CS 251
Intermediate Programming
Intro to Collections and Generics

Brooke Chenoweth

University of New Mexico

Spring 2025

Lists

Ordered sequence, similar to an array, but more general.

- `add(item)` Add item to end of list
- `add(index, item)` Add item at specified position in list
- `contains(object)` Returns true if list contains given object
- `get(index)` Get item at specified position
- `remove(item)` Remove item from list
- `remove(index)` Remove item at position from list
- `size()` Get size of list

String List Interface

```
public interface StringList {
  // Add s to end of list.
  void add(String s);

  // Add s at position index
  void add(int index, String s);

  // True if s in list
  boolean contains(String s);

  // Return item at index
  String get(int index);

  // Remove s, return true if successful.
  boolean remove(String s);

  // Remove and return item at index.
  String remove(int index);

  // Return size of list
  int size();
}
```

Abstract Data Types

- List – sequence
- Set – no duplicates
- Queue – First In, First Out (FIFO)
- Stack – Last In, First Out (LIFO)
- Deque – double-ended queue (pronounced "deck")
- Map – associates keys with values (some languages call it a *dictionary*)

Comparable interface (sort of)

```
public interface Comparable {

   // Compares this with other object
   // Returns negative, zero, or positive integer when
   // less than, equal to, or greater than other
   int compareTo(Comparable o);
}
```

Using Comparable

```
public static void printGreatest(Comparable a,
                                 Comparable b) {
  if(a.compareTo(b) > 0) {
    System.out.println(a);
  } else {
    System.out.println(b);
  }
}
```

Implementing Comparable

```
public class MyClass implements Comparable {
  // Value used to compare instances
  private int value;

  public MyClass(int value) {
    this.value = value;
  }

  public int compareTo(Comparable o) {
    MyClass other = (MyClass)o;
    return this.value - other.value;
  }
}
```

Comparable interface with generics

```
// T is type of objects that this
// object may be compared to
public interface Comparable<T>{

   // Compares this with other object
   // Returns negative, zero, or positive integer when
   // less than, equal to, or greater than other
   int compareTo(T o);
}
```

Implementing Comparable

```
// Can compare MyClass objs to other MyClass objs
public class MyClass implements Comparable<MyClass> {
  // Value used to compare instances
  private int value;

  public MyClass(int value) {
    this.value = value;
  }

  // Look Ma! No casting!
  public int compareTo(MyClass other) {
    return this.value - other.value;
  }
}
```

Non-generic Box

```
public class Box {
  private Object object;

  public void set(Object object) {
    this.object = object;
  }

  public Object get() {
    return object;
  }
}
```

Could put anything in the box!

Using Box

```
String myString = "Hello!";

Box myBox = new Box();
myBox.set(myString);

Object obj = myBox.get();
String str = (String)myBox.get();
```

We've lost our type information!

Generic Box

```
public class Box<T> {
  private T object;

  public void set(T object) {
    this.object = object;
  }

  public T get() {
    return object;
  }
}
```

T is the *type parameter*.

Using Generic Box

```
String myString = "Hello!";

Box<String> myBox = new Box<String>();
myBox.set(myString);

String str = myBox.get();
```

Specifying a *type argument* of `String` to say that `myBox` is a `Box` of `String`.

Why Generics?

- Stronger type checks at compile time. It is easier to fix compile-time errors than runtime errors.
- Elimination of casts.
- Enables generic algorithms.
- Collections!

Generic List Interface

```
public interface List<T> {
  // Add t to end of list.
  void add(T t);

  // Add t at position index
  void add(int index, T t);

  // True if t in list
  boolean contains(T t);

  // Return item at index
  T get(int index);

  // Remove t, return true if successful.
  boolean remove(T t);

  // Remove and return item at index.
  T remove(int index);

  // Return size of list
  int size();
}
```