

Arrays and Vectors

The abstract concept of an array has some similarity with the mathematical notion of a sequence: a list of elements indexed by natural numbers and all of the same type. It was implemented in the first high level language (FORTRAN) as the only structured data type.

An array is a means of grouping a collection of variables together. It can be considered as a single object, or as a collection of elements. Both these views of an array are reflected in the following specification of an ADT Array:

Domain: A component data type is specified and all elements are of this component type. An index type, which must be linear, is also specified and there is a 1-1 correspondence between the values of the index type and the component element values.

Operations:

create (A [i : indexType] , size:integer) :

arrayOfComponentType

pre: none

post: reference created to array A and storage allocated for *size* elements of ComponentType

copy (c: ComponentType, A[i])

pre: A is non-empty

post: c contains the value of A's *i*th component

update (A[i], e)

pre: e is an expression of ComponentType

post: A's ith component contains the value of e

assign(A, B :arrayOfComponentType)

pre: A, B have the same size and component type

post: entire array B is copied into array A

Arrays are a special data type in the Java language. An array is an object that is a container holding a collection of values all of the same type. An array object must be created using the `new` operator. But there is no Array class of which array objects are instances. (Confusingly there is an Array class in `java.lang.reflect` but it does not define array objects!)

Arrays in Java are pseudo-primitive types: they are managed explicitly by the compiler responding to the `[]` notation for declarations and indexing. The index type must be natural numbers starting with 0. The index is also called a *subscript* and the elements may be called *components*.

For example:

```
Stack [] s ; // s is a variable of  
//type array of Stack
```

```
s = new Stack[5]; //creates array  
//object holding 5 elements.
```

```
s[0] = new Stack(); //assign a Stack
reference
```

Arrays can also be initialised with a special notation:

```
int [] i = {1, 4, 1, 5, 9};
```

```
Stack [] triad = {new Stack(); new
Stack(); new Stack() };
```

There is an important difference between declarations of an array of primitive type and one of a reference type. For example:

```
int [] arrayOfInt = new int [7];
```

allocates memory to hold seven integers. But

```
String [] arrayOfString = new
String[5];
```

allocates handles (references) to five strings; it does not allocate the memory for the String objects themselves. This can be done, for example, by

```
arrayOfString[0] = new String
("Hello");
```

.....

etc. (cf. The declarations of the Board class in the snakes program.) All Java arrays use a zero-based index:

Array:

John	Paul	George	Ringo
------	------	--------	-------

Index: 0 1 2 3

An array also has a special public instance variable `length` that is not changeable during the life of the object. An array is a static object.

For example, in any `main` method there is the declaration:

```
public static void main(String []  
args) {
```

```
// various statements  
}
```

If there are any arguments `main` could print them out using a `for` loop as :

```
for (int i = 0; i < args.length; i++)  
    System.out.println(args[i]);
```

Passing an array to a method – as in lecture.
See Watt&Brown Ch.3 for summary of arrays.
Use an array whenever:

- all elements of collection are of the same type;
- the collection is of a known, fixed size;
- the collection is a non-sorted data set (so data is not being inserted into collection).

If any of these conditions are not satisfied it may be better to use a Vector object.

A vector is a practical and flexible collection of data that is similar to an array but is dynamic – storage space can be added or deleted as required while the program is running. Vector objects are instances of the class Vector. Some of the most useful methods in the Vector class are:

```
public Vector()
// post : constructs an empty vector

public Vector(int initialCapacity)
//pre: initialCapacity >= 0
//post: constructs an empty vector
//with given initial capacity

public void addElement(Object obj)
//post: adds a new element to the end
//of the vector

public Object elementAt(int index)
//pre: 0 <= index && index < size()
//post: returns the element stored in
//location index

public void insertElementAt(Object
obj, int index)
//pre: 0 <= index < size()
//post: inserts new value in vector
//with index specified
```

```
public boolean isEmpty()  
//post: returns true iff there are no  
//elements in the vector
```

```
public boolean removeElement(Object  
element)  
//post: element equal to parameter is  
//removed, true if found, else false
```

```
public void removeElementAt(int where)  
//pre: 0 <= where && where < size()  
//post: element at 'where' is removed  
//and size decreases by 1
```

```
public void setElementAt(Object obj,  
int index)  
//pre: 0 <= index && index < size()  
//post: element value changed to obj
```

```
public int size()  
//post: returns the size of the vector
```

A vector object begins empty and expands as necessary. It has an end where the method `addElement` adds a value. To insert a new value in another position we can use `insertElementAt` and to access an element we use `elementAt`. The `removeElement` and `removeElementAt` methods reduce the logical size of the vector by one. The method `setElementAt` can be used to update a value. The methods `size` and `isEmpty` keep track of how many values are stored in the vector.

See a sample implementation of Vector by means of arrays in Bailey Chapter 3. The vector's ability to expand as necessary comes at the price of creating and copying the data to a new internal array. Each access to an element in a vector requires a method call (rather than the simple use of index notation).

By default a vector doubles in size each time it runs out of space. This represents a trade-off between speed and space management that is often efficient. Example in lecture. In some applications we might know the size will change in certain definite chunks and it will be more efficient to set the increment ourselves. This is possible through a third constructor in the Vector class (in addition to the two given above) as follows:

```
public Vector(int initialCapacity, int
capacityIncrement)
```

This uses the member field `capacityIncrement` to give control over how much the vector grows when memory is needed. There are many other methods provided – see the documentation. There are three instance variables:

```
protected Object elementData[];
```

```
protected int elementCount;
```

```
protected int capacityIncrement;
```

`elementData[]` is a generic Object array reference which holds the internal array storage for the collection.

It is possible, and convenient, in Java to have more than one method in the same class with the same name. For example, the three constructors in the Vector class. However, they must have different signatures – that is, different argument lists. Using methods with the same name in a class is called *method overloading*. Methods inherited from a superclass can also be overloaded. A subclass may even contain a method with an identical signature to a method in the superclass. This is called *method overriding* and allows a subclass to have its own version of a method supplied in the superclass.

A method which can be overridden is called a *virtual method*. By default all methods are virtual. The system uses a look-up table at runtime to determine which virtual function to use. Methods declared as final cannot be overridden and do not make use of this function table.

In the Vector class most of the methods are declared final. This prevents overriding and allows for faster access. The instance variables of Vector which give access to the internal representation are available to a subclass so although the core functionality cannot be changed we can add functionality (such as a sorting method) to the class.