Collections

- A collection is an object that helps us organize and manage other objects.
- We will explore:
  - the concept of a collection
  - separating the interface from the implementation
  - dynamic data structures
  - linked lists
  - queues and stacks
  - trees and graphs
  - generics

Abstract Data Types

- An abstract data type (ADT) is an organized collection of information and a set of operations used to manage that information.
- The set of operations defines the interface to the ADT.
- In one sense, as long as the ADT fulfills the promises of the interface, it doesn’t matter how the ADT is implemented.
- Objects are a perfect programming mechanism to create ADTs because their internal details are encapsulated.

Dynamic Structures

- A static data structure has a fixed size.
- This meaning is different from the meaning of the static modifier.
- Arrays are static; once you define the number of elements it can hold, the size doesn’t change.
- A dynamic data structure grows and shrinks at execution time as required by its contents.
- A dynamic data structure is implemented using links.
Object References

- Recall that an object reference is a variable that stores the address of an object.
- A reference also can be called a pointer.
- References often are depicted graphically:

References as Links

- Object references can be used to create links between objects.
- Suppose a Student class contains a reference to another Student object.

Intermediate Nodes

- The objects being stored should not be concerned with the details of the data structure in which they may be stored.
- For example, the Student class should not have to store a link to the next Student object in the list.
- Instead, we can use a separate node class with two parts: 1) a reference to an independent object and 2) a link to the next node in the list.
- The internal representation becomes a linked list of nodes.

Magazine Collection

- Let’s explore an example of a collection of Magazine objects, managed by the MagazineList class, which has an private inner class called MagazineNode.
- Because the MagazineNode is private to MagazineList, the MagazineList methods can directly access MagazineNode data without violating encapsulation.
- See Magazine example source code on Schedule.

Other Dynamic Representations

- It may be convenient to implement as list as a doubly linked list, with next and previous references.
### Other Dynamic Representations
- It may be convenient to use a separate header node, with a count and references to both the front and rear of the list.

![Diagram of a list with a header node](image)

### Classic Data Structures
- Now we'll examine some classic data structures.
- **Classic linear data structures** include queues and stacks.
- **Classic nonlinear data structures** include trees and graphs.

### Queues
- **A queue** is similar to a list but adds items only to the rear of the list and removes them only from the front.
- It is called a FIFO data structure: First-In, First-Out.
- **Analogy:** a line of people at a bank teller’s window.

![Diagram of a queue with enqueue and dequeue operations](image)

- We can define the operations for a queue:
  - enqueue - add an item to the rear of the queue
  - dequeue (or serve) - remove an item from the front of the queue
  - empty - returns true if the queue is empty
- As with our linked list example, by storing generic Object references, any object can be stored in the queue.
- Queues often are helpful in simulations or any situation in which items get "backed up" while awaiting processing.

### Stacks
- **A stack ADT** is also linear, like a list or a queue.
- Items are added and removed from only one end of a stack.
- It is therefore LIFO: Last-In, First-Out.
- **Analogy:** a stack of plates in a cupboard, a stack of bills to be paid, or a stack of hay bales in a barn.

### Queues
- A queue can be represented by a singly-linked list; it is most efficient if the references point from the front toward the rear of the queue.
- A queue can be represented by an array, using the remainder operator (%) to “wrap around” when the end of the array is reached and space is available at the front of the array.
Stacks

- Stacks often are drawn vertically:

```
push                   pop
```

Stacks

- Some stack operations:
  - push - add an item to the top of the stack
  - pop - remove an item from the top of the stack
  - peek (or top) - retrieves the top item without removing it
  - empty - returns true if the stack is empty

- A stack can be represented by a singly-linked list; it doesn't matter whether the references point from the top toward the bottom or vice versa
- A stack can be represented by an array, but the new item should be placed in the next available place in the array rather than at the end

Stacks

- The `java.util` package contains a `Stack` class
- Like `ArrayList` operations, the `Stack` operations operate on `Object` references
- See Decode example on Schedule

Trees

- A tree is a non-linear data structure that consists of a root node and potentially many levels of additional nodes that form a hierarchy
- Nodes that have no children are called leaf nodes
- Nodes except for the root and leaf nodes are called internal nodes
- In a general tree, each node can have many child nodes

Binary Trees

- In a binary tree, each node can have no more than two child nodes
- A binary tree can be defined recursively. Either it is empty (the base case) or it consists of a root and two subtrees, each of which is a binary tree
- Trees are typically represented using references as dynamic links, though it is possible to use fixed representations like arrays
- For binary trees, this requires storing only two links per node to the left and right child

Graphs

- A graph is a non-linear structure
- Unlike a tree or binary tree, a graph does not have a root
- Any node in a graph can be connected to any other node by an edge
- Analogy: the highway system connecting cities on a map
Digraphs

- In a directed graph or digraph, each edge has a specific direction.
- Edges with direction sometimes are called arcs.
- Analogy: airline flights between airports.

Representing Graphs

- Both graphs and digraphs can be represented using dynamic links or using arrays.
- As always, the representation should facilitate the intended operations and make them convenient to implement.

Collection Classes

- The Java standard library contains several classes that represent collections, often referred to as the Java Collections API.
- Their underlying implementation is implied in the class names such as ArrayList and LinkedList.
- Several interfaces are used to define operations on the collections, such as List, Set, SortedSet, Map, and SortedMap.

Generics

- Java supports generic types, which are useful when defining collections.
- A class can be defined to operate on a generic data type which is specified when the class is instantiated:

  ```java
  LinkedList<Book> myList = new LinkedList<Book>();
  ```

- By specifying the type stored in a collection, only objects of that type can be added to it.
- Furthermore, when an object is removed, its type is already established.