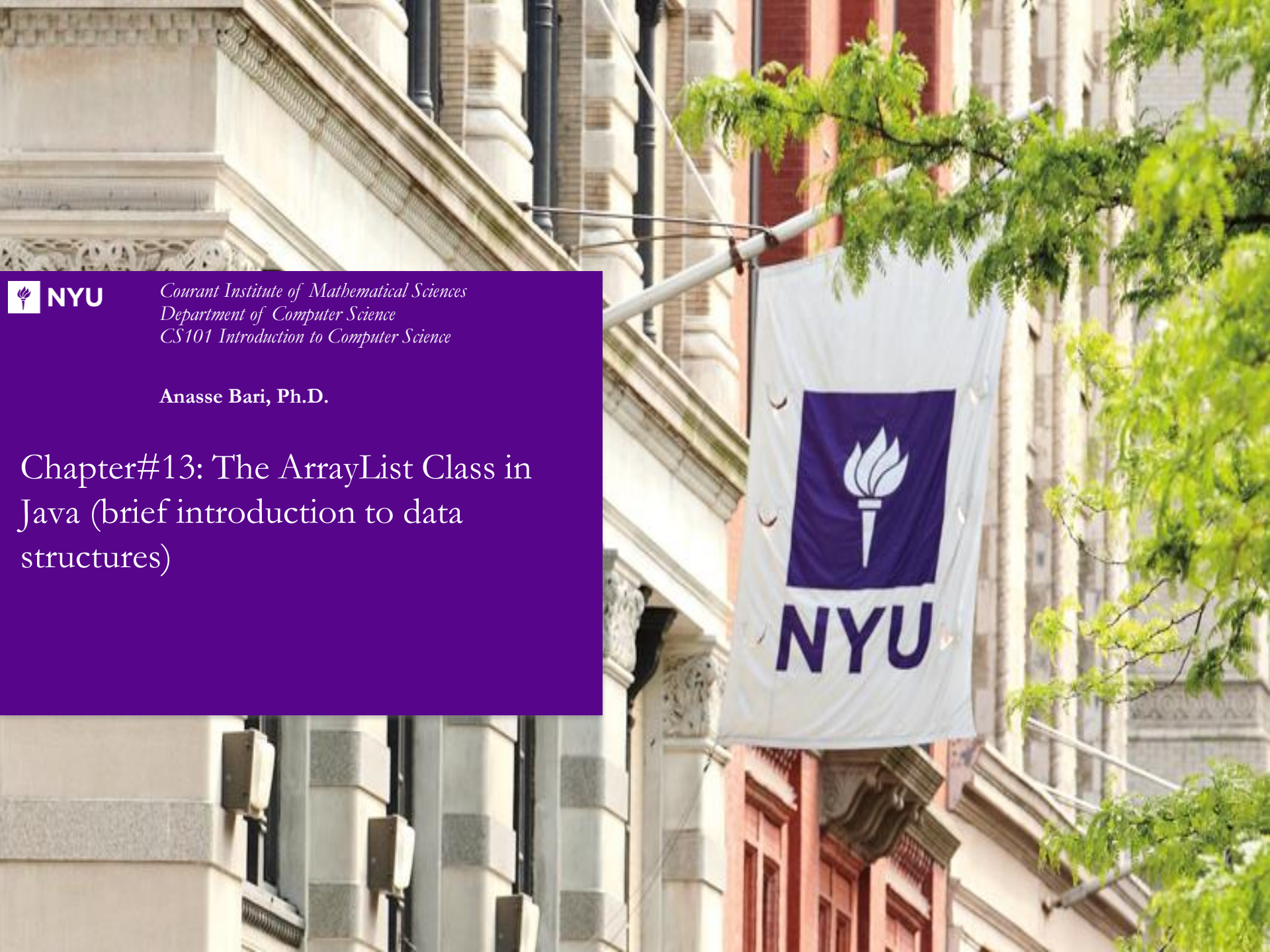

NYU

Courant Institute of Mathematical Sciences
Department of Computer Science
CS101 Introduction to Computer Science

Anasse Bari, Ph.D.

Chapter#13: The ArrayList Class in Java (brief introduction to data structures)

Learning Outcomes

- Learning and applying the ArrayList class
- Introducing the concept of generic type in Java, boxing and unboxing
- Learning how to implement an ArrayList of objects
- Exposure to the concept of data structures

Introducing The **ArrayList** Class

- Although arrays are conceptually important as a data structure, they are not used as much in Java as they are in most other languages. The reason is that the **java.util** package includes a class called **ArrayList** that provides the standard array behavior along with other useful operations.
- The main differences between Java arrays and **ArrayList**s stem from the fact that **ArrayList** is a Java class rather than a special form in the language. As a result, all operations on **ArrayList**s are indicated using method calls. For example, the most obvious differences include:
  - You create a new **ArrayList** by calling the **ArrayList** constructor.
  - You get the number of elements by calling the **size** method rather than by selecting a **length** field.
  - You use the **get** and **set** methods to select individual elements.
- The next slide summarizes the most important methods in the **ArrayList** class. The notation **<T>** indicates the base type.

Generic Types in Java

- The **<T>** notation used on the preceding slide is a new feature of Java that was introduced with version 5.0 of the language. In the method descriptions, the **<T>** notation is a placeholder for the element type used in the array. Class definitions that include a type parameter are called **generic types**.
- When you declare or create an **ArrayList**, it is a good idea to specify the element type in angle brackets. For example, to declare and initialize an **ArrayList** called **names** that contains elements of type **String**, you would write

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

- The advantage of specifying the element type is that Java now knows what type of value the **ArrayList** contains. When you call **set**, Java can ensure that the value matches the element type. When you call **get**, Java knows what type of value to expect, eliminating the need for a type cast.

Methods in the **ArrayList** Class

`boolean add(<T> element)` Adds a new element to the end of the **`ArrayList`**; the return value is always **`true`**.
`void add(int index, <T> element)` Inserts a new element into the **`ArrayList`** before the position specified by **`index`**.
`<T> remove(int index)` Removes the element at the specified position and returns that value.
`boolean remove(<T> element)` Removes the first instance of **`element`**, if it appears; returns **`true`** if a match is found.
`void clear()` Removes all elements from the **`ArrayList`**.
`int size()` Returns the number of elements in the **`ArrayList`**.
`<T> get(int index)` Returns the object at the specified index.
`<T> set(int index, <T> value)` Sets the element at the specified index to the new value and returns the old value.
`int indexOf(<T> value)` Returns the index of the first occurrence of the specified value, or −1 if it does not appear.
`boolean contains(<T> value)` Returns **`true`** if the **`ArrayList`** contains the specified value.
`boolean isEmpty()` Returns **`true`** if the **`ArrayList`** contains no elements.

Boxing and Unboxing

- Generic types benefit substantially from the technique of **boxing and unboxing**.

- As of Java Standard Edition 5.0, Java automatically converts values back and forth between a primitive type and the corresponding wrapper class. This feature makes it possible to store primitive values in an **`ArrayList`**, even though the elements of any **`ArrayList`** must be a Java class.

- For example, suppose that you execute the following lines:

```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(42);
int answer = list.get(0);
```

- In the second statement, Java uses boxing to enclose 42 in a wrapper object of type **`Integer`**. When Java executes the third statement, it unboxes the **`Integer`** to obtain the **`int`**.

In The Art and Science of Java, Stanford professor and well-known leader in CS Education Eric Roberts

Example One: ArrayLists of Integers

```
import java.util.ArrayList;

public class Program {
    public static void main(String[] args) {

        // Create new ArrayList.
        ArrayList<Integer> elements = new ArrayList<>();

        // Add three elements.
        elements.add(10);
        elements.add(15);
        elements.add(20);

        // Get size and display.
        int count = elements.size();
        System.out.println("Count: " + count);

        // Loop through elements.
        for (int i = 0; i < elements.size(); i++) {
            int value = elements.get(i);
            System.out.println("Element: " + value);
        }
    }
}
```

Example Two: ArrayLists of Products

```
// Product class
public class Product
{
    private String name;
    private double price;
    private int quantityInStock;

    public Product (String nameIn, double priceIn, int quantityIn)
    {
        name = nameIn;
        price = priceIn;
        quantityInStock = quantityIn;
    }

    public void print()
    {
        System.out.println("Name=" + name + " Price=" + price + " Quantity in stock=" + quantityInStock);
    }
}
```

Example Two: ArrayLists of Products

```
// Main class
import java.util.ArrayList;

public class ArrayListExample1
{
    public static void main (String[] args)
    {
        ArrayList<Product> products = new ArrayList<Product> ();

        Product product1 = new Product("Smartphone", 299.99, 5);
        Product product2 = new Product("Tablet", 199.99, 3);
        Product product3 = new Product("SD Card", 9.99, 100);

        products.add(product1);
        products.add(product2);
        products.add(product3);

        for(int count=0; count<products.size(); count++)
        {
            Product currentProduct = products.get(count);
            currentProduct.print();
        }
    }
```

More on ArrayLists

http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

http://www.dummies.com/how-to/content/use-array-lists-in-java.html

NYU

Courant Institute of Mathematical Sciences
Department of Computer Science
CS101 Introduction to Computer Science

Anasse Bari, Ph.D.

End Chapter#13: The ArrayList Class in Java (brief introduction to data structures)