Програмування-1

Лекція 10

Колекції

План лекції

- Загальні відомості про колекції
- Основні інтерфейси Collections Framework
- Set, TreeSet, HashSet
- Хешування
- List, ArrayList, LinkedList
- Перегляд колекцій
- Generic-типи та типізовані колекції
- Інтерфейс RandomAccess
- SubList, SubSet
- Застарілі колекції з JDK 1.0

Загальні відомості про колекції

- Колекції призначені для зберігання посилань на об’єкти

Масиви	Колекції
Об’єкти та примітиви	Об’єкти
Фіксований розмір	Розмір змінюється динамічно
Найбистріший Random Access	Наявність Random Access, способи та час доступу залежить від типу колекції
Реалізовані на рівні мови	Реалізовані на рівні бібліотек класів та інтерфейсів (java.util.*). Деякі конструкції Java по особливому працюють з колекціями

Версії колекцій

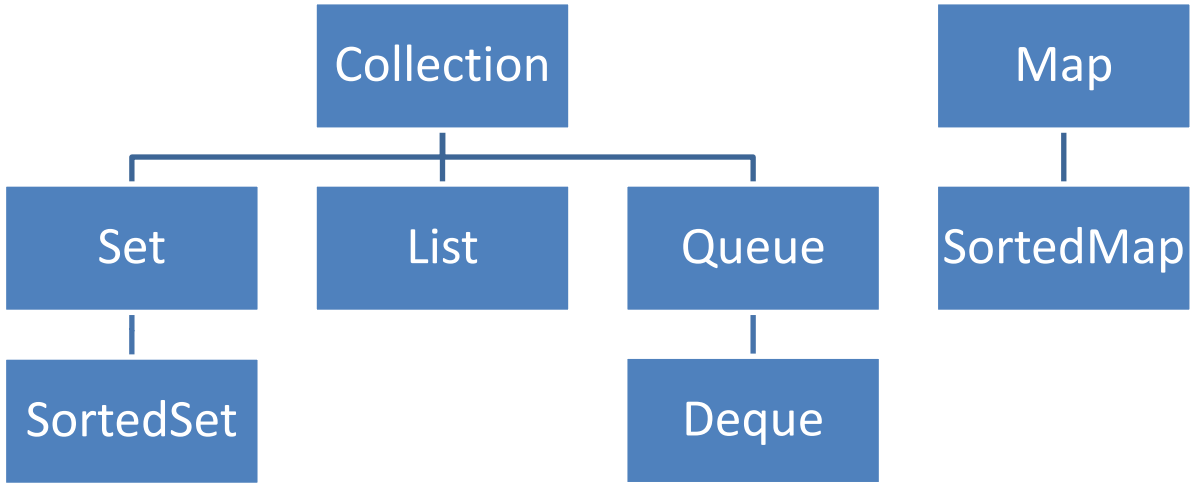
- Застарілі (Java 1.0)
  - Vector
  - Stack
  - Hashtable
  - Enumeration
- Collections framework (Java 1.2+)
  - Collection, Set, List, Queue, Map
  - ArrayList, LinkedList, TreeSet, HashSet,...

Термінологія

- collection ≠ Collection ≠ Collections ≠ Collections Framework

  – collection – колекція
  – Collection – інтерфейс
  – Collections – утилітний клас
  – Collections Framework – бібліотека

Основні інтерфейси



- **Collection** – посилання на окремі об’єкти
- **Map** – пари ключ-значення (асоціативні масиви)

  - **Collection**, **Set**, **SortedSet**, **List**, **Map**, **SortedMap** – Java 1.2+
  - **Queue** – Java 5+
  - **Deque** – Java 6+

Основні класи

Спосіб реалізації	Інтерфейси			
	Set	**List**	**Deque**	**Map**
Resizable Array		**ArrayList**	**ArrayDeque**	
Linked List		**LinkedList**		
Balanced Tree	**TreeSet**			**TreeMap**
Hash Table	**HashSet**			**HashMap**
Hash Table + Linked List	**LinkedHashSet**			**LinkedHashMap**

Інтерфейс Collection

- **Collection** – базовий інтерфейс для усіх колекцій
  - крім асоціативних масивів
- Найвищій рівень абстракції
  - відсутня інформація про тип колекції (Set / List / Queue / Deque / ...)
- Операції:
  - з окремим елементом «**e**»
    - **add(e), remove(e), clear()**
  - перегляд
    - **isEmpty(), size(), contains(e), iterator()**
  - групові операції (над даною **this** та іншою колекцією **collection**)
    - **addAll(collection)**
    - **removeAll(collection)**
    - **retainAll(collection)**
    - **containsAll(collection)**
  - перетворення у масив
    - **toArray()**

Інтерфейс List

- Це інтерфейс
- Представляє собою список
- Є нащадком інтерфейсу Collection
- Крім того має свої унікальні методи
    - get, set (по індексу)
    - add, remove (по індексу)
    - indexOf, lastIndexOf
    - listIterator
    - subList
- Дозволяє зберігати дублікати та null
- Порядок елементів при перегляді гарантується
- Швидкість операцій суттєво залежить від типу операції та реалізації
    - (далі буде приклад ArrayList VS LinkedList)

Класс ArrayList

- Представляє собою «масив змінної довжини»
- В середині прихований звичайний масив
- Реалізує інтерфейс List
- Додатково має такі методи:
  - ensureCapacity(int minCapacity)
  - trimToSize()
- Реалізує маркерний інтерфейс RandomAccess
- Реалізує маркерні інтерфейси Serializable, Cloneable

- Рекомендації:
  - get(), set() працюють дуже швидко
  - add(), remove() на початку та в середині колекції призводять до зсуву елементів у масиві
  - add() в конці колекції працює швидко при size<capacity. При size=capacity працює повільно через необхідність збільшити розмір масиву.

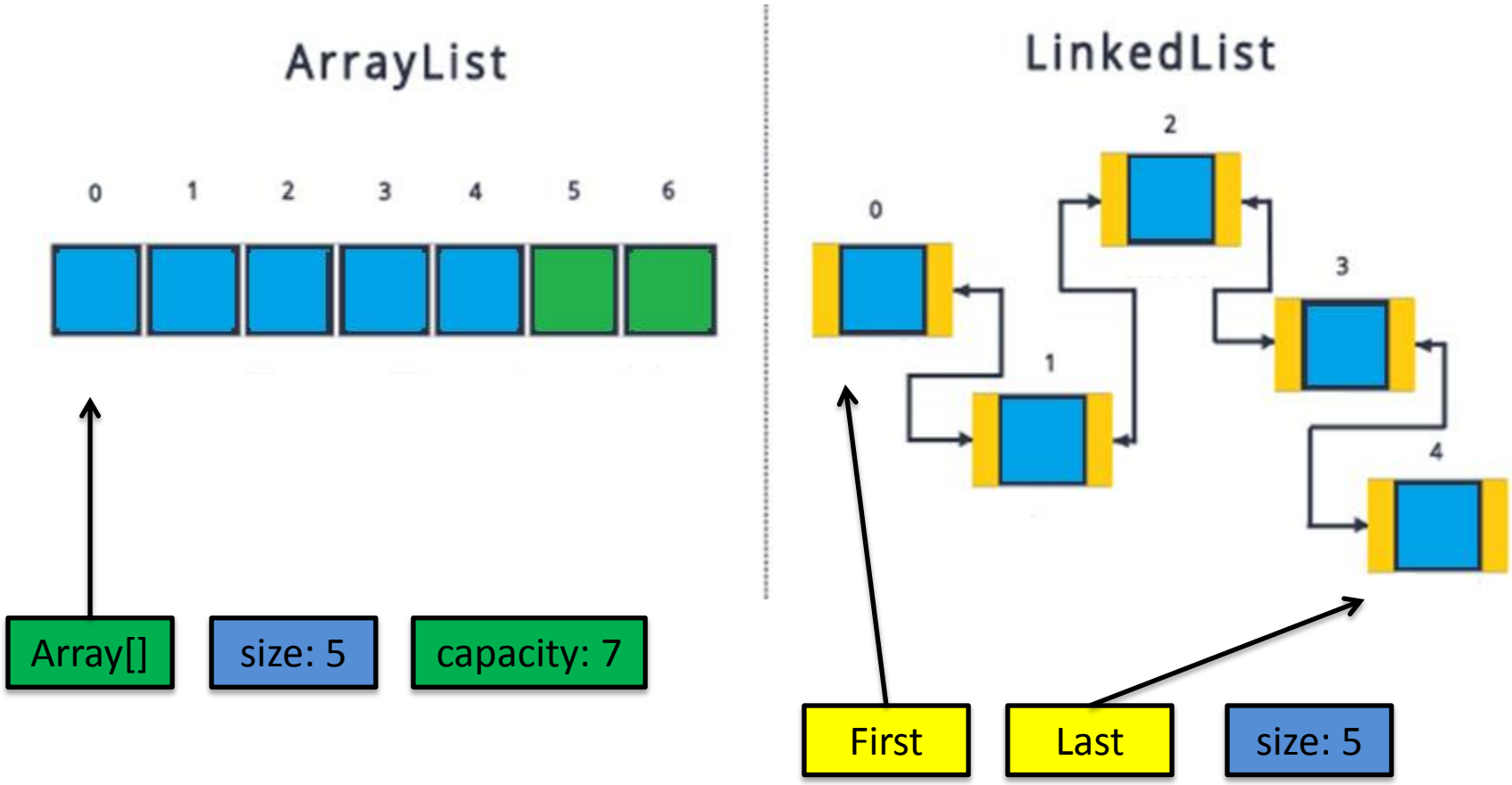
Клас ArrayList – приклад

```
import java.util.*;

public class ArrayListDemo {

        public static void main(String[] args) {
                List list = new ArrayList();
                list.add(new Integer(1));
                list.add(null);
                list.add("one");
                list.add("two");
                list.add("three");
                list.add("four");
                list.add("one");
                list.add("two");
                // check order, nulls, duplicates
                System.out.println(list);
        }
}
```

Класс LinkedList

- Реалізує інтерфейс List
- Реалізує інтерфейс Deque
  - можна створювати черги FIFO або FILO
- В середині реалізований як двобічно зв’язаний список
- Не реализує маркерный интерфейс RandomAccess
- Реалізує маркерні інтерфейси Serializable, Cloneable
- Рекомендации:
  - get(), set() працюють дуже повільно. Їх час роботи залежить від того як далеко елемент знаходиться від середини
  - add(), remove() в початок або кінець працюють дуже швидко

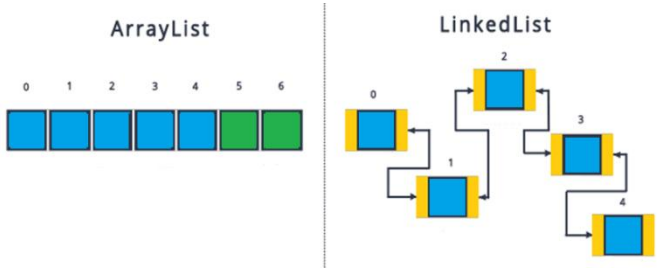
ArrayList vs LinkedList



ArrayList vs LinkedList

	ArrayList	LinkedList
get(index), set(index, element)	O(1)	~ O(n/4)
add(element)	size < capacity: O(1)	O(1)
	size = capacity: O(n)	
add(index, element)	~ O(n/2)	~O(n/4)
		index=0: O(1)
remove(element)	~O(n/2)	~O(n/4)
Iterator.remove()	~O(n/2)	O(1)
ListIterator.add(element)	~O(n/2)	O(1)
Iterator.next()	O(1)	O(1)

* ~ — в середньому



Інтерфейс Iterator

- Використовується для перегляду елементів у колекції
- Має наступні методи:
  - hasNext()
  - next()
  - remove() – опціонально
- У кожної колекції є власна реалізація цього інтерфейсу
- Зазвичай використовується наступним чином:

```
List list = new ArrayList();
list.add("1"); list.add("2"); list.add("3");

//...

for (Iterator i = list.iterator(); i.hasNext();) {
   Object o = i.next();
   // Do something with object
   System.out.println(o);
}
```

Цикл for-each

- У Java 5 з’явився цикл типу for-each
- Є «синтаксичним цукром»
  - компілятор генерує для нього такий саме байт-код як і для ітератора
- Тому не виграє і не програє ітератору з точки зору швидкодії
- Значно підвищує читабельність коду
- Не дозволяє видаляти елементи при перегляді колекції

```
List list = new ArrayList();
list.add("1"); list.add("2"); list.add("3");

//...

for (Object o:list) {
   System.out.println(o);
}
```

Типізовані колекції: for-each

- У Java 5 з’явилися generic-типи

Java 1.4	Java 1.5

`List list = new ArrayList();`
 	
`list.add("1");`
`list.add("22");`
`list.add("333");`
 	
`for (Object o:list) {`
`  if (o instanceof String) {`
`  String s = (String) o;`
`  // Do something with string s`
`  System.out.println(s.length());`
`  }`
`}`
	`List<String> list = new ArrayList<String>();`
	
	`list.add("1");`
	`list.add("22");`
	`list.add("333");`
	
	`for (String s:list) {`
	
	`  // Do something with string s`
	`  System.out.println(s.length());`
	`}`

Типізовані колекції: iterator

Java 1.4	Java 1.5

`List list = new ArrayList();`
`list.add("1");`
`list.add("22");`
`list.add("333");`
 	
`for (Iterator i = list.iterator();`
`  i.hasNext();) {`
`  Object o = i.next();`
`  if (o instanceof String) {`
`  String s = (String) o;`
`  // Do something with string s`
`  System.out.println(s.length());`
`  }`
`}`
	`List<String> list = new ArrayList<String>();`
	`list.add("1");`
	`list.add("22");`
	`list.add("333");`
	
	`for (Iterator<String> i = list.iterator();`
	`  i.hasNext();) {`
	`  String s = i.next();`
	
	`  // Do something with string s`
	`  System.out.println(s.length());`
	`}`

Інтерфейс RandomAccess

- RandomAccess – маркерний інтерфейс
  – пустий (не містить жодних елементів)
- Якщо він реалізований у класі колекції, то методи get(index), set(index) гарантовано працюють за O(1)
- Присутній у класах ArrayList, Stack, Vector
- Відсутній у LinkedList

Інтерфейс RandomAccess

```
List<String> list1 = new ArrayList<String>();
List<String> list2 = new LinkedList<String>();
// Заповнення колекцій
// спробуйте 10, 100, 1000, 10000, 100000
for (int i=0; i<10; i++) {
    list1.add("" + i);
    list2.add("" + i);
}
// Random access test
long t0=System.nanoTime(); // get list1
for (int i=0, n=list1.size(); i<n; i++) {
    String s = list1.get(i);
}
long t1=System.nanoTime(); // get list2
for (int i=0, n=list2.size(); i<n; i++) {
    String s = list2.get(i);
}
long t2=System.nanoTime();
// Результати
System.out.println("RandomAccess :" + (list1 instanceof RandomAccess) + " "
                                    + (list2 instanceof RandomAccess));
System.out.println("get(i)       :" + (t1-t0) + "  " + (t2-t1));
```