Collections

Lists, Queues, Vectors, Stacks

Before we get started...

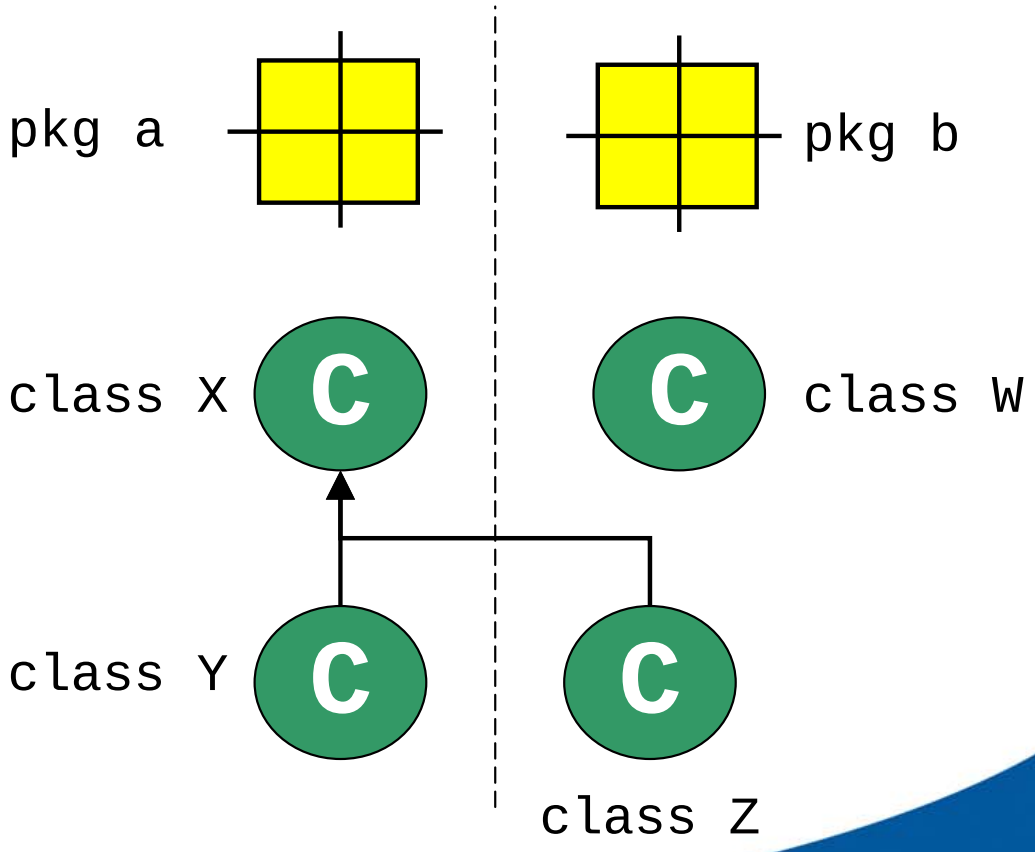
- **protected** keyword

```
public class X {

   private int id;

   public void method() {
       ...

   protected void funct() {
       ...
```



Arrays of Objects

- Defining an array: user accounts

```
int[] userIds;
double[] accountValues;
String[] userNames;
```

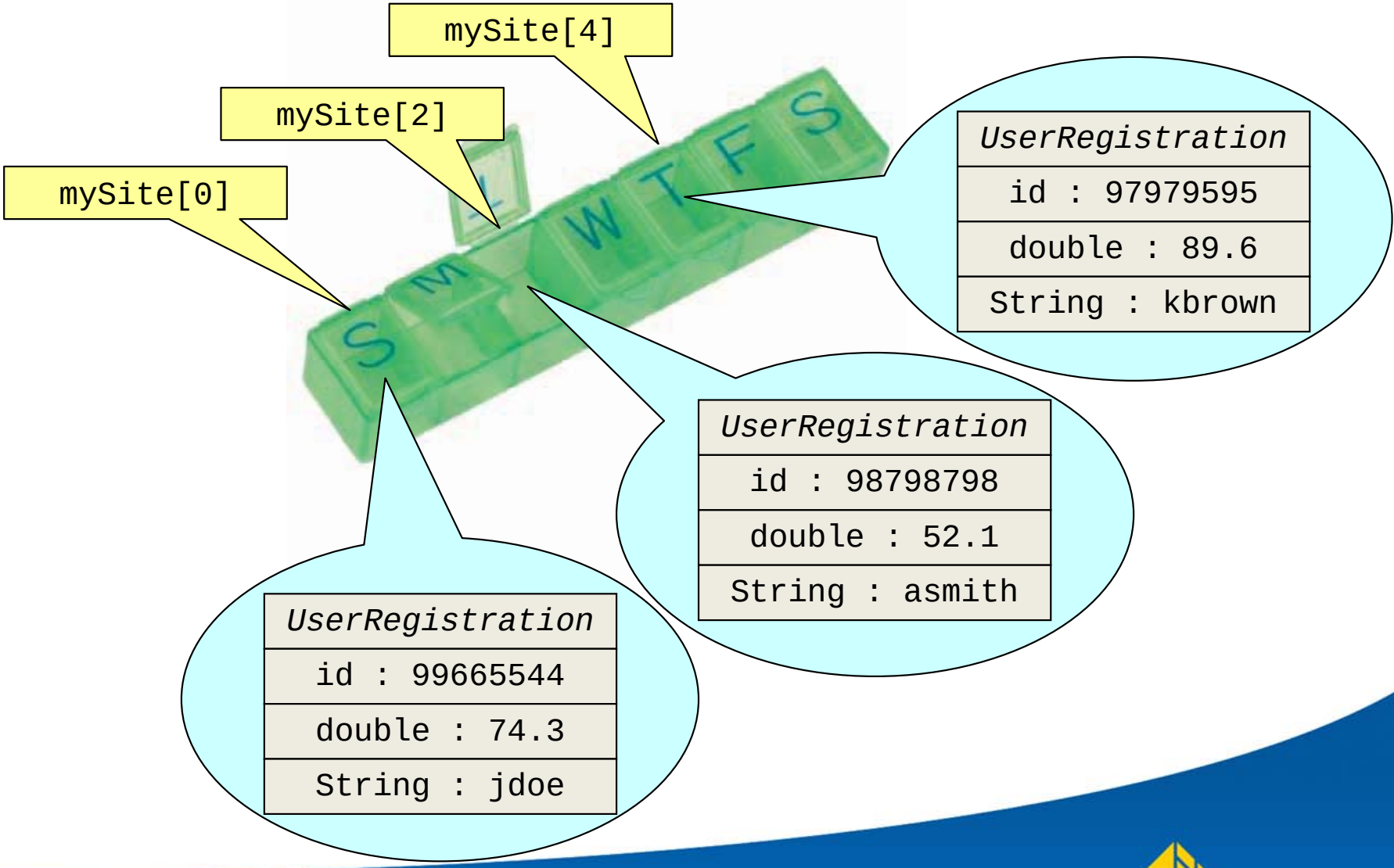
- Defining the user registration class:

```
public class UserRegistration {
    private int userId;
    private double balance;
    private String userName;
}
```

- The better way to define user accounts:

```
UserRegistration[] mySite;
```

Arrays of Objects



Sheridan

Arrays of Objects

- Things to remember:
  - You can't use curly brackets { } to initialize the array
  - Using *new* does not call a constructor

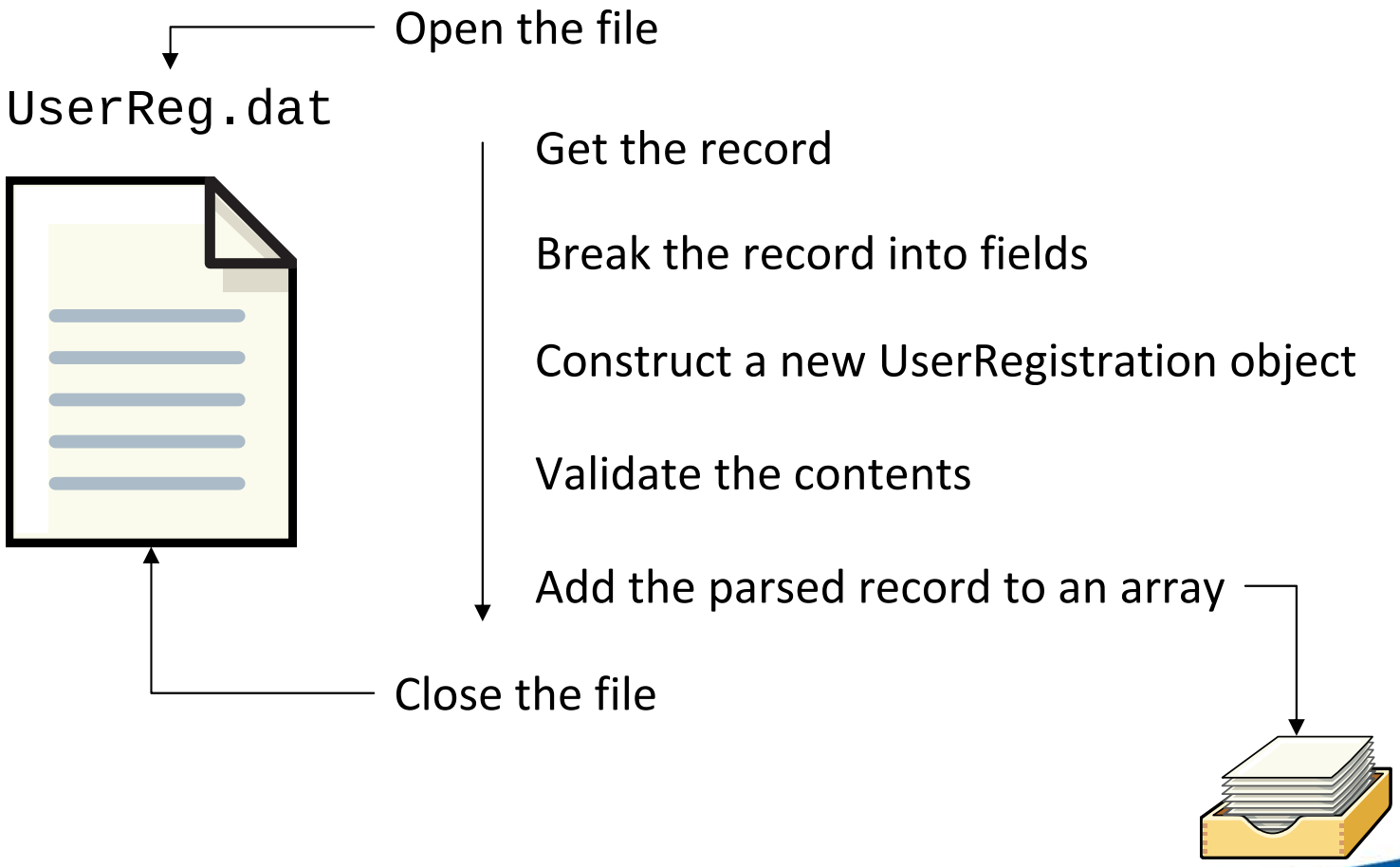
```
Student[] myClass = new Student[21]; // 21 null spaces
```

  - Polymorphism is really handy!

```
Farmer[] pettitRd = new Farmer[6];
pettitRd[0] = new SodFarmer();
pettitRd[1] = new StrawberryFarmer();
pettitRd[2] = new CattleRancher();
```

  - Really useful for input/output...

Using Object arrays with I/O



Collection classes

Collections

- Collection is an interface
  - One type stores elements (*collection*)
  - One type stores key/value pairs (*map*)
- Collections can be easier than arrays
  - Specific types with unique qualities
  - Can grow or shrink
  - Useful functions to work with
  - Con: take up more memory

How to initialize a collection

The collection class

Any generic class to collect

```
Collection<Element> var = new Collection<Element>();
```

```
ArrayList<String> names = new ArrayList<String>();

Vector<Employee> office = new Vector<Employee>();
```

How to iterate through collections

- Use an Iterator
  - Don't use 'for' loops!

```
Iterator<Element> var = collection.iterator();

while(var.hasNext()) {
      System.out.print(var.next());
}
```

Useful methods

+get(index : int) : Element
+add(e:Element) : boolean
+clear() : void
+remove(o:Object) : boolean
+size() : int
+toArray() : Object[]

+set(e:Element) : void
+addAll(c : Collection) : boolean
+contains(o : Object) : boolean
+removeAll(c : Collection) : boolean
+isEmpty() : boolean

Collections.sort(c:Collection) : void
Collections.reverse(c:Collection):void
Collections.shuffle(c:Collection):void
Collections.max(c:Collection):Object

Collections.binarySearch(c:Coll,
key:Object) : int
Collections.copy(dest:C, src:C) : void
Collections.min(c:Collection):Object

Collection types

- Sets
  - Group of non-duplicate elements
- Lists
  - Ordered collection of elements
- Queues
  - Collection of elements stored for processing

Lists

- Ordered elements
- Great for sorting elements sequentially
- Common methods:

  +indexOf(o:Object) : int

  +listIterator() : ListIterator<E>

  +subList(start:int, end:int) : List<E>
- Types of Lists
  - LinkedList
  - ArrayList
  - Vector extends AbstractList (synchronized array list)
  - Stack extends Vector

Stacks

- Type of List
- First in, last out

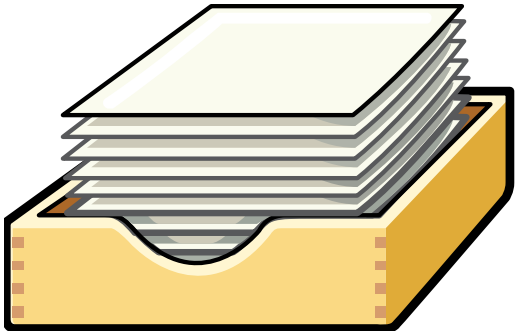
  +peek() : Element

  +push(o: Element) : Element

  +pop() : Element

  +search(o:Object) : int

  +empty() : boolean



Queues

- An interface
  - Also see Deque
- First in, first out

  +offer(e:Element) : boolean

  +poll() : Element

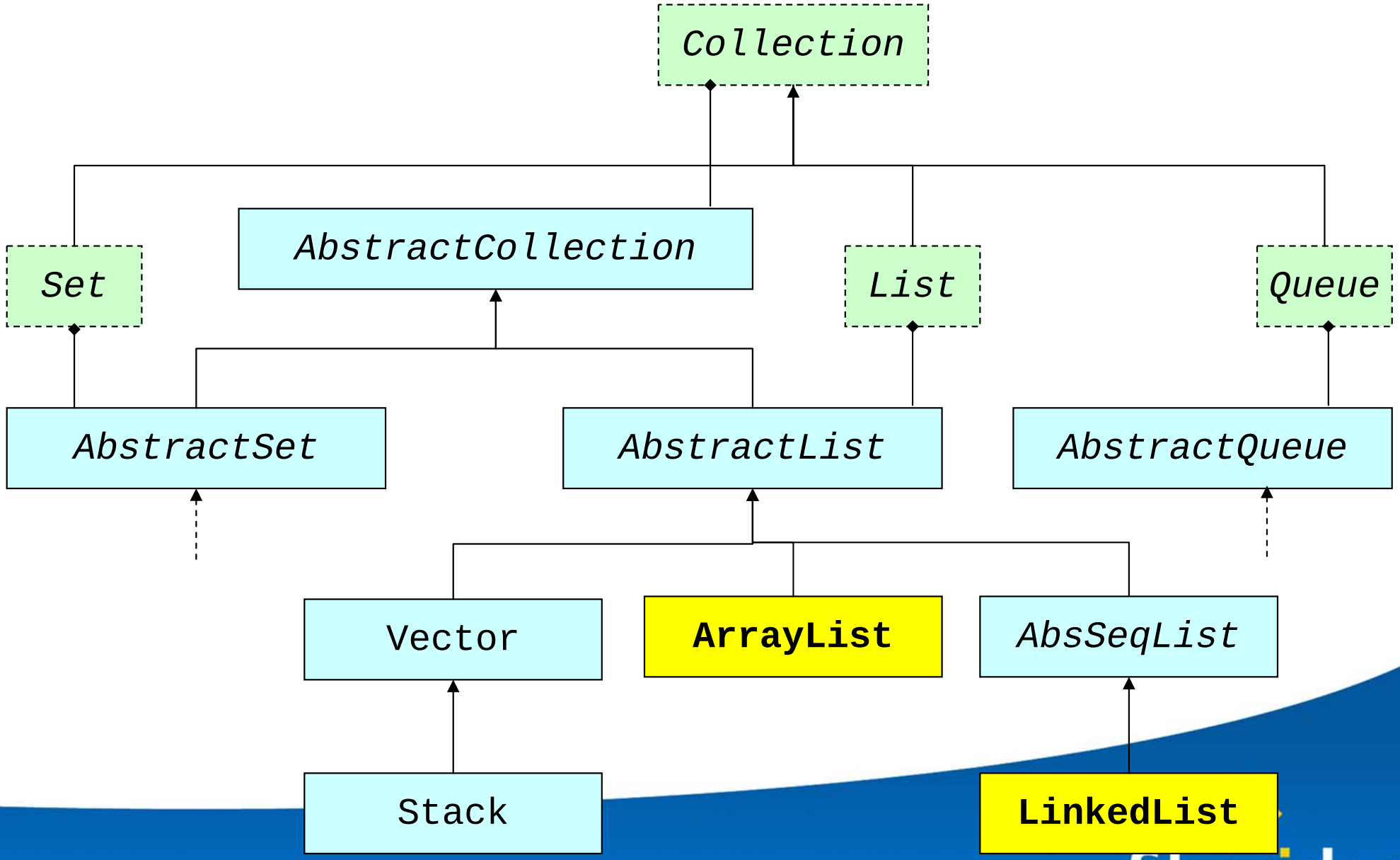
  +remove() : Element

  +peek() : Element

  +element() : Element

ArrayList

Collection hierarchy



ArrayList

- List, holds arrays of objects (any object!)
- Easier to use than arrays
- Has a capacity:
  - Not quite like size
  - Default capacity is 10
  - ensureCapacity(int) changes capacity

CRUD methods for ArrayList

- Insert an element
  - add(Element : e) : void / add(int : index, Element : e) : void
- Retrieve an element
  - get(int : index) : Element
- Change an element
  - set(int : index, Element : e) : Element
- Remove an element
  - remove(int : index) : boolean / remove(int : index) : Object
  - clear() : void