Tema 4 Listas

ESTRATEGIAS DE PROGRAMACION Y ESTRUCTURAS DE DATOS

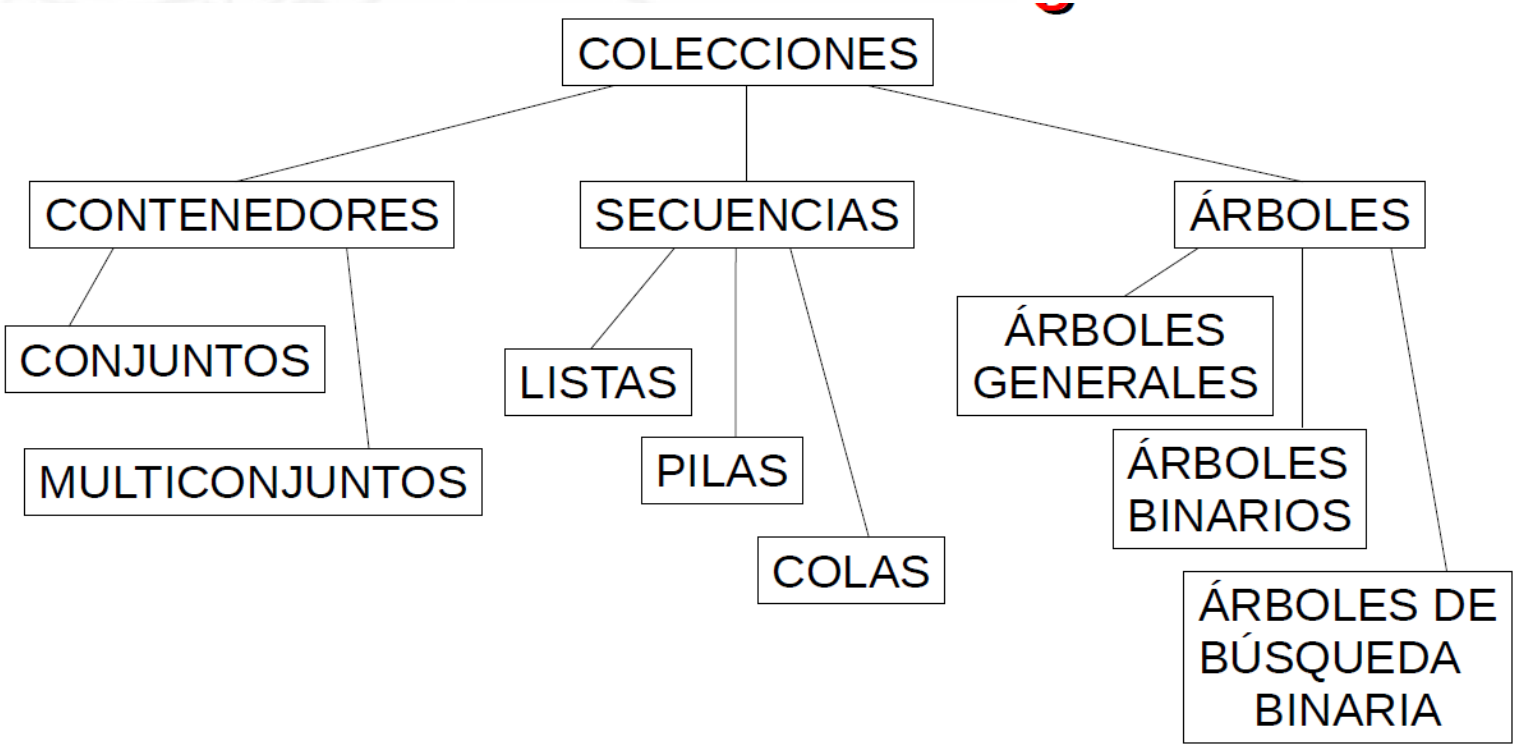
CA Guadalajara (UNED)

Índice

Implementación de Colecciones y Secuencias

Implementación de Listas

Implementación de Colecciones y Secuencias



Colecciones

- Sin restricciones adicionales: sólo importa si un elemento está o no
  - TODOS los demás TAD extienden Colección
- ¿Qué operaciones hacen falta?
  - Tamaño
  - Vaciar, ¿está vacía?
  - ¿Pertenece un elemento?

Colecciones

```
public abstract class Collection<E> implements CollectionIF<E> {
  protected int size;

  /* Constructor por defecto de una colección */
  public Collection () { size = 0; }

  /* Devuelve el número de elementos de la colección */
  public int size() { return size; }

  /* Nos dice si la colección está vacía o no */
  public boolean isEmpty() { return size == 0; }

  /* Vacía la colección */
  public void clear() { size = 0; }

  abstract public boolean contains(E e);
}
```

Secuencias

Los datos forman una secuencia: elementos con orden lineal explícito

- Relaciones de adyacencia: antecesión y sucesión
- Elemento antecesor y elemento sucesor (si hay)
- Organización no condiciona representación
- No hay orden implícito (aunque las listas podrían)

¿Qué operaciones hacen falta?

- Recorrer los elementos

Secuencias: Nodo – estructura de datos

Un nodo nos permite almacenar un valor

Un nodo además almacena cuál es el siguiente nodo

- Java → los objetos son “punteros” al objeto
- Un nodo almacena el puntero al siguiente nodo

Una secuencia → un nodo inicial

Secuencias

```
public abstract class Sequence<E> extends Collection<E> implements
SequenceIF<E> {
  /* Clase privada que implementa la estructura de nodos *
   * de la secuencia                                     */
  protected class NodeSequence { … }

  /* Clase privada que implementa un iterador para la secuencia */
  private class SequenceIterator implements IteratorIF<E> { … }

  protected NodeSequence firstNode;

  …
}
```

Secuencias: Clase Nodo

```
protected class NodeSequence {

  private E value;
  private NodeSequence next;

  NodeSequence(){ this.value = null; this.next = null; }

  NodeSequence(E e){ this.value = e; this.next = null; }

  public E getValue(){ return this.value; }

  public void setValue(E e){ this.value = e; }

  public NodeSequence getNext(){ return this.next; }

  public void setNext(NodeSequence n){ this.next = n; }

}
```

Secuencias: Iterador

```
private class SequenceIterator implements IteratorIF<E> {

  private NodeSequence currentNode;

  SequenceIterator(){ this.currentNode = firstNode; }

  public E getNext() {
    E elem = this.currentNode.getValue();
    this.currentNode = this.currentNode.getNext();
    return elem;
  }
  public boolean hasNext() { return this.currentNode != null; }
  public void reset() { this.currentNode = firstNode; }
}
/* Devuelve un iterador para la secuencia */
public IteratorIF<E> iterator() {
  return new SequenceIterator();
}
```

Secuencias: Métodos Auxiliares

```
/* Devuelve el primer nodo de la secuencia */
private NodeSequence getFirstNode() {
  return this.firstNode;
}

/* Devuelve el nodo i-ésimo de la secuencia          *
 * @Pre: 1 <= i <= size()                            */
protected NodeSequence getNode(int i){
  NodeSequence node = this.firstNode;
  for ( int aux = 1 ; aux < i ; aux++ ) {
    node = node.getNext();
  }
  return node;
}
```

Secuencias: Constructores

```
/* Constructor de una secuencia vacía */
public Sequence () { super(); this.firstNode = null; }

/* Constructor por copia */
public Sequence (Sequence<E> s) {
  this();
  if ( ! s.isEmpty() ) {
    this.size = s.size();
    NodeSequence nodeS = s.getFirstNode();
    NodeSequence pNode = new NodeSequence(nodeS.getValue());
    this.firstNode = pNode;
    while ( nodeS.getNext() != null ) {
      nodeS = nodeS.getNext();
      NodeSequence nNode = new NodeSequence(nodeS.getValue());
      pNode.setNext(nNode);
      pNode = nNode;
    }
  }
}
```

Secuencias: Métodos Clear y Contains

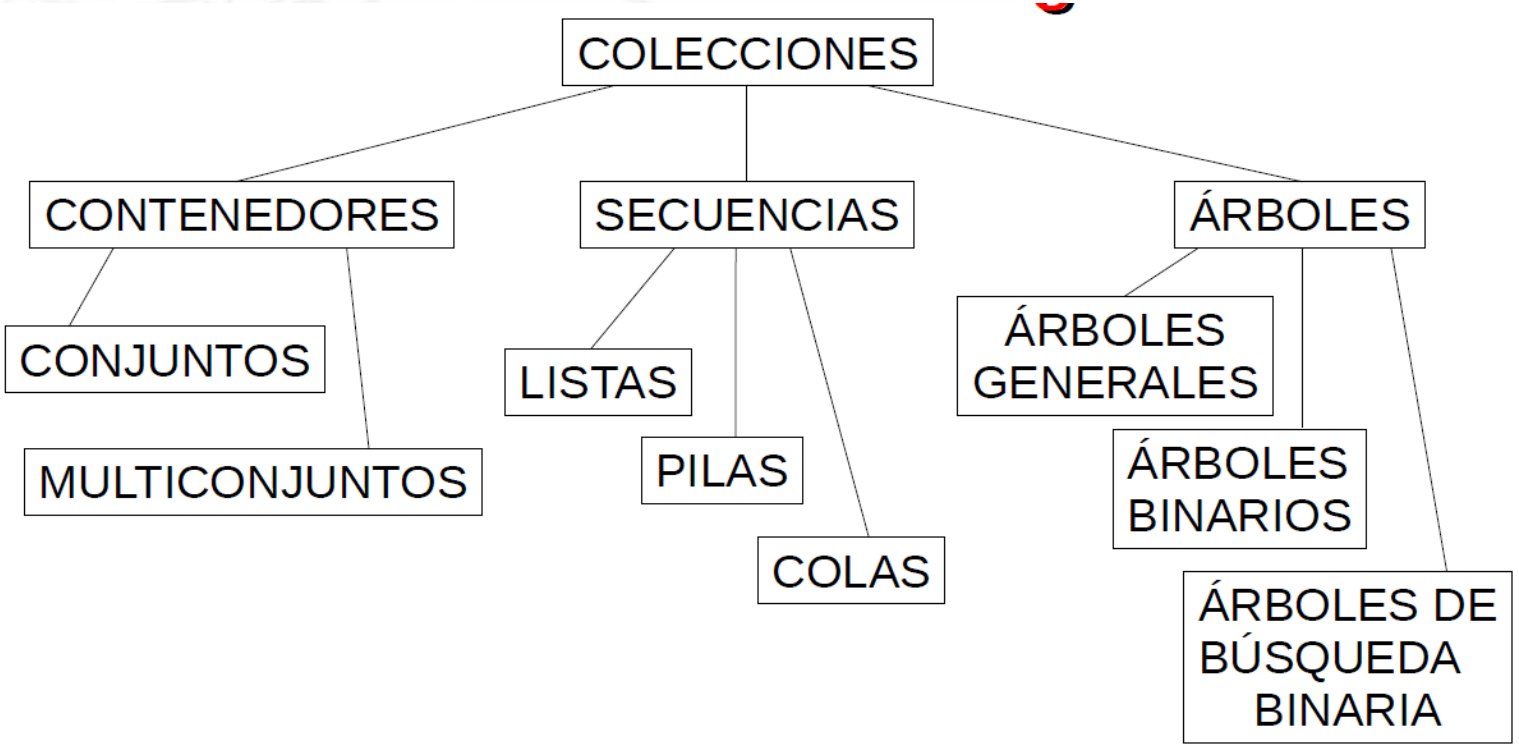
```
/* Reimplementación/Especialización de métodos de Collection */

/* Elimina todos los elementos de la secuencia */
public void clear () { super.clear(); this.firstNode = null; }

/* Métodos heredados de CollectionIF */

/* Comprueba si la secuencia contiene el elemento */
public boolean contains(E e) {
  NodeSequence node = this.firstNode;
  while(node!=null){
    E next = node.getValue();
    if(next.equals(e)){ return true; }
    node = node.getNext();
  }
  return false;
}
```

Implementación de Listas



Implementación de Listas

Acceso arbitrario a los elementos por un punto

Recorrido NO destructivo

Modelo de implementación

- **Acceso por posición.**

Implementación de Listas: Constructores

```
public class List<E> extends Sequence<E> implements ListIF<E> {

  /* Constructor por defecto: crea una lista vacía */
  public List(){ super(); }

  /* Constructor por copia: delega en el constructor por copia *
   * de la secuencia                                           */
  public List(List<E> s) { super(s); }
```

Implementación de Listas: Acceso y Modificación

```java
/* Devuelve el elemento pos-ésimo de la lista */
public E get(int pos) {
  NodeSequence node = getNode(pos);
  return node.getValue();
}

/* Modifica el elemento pos-ésimo de la lista */
public void set(int pos, E e) {
  NodeSequence node = getNode(pos);
  node.setValue(e);
}
```

Implementación de Listas: Inserción

```
/* Inserta un nuevo elemento en la lista en la posición *
 * indicada                                              */
public void insert(int pos, E elem) {
  NodeSequence newNode = new NodeSequence(elem);
  if(pos==1){
    newNode.setNext(this.firstNode);
    this.firstNode = newNode;
  }else{
    NodeSequence previousNode = getNode(pos-1);
    NodeSequence nextNode = previousNode.getNext();
    previousNode.setNext(newNode);
    newNode.setNext(nextNode);
  }
  this.size++;
}
```

Implementación de Listas: Eliminación

```
  /* Elimina el elemento pos-ésimo de la lista */
  public void remove(int pos) {
    if(pos==1){
      this.firstNode = this.firstNode.getNext();
    }else{
      NodeSequence previousNode = getNode(pos-1);
      NodeSequence nextNode = previousNode.getNext().getNext();
      previousNode.setNext(nextNode);
    }
    this.size--;
  }
}
```

Tema 4 Listas

ESTRATEGIAS DE PROGRAMACION Y ESTRUCTURAS DE DATOS

CA Guadalajara (UNED)