Arrays, Contenedores y Colecciones

Introducción al desarrollo del Software con la plataforma J2EE

Curso de Extensión Universitaria

Gijón, 2006

Justificación

- Es necesario guardar objetos (referencias a objetos) para manipularlos desde los programas.
- J2EE proporciona dos (tres) mecanismos:
  - Arrays
  - Contenedores (colecciones)

Arrays

- Secuencia lineal de referencias a objetos.
- Ventajas: eficiencia y manejo de tipo
- Inconveniente: tamaño FIJO
- Existen arrays de tipos primitivos y arrays de objetos (almacenan referencias a objetos).

Ejemplos

- ManejoArrayPrimitivo
  - Realiza la definición de diferentes arrays de elementos de tipo “int”.
- ManejoArrayMultidimensional
  - Define arrays multidimensionales de tipo primitivo
- ManejoArrayObjetos
  - Realiza la definición de arrays de objetos de la clase Animal

Clase Arrays

- Mantiene una serie de métodos estáticos para manipular arrays (java.util.Arrays)
- Métodos fundamentales:
  - equals(): dos arrays son iguales
  - fill(): para rellenar un array con un valor
  - sort(): para ordenar un array
  - binarySearch: para encontrar un valor en un array ordenado
- Sobrecargados para tipos primitivos y objetos
- Método asList(): convierte un array en un contenedor List.

Ejemplo Clase Arrays

- Crea arrays de String
- Utiliza los métodos de Arrays para:
  - Rellenarlos
  - Compararlos
  - Ordenarlos
  - Buscar un String

Colecciones

- Permite *almacenar* y organizar *objetos* de manera útil para un acceso eficiente.
- Se encuentran en el paquete `java.util`
- Núcleo de abstracciones de colecciones de utilidad (interfaces) e implementaciones ampliamente útiles.

Colecciones

- Las interfaces proporcionan métodos para todas las operaciones comunes y las implementaciones concretas especifican la decisión de las operaciones no permitidas.

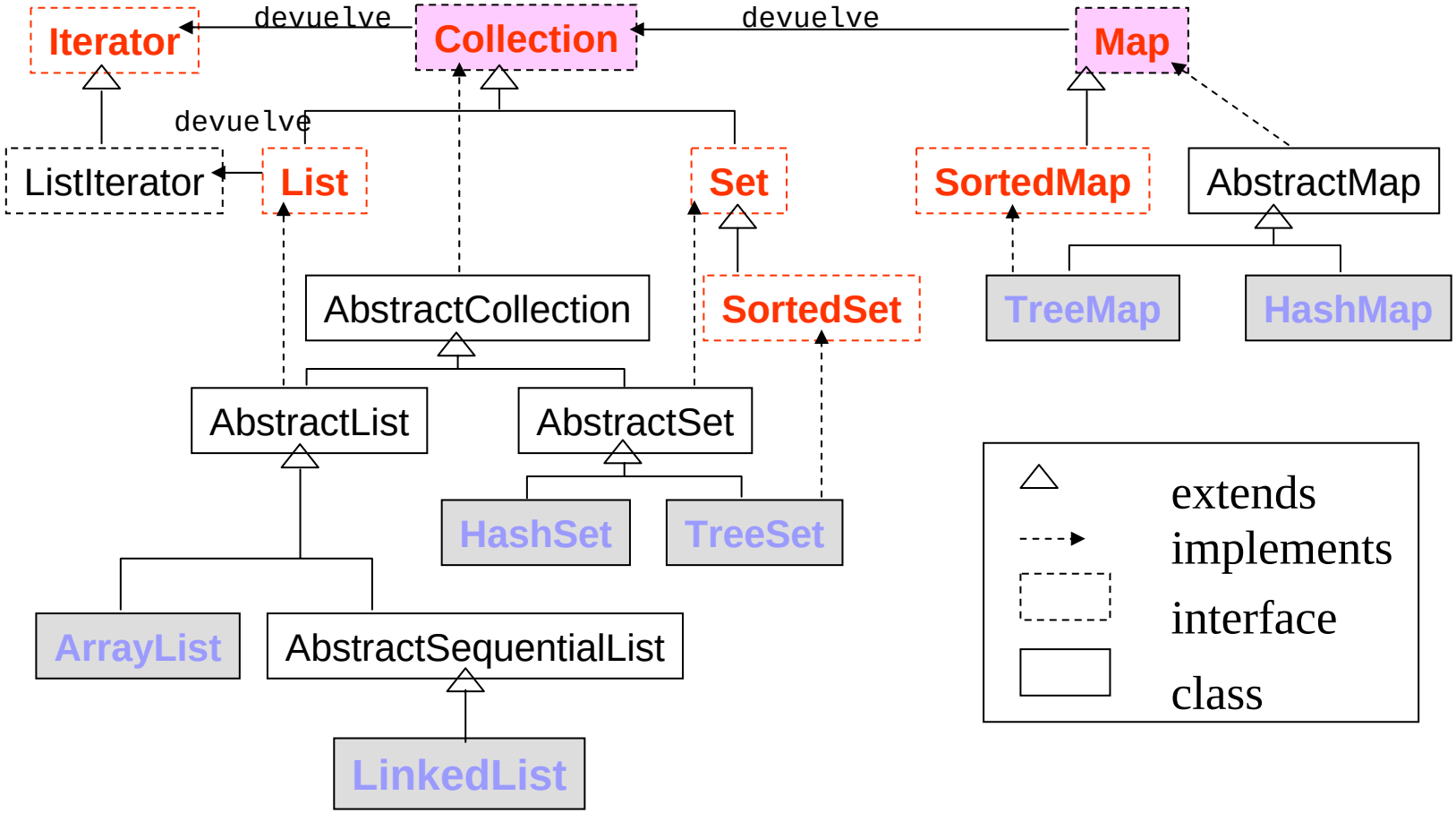
  (`java.lang.UnsupportedOperationException`)

- Sobre los elementos se puede iterar (`Iterator`)

Colecciones

- Inconvenientes:
  - Almacenan referencias a Object, se pierde la información de Tipo
  - Hay que realizar una conversión de tipo antes de utilizar las referencias.

Jerarquía de colecciones



Colecciones e iteradores

- Interfaz `Collection` (elementos individuales):
  - Interfaz `List`: elementos con una secuencia concreta. Clases `ArrayList` y `LinkedList`
  - Interfaz `Set`: no admite duplicados. Clases `HashSet` y `TreeSet`
- Interfaz `Map` (grupo de pares clave-valor)
  - Clases `TreeMap`, `HashMap`
- Iteratores: interfaz `Iterator`

Ejemplo

- PrimerContenedor.java
  - Crea tres contenedores:
    - ArrayList
    - HashSet
    - HashMap

Interfaz `Collection`

- Métodos
  - int size()
  - boolean empty()
  - boolean contains(Object elem)
  - Iterator iterator()
  - Object[] toArray(), Object[] toArray(Object dest[])
  - boolean add(Object elem),
  - boolean remove(Object elem)
  - void clear()

Interfaz `List` (`extends Collection`)

- Una colección cuyos elementos permanecen en un orden particular a menos que se modifique la lista (no significa lista enlazada aunque es una posible implementación).
  - `void add(int index, Object element)`
  - `Object remove(int index)`
  - `Object get(int index)`
  - `Object set(int index, Object element)`
  - `int indexOf(Object o)`
  - `int lastIndexOf(Object o)`
  - `List subList(int min, int max)`

Interfaz `Set` (`extends Collection`)

- Una colección (conjunto) donde no puede haber elementos repetidos, y cuyos elementos no se almacenan necesariamente siguiendo un orden particular.
  - Mismos métodos que `Collection`.

Interfaz `SortedSet` (`extends Collection`)

- Conjunto con elementos ordenados.
  - `Object` **`first`**`()`
  - `Object` **`last`**`()`
  - `SortedSet` **`subSet`**`(Object fromElement, Object toElement)`
  - `SortedSet` **`headSet`**`(Object toElement)`
  - `SortedSet` **`tailSet`**`(Object fromElement)`

Interfaz **`Iterator`**

- Permite iterar entre los elementos de una colección
  - `boolean hasNext();`
  - `Object next();`
  - `void remove();`

- La interfaz **`ListIterator`** extiende a `Iterator` y maneja un objeto `List` ordenado. Permite iterar hacia delante y hacia atrás.

Ejemplo de uso de Iteradores

- Cálculo del gasto total de un departamento. Plantilla es una colección que implementa la interfaz `Collection`

```
public double gastoDpto(){
   double gasto=0;
   Iterator it=plantilla.iterator();
   while (it.hasNext()){
       gasto+=((Empleado)it.next()).getSueldo();
   }
   return gasto;
}
```

Implementaciones de `Collection`

- Las implementaciones de la interfaz `Collection` son:
  - `LinkedList`
  - `ArrayList`
  - `HashSet`
  - `TreeSet`
- Todas son Serializable

LinkedList

- Una implementación de una lista doblemente enlazada.
- La modificación es poco costosa para cualquier tamaño, pero el acceso aleatorio es lento.
- Útil para implementar colas y pilas.
  - `getFirst`
  - `getLast`
  - `removeFirst`
  - `removeLast`
  - `addFirst`
  - `addLast`

Ejemplos de LinkedList

- 01-LinkedList
  - Implementación y manejo de un contenedor de tipo LinkedList.
  - Utiliza métodos e iteradores
- 02- Pila
  - Implementación y manejo de una pila implementado con LinkedList.
- Ejercicio: Realizar la implementación de

ArrayList

- Una lista implementada utilizando un array de dimensión modificable.

- Es **costoso añadir o borrar** un elemento cerca del principio de la lista si ésta es grande, pero es relativamente poco costoso de crear y **rápido** para **acceso aleatorio**.

HashSet

- Un Set implementado mediante una tabla *hash*.

- Es una buena implementación de propósito general por lo que la búsqueda, la adición y eliminación son **insensibles al tamaño** de los contenidos.

TreeSet

- Un SortedSet implementado utilizando un árbol binario equilibrado.
- Es **más lento para buscar o modificar que un HashSet**, pero mantiene los elementos ordenados.
- Asume que los elementos son *comparables* si no se le ha pasado un *comparator* en el constructor.

Interfaz **Map**

- Un objeto que asocia claves con valores.
- No puede tener claves duplicadas.
  - `Object` **`put`**`(Object key, Object value);`
  - `Object` **`remove`**`(Object key);`
  - `Object` **`get`**`(Object key);`
  - **`containsKey`**, **`containsValue`**, **`isEmpty`**, **`size`**

Interfaz **`Map`**

- Proporciona tres vistas de colección: colección de claves (keySet), colección de valores (values), colección de asociaciones clave-valor (entrySet).

Interfaz `SortedMap` (`extends Map`)

- Un mapa cuyas claves están ordenadas.
  - `Object firstKey(),`
  - `Object lastKey(),`
  - `SortedMap subMap(Object minKey, Object maxKey),`
  - `SortedMap headMap(Object maxKey),`
  - `SortedMap tailMap(Object minKey)`

Implementaciones de `Map`

- Las implementaciones de la interfaz `Map` son:
  - `HashMap`
  - `TreeMap`

HashMap

- Una implementación de `Map` con una *tabla hash*.
- El método `hashCode` de cada clave se utiliza para seleccionar un lugar en la tabla
- Una colección de utilidad muy general con tiempos relativamente cortos de búsqueda e inserción.

Ejemplo de HashMap

- 03- HashMap: Crea una agenda como un HashMap donde relaciona claves y objetos de la clase persona.
  - Utiliza iteradores

TreeMap

- Una implementación de `SortedMap` utilizando un árbol binario equilibrado que mantiene sus elementos ordenados por clave.
- Útil para conjuntos de datos ordenados que requieren una búsqueda por clave moderadamente rápida.
  - Asume que los elementos son *comparables* si no se le ha pasado un *comparator* en el constructor.

Convenciones sobre excepciones

- `UnsupportedOperationException`
  - Métodos opcionales en la implementación de una interfaz
- `ClassCastException`
  - El tipo del elemento que se desea insertar no es del tipo apropiado

Convenciones sobre excepciones

- `IllegalArgumentException`
  - El valor del elemento no es apropiado para la colección
- `NoSuchElementException`
  - La colección de la que se quiere devolver un elemento está vacía
- `NullPointerException`
  - Se pasa como argumento una referencia con valor `null` cuando la colección no admite este valor.

Conversiones de tipos

- Ejemplo: ArrayListTipoObjeto
  - Dos clases básicas: Perro y Gato
  - Crea un ArrayList de la clase Gato
  - Rellena el ArrayList con gatos
  - Introduce un perro
  - Muestra el contenido del ArrayList

Declaración de colecciones

```
import java.util.*;

public class ColeccionSimple {
   public static void main( String args[]){
        List c = new ArrayList();
        for( int i=0; i < 10; i++ )
            c.add(new Integer(i));
        Iterator it = c.iterator();
        while( it.hasNext() )
          System.out.println(it.next());
   }
}
```

Clase concreta

interfaz

Las utilidades de `Collections`

- `public static Object **min**(Collection col)`
- `public static Object **max**(Collection col)`
- `public static Object **min**(Collection col, Comparator comp)`
- `public static Object **max**(Collection col, Comparator comp)`
- `public static void **reverse**(List lista)`

Las utilidades de `Collections`

- `public static void` **`copy`**`(List dst, List fnt)`
- `public static void` **`sort`**`(List lista)`
- `public static void` **`sort`**`(List lista, Comparator comp)`
- `public static int` **`binarySearch`**`(List lista, Object clave)`
- `public static int` **`binarySearch`**`(List lista, Object clave, Comparator comp)`

Conclusiones

- Si un método tiene que devolver (pasar como parámetro) una colección de objetos, el tipo será `Iterator` o cualquiera de las interfaces de colección.

Conclusiones

- El tipo de la declaración de los atributos y variables locales será cualquiera de las interfaces de colección.
  - `List lista = new ArrayList();`
  - Excepción: `LinkedList` si la utilizamos como pila o cola.
- Utilizar SIEMPRE `Iterator` para el recorrido de cualquier colección.

Ejercicio

- Realizar un ArrrayList de Mascotas, que sea “consciente” del tipo y que encapsule la funcionalidad del ArrayList y de Collections.
- Todas las mascotas de la práctica deben estar dentro de la Caseta.