Object-oriented Software Design and Development
CCP114N

Week 4:

The Object - Oriented Software Design Process (cont.)

- Case study: a simple arithmetic calculator

Java Remedy

- Array, Linked List and Queue in Java

Last week

- Software Analysis-Design-Implementation
- What are Objects and Classes
  - How to identify them?
    - Identity
    - Behaviour - Responsibilities
    - States
- Use Cases: Scenarios of interactions between (external) actors (the user) and the application (the software system)
- UML Diagrams: Class diagram, Sequence diagram & State diagram
- Using **javadoc** for Design Documentation

Case Study: a simple arithmetic calculator (1/2)

- Consider the development of a simple application that simulates a handheld electronic calculator which can perform 4 operations: addition, subtraction, multiplication and division on any two integers, which are entered by the user via the calculator's keypad.

- To design the (software) system we go thru these steps
  - (1) Determine use cases
    - ❏ Incl. variations
  - (2) Identify class candidates
  - (3) Build CRC cards

  } (4) Use cases analysis blends UC & CRC to revise and refine them; relationships between the classes are revealed.

  - ❏ Assigning every use case action to a class/actor as responsibility
    - New classes may be needed (to play a role, such as Connection class)

Case Study (2/2)

- What are the outcomes from the design phase?

  (for examples also see the lecture notes for diagrams of the Voice Mail system)

  - Use cases and classes are refined, even re-defined
  - UML Class diagrams
  - Sequence diagram for every use cases
  - Any required state diagrams
  - Observations:
    - Use case system actions become service (methods) provided by classes, being represented as arrows in sequence diagrams
    - Class dependency is also represented by arrows in sequence diagrams
    - Association is represented by class attributes e.g. Connection->Mailbox,
    - Composition is represented by object collections in a class e.g. a Mailbox is composed of 2 MessageQueues that do not exit outside the Mailbox.

Java remedy

Array, Linked List, List and Queue in Java

Array

- Collection of data elements of the same data types with random access via an index
- A static data structure i.e. the array size cannot be changed at runtime

```
String [ ] FavouriteColour = {“Green”, “Blue”, “Grey”, “Orange”};
```

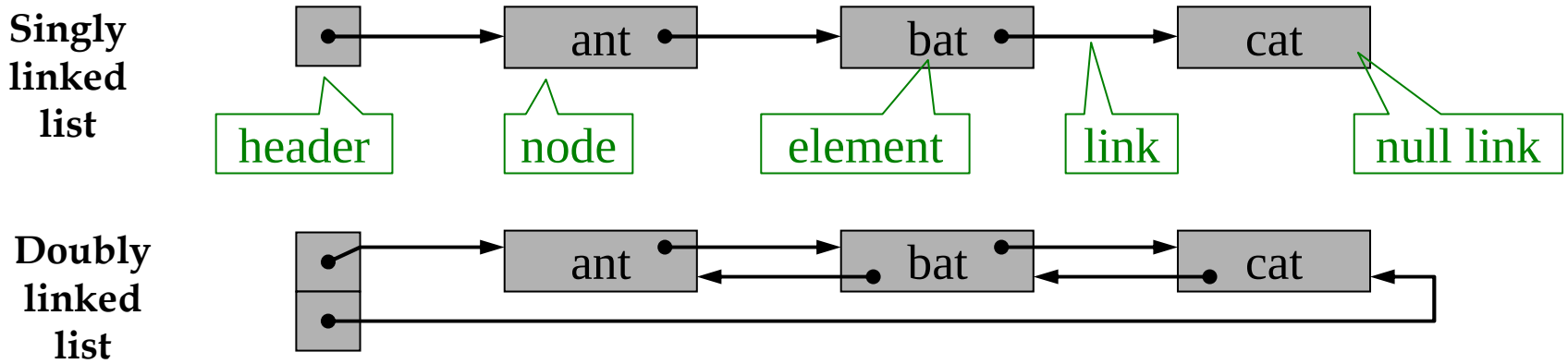
- In Java an array can be used to hold data of any data types (How?)

```
Object FavouriteThing [ ] = {“Green”, 14, 9.99}; //String, int, double
```

Java remedy

Array, Linked List, List and Queue in Java

- A dynamic data structure
- A **linked list** consists of a sequence of **nodes** connected by **links**, plus a **header**.
- Each node (except the last) has a **successor**, and each node (except the first) has a **predecessor**.
- Each node contains a single **element** (object or value), plus links to its successor and/or predecessor.



Java remedy

Array, Linked List, List and Queue in Java

- Do not confuse the **list** as an abstract data type (ADT) with the **linked-list** data structure.
  - The List ADT
    - A **list** is a sequence of elements, whose order is significant.
    - Elements can be added and removed anywhere in the list.
    - The **length** or **size** of a list is the number of elements it contains.
    - Operations: create, add, insert, remove, get, traversal, etc.
  - A list ADT can be implemented using different data structures (arrays, linked lists).
- Linked-list data structures can be used to implement many ADTs (e.g. lists, queues, stacks, sets)
  - Queue: FIFO first-in first out
  - Stack: LIFO last-in first-out

Java remedy

Array, Linked List, List and Queue in Java

- Lists in the Java Collections Framework
- The Java *interface* **`java.util.List`** specifies a generic List ADT.
  - Implemented as
    - ArrayList
    - Vector
    - LinkedList
  - For details of the List interface consult the Java document and/or the Sun Java Tutorial.
- The Java class **`java.util.ArrayList`** implements the `java.util.List` interface, representing each list by an array.
  - called smart list as its size will be doubled as the current size being reached, so its more flexible than normal arrays
- The Java class **`java.util.LinkedList`** implements the `java.util.List` interface, representing each list by a linked list
- Since Java 1.5: *interface* **`java.util.Queue`**

Summary

- A case study of an OO design
- Java remedy
  - Array
  - Linked List
  - List & Queue