

# Data Structures & Java Generics

---



**Nelson Padua-Perez**  
**Chau-Wen Tseng**

**Department of Computer Science**  
**University of Maryland, College Park**

## Algorithms and Data Structures

### ■ Algorithm

- Sequence of steps used to solve a problem
- Operates on **collection** of data
- Each **element** of collection  $\Rightarrow$  **data structure**

### ■ Data structure

- Combination of simple / composite data types
- Design  $\Rightarrow$  information stored for each element
- Choice affects characteristic & behavior of algorithm
- May severely impact **efficiency** of algorithm

## Data Structures

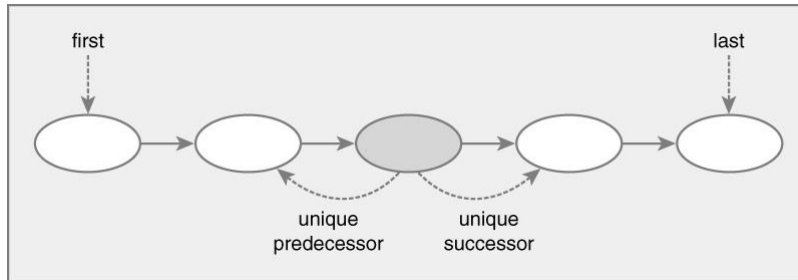
- **Taxonomy**
  - Classification scheme
  - Based on relationships between element
- **Category**                      **Relationship**
  - Linear                              one  $\Rightarrow$  one
  - Hierarchical                      one  $\Rightarrow$  many
  - Graph                                many  $\Rightarrow$  many
  - Set                                      none  $\Rightarrow$  none

## Data Structures

- **Core operations**
  - Add element
  - Remove element
  - Iterate through all elements
  - Compare elements

## Linear Data Structures

- One-to-one relationship between elements
  - Each element has **unique** predecessor
  - Each element has **unique** successor

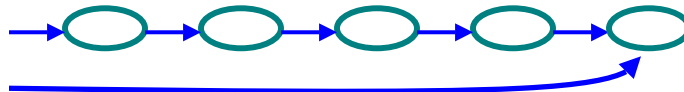


## Linear Data Structures

- Core operations
  - Find first element (head)
  - Find next element (successor)
  - Find last element (tail)
- Terminology
  - Head  $\Rightarrow$  no predecessor
  - Tail  $\Rightarrow$  no successor

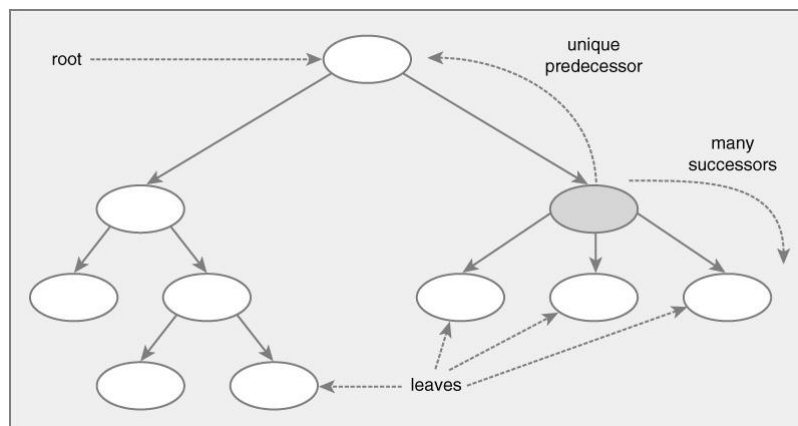
## Example Linear Data Structures

- **List**
  - Collection of elements in order
- **Queue**
  - Elements removed in order of insertion
  - First-in, First-out (FIFO)
- **Stack**
  - Elements removed in **opposite** order of insertion
  - First-in, Last-out (FILO)



## Hierarchical Data Structures

- **One-to-many relationship between elements**
  - Each element has **unique** predecessor
  - Each element has **multiple** successors



# Hierarchical Data Structures

## ■ Core operations

- Find first element (root)
- Find successor elements (children)
- Find predecessor element (parent)

## ■ Terminology

- Root  $\Rightarrow$  no predecessor
- Leaf  $\Rightarrow$  no successor
- Interior  $\Rightarrow$  non-leaf
- Children  $\Rightarrow$  successors
- Parent  $\Rightarrow$  predecessor

## Example Hierarchical Data Structures

### ■ Tree

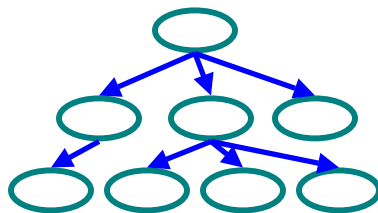
- Single root

### ■ Forest

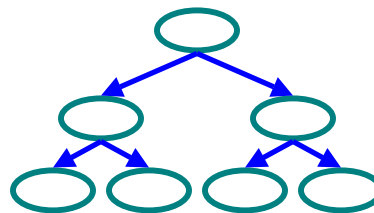
- Multiple roots

### ■ Binary tree

- Tree with 0–2 children per node



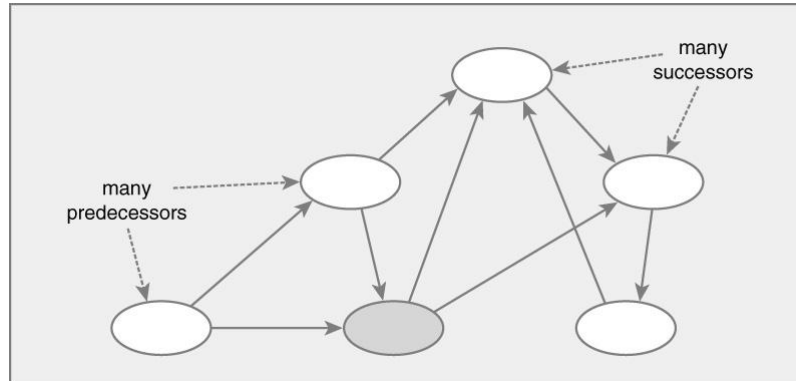
Tree



Binary Tree

## Graph Data Structures

- **Many-to-many relationship between elements**
  - Each element has **multiple** predecessors
  - Each element has **multiple** successors

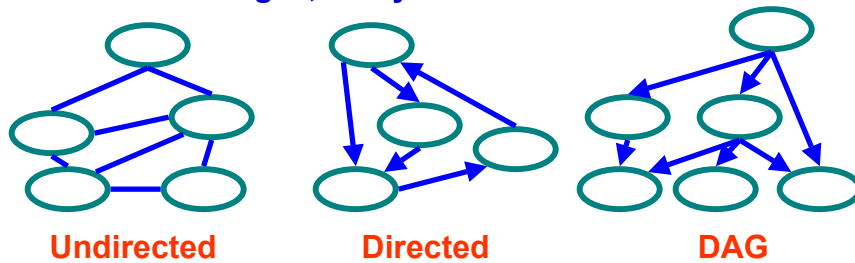


## Graph Data Structures

- **Core operations**
  - Find successor nodes
  - Find predecessor nodes
  - Find adjacent nodes (neighbors)
- **Terminology**
  - Directed  $\Rightarrow$  traverse edges in one direction
  - Undirected  $\Rightarrow$  traverse edges in both directions
  - Neighbor  $\Rightarrow$  adjacent node
  - Path  $\Rightarrow$  sequence of edges
  - Cycle  $\Rightarrow$  path returning to same node
  - Acyclic  $\Rightarrow$  no cycles

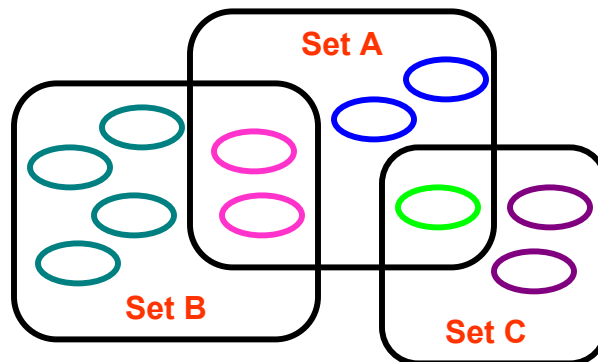
## Example Graph Data Structures

- Undirected graph
  - Undirected edges
- Directed graph
  - Directed edges
- Directed acyclic graph (DAG)
  - Directed edges, no cycles



## Set Data Structures

- No relationship between elements
  - Elements have **no** predecessor / successor
  - Only **one** copy of element allowed in set



# Set Data Structures

## ■ Core operations

- Set versions of core operations
- Add set, remove set, compare set

## ■ Terminology

- Subset  $\Rightarrow$  elements contained by set
- Union  $\Rightarrow$  select elements in either set
- Intersection  $\Rightarrow$  select elements in both sets
- Set difference  $\Rightarrow$  select elements in one set only

## Example Set Data Structures

### ■ Set

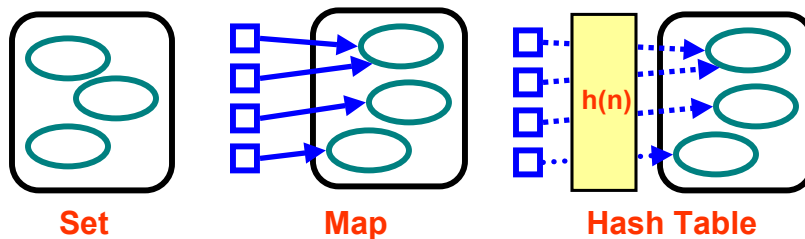
- Basic set

### ■ Map

- Map value to element in set

### ■ Hash Table

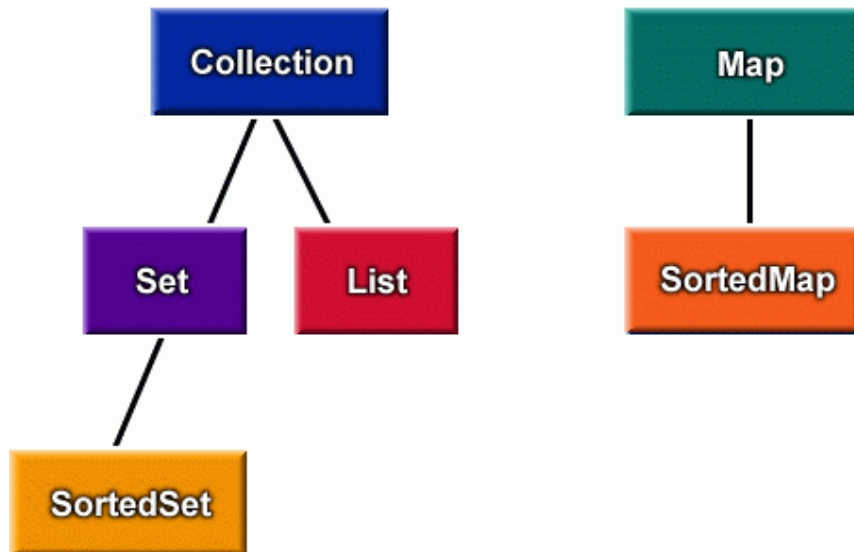
- Maps value to element in set using **hash** function



# Java Collections Framework

- **Collection**
  - Object that groups multiple **elements** into one unit
  - Also called container
- **Collection framework** consists of
  - **Interfaces**
    - Abstract data type
  - **Implementations**
    - Reusable data structures
  - **Algorithms**
    - Reusable functionality


## Core Collection Hierarchy



## Collections Interface Implementations

- **General implementations**
  - Primary public implementation
  - Example
    - List – ArrayList, LinkedList
    - Set – TreeSet, HashSet
    - Map – TreeMap, HashMap
- **Wrapper implementations**
  - Combined with other interfaces
  - Example
    - **synchronizedArrayList**, **unmodifiableHashMap**

## New Features in Java 1.5

- Enumerated types
- Enhanced for loop
- Autoboxing & unboxing
- Scanner
- Generic types 
- Variable number of arguments (varargs)
- Static imports
- Annotations

## Generics – Motivating Example

### ■ Problem

- Utility classes handle arguments as Objects
- Objects must be cast back to actual class
- Casting can only be checked at runtime

### ■ Example

```
class A { ... }  
class B { ... }  
List myL = new List();  
myL.add(new A()); // Add an object of type A  
...  
B b = (B) myL.get(0); // throws runtime exception  
// java.lang.ClassCastException
```

## Solution – Generic Types

### ■ Generic types

- Provides abstraction over types
- Can parameterize classes, interfaces, methods
- Parameters defined using `<x>` notation

### ■ Examples

- `public class foo<x, y, z> { ... }`
- `public class List<String> { ... }`

### ■ Improves

- Readability & robustness

### ■ Used in Java Collections Framework

## Generics – Usage

### ■ Using generic types

- Specify <type parameter> for utility class
- Automatically performs casts
- Can check class at compile time

### ■ Example

```
class A { ... }  
class B { ... }  
List<A> myL = new List<A>();  
myL.add(new A()); // Add an object of type A  
A a = myL.get(0); // myL element ⇒ class A  
...  
B b = (B) myL.get(0); // causes compile time error
```

## Generics – Issues

### ■ Generics and subtyping

- Even if class A extends class B
- List<A> does not extend List<B>

### ■ Example

```
class B { ... }  
class A extends B { ... } // A is subtype of B  
B b = new A(); // A used in place of B  
List<B> myL = new List<A>(); // compile time error  
// List<A> used in place of List<B>  
// List<A> is not subtype of List<B>
```