

Abstract Data Types

Algorithms and Data Structures

COMP3506 / 7505

Lecture 3 - Abstract Data Types

- Definition and purpose
- Specifying ADTs in Java
- Generics
- Factory Pattern
- Implementation
- ADT Commonalities

Definition

- An ADT is a formal model of a data structure that specifies:
 - the type of data stored
 - the operations supported
 - the types of parameters of the operations
- Specifies *what* each operation does, *not how* it does it

Purpose

- Behavioural contract with user
- Isolation
 - User code and ADT implementation can both be changed without affecting each other
- Encapsulation
 - Implementation details are hidden
- Flexibility
 - Alternative implementations are interchangeable
 - Optimized for specific purpose

ADTs in Java without Generics

- In Java an interface is used to specify abstract methods that an implementing class must implement
- Interfaces may extend other interfaces to add functionality
- Operations take Object class as parameter type of data stored

Stack ADT

```
1 package au.edu.uq.itee.comp3506.adt;
2
3 import java.util.NoSuchElementException;
4
5 public interface IStack {
6
7     public void    push( Object anObject );
8     public Object pop() throws NoSuchElementException;
9
10    public Object  top() throws NoSuchElementException;
11    public int     size();
12    public boolean isEmpty();
13
14 }
```

ADTs in Java without Generics (cont)

- Allows ADTs to be used regardless of derived class
- Element access requires casting at runtime
 - lose type safety

Using a Stack ADT

```
1 public class Example
2 {
3     public static void main( String[] args )
4     {
5         IStack stack = new StackImpl();
6         stack.push( "A String" );
7         stack.push( "Another String" );
8
9         String str = (String) stack.pop();
10
11     }
12 }
```

Java Example 6

```
1 package au.edu.uq.itee.comp3506.example6;
2
3 import java.util.*;
4 public interface IFixedStack
5 {
6     public void    push( Object anObject ) throws Exception;
7     public Object pop() throws NoSuchElementException;
8 }
```

Java Example 6

```
1 package au.edu.uq.itee.comp3506.example6
  ...
4 public class FixedStack implements IFixedStack {
5
6     public FixedStack( int aCapacity ) {
7         this.capacity = aCapacity;
8         this.stack = new Object[capacity];
9     }
10
11    public void push( Object anObject ) throws Exception {
12        if ( this.size == capacity ) throw new Exception();
13
14        this.stack[this.size] = anObject;
15        this.size++;
16    }
17
18    public Object pop() throws NoSuchElementException {
19        if ( 0 == this.size ) throw new NoSuchElementException();
20        this.size--;
21        return this.stack[this.size];
22    }
23
24    Object[]    stack;
25    private int size;
26    private int capacity; }
```

Java Example 6

```
1 import au.edu.uq.itee.comp3506.example6.Stack;
2
3 public class Example6a {
4
5     public static void main( String[] args )
6     {
7         IFixedStack stack = new FixedStack( 10 );
8
9         try
10        {
11            stack.push( new Integer( 1 ) );
12            stack.push( new String( "Second" ) );
13
14            String second = (String) stack.pop();
15            String first = (String) stack.pop(); // Will throw class cast exception
16
17            System.out.println( first + second );
18        }
19        catch ( Exception ex )
20        {
21            System.out.println( ex.getMessage() );
22        }
23    }
24 }
```

Generics

- In Java 5 (aka 1.5) Generics were added
- Instead of using Object class as type of data store, a parameterised type is specified
- A parameterised type is specified by user code when the ADT is used
 - i.e. the type of the class is passed in as a parameter

Generic Stack ADT

```
1 package au.edu.uq.itee.comp3506.adt;
2
3 import java.util.NoSuchElementException;
4
5 public interface IStack<E> {
6
7     public void      push( E anObject );
8     public E        pop() throws NoSuchElementException;
9
10    public E         top() throws NoSuchElementException;
11    public int       size();
12    public boolean   isEmpty();
13
14 }
```

Using a Generic Stack ADT

```
1 public class Example
2 {
3     public static void main( String[] args )
4     {
5         IStack stack = new StackImpl<String>();
6         stack.push( "A String" );
7         stack.push( "Another String" );
8
9         String str = stack.pop();
10    }
11 }
12
```

During compilation... treated as

```
1 package au.edu.uq.itee.comp3506.adt;
2
3 import java.util.NoSuchElementException;
4
5 public interface IStack {
6
7     public void      push( String anObject );
8     public String    pop() throws NoSuchElementException;
9
10    public String     top() throws NoSuchElementException;
11    public int        size();
12    public boolean    isEmpty();
13
14 }
```

Side-bar: type-erasure

- At compile time, `Stack<String>` is checked as though arguments were specified as `Strings`
- At run time, their types are lost; everything becomes an `Object`
- For example, this won't work:

```
T instantiateElementType(List<T> arg) {  
    return new T();  
}
```

Java Example 6

```
1 package au.edu.uq.itee.comp3506.example6;
2
3 import java.util.*;
4
5 public interface IGFixedStack<E> extends IFixedStack
6 {
7     public void gpush( E e ) throws Exception;
8     public E gpop() throws NoSuchElementException;
9 }
```

Java Example 6

```
1 package au.edu.uq.itee.comp3506.example6;
  ...
5 public class GFixedStack<E> implements IFixedStack<E> {
6
7     public GFixedStack( int aCapacity ) {
8         this.stack = new Stack( aCapacity );
9     }
10
11    public void push( Object anObject ) throws Exception {
12        this.stack.push( anObject );
13    }
14
15    public void gpush( E anObject ) throws Exception {
16        this.stack.push( anObject );
17    }
18
19    public Object pop() throws NoSuchElementException {
20        return this.stack.pop();
21    }
22
23    public E gpop() throws NoSuchElementException {
24        return (E) this.stack.pop();
25    }
26    IStack    stack;
27 }
```

Java Example 6

```
1 import au.edu.uq.itee.comp3506.example6.GStack;

5 public class Example6b {
6
7 public static void main( String[] args )
8 {
9     GFixedStack<String> stack = new GFixedStack<String>( 10 );
10
11     try {
12         stack.gpush( new Integer( 1 ) );           // Flagged by compiler as error
13         stack.gpush( new String( "Second" ) );
14
15         String first = stack.gpop();
16         String second = stack.gpop();
17
18         System.out.println( first + second );
19     }
20     catch ( Exception ex )
21     {
22         System.out.println( ex.getMessage() );
23     }
24 }
25 }
```

Factory Pattern

- The Factory Pattern (GoF) can be used to allow different ADT implementations to be instantiated at runtime

ADTFactory

```
1 package au.edu.uq.itee.comp3506.adt;  
2  
3 public interface ADTFactory<E>  
4 {  
5     public IStack<E>    createStack( String type );  
6  
7     public IVector<E>  createVector( String type );  
8  
9     public IQueue<E>   createQueue( String type );  
10 }
```

COMP3506ADTFactory

```
1 package au.edu.uq.itee.comp3506.adt.impl;
2
3 import au.edu.uq.itee.comp3506.adt.*;
4
5 public class COMP3506ADTFactory<E> implements ADTFactory<E>
6 {
7     public IStack<E> createStack( String type )
8     {
9         if ( type.equals( "LLStack" ) )
10        {
11            return new LLStack<E>();
12        }
13    }
14
... ..
```

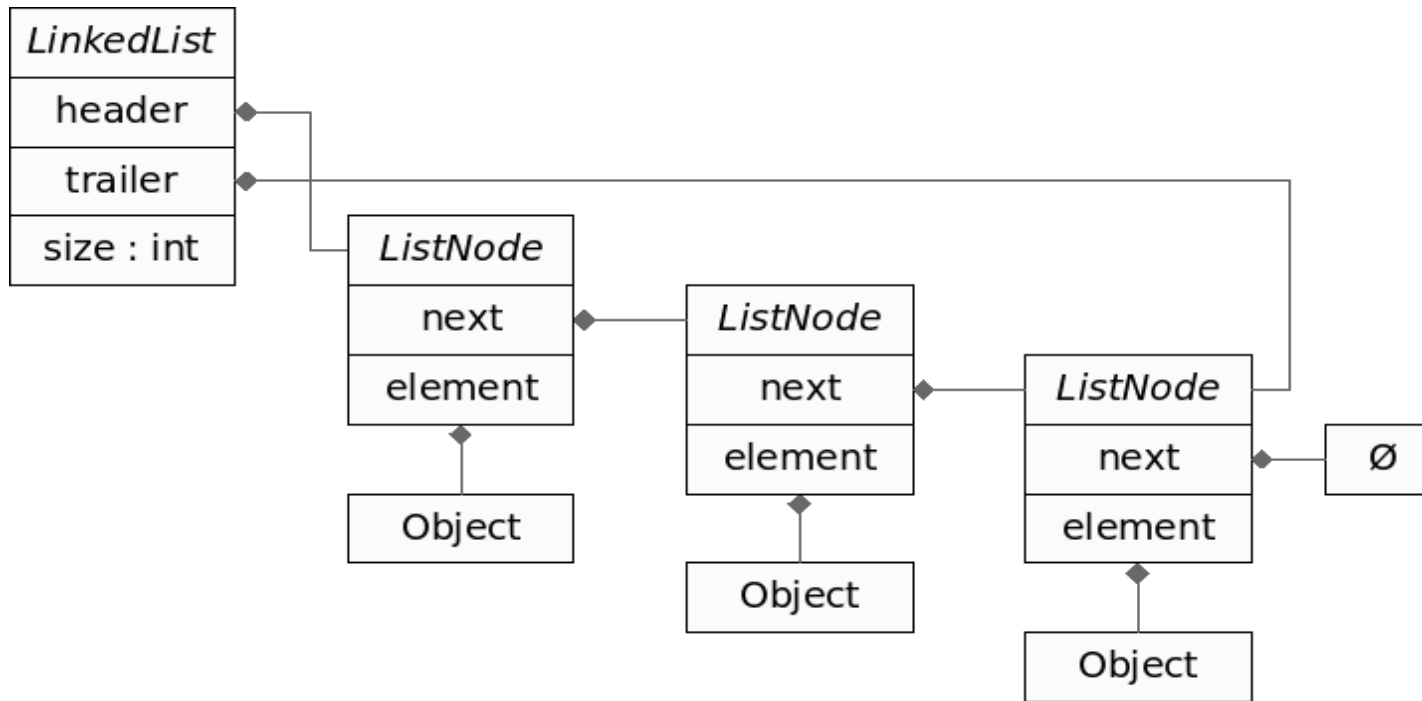
Implementation

- ADTs are implemented using arrays or linked data structures
 - Many ADTs may be implemented with either
 - Some ADTs are implemented using a combination of both
- ADTs are often implemented using other ADTs

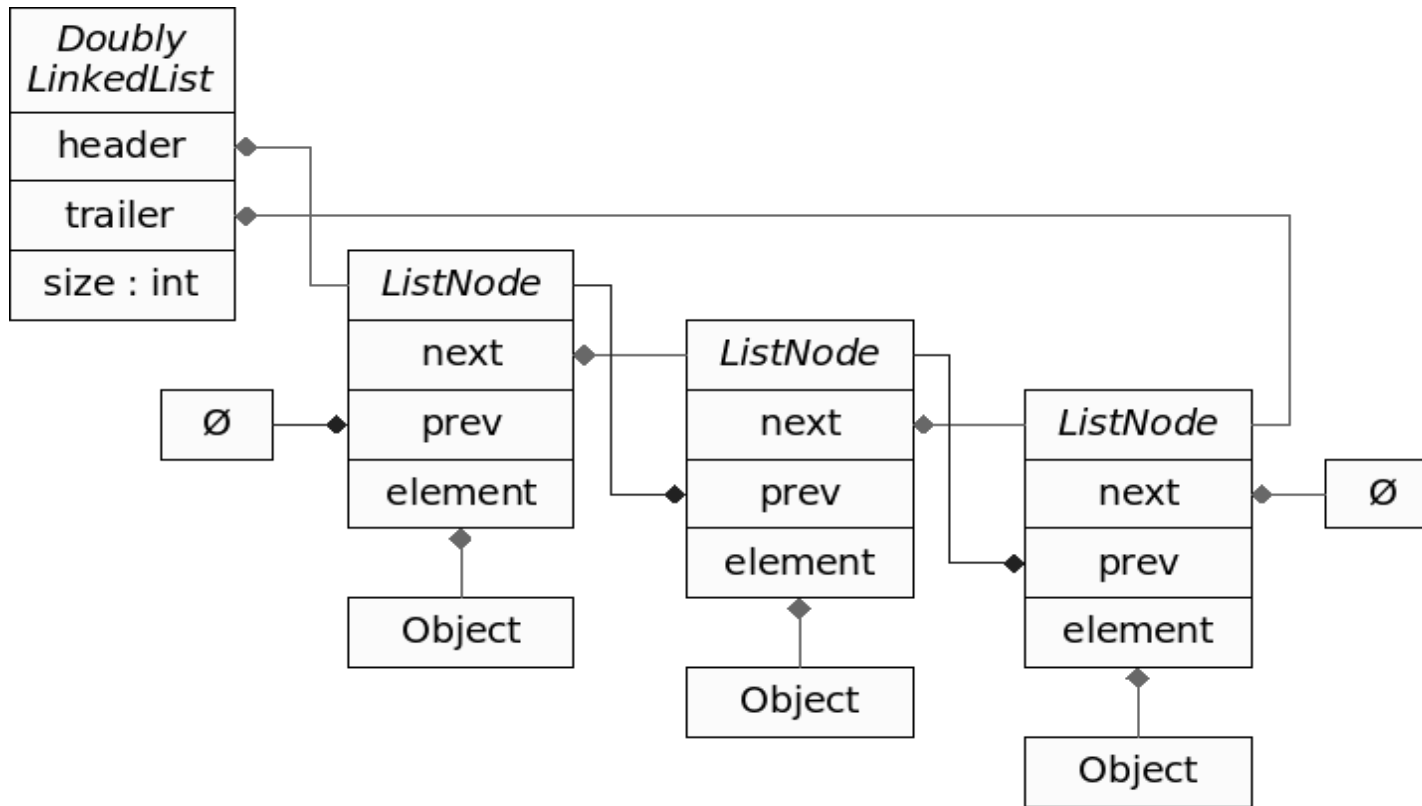
Common Linked Data Structures

- Singly Linked List
- Doubly Linked List
- Circular Linked List
- Tree

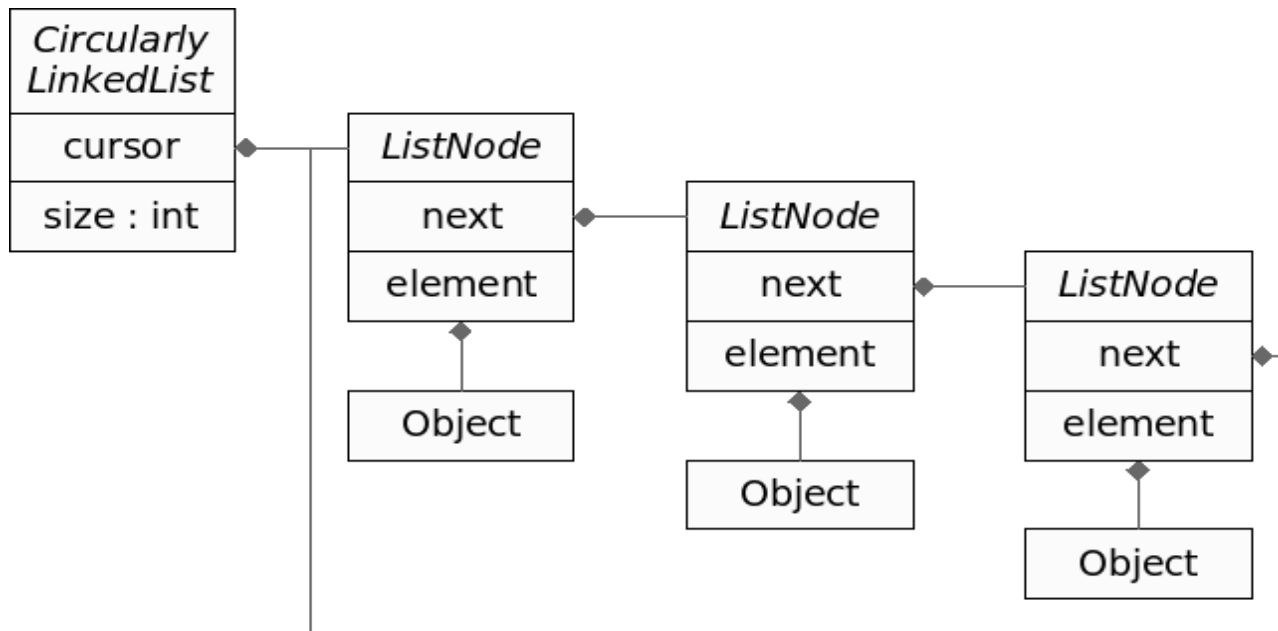
Singly Linked List



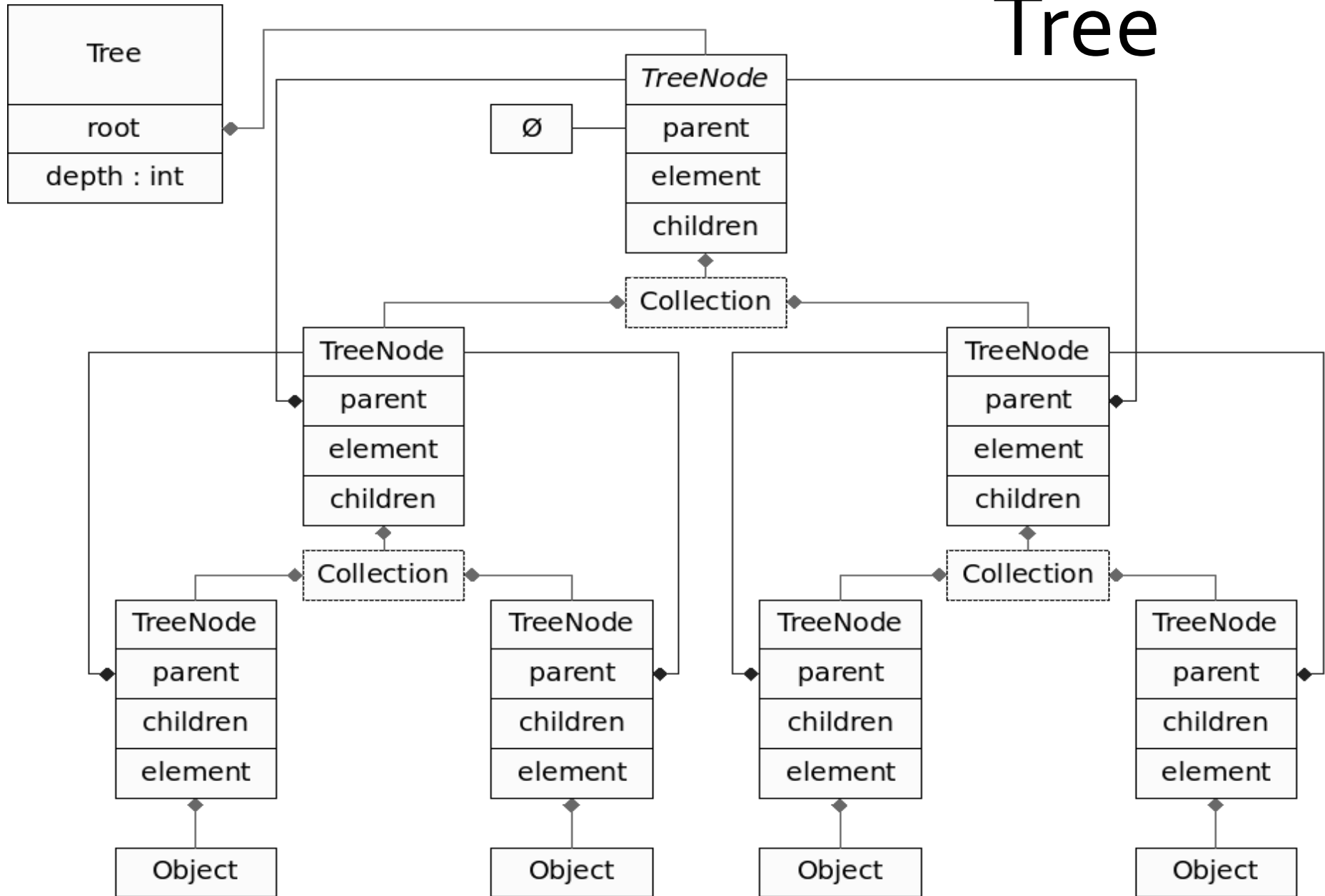
Doubly Linked List



Circularly Linked List



Tree



ADT Commonalities

- In the coming weeks we will be looking at how these data structures are used to implement common ADTs
- Common operations
 - Insertion/removal of data elements
 - Iteration/traversal of data elements
 - Search for specific data element

ADT Commonalities

ADT	Modification	Accessor	Iterators
Stack	push pop	top	
Queue	enqueue dequeue	front	
Deque	addLast removeFront addFront removeLast	getFirst getLast	
Tree	replace	root parent children	iterator
Map	put remove	get	keys values entries
Dictionary	insert remove	find	findAll entries
Set	union intersect subtract		

Lecture Summary

- ADTs provide behavioral contract and separation of implementation
- Specifying ADTs in Java using interfaces
- Generics
- Factory Pattern
- Implementation
- ADT Commonalities