

Structures in Java

- In this Section:
 - Organizing data - introduction
 - "Records" (in other languages, but not an official Java term!)

Introduction

The aim of this part of the module is to deal with data and structure issues in Java. This includes:

- **Data Structures and Records:** collecting related information, using classes as records
- **Arrays:** declaration, initialisation, access, searching
- **Strings:** declaration, initialisation, access, string manipulation
- **Files:** opening, closing, reading, writing, other files

Records: Motivation

- Simple data types are used for things like numbers, characters and booleans.
- These are sufficient and common for isolated pieces of information.
- However it is often desirable to group related information items, keeping them together under one name.
- This gives rise to **data structures**, which are collections of information in a well-defined relationship.

- If the information items are *dissimilar* but *naturally go together*.
 - They are grouped in what is traditionally called a **record** with a number of **fields**
- Examples:
 - the personal details of an employee: name, address, date of birth, position, salary, etc.
 - the academic details of a student: name, date of enrolment, degree, course modules, grades, etc.
 - the commercial details of a product: code, description, manufacturing cost, selling price, etc.

- If the information items are *identical in type*, they may be grouped in what is called an **array**:
 - heights for children aged 0, 1, 2, etc.
 - number of days for each calendar month
 - the date of Easter for each calendar year
- These are all examples of data structures; there are many others
- In **data-driven design**, the focus is on the information that a program holds and how it can be manipulated

Classes and Records (JFS, Chap 9)

- **Classes** support the idea of **encapsulation**
 - The information they hold is closely related to operations on this information
- Classes also support the idea of **information hiding**
 - What information is held and how it is stored can be hidden behind public methods, making future changes easier
- Records are a common and simple case so they will be discussed first
- A class may have **instance variables** and **methods**
- It is normally desirable to make instance variables **private** so that other classes do not know how information is stored (or even what information is held)

- Public methods are then provided to operate on data held by the class, typically **set...** and **get...** methods
- A telephone directory entry, for example, might be defined by:

```
// directory entry as record
public class Entry {
    private String name;
    private int number;
    private boolean unlisted; // Ex-directory?
    set...    get...
}
```

- A directory entry might be created and set up as follows:

```
subscriber = new Entry();
subscriber.setName("Jo Smith");
subscriber.setNumber(467420);
subscriber.setUnlisted(true);
```

- It is inconvenient to set up the fields individually, so an explicit constructor method would be useful as well as field methods
 - See next slide

Full directory entry class

```
// directory entry as record
public class Entry {
    private String name;
    private int number;
    private boolean unlisted;    // Ex-directory?
    // Constructor
    public Entry(String person, int phone,
                  boolean exdir) {

        name = person;
        number = phone;
        unlisted = exdir;
    }
}
```

```
public void setName(String person) {
    name = person;
}
public String getName() {
    return name;
}
public void setNumber(int phone) {
    number = phone;
}
public int getNumber() {
    return number;
}
}
```

```
public void setUnlisted(boolean exdir) {  
    unlisted = exdir;  
}  
public boolean isUnlisted() {  
    return unlisted;  
}  
} \\ End of Entry class
```

Basic use of Entry

```
Entry subscriber =  
    new Entry("Jo Smith", 467420, true);  
  
String information =  
    subscriber.getName()  
    + " is " + subscriber.getNumber();
```

Example directory application - actionPerformed

```

public void actionPerformed(ActionEvent event) {
    String name, phone;
    if (event.getSource() == enterButton) { // new entry?
        name = nameText.getText();           // get name
        phone = phoneText.getText();          // get phone
        directoryEntry = new Entry(name, phone); // create
    }
    else if (event.getSource() == retrieveButton) {
        name = nameText.getText();           // get name
        if (name.equalsIgnoreCase(directoryEntry.getName())) {
            // name matches?
            phone = directoryEntry.getPhone(); // get phone
            phoneText.setText(phone);          // display
        }
        else
            phoneText.setText("???");         // name not found
            // display unknown
    }
}

```

ITNP001 Records
© University of Stirling 2015

13

Better: We should *only* allow retrieval of directory entries that are *not "unlisted"*:

```

if (name.equalsIgnoreCase(directoryEntry.getName())
    && !directoryEntry.isUnlisted()) { // name matches?
    phone = directoryEntry.getPhone(); // get phone
    phoneText.setText(phone);          // display
}
else
    phoneText.setText("???");           // name not found
    // display unknown
}
}

```

ITNP001 Records
© University of Stirling 2015

14

Record Example



Demo

End of section