Part 9: ARRAYS, LISTS AND MAPS

Programs often deal with thousands or even millions of items of information. We cover array data structure which can be used for such bulk data. Also generalisation of array provided by List interface, and related interface called Map which can also be very useful.

(CCJE, chapter 11)

- Arrays.
- The List interface.
- A simple database program.
- The Map interface.

Arrays

An array is a special kind of object. It consists of sequence of variables – called the *elements* of the array.

Elements of array do not have individual names, instead are numbered: 0, 1, 2, ... The number of an element is called its *index* (or subscript).

An array element can be any type of variable, but all elements in array must be of same type. So, for example, can have array whose elements are all of type *int*. Type of this array is denoted by *int*[]. Similarly can have array of type *String*[] whose elements hold references to strings, or array of type *Student*[] which holds references to *Student* objects, and so on.

Arrays

Number of elements in an array is called its *length*. Once an array has been created, length is fixed. Can change values stored in array elements, but cannot extend or shrink array. If a denotes an array, $a.length$ denotes its length.

If variable *row* contains reference to array of type $int[]$ of length 5, element at index i is denoted by $row[i]$. Can treat it like any other *int* variable. Can assign to $row[2]$:

```
row[2] = 0;
```

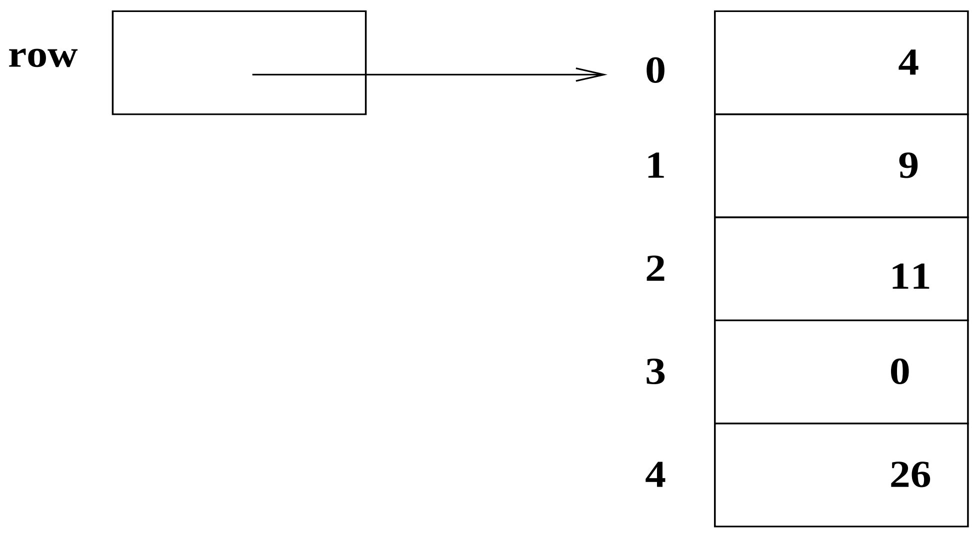
or output value of $row[4]$:

```
System.out.println(row[4]);
```

Or output values of all elements of *row*:

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


Layout of arrays in memory



Arrays

Whenever expression such as $row[4]$ is used to denote an array element, Java will check that array really does have element with the given index (4 in this case). If not, a run-time error *IndexOutOfBoundsException* occurs.

To declare integer array of length 5:

```
int[] row = new int[5];
```

The 5 elements of this array will each be given a default value. Default value for numbers is 0, default value for objects is null, and so on.

Declaration with initial values for elements:

```
int[] row = {4, 9, 11, 0, 26};
```

This time an array of length 5 will be created, with initial values shown.

Can just create variable *row* on its own. E.g.

```
int[] row;
```

This says *row* will hold references to *int* arrays.

Initialisation of string arrays:

```
String[] animals = { "cat", "dog", "fox" };
```

Simple Array Application

Application reads in rainfall figures (in millimetres) for each month of year, then displays data as a histogram.

Here is output from a run. In histogram, rainfall for each month is represented by length of line of Xs for that month. Each X represents two millimetres of rain.

```
Enter rainfall data.
Jan: 55
Feb: 41
Mar: 36
Apr: 35
May: 47
Jun: 45
Jul: 60
Aug: 62
Sep: 50
Oct: 57
```



```
Nov: 65
Dec: 48

Jan  XXXXXXXXXXXXXXXXXXXXXXXXXXX
Feb  XXXXXXXXXXXXXXXXXXXX
Mar  XXXXXXXXXXXXXXXXXX
Apr  XXXXXXXXXXXXXXXXX
May  XXXXXXXXXXXXXXXXXXXXXXX
Jun  XXXXXXXXXXXXXXXXXXXXXX
Jul  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Aug  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Sep  XXXXXXXXXXXXXXXXXXXXXXXXX
Oct  XXXXXXXXXXXXXXXXXXXXXXXXXXXX
Nov  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Dec  XXXXXXXXXXXXXXXXXXXXXXXX
     0    10   20   30   40   50   60   70   80   90
```

Array Application

Program reads data typed by user and stores it in array called *rainData* which has an element for each month, e.g. *raindata*[0] holds rainfall for January, *rainData*[1] for February, etc. Following statement creates the rainData array:

```
int[] rainData = new int[12];
```

A `for` statement is used to read each figure and store it in appropriate element:

```
for (int i = 0; i < rainData.length; i++)
{  Read next number.
   Store the number in rainData[i].
}
```


Before it reads data for a month, program prompts user with name of the month. Names stored in array called *month*:

```
String[] month =
   { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
     "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };
```

Here is complete loop that reads rainfall figures. Input is obtained from a *ConsoleReader* object, *input*:

```
System.out.println("Enter rainfall data.");
for (int i = 0; i < rainData.length; i++)
{  System.out.print(month[i] + ": ");
   rainData[i] = input.readInt();
}
```

Once data read, 2nd `for` statement displays array values:

```
for (int i = 0; i < rainData.length; i++)
{  Output the name month[i].
   Output a line of Xs that represents
   the value rainData[i].
}
```

Following statement outputs a line of num Xs:

```
for (int n = 0; n < num; n++)
{ System.out.print("X"); }
```

In this program number of Xs output is $rainData[i]/2$:

```
for (int i = 0; i < rainData.length; i++)
{  System.out.print(month[i] + "  ");
   for (int n = 0; n < rainData[i]/2; n++)
   { System.out.print("X"); }
   System.out.println();
}
```


EXAMPLE 9.1

```
public static void main(String[] args)
{  String[] month =
      { "Jan", "Feb", "Mar", "Apr", "May", "Jun",
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };
   ConsoleReader input =
      new ConsoleReader(System.in);

   /* Read rainfall figures. */
   int[] rainData = new int[12];
   System.out.println("Enter rainfall data.");
   for (int i = 0; i < rainData.length; i++)
   {  System.out.print(month[i] + ": ");
      rainData[i] = input.readInt();
   }
```

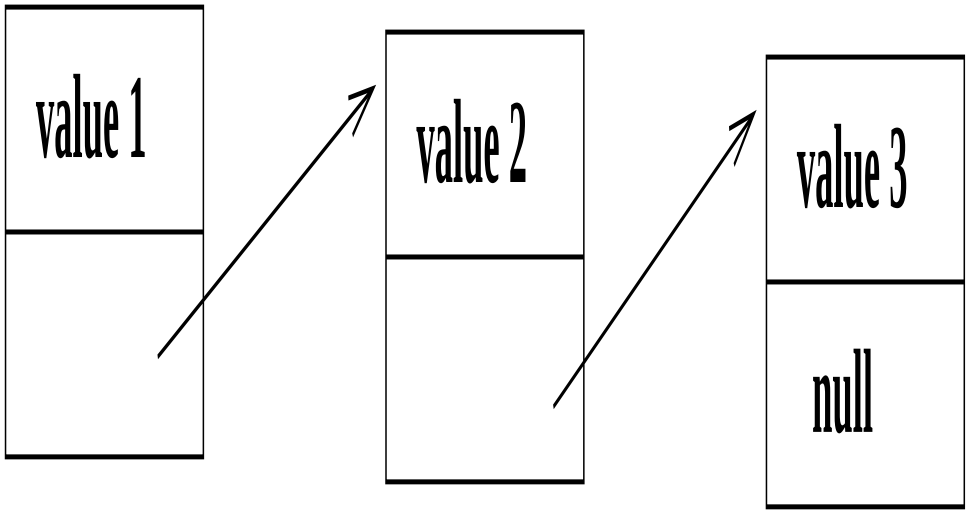
```
   /* Display histogram of rainfall data. */
   System.out.println();
   for (int i = 0; i < rainData.length; i++)
   {  System.out.print(month[i] + "  ");
      for (int n = 0; n < rainData[i]/2; n++)
      { System.out.print("X"); }
      System.out.println();
   }
   /* Print scale. */
   System.out.print("     ");
   for (int p = 0; p < 100; p = p+10)
   { System.out.print(p + "   "); }
   System.out.println();
}
```


The List Interface

Linked lists are an alternative way of storing a long sequence of values such as you might store in an array. Can define a class called *DataList*, say, which has a field containing a data value, and another field containing reference to another *DataList* object. This second field can be used to link together a number of these objects in a long chain.

For some situations linked lists more suitable than arrays. This is the case when elements frequently added to, or removed from interior of a list. But if we want to access value quickly using their index numbers, then arrays will be much more efficient.

Linked List versus Array Storage



List Interface in Java

Java provides an interface called *List* which describes set of methods for handling sequences of values independent of implementation (array or linked list) of the sequences.

Can declare your variables that refer to sequences of values as being of type *List*, then use any of wide range of methods in *List* interface.

Only time you have to specify the implementation that is used for a particular List is when calling a constructor to create the object.

List Interface Methods

- $l.add(v)$ – Add value v to end of List l. v can be any object (but not a primitive value such as an *int*). New values can be added as often as you like. You do not have to check if there is enough space for them.
- $l.get(i)$ – Return value stored in element i of List l. Elements of Lists are numbered 0, 1, ... The value returned will be of type Object. $0 \leq i < l.size()$ required.
- $l.size()$ – Return number of values stored in List l.

For an array-based implementation of *List* use constructor *ArrayList*(). For linked-list based implementation use constructor *LinkedList*().

Using List Interface

- To store numbers in lists use 'wrapper' classes: *Integer* for *int*s, *Double* for doubles. To turn a double value d into a Double object that just contains d, use:

  ```
  Double x = new Double(d);
  ```

  x can be stored in a list. To extract its double value use expression

  ```
  x.doubleValue()
  ```

- The *get* method has return type *Object*. So compiler only knows that values extracted from a list are *Object*s. If you know value is a *Point2D* object, and you are going to apply methods to it that can only be applied to *Point2D* objects (e.g. getX()), then must tell compiler this, by applying (Point2D) cast to it. E.g.

  ```
  Point2D pp = (Point2D) points.get(i);
  ```

- *List* interface is defined in `java.util` package, but has same name as GUI List class defined in `java.awt`. So if you do:

  `import java.awt.*;`

  must use full name

  `java.util.List`

  for list data structure.

EXAMPLE 9.2

```
/** User database program. */

public class CSDatabase
{ public static void main(String[] args)
  { Curator users = new Curator();
    ConsoleReader input =
       new ConsoleReader(System.in);
    System.out.println();
    System.out.println
       ("Welcome to the CS Department Database.");
    while (true)
    {  /* Get user to select an option. */
      System.out.println();
      System.out.print("Action (lookup,add,delete,quit): ");
      String option = input.readLine();
```

```
if (option.equalsIgnoreCase("lookup"))
{ /* Lookup user. */
  System.out.print("Username: ");
  String userName = input.readLine();
  User u = users.getUser(userName);

  if (u == null)
  { System.out.println("There is noone with " +
                       "that username.");
  }
  else
  { System.out.println();
    u.printDetails();
  }
}
else if (option.equalsIgnoreCase("add"))
{ /* Add user. */
  System.out.println("Enter details of user.");
```



```
  User u;
  System.out.print("Username: ");
  String userName = input.readLine();
  System.out.print("Name: ");
  String name = input.readLine();
  System.out.print("Is user a student? (y/n): ");
  if (input.readLine().equals("y"))
  { /* Read student details. */
    System.out.print("Student ID number: ");
    String idNumber = input.readLine();
    System.out.print("Degree programme: ");
    String degree = input.readLine();
    System.out.print("Year: ");
    int year = input.readInt();
    u = new Student(userName, name, idNumber,
                    degree, year);
  }
  else
```

```
        { /* Read staff details. */
          System.out.print("Room number: ");
          String room = input.readLine();
          System.out.print("Extension: ");
          String phone = input.readLine();
          u = new Staff(userName, name, room, phone);
        }
        users.addUser(u);
        System.out.println("User added.");
      }
      else if (option.equalsIgnoreCase("delete"))
      { /* Delete user. */
        System.out.print("Username: ");
        String userName = input.readLine();
        if (users.deleteUser(userName))
        { System.out.println("User deleted."); }
        else
        { System.out.println("There is noone with username " +
```



```
                              userName + ".");
        }
      }
      else if (option.equalsIgnoreCase("quit"))
      { /* Finish. */
        System.out.println("Bye");
        return;
      }
      else
      { System.out.println("Please choose an action."); }
    }
  }
}
```

```
/** A Curator object manages a list of users. */

public class Curator
{  private User[] users = new User[100];
   //  All users in the database.
   private int usersSize = 0;
   //  Number of users in the 'users' array.

   /** Return user with username un.
       If no user with this username, return null.
   */
   public User getUser(String un)
   {  for (int i = 0; i < usersSize; i++)
      { if (users[i].getUserName().equals(un))
        { return users[i]; }
      }
      return null;
   }
```



```
   /** Add the User object u to the list of users.
   */
   public void addUser(User u)
   {  /* If array is full, increase its size. */
      if (usersSize >= users.length)
      {  int len = users.length;
         User[] newArray = new User[len+50];
         for (int i = 0; i < len; i++)
         { newArray[i] = users[i]; }
         users = newArray;
      }
      /* Store new user in users array. */
      users[usersSize] = u;
      usersSize++;
   }
```

```
   /** Delete from the list of users
       the user with username un, and return true.
       If username not found, return false.
   */
   public boolean deleteUser(String un)
   {  for (int i = 0; i < usersSize; i++)
      {  if (users[i].getUserName().equals(un))
         {  users[i] = users[usersSize - 1];
            usersSize--;
            return true;
         }
      }
      return false;
   }
}
```


List version of CSDatabase

Can replace array by list. *getUser* is similar, but *addUser* is much simpler than array version, since no need to extend array. *deleteUser* can use *List remove* method:

```
public void addUser(User u)
{ users.add(u); }

public boolean deleteUser(String un)
{ User user = getUser(un);
  if (user != null)  /* User was in database */
  { users.remove(user); /* removes all occurrences */
    return true;
  }
  return false;
}
```

Modified Curator class

```
import java.util.ArrayList;
import java.util.List;

public class Curator2
{  private List users = new ArrayList();
   //  All users in the database.

   /** Return user with username un.
       If no user with this username, return null. */
   public User getUser(String un)
   {  for (int i = 0; i < users.size(); i++)
      { User user = (User) users.get(i);
        if (user.getUserName().equals(un))
        { return user; }
      }
      return null;
```



```
   }

   /** Add the User object u to the list of users. */
   public void addUser(User u)
   { users.add(u); }

   /** Delete from the list of users
       the user with username un, and return true.
       If username not found, return false. */
   public boolean deleteUser(String un)
   { User user = getUser(un);
     if (user != null)  /* User was in database */
     { users.remove(user); /* removes all occurrences */
       return true;
     }
     return false;
   }
}
```

No change to *CSDatabase* except use *Curator*2 in *CSDatabase*2:

```
public class CSDatabase2
{ public static void main(String[] args)
  { Curator2 users = new Curator2();
  ...
  }
}
```

Notice that can add several entries with *same* username! Probably not what was intended. Could add check in *add* to see if already such a user, or use *Map* interface instead of array or List.

Arrays versus Lists

Property	*array*	*List*
Extensible	Manually	Automatically
Element type	Any type	Not primitive types
Getting elements	$arr[ind]$	$(Type)\ l.get(ind)$
Insertion	Complex	Library method *insert*
Deletion	Complex	Library method *remove*

Use arrays if data size is fixed and element type is primitive, otherwise use list.

The Map Interface

List interface is one of a number of Java interfaces that represent collections of data. They are known as *Collections*. For databases the *Map* interface is often more suitable than List.

Objects which implement Map interface behave like mappings from one set of values, called *keys*, to another set called *values*. Unlike mappings in mathematics, a map can only be applied to a finite number of keys.

$$keys \rightarrow values$$

Can think of a map as a lookup table. Given a key, map tells you the value the key maps to/identifies.

Some *Map* methods:

- $m.put(k, v)$ – Extend map m by adding mapping of key k to value v.
- $m.get(k)$ – Return value that m maps key k to. If no entry for k in m, return *null*.
- $m.remove(k)$ – Remove mapping for k. Returns value that m maps k onto. If m does not map k onto anything, returns *null*.

Using Map

Two standard implementations of *Map* interface, *HashMap* and *TreeMap*. *HashMap* is good choice if not interested in a particular ordering on keys.

Curator is almost a map, where usernames are keys, and User objects as corresponding values. It provides:

- Method that adds key and its value (*addUser*).
- Method that looks up value for a given key (*getUser*).
- Method that removes a key and its value (*deleteUser*).

So can forget *Curator* object completely, and replace it with a *HashMap* object. *addUser* method can be replaced by *put*, *getUser* can be replaced by *get*, and *deleteUser* by *remove* (with an adjustment to the code).

Map version of CS Database

```
import java.util.*;

public class CSDatabase3
{ public static void main(String[] args)
  { Map users = new HashMap();
    ConsoleReader input =
       new ConsoleReader(System.in);

    System.out.println();
    System.out.println
       ("Welcome to the CS Department Database.");

    while (true)
    {  /* Get user to select an option. */
       System.out.println();
       System.out.print
```

```
      ("Action (lookup,add,delete,quit): ");
    String option = input.readLine();

    if (option.equalsIgnoreCase("lookup"))
    {  /* Lookup user. */
       System.out.print("Username: ");
       String userName = input.readLine();
       User u = (User) users.get(userName);
       if (u == null)
       { System.out.println("There is noone with " +
                            "that username."); }
       else
       {  System.out.println();
          u.printDetails();
       }
    }
    else if (option.equalsIgnoreCase("add"))
    {  /* Add user. */
```



```
       System.out.println("Enter details of user.");
       User u;
       System.out.print("Username: ");
       String userName = input.readLine();
       System.out.print("Name: ");
       String name = input.readLine();
       System.out.print("Is user a student? (y/n): ");
       if (input.readLine().equals("y"))
       {  /* Read student details. */
          System.out.print("Student ID number: ");
          String idNumber = input.readLine();
          System.out.print("Degree programme: ");
          String degree = input.readLine();
          System.out.print("Year: ");
          int year = input.readInt();
          u = new Student(userName, name, idNumber,
                          degree, year);
       }
```

```
          else
          {  /* Read staff details. */
             System.out.print("Room number: ");
             String room = input.readLine();
             System.out.print("Extension: ");
             String phone = input.readLine();
             u = new Staff(userName, name, room, phone);
          }
          users.put(userName,u);
          System.out.println("User added.");
       }
       else if (option.equalsIgnoreCase("delete"))
       {  /* Delete user. */
          System.out.print("Username: ");
          String userName = input.readLine();
          if (users.remove(userName) != null)
          { System.out.println("User deleted."); }
          else
```



```
          { System.out.println("There is noone with username " +
                                 userName + ".");
          }
       }
       else if (option.equalsIgnoreCase("quit"))
       {  /* Finish. */
          System.out.println("Bye");
          return;
       }
       else
       { System.out.println("Please choose an action."); }
    }
  }
}
```

Vectors

Vector class also implements *List* interface, so *add*, *contains*, *remove*, *get*, *size* can be used on *Vector* objects:

```
Vector v = new Vector();

v.add("rex");
v.add("fred");
v.add("jim");
v.add("vijay");

if (v.contains("fred"))
{ v.remove("rex"); }

System.out.println(v.get(2)); /* who? */
```


Iterating through a Vector

Typical loop is:

```
for (int i = 0; i < v.size(); i++)
{ Type x = (Type) v.get(i);
  /* do something with x */
}
```

All elements of v must be objects of class *Type* (or a subclass of *Type*).

This is an example of a *programming idiom* in Java.

Vectors very similar to *ArrayList*: more time-efficient for *get* and *set* than *LinkedList*, less time-efficient for *add* and less space-efficient.

Coursework 6. Hand-in: Friday 13th December, 4pm. Coursework office

1 Redefine the class *YearMarks* from coursework 3 to use a single integer array variable of size 8, in place of the 8 individual variables. Initialise this variable using an array initialiser in its declaration, and modify the method *calculateAverage*() to use this array.

2 Define a method *public int numberPassed*() of *YearMarks* which uses a loop to count how many courses the student has passed, and returns this count. A course is passed if the marks for it are 40 percent or greater, so 4 courses are passed using the example data.

Coursework Received

Get the coursework office to stamp/sign each coursework receipt when you hand it in to confirm it's been handed in.

Coursework 1: 4.10.2002

Coursework 2: 18.10.2002

Coursework 3: 1.11.2002

Coursework 4: 15.11.2002

Coursework 5: 29.11.2002

Mini Group Project: 6.12.2002

Coursework 6: 13.12.2002