CSE 143
Lecture 25 (b)

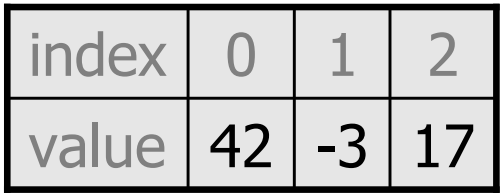
Generic collections

read 11.1, 15.3-15.4, 16.4-16.5

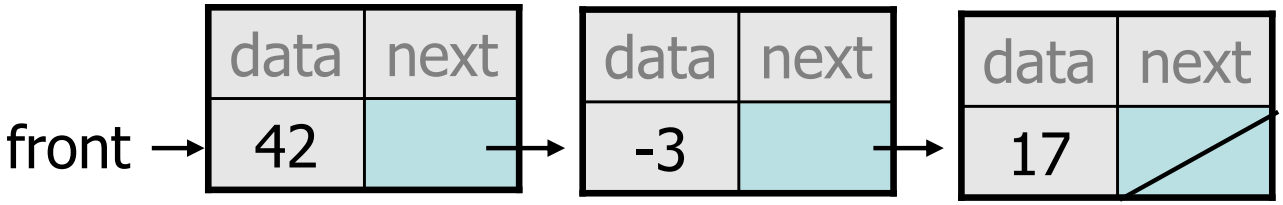
slides created by Marty Stepp
http://www.cs.washington.edu/143/

Our list classes

- We implemented the following two list classes:
  - `ArrayIntList`



  - `LinkedIntList`



  - Problem:
    - We should be able to treat both lists the same way in client code.

Recall: ADT interfaces (11.1)

- **abstract data type (ADT)**: A specification of a collection of data and the operations that can be performed on it.
  - Describes *what* a collection does, not *how* it does it.

- Java's collection framework describes ADTs with interfaces:
  - `Collection`, `Deque`, `List`, `Map`, `Queue`, `Set`, `SortedMap`

- An ADT can be implemented in multiple ways by classes:
  - `ArrayList` and `LinkedList` implement `List`
  - `HashSet` and `TreeSet` implement `Set`
  - `LinkedList` , `ArrayDeque`, etc. implement `Queue`

- Exercise: Create an ADT interface for the two list classes.

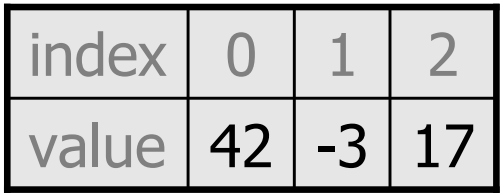
An `IntList` interface (16.4)

```
// Represents a list of integers.
public interface IntList {
    public void add(int value);
    public void add(int index, int value);
    public int get(int index);
    public int indexOf(int value);
    public boolean isEmpty();
    public void remove(int index);
    public void set(int index, int value);
    public int size();
}
```

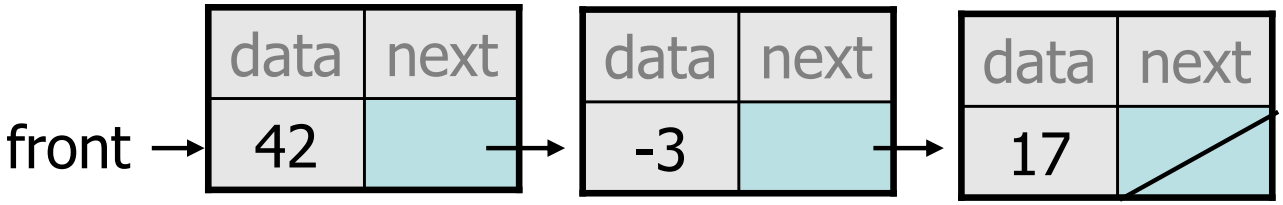
```
public class ArrayIntList implements IntList { ...
public class LinkedIntList implements IntList { ...
```

Our list classes

- We have implemented the following two list collection classes:
  - `ArrayIntList`



  - `LinkedIntList`



  - Problem:
    - They can store only `int` elements, not any type of value.

Type Parameters (Generics)

```
ArrayList<Type> name = new ArrayList<Type>();
```

- Recall: When constructing a `java.util.ArrayList`, you specify the type of elements it will contain between `<` and `>`.
  - We say that the `ArrayList` class accepts a **type parameter**, or that it is a **generic** class.

```
ArrayList<String> names = new ArrayList<String>();
names.add("Marty Stepp");
names.add("Stuart Reges");
```

Implementing generics

```
// a parameterized (generic) class
public class name<Type> {
    ...
}
```

- By putting the **Type** in < >, you are demanding that any client that constructs your object must supply a type parameter.
  - You can require multiple type parameters separated by commas.
- The rest of your class's code can refer to that type by name.

- Exercise: Convert our list classes to use generics.

Generics and arrays (15.4)

```
public class Foo<T> {
    private T myField;                          // ok

    public void method1(T param) {
        myField = new T();                      // error
        T[] a = new T[10];                      // error
    }
}
```

- You cannot create objects or arrays of a parameterized type.

Generics/arrays, fixed

```
public class Foo<T> {
    private T myField;                          // ok

    public void method1(T param) {
        myField = param;                        // ok
        T[] a2 = (T[]) (new Object[10]);        // ok
    }
}
```

- But you can create variables of that type, accept them as parameters, return them, or create arrays by casting `Object[]`.

Comparing generic objects

```
public class ArrayList<E> {
    ...
    public int indexOf(E value) {
        for (int i = 0; i < size; i++) {
        // if (elementData[i] == value) {
            if (elementData[i].equals(value)) {
                return i;
            }
        }
        return -1;
    }
}
```

- When testing objects of type E for equality, must use `equals`

Generic linked list nodes

```
public class ListNode<E> {
    public E data;
    public ListNode<E> next;

    ...
}
```

- For a generic linked list, the node class must also accept the type parameter `E`

Generic interface (15.3, 16.5)

```
// Represents a list of values.
public interface List<E> {
    public void add(E value);
    public void add(int index, E value);
    public E get(int index);
    public int indexOf(E value);
    public boolean isEmpty();
    public void remove(int index);
    public void set(int index, E value);
    public int size();
}

public class ArrayIntList<E> implements IntList<E> { ...
public class LinkedIntList<E> implements IntList<E> { ...
```