



Lecture 12: Collections and Generics

CS 2110

October 1, 2026

Today's Learning Outcomes

- 48. Describe the differences between *data structures* and *abstract data types*.
- 49. Implement a generic class or method with one or more generic type parameters. Use generic classes in client code.
- 50. Describe the semantics of *auto-boxing* and *auto-unboxing* and identify where they happen in a code snippet.
- 52. Compare and contrast the behaviors of ordinary Java arrays and dynamic array types such as Java's ArrayList.

Arrays as Collections

A **collection** is a type that groups together objects of another type.

First Example: Arrays

`String[]` is a collection of `Strings`

State:

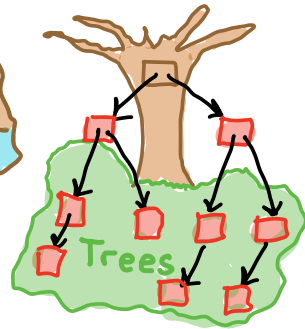
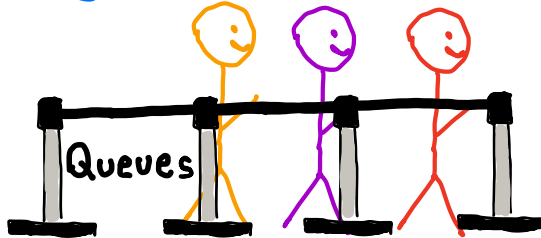
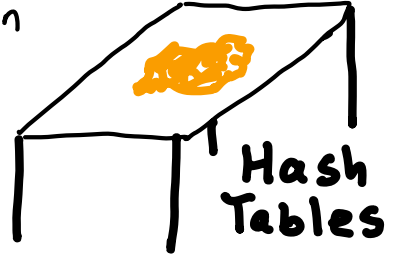
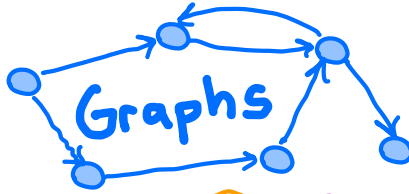
Behaviors:

Other Collections



Each has its own

- structure/state representation
- behaviors / invariants
- performance guarantees



Abstract Data Types vs. Data Structures

The client and implementer interact with collections in different ways.
(i.e., collections impose an *abstraction barrier*)

Client:

- What operations are supported

How is the collection

- modified (add, remove, etc.)

- queried

- (runtime guarantees for behaviors)

Implementer:

How do we

- represent data in memory

- define promised operations

- meet runtime guarantees

The List ADT



A **list** is an ordered collection that grows to accommodate an arbitrary number of elements. Its elements are accessed by index.

Behaviors:



Coding Demo: `StringList` Interface



Using StringList as a Client

Use the StringList methods to complete the definition of the following method:

```
/** Replaces all instances of the given `word`  
 * with "****" in these `lyrics`. */  
static void censor(StringList lyrics, String word) {  
  
  
  
  
  
  
}
```

StringList

```
add(String elem): void  
size(): int  
get(int index): String  
contains(String elem): boolean  
indexOf(String elem): int  
set(int index, String elem): void  
remove(int index): String  
delete(String elem): void
```

Generic Classes

What if we want to make a list of something other than `Strings`?

- `Accounts`, `Points`, other lists...



Coding Demo: Generic CS2110List



Using Generic Types

Generic types can take the place of a type name almost everywhere in a class.

- Fields

```
private T elem;  
private T[] storage;
```

cannot:

- invoke methods on objects of type T (for now)

- Local variables

- Parameters

```
public void add(T elem) {}
```

- Return types

```
public T get(int index) {}
```

Brief Aside: Auto-(un)boxing

Generic types parameters can only be assigned **reference** types, not primitive types.

What if we want a list of ints?

The Dynamic Array Data Structure

How can we use an array to represent the state of a list?



Coding Demo: `DynamicArrayList` Design



invariantSatisfied() Methods

Data structures often rely on intricate class invariants to ensure correctness and achieve good performance.

Recall: The class invariant must hold at the start/end of every public method call.

`invariantSatisfied()` methods are a good development tool

- packages up checks for class invariant into one boolean method
- `assert invariantSatisfied()` before returning from any public mutating methods
- *"defensive programming against yourself"*



Coding Demo: `DynamicArrayList` Methods



Space Complexity of `DynamicArrayList`

Time Complexity of DynamicArrayList

DynamicArrayList

`insert(int index, T elem): void`

`remove(int index): T`

`size(): int`

`get(int index): T`

`set(int index, T elem): void`

`contains(T elem): boolean`

`indexOf(T elem): int`

`delete(T elem): void`

`add(T elem): void`