

## Topic 2

# Collections

08/28/11

2-1

## Objectives

- Define the concepts and terminology related to collections
- Discuss the abstract design of collections

08/28/11

2-2

2-2

## Collections

**Collection** : a group of items that we wish to treat as a conceptual unit

- **Example**: a stamp collection
- In computer science, we have collections of items also
  - **Examples**: stack, queue, list, tree, graph
- *The proper choice of a collection for a given problem can affect the efficiency of a solution!*

08/28/11

2-3

2-3

## Examples of Collections

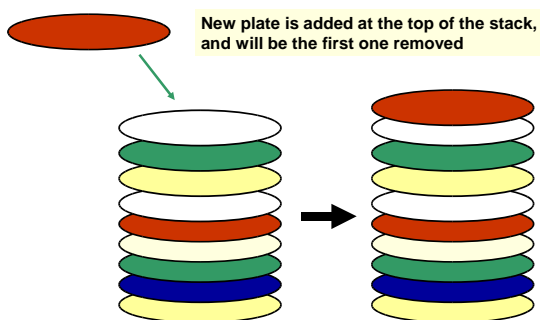
- What do collections look like?
  - **Queue** : first item in is first item out
    - e.g. a lineup at a checkout counter
  - **Stack** : last item in is first item out
    - e.g. a stack of plates in the cafeteria
  - **List** : we can have ordered lists or unordered lists
    - e.g. a shopping list; a list of names and phone numbers; a to-do list

08/28/11

2-4

2-4

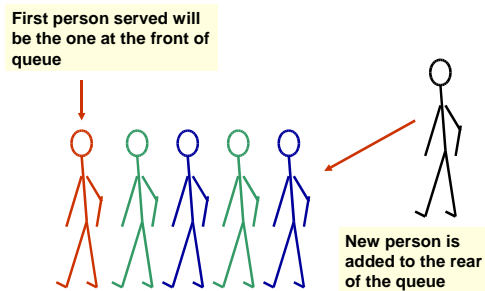
### Example: stack of plates



08/28/11

2-5

### Example: queue at checkout

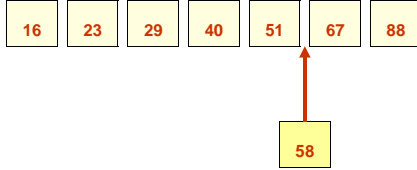


08/28/11

2-6

## Example: ordered list of numbers

New number must be added so that the ordering of the list is maintained



08/28/11

2-7

## Examples of Collections

- The previous examples are **linear collections**: items are organized in a “straight line”
- Each item except the first has a unique **predecessor**, and each item except the last has a unique **successor** within the collection

08/28/11

2-8

2-8

## Examples of Collections

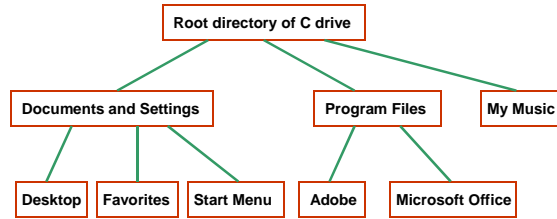
- We also have **nonlinear collections**
- Hierarchical collections: trees**
  - items are ordered in an “upside down tree”
  - Each item except the one at the top has a unique predecessor but may have many successors
  - Examples**: taxonomies, computer file systems

08/28/11

2-9

2-9

## Example of a tree: computer file system



08/28/11

2-10

## Examples of Collections

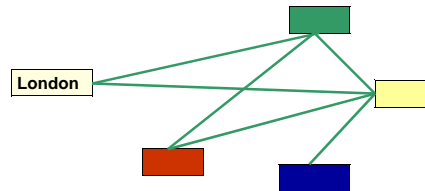
- Another nonlinear collection is a **graph**: items can have many predecessors and many successors
- Example**: maps of airline routes between cities

08/28/11

2-11

2-11

## Example of a graph: airline routes between cities



08/28/11

2-12

2-12

## Abstraction

- In solving problems in computer science, an important design principle is the notion of **abstraction**
- Abstraction separates the **purpose** of an entity from its **implementation**
  - **Example in real life**: a car (we do not have to know how an engine works in order to drive a car)
  - **Examples in computer systems**: a computer, a file
  - **Example in Java**: class, object

08/28/11

2-13

2-13

## Abstraction

- Abstraction provides a way of dealing with the complexity of a **large** system
- **We will deal with each collection in a general way, as a data abstraction**
- We will think of **what** we want to do with the collection, **independently of how** it is done
  - For example, we may want to add something to a queue, or to remove it from the queue

08/28/11

2-14

2-14

## Collection as an Abstraction

- Example: think of a queue of customers
  - Suppose **what** we want to do is to deal with the first customer in the queue, i.e. *dequeue* a customer
  - **How** is this dequeue done?
    - We may not need to know, if someone else looked after the details
    - Or, if we are involved in the "how"
      - We may choose to program in Java or some other language
      - There may be several ways of implementing a queue that differ in efficiency

08/28/11

2-15

2-15

## Collection as an Abstraction

- In other words, we want to *separate*
  - The **interface** of the collection: **what** we need in order to interact with it, i.e. the operations on the collection
    - This is from the perspective of a **user** of the collection
  - The **implementation** of the collection: the underlying details of **how** it is coded
    - This is from the perspective of a **writer** of the collection code

08/28/11

2-16

2-16

## Issues with Collections

- For each collection that we examine, we will consider:
  - How does the collection operate conceptually?
  - How do we formally define its interface?
  - What kinds of problems does it help us solve?
  - In what ways might we implement it?
  - What are the benefits and costs of each implementation?

08/28/11

2-17

2-17

## Abstract Data Types (ADTs)

- **Data type**: a set of values and the operations defined on those values
  - Example: integer data type (**int**)
    - Values? operations?
- **Abstract data type**: a *collection of data* together with the *set of operations* on that data
  - Why **abstract**? It's a data type whose values and operations are **not** inherently defined in a programming language
  - **Examples**: stack, queue, list, tree

08/28/11

2-18

2-18

## Data Structures

- **Data structure:** a construct within a programming language, used to implement a collection
  - **Example:** array, C++ structure
- So, what is the difference between the terms “**abstract data type**” and “**data structure**”?
  - (Note that sometimes the terms are used interchangeably, in generalizations about “data structures”)