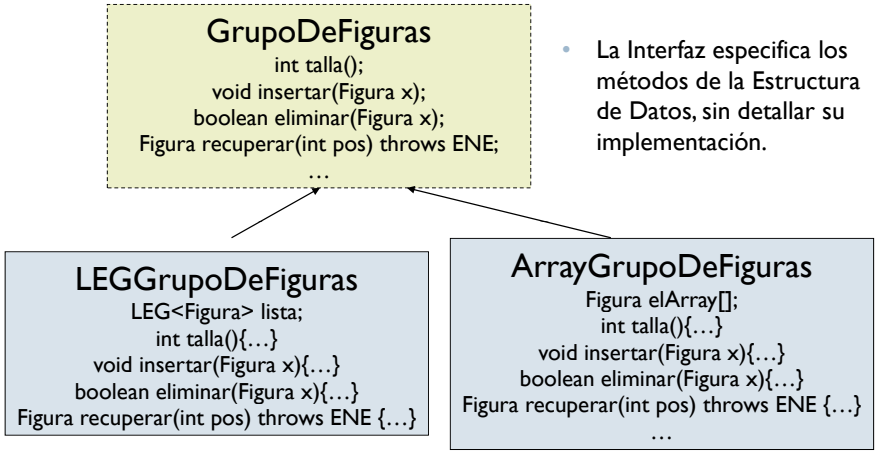
- La interfaz GrupoDeFiguras define las operaciones que soporta un grupo de figuras pero no especifica su implementación.
  - Para crear y utilizar un GrupoDeFiguras no nos hace falta saber cómo está implementado. Tan solo es necesario conocer la funcionalidad de cada operación.

Ejemplo de Especificación vs Implementación (II)



- Dado un Modelo (Especificación), se pueden realizar múltiples implementaciones del modelo en base a diferentes criterios, como por ejemplo, la eficiencia.

Ejemplo de Especificación vs Implementación (III)



- La Interfaz especifica los métodos de la Estructura de Datos, sin detallar su implementación.

- Si se preve realizar múltiples operaciones de borrado, interesa una representación enlazada ya que el coste es menor.

Ejemplo: Urgencias Hospitalarias

- Modelización de un servicio de recepción de urgencias hospitalario.
- Los *Pacientes* acuden al centro y un *Gestor* se encarga de decidir quien es el siguiente paciente a ser atendido, hasta que no queden más pacientes por tratar.
- A determinar:
  - Representación adecuada del grupo de pacientes.
  - Funcionalidad mínima sobre el grupo de pacientes:
    - void insertar(Paciente p); //Añade un nuevo Paciente al Grupo.
    - Paciente recuperar(); // Obtiene el próximo Paciente a tratar.
    - boolean borrar(Paciente x); // Elimina el Paciente que ya ha sido atendido.

Ejemplo: Urgencias Hospitalarias (II)

- Alternativas de implementación:
  - Todos los Pacientes tienen la misma prioridad → gestión de forma FIFO (**F**irst-**I**n **F**irst-**O**ut). El Gestor se comporta como una **Cola**.
  - No todos los pacientes de urgencias tienen la misma prioridad (ej.: Paro Cardiaco vs Resfriado Común). En cada momento, se debe atender al paciente de máxima prioridad (mínimo tiempo de espera).
- Esquema de **Cola de Prioridad**:
  - void insertar(Paciente p);
  - Paciente recuperarMin();
  - Paciente eliminarMin();
- Un Gestor de Trabajos se puede generalizar y adaptar a diferentes situaciones:
  - Trabajos de Impresión, tareas de un Sistema Operativo.
  - Venta de entradas de cine, etc.

Especificación de una EDA en Java

- EDA = Modelo + Implementación
- El Modelo debe ser independiente de la Implementación.
- La forma más conveniente de especificar un modelo en Java es mediante una *Interfaz*.
  - Todos sus métodos son abstractos.
  - No permite la definición de constructores ni atributos no estáticos.
- Una clase puede implementar todas las *interfaces* que quiera.

Ejemplos de Especificación de una EDA

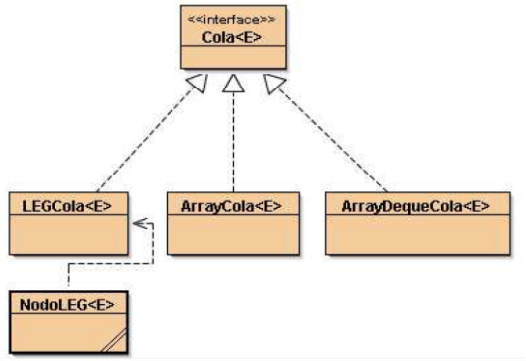
```
public interface Cola<E>{
   void encolar(E x);
   E desencolar();
   E primero();
   boolean esVacia();
}
```

```
public interface ColaPrioridad<E
extends Comparable<E>>{
    void insertar(E x);
    E eliminarMin();
    E recuperarMin();
    boolean esVacia();
}
```

- La **Especificación** de la EDA muestra sus operaciones principales.
- Posteriormente, la **Implementación** de una EDA supone:
  1. Elegir la representación interna de la colección de datos.
  2. Dar código a los métodos especificados por la interfaz implementada.

Ejemplo: La EDA Cola

- Múltiples implementaciones de la interfaz Cola<E>
  - Utilizando una Lista Enlazada Genérica, un Array, o un ArrayDeque (ver Java 1.6 API).



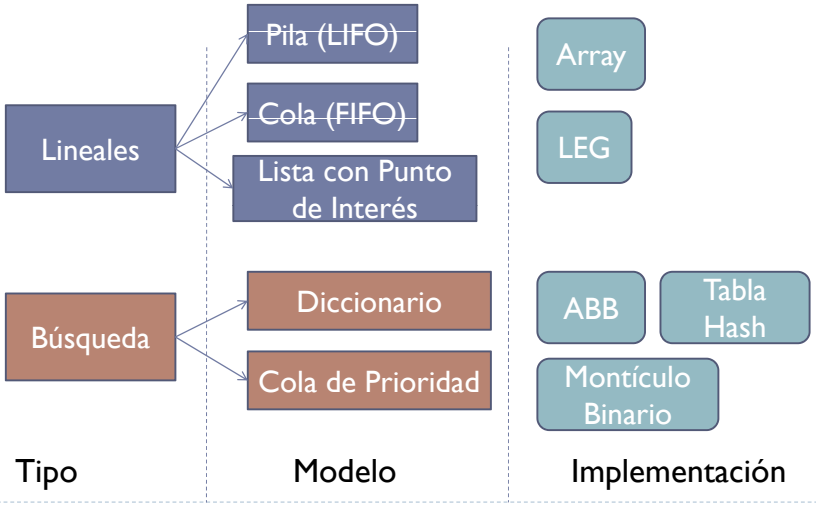
Atisbo de Implementación de Cola<E> con un Array

```
public class ArrayCola<E> implements Cola<E> {
    protected E elArray[];
    protected int talla;
    //Resto de atributos
   public ArrayCola(){...}
   public void encolar(E x){...}
   public E desencolar(){...}
   public E primero(){ ... }
   public boolean esVacia(){ return ( talla == 0 ); }
   public String toString(){…}
}
```

Uso de una EDA

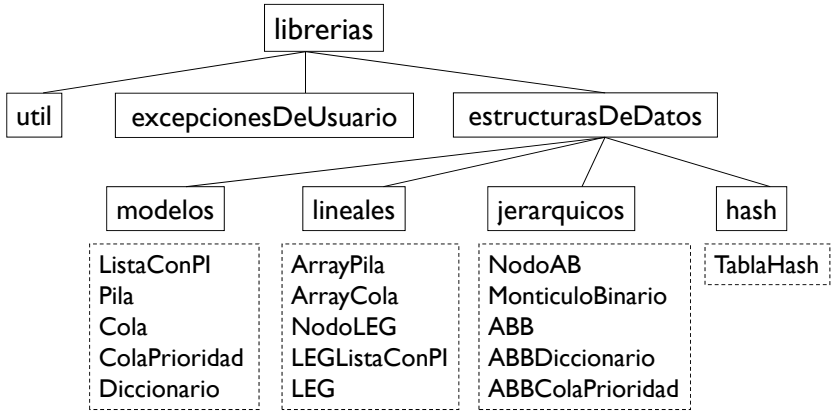
```
package gestionEDACola;
import modelos.*;
import lineales.*;
public class TestEDACola {
   public static void main(String args[]){
      Cola<Integer> q = new ArrayCola<Integer>();
      q.encolar(new Integer(10));
      q.encolar(new Integer(20));
      System.out.println("Extraigo el elemento:" + q.desencolar());
      System.out.println("La Cola de Integer actual es q = [ "+q.toString()+"]");
   }
}
```


Clasificación General de EDAs



Librerías de Usuario para el manejo de EDAs

- Jerarquía de directorios utilizada en prácticas:



Extensión de una EDA

- Es posible aumentar la funcionalidad de una EDA para añadir nuevos métodos útiles para nuestras aplicaciones.
- La extensión realizada debe seguir el principio de ocultación de información para garantizar que el software sea reutilizable.
- La extensión se puede llevar a cabo de dos maneras:
  - Mediante una subclase de la implementación de la EDA que incluya el nuevo método:
    - Accediendo a la implementación.
    - Sin acceso a la implementación.
  - Utilizando clases repositorio.

Extensión de una EDA: Con Acceso a Implementación

- Añadir un método para vaciar una Cola

```
public interface ColaExt<E> extends Cola<E>{
    void vaciar();
}
```

```
public class  ArrayColaExt<E> extends ArrayCola<E> implements
    ColaExt<E>{
    public void vaciar(){
          talla = 0;
        //Resto de acciones
    }}
```

- El acceso a la implementación permite la implementación eficiente.
- No siempre es posible acceder a la implementación (i.e., atributo talla definido como privado).

Extensión de una EDA: Sin Acceso a Implementación

- Añadir un método para vaciar una Cola

```
public interface ColaExt<E> extends Cola<E>{
    void vaciar();
}
```

```
public class  ArrayColaExt<E> extends ArrayCola<E> implements
    ColaExt<E>{
    public void vaciar(){
          while (!esVacia()) desencolar();
    }
}
```

- La implementación se realiza exclusivamente con los métodos definidos por la interfaz.
- Coste temporal: Lineal con el número de elementos de la Cola.

Extensión de una EDA: Clases Repositorio

- Una clase repositorio implementa nuevos métodos accediendo únicamente a la funcionalidad de la EDA a través de su interfaz.

```
public abstract class RepoColaExt<E> implements
    ColaExt<E>{
    public void vaciar(){
          while (!esVacia()) desencolar();
    }
}
```

- La clase repositorio se convierte en un almacén de operaciones.
- Para disponer de una implementación completa, se deberá realizar una subclase que proporcione código al resto de métodos de la interfaz Cola.