UBC OKANAGAN

COSC 121
Computer Programming II

ArrayLists and Intro to Generics

Part 1/2

Dr. Mostafa Mohamed

Acknowledgement: The slides mainly rely on the textbook of Y. D. Liang titled “Introduction to Java Programming, 10th Ed.”, Pearson Edu. Inc.

Previous Lecture

- Recursion:
  - How it really works
  - Recursive Helper Methods
  - More example problems
    - Ones that have a nice recursive solution
    - Ones that really need recursion
  - Tail-Recursive Method
  - Recursion vs. Iteration

Outline

Today:

- Intro to Java Collection Framework
- The `ArrayList` Class
- Implementing a `Stack` using `ArrayList`
- Sample applications

Next lecture:

- Iterating Over an `ArrayList`
- Random Access in `ArrayLists`
- Useful Methods for Lists
  - `Arrays` and `Collections` classes
- Intro to Generics

121 Topics

So far, you learned techniques of "**how to do things**":

- Basic Java constructs (if, while, for, …)
- OOP basics (Encapsulation, Inheritance, Polymorphism)
- Exception Handling
- I/O streams
- Recursion

Next, lets look at "how to do things ***efficiently***"

- Data structures
  - what is the best and most ***efficient way to store data in memory***?
    - We have a course COSC222 dedicated for data structures
- Algorithms – e.g. Searching
  - What is the most ***efficient algorithm*** to solve a problem (sorting)?
    - We have a course COSC 320 dedicated for algorithms

Java Collections Framework

Java Collections Framework

A ***data structure*** is a collection of data organized somehow.

- It is a ***container object*** that stores other objects (data).
- A data structure is defined by a class.
  - data fields – data objects
  - methods – operations for managing the data.

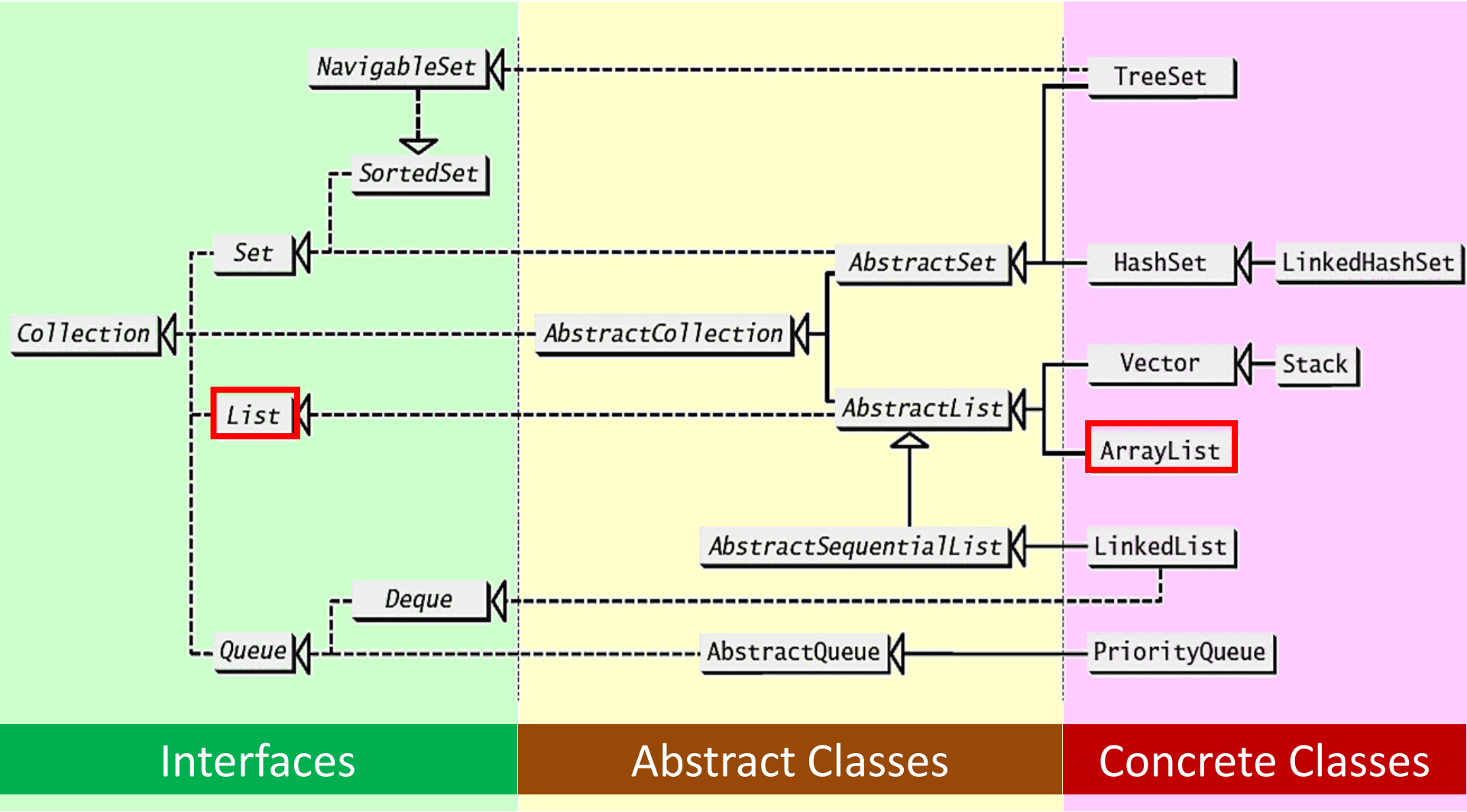
Java Collections Framework includes several data structures for storing and managing data (objects of any type).

It supports two types of containers:

- **Collection***:* for storing a collection of elements.
  - e.g., **Lists** (ordered), **Stacks** (LIFO), and **Queues** (FIFO).
- **Map***:* for storing key/value pairs.
  - e.g., **HashMap**

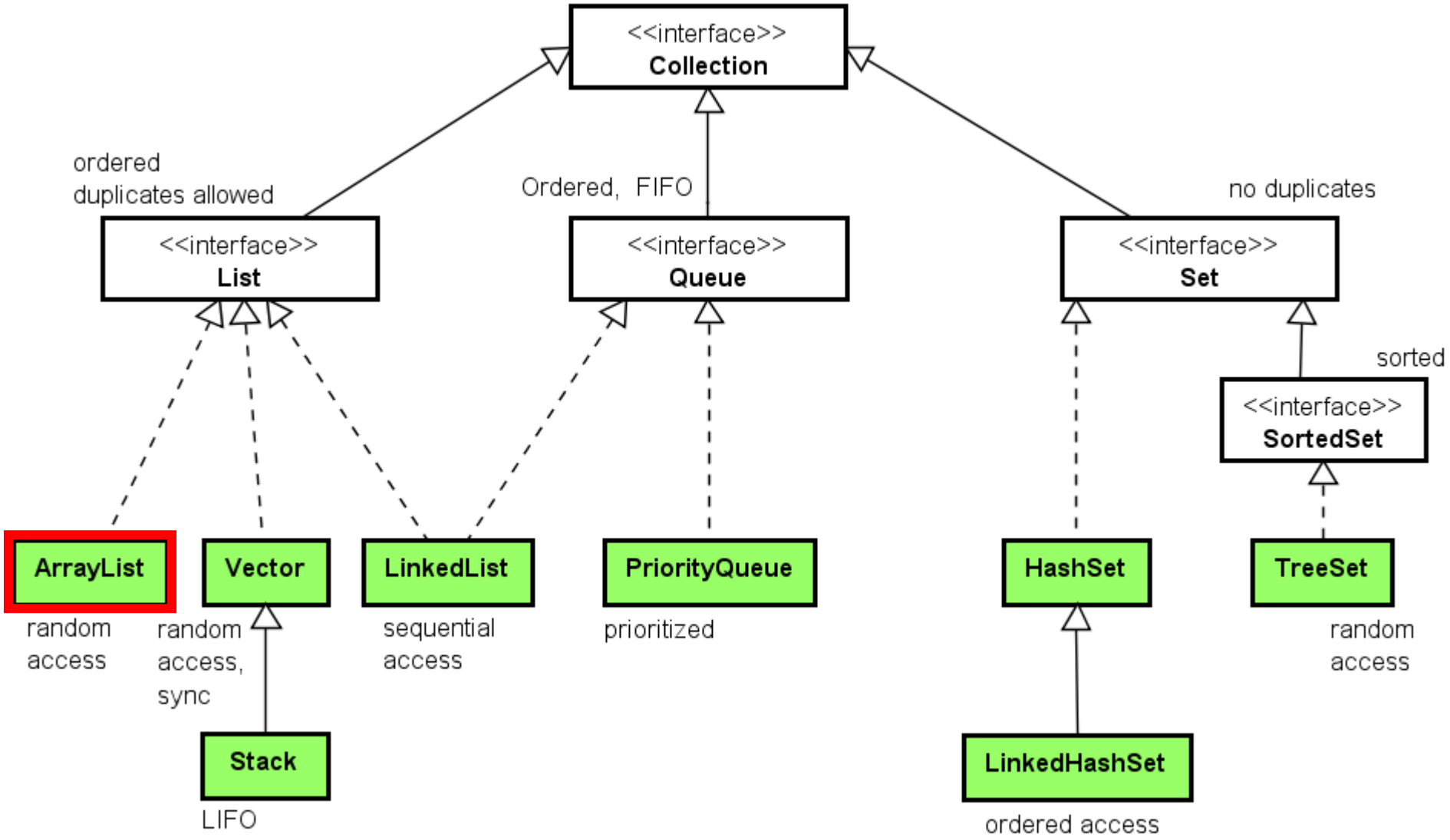
All concrete classes in the Collection framework implements the *Serializable* interface.

Collection Class Hierarchy



Will discuss more on this in Chapter 20

Collections Class Hierarchy (simplified)



*Note: This diagram is **not accurate**. There are several abstract classes and interfaces omitted, but it helps show the big picture.*

The `ArrayList` Class

Intro to `ArrayList` Class

An **`ArrayList`** object

- can be used to store a list of objects (**similar to an array**).
- its **capacity** can **dynamically grow** during the runtime.
  - **capacity**: the maximum number of items an array list can currently hold without being extended.
  - **size** :the current number of items stored in the ArrayList.

- **internally, it is implemented using an array**
  - **Arrays are fixed-size** data structures

Array vs. `ArrayList`

Objects of the generic class **`ArrayList`** are similar to arrays, but have differences:

- Similarities
  - Both can store a number of references
  - The data can be accessed via an **index**.
- Differences
  - `ArrayList` can **automatically increase in capacity** as necessary
    - Will request more space from the Java run-time system, if necessary.
  - `ArrayList` have methods to do many actions
    - e.g. can insert/remove elements anywhere in the list
  - `ArrayList` is **not ideal in its use of memory**.
    - As soon as an `ArrayList` requires more space than its current capacity, the system will allocate more memory space for it. The extension is more than required for the items about to be added.

How to declare an `ArrayList`?

```
ArrayList<String> list1 = new ArrayList<String>();
```

- `list1` holds String references
- This declaration sets up an empty ArrayList The initial **capacity** is set to a **default value**.

```
ArrayList<Date> list2 = new ArrayList<Date>(100);
```

- `list2` holds Date references
- This declaration can be used to specify the initial capacity – in this case 100 elements.
  - This can be more efficient in avoiding repeated work by the system in extending the list, if you know approximately the required initial capacity.

Whenever you use array lists, you must import java.util.ArrayList.

How to declare an `ArrayList`?

Since Java 7, the statement

```
ArrayList<String> list1 = new ArrayList<String>();
```

can be simplified to

```
ArrayList<String> list1 = new ArrayList<>();
```

compiler is able to infer the type from the variable declaration.

If `Type` is omitted, the default type is `Object`

```
ArrayList list3 = new ArrayList();
```

list3 has references of the type Object, which means we can store any object types in it (*WHY?*)

Initializing an `ArrayList`

Once you create an array list, you can start adding elements to it using its add method:

```
ArrayList<String> list = new ArrayList< >();
list.add("R");     //add "R" to end of list
list.add("B");       //add "B" to end of list
list.add(1,"G");   //add "G" between R and G
```

You can initialize an array list using another collection:

```
ArrayList<Type> list = new ArrayList<>(aCollection);
```

We can use `Arrays.asList` method to create a fixed-size list backed by specified elements (more about this next class).

```
ArrayList<String> list
      = new ArrayList< >(Arrays.asList("R","G"));
```

SOME methods *(more will be discussed later)*

boolean add(E o)	Appends a new element o at the end of this list.
void add(int index, E o)	Adds a new element o at the specified index in this list.
E set(int index, E e)	Sets the element at the specified index.
E get(int index)	Returns the element from this list at the specified index.
boolean remove(Object e)	Removes the first occurrence of element o from this list.
E remove(int index)	Removes the element at the specified index.
void clear()	Removes all the elements from this list.
int indexOf(Object e)	Returns the index of the first matching element in this list
int lastIndexOf(Object e)	Returns the index of the last matching element in this list.
boolean contains(Object e)	Returns true if this list contains the element o.
boolean isEmpty()	Returns true if this list contains no elements.
int size()	Returns the number of elements in this list.
void trimToSize()	Trims the capacity of this ArrayList instance to the current size.
boolean equals(Object x)	Returns true if **x** is a list and both lists have the same size, and all corresponding pairs of elements are *equal*.
String toString()	Returns a string representation of the list elements.

Practice Questions

Practice 1

Write Java code to do the following:

a. Create an **`ArrayList`** for storing double values

b. Append two doubles to a list

c. Insert a double at the beginning of a list

d. Find the number of objects in a list

e. Remove a given object from a list

f. Remove the last object from the list

g. Check whether a given object is in a list

h. Retrieve an object at a specified index from a list

Practice 2

Write a program that reads from the user a set of integers and stores them in an `ArrayList`. The input ends with 0 where the program displays all stored elements. Your list should have **no duplicates**. For example:

```
Enter integers (input ends with 0).
41 20 5 20 20 6 0
41 20 5 6
```

```
ArrayList<Integer> list = new ArrayList<>();
int value = 1; // to enter the loop
//read numbers
Scanner input = new Scanner(System.in);
System.out.println("Enter integers (input ends with 0). ");
while (value != 0) {
    value = input.nextInt(); // Read a value from the input
    if (!list.contains(value) && value != 0)
        list.add(value); // Add the value if it is not in the list
}
// Display the distinct numbers
for (int i = 0; i < list.size(); i++)
    System.out.print(list.get(i) + " ");
```

Practice 2 – cont'd

Simpler code

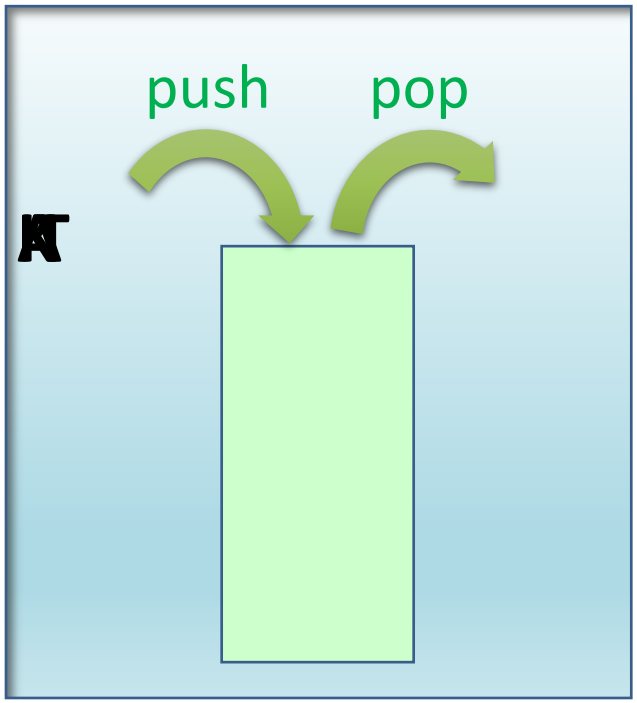
```
ArrayList<Integer> list = new ArrayList<>();
int value;

//read numbers
Scanner in = new Scanner(System.in);
System.out.println("Enter integers (0 to end): ");
while((value = in.nextInt()) != 0)
  if(!list.contains(value))
    list.add(value);

//display distinct numbers
System.out.println(list);
```

The Stack

A Stack represents a LIFO (last-in-first-out) data structure. The elements are accessed only from the top of the stack. That is, you can only retrieve, insert, or remove an element from the top of the stack.



Practice 3

Create a custom stack class, `MyStack`, that can hold objects according to the following:

MyStack	
-list: ArrayList	A list to store elements.
+isEmpty(): boolean	**Returns true if this stack is empty.**
+size(): int	**Returns the number of elements in this stack.**
+peek(): Object	**Returns the top element in this stack.**
+pop(): Object	**Returns and removes the top element in this stack.**
+push(o: Object): void	**Adds a new element to the top of this stack.**

Solution

```
class MyStack {
    private ArrayList list = new ArrayList<>();
    public boolean isEmpty() {
        return list.isEmpty();
    }
    public int size(  ) {
        return list.size();
    }
    public void push(Object e) {
        list.add(e);
    }
    public Object pop() {
        if(list.size()>0)
            return list.remove(list.size() - 1);
        else
            return null;
    }
    public Object peek() {
        if(list.size()>0)
            return list.get(list.size() - 1);
        else
            return null;
    }
}
```

Sample Applications in Gaming

Practical Example

The following example creates an 'army' of robots, e.g. in a game and then control all of them using a loop (e.g., move forward)

```
ArrayList<Robot> robots = new ArrayList<>(5);
robots.add(new Robot(1,1));
robots.add(new Robot(2,1));
robots.add(new Robot(3,1));

System.out.println("BEFORE:" + robots);
for(Robot r: robots) //move your robot army forward
   r.moveForward();
System.out.println("AFTER :" + robots);
```

```
class Robot{
  private int x, y;
  public Robot(int x, int y) {this.x = x;this.y = y;}
  public void moveForward(){y++;}
  public String toString() {return "("+x+","+y+")";}
}
```

Pseudo code for a space shooter game

```
Create an ArrayList<Star> stars
Create an ArrayList<Enemy> enemies
Create an ArrayList<Bullet> bullets

In each frame{
   for(Star star: stars) {star.move() }
   for(Enemy enemy:enemies){enemy.move()}
   for(Bullet bullet: bullets) {
      bullet.move();
      for(Enemy enemy: enemies)
         if (bullet.hits(enemy)){
            bullets.remove(bullet);
            enemies.remove(enemy);
            score++
         }
      }
}
```