# Classes with Generics

Deborah A. Trytten

CS 1323/1324

# Data Structures

- Ways of organizing data for efficient storage and retrieval
- ArrayList is our first data structure
  - Resizable array (special case of oversize array)
- Part of Java Collections Framework (JCF)
  - Library of Java data structures
  - API will look a bit odd at first
    - Uses advanced programming techniques
- There is a companion class of static methods
  - Collections

# Why Do This Now?

- We can have more fun
  - With this class we can store large amounts of data
  - We can write programs more like the ones you really use
- Continue to improve OOP vocabulary
  - Class versus object
  - Class methods versus instance (object) methods
  - Parameters versus arguments
- Have a fantastic example of a mutable class

# What is an ArrayList?

- A way of storing multiple objects in a sequence
- Objects are zero indexed
- No gaps permitted in sequence
- All of the objects must be the same type
  - Homogeneous
- Familiar properties hint at what's hidden inside

# Construct an ArrayList

- Construct an ArrayList that stores String objects
- Read API. What is <E>?
  - A variable that stands in for the type of data stored
  - We'll use String and wrapper classes
- Declare a reference:

```
ArrayList<String> list;
```

- Construct an object:

```
list = new ArrayList<String>();
```

- Draw memory diagram
  - list contains 10 null references
  - list does not contain any objects initially

# iClicker Question

Which line of code constructs an ArrayList that can store rainfall in inches?

```
a) ArrayList<double> list = new ArrayList();
b) ArrayList<double> list = new ArrayList<double>();
c) ArrayList<Double> list;
d) ArrayList<Double> list = new ArrayList();
e) ArrayList<Double> list = new ArrayList<Double>();
```

# ArrayList: Capacity and Size

- Default constructor creates ArrayList object with an initial capacity of 10
  - Number of objects that can be stored without resizing
- Other constructor lets us set the initial capacity
  - Use if capacity is known in advance (save time)
- Size of ArrayList objects is initially 0
  - No objects are stored in it
- Size increases as objects added
- Capacity increases automatically as necessary
  - Done behind the scenes (encapsulated)

# iClicker Question

Suppose we create an ArrayList<Integer> object using the no-argument constructor and add three Integer objects (1, 3, and 5).

Which of the following is true?

a) The size is 3 and the capacity is 10.
b) The size is 3 and the capacity is 3.
c) The size is 10 and the capacity is 3.
d) The size is 10 and the capacity is 10.

# Shopping List Example: Add Items

- Look for method to add items in the API
  - Accessor or mutator?
- Add items at end of list
- Add items at start of list
- Show memory diagram
- Where are we allowed to add items?
  - Read API for add(int index, E Element)

# Shopping List Example: Display Items

- Use method toString
- How to access individual elements?
- How many elements are in the list?
- Write method to display elements like this:
  - 1. First item
  - 2. Second item
  - …

# Shopping List Example: Delete Items

- Delete by index
  - How to avoid leaving gaps?
- Delete by name
- Delete all items
  - Multiple ways

# Collections Class

- Similar to Arrays class
- Contains only static methods
  - Some crazy syntax
    - Use common sense
- ArrayList objects are passed by sharing
  - Exactly like arrays
- If new ArrayList is constructed inside method, it must be returned

# Collections Rules

- ArrayList can be an argument for any parameter with data type Collection<> or List<>
- Ignore methods that deal with other classes like Map<> and Set<>
- Methods that need to compare elements only work on objects that have compareTo() methods
  - String, Integer, Double, Character, other wrappers
  - What would it mean to sort Point objects?

# Lots of Great Methods

sort()
copy()
disjoint()
fill()
frequency()
max()
min()
reverse()

- Print the shoppingList in sorted order

# iClicker Question

Suppose the Collections class had a method that performed linear search for a target value. What would the signature be?

a) `boolean contains()`
b) `boolean contains(E target)`
c) `boolean contains(ArrayList<E> list, E target)`
d) `boolean contains(ArrayList<E> list)`
e) `void contains(ArrayList<E> list, E target, boolean isInThere)`

# Using Wrapper Classes

- Suppose that we're storing examination scores
  - `ArrayList<Integer> scores;`
- Add the scores 90 to 99 to an ArrayList object
  - Add 93 four times
- Play with some Collections methods
  - Find the maximum
  - Find how many times 93 occurred
  - Shuffle the list

# Binary Search

- Same as in Arrays class
- Requires sorted data
  - If not sorted the results are undefined
- Analogy to dictionary
- Algorithm same as for arrays
- Very, very fast when compared to linear search

# iClicker Question

Suppose we execute the following code:

```
ArrayList<Integer> list = new ArrayList<Integer>();
for (int i = 15; i > 0; --i) {
    list.add(new Integer(i));
}
int idx;
idx = Collections.binarySearch(list, new Integer(2));
```

Which of the following is true about idx?

a) idx will be positive
b) idx will be zero
c) idx will be negative

# Autoboxing

- Java knows the relationship between primitive data types and wrapper classes
  - If you insert primitive data instead of an object, Java constructs the object automatically
- Rework the example from the last slide with int instead of Integer
  - Remember that objects are being constructed
- Similarity to hidden String constructor

# ArrayList and Mutable Objects

- ArrayLists and mutable objects can cause trouble
- Create ArrayList of StringBuilder objects
  - Sort it
  - Save the reference to the second object
  - Change it
  - What happens to the object in the ArrayList?
- Show memory diagram
- Try sorting and binary search
- Why would JCF work this way?
- Note Java strongly prefers immutable objects
  - Avoids subtle debugging problems