Introduction to Information Systems and Programming

Static Typing

ArrayList, LinkedList

Iterating

Objectives

- Static Typing Concept
- ArrayList
- LinkedList
- Generic
- Iterating List Using Iterator

Python is not statically-typed

- No declaration, variable types depend on results of assignments / program execution
- Sometimes impossible to resolve the variable type before program execution

```
var1 =1

userInput = int(input("enter a number:  "))
if userInput > 1:
    var2 = "hello"
else:
    var2 = 10

var3 = var1 + var2
Print(var3)
```

Static Typing

- Java is a **statically-typed** language
- The types of all variables are known at compile time
- The variable type stipulates: (i) the set of **values** that can be taken and (ii) the **operations** that can be performed on those values
- Many ideas in this course / modern programming language is to eliminate bugs from the code, and static typing is one idea

```
int e1 = 24;
String e2 = "hello";

e2 = e1;
/* found before program
execution */
```

Discussion:
Compile time vs Runtime?

Primitive Types

Type	Size	Range	Default*
`boolean`	1 bit	`true` or `false`	`false`
`byte`	8 bits	[-128, 127]	`0`
`short`	16 bits	[-32,768, 32,767]	`0`
`char`	16 bits	['\u0000', '\uffff'] or [0, 65535]	`'\u0000'`
`int`	32 bits	[-2,147,483,648 to 2,147,483,647]	`0`
`long`	64 bits	$[-2^{63}, 2^{63}-1]$	`0`
`float`	32 bits	32-bit IEEE 754 floating-point	`0.0`
`double`	64 bits	64-bit IEEE 754 floating-point	`0.0`

int number = 0; vs int[] number = {1,2,3};

ArrayList, LinkedList

- **Array**: once the array is created, its size is fixed
- Example: int [] a = new int [10];
- **List** are resizable, which means we can add or remove items even after the object is initialized python user definitely not impressed here
- There are 2 types of list in Java: ArrayList and LinkedList
- List can only contain 1 type of elements

ArrayList, LinkedList

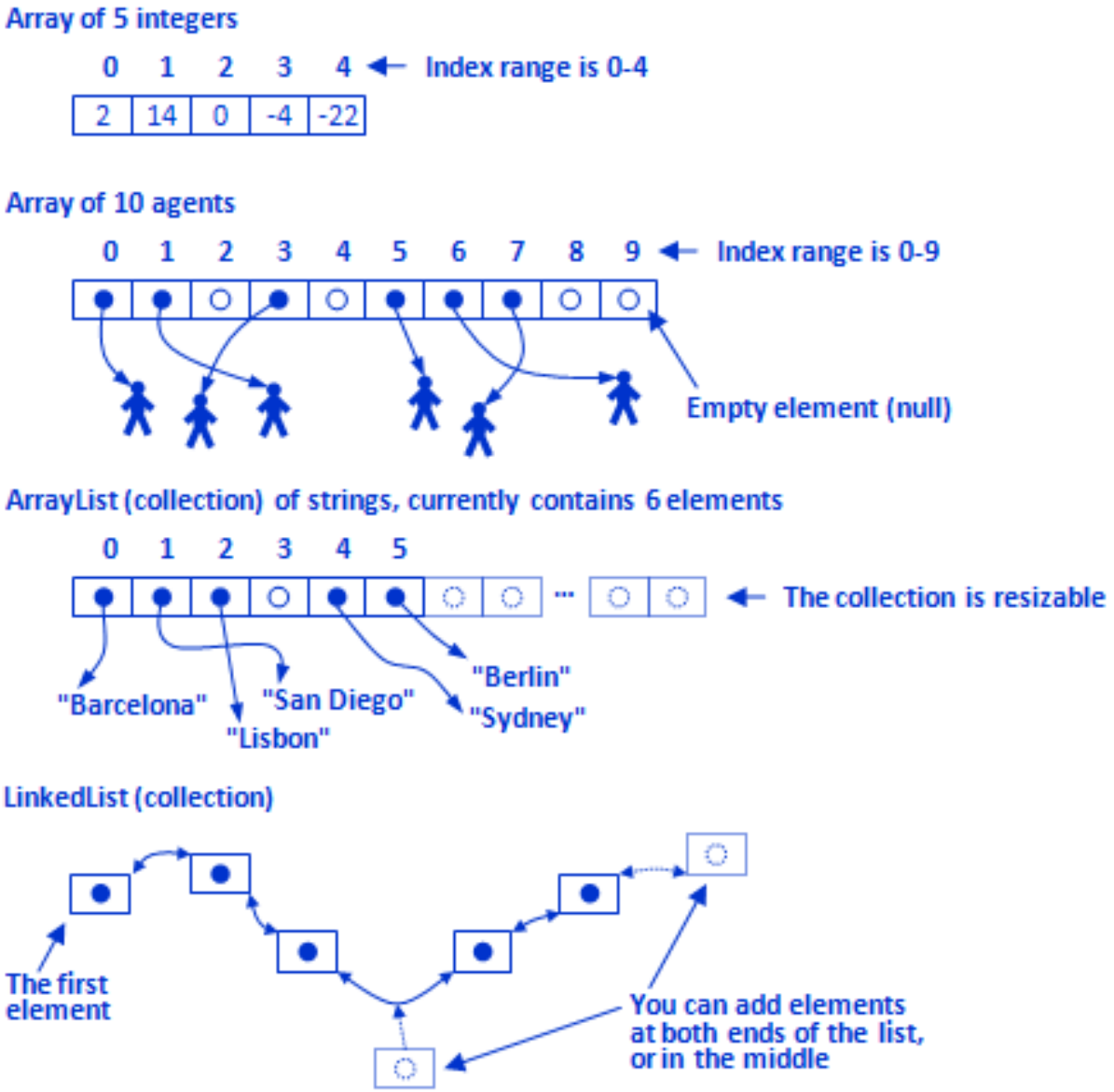
TABLE 11.1 Differences and Similarities between Arrays and `ArrayList`

Operation	*Array*	*ArrayList*
Creating an array/ArrayList	`String[] a = new String[10]`	`ArrayList<String> list = new ArrayList<>();`
Accessing an element	`a[index]`	`list.get(index);`
Updating an element	`a[index] = "London";`	`list.set(index, "London");`
Returning size	`a.length`	`list.size();`
Adding a new element		`list.add("London");`
Inserting a new element		`list.add(index, "London");`
Removing an element		`list.remove(index);`
Removing an element		`list.remove(Object);`
Removing all elements		`list.clear();`

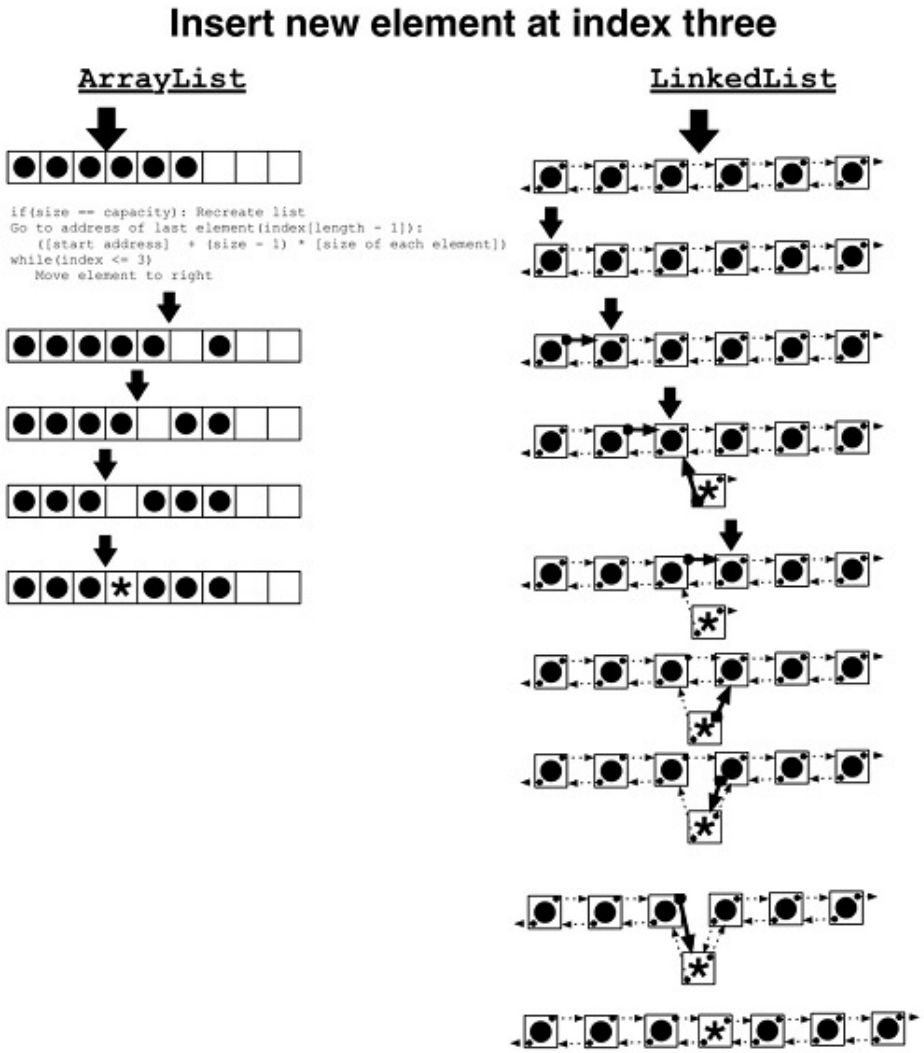
ArrayList, LinkedList

- Operations for ArrayList and LinkedList are **similar** (both implemented the List interface)
- Underlying mechanisms are different:
  - ArrayList stores elements in an **array**; if the capacity is exceeded, a larger new array will be created and all the elements are copied to the new array
  - LinkedList stores elements in a **linked** list data structure
  - Have different performance for various operations, e.g. ArrayList is more efficient to support random access through an index

Array, ArrayList, LinkedList



Insert: ArrayList vs LinkedList



http://cdn.differencebetween.net/wp-content/uploads/2017/07/Difference-Between-Array-List-and-Linked-List.png

Primitive Datatypes Wrapper

- ArrayLists / LinkedList cannot hold primitive data types. It can only contain objects.
- We need a wrapper for the primitive datatypes so it behaves like an object

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Wrapper class = Object version of primitive datatypes

Performance Comparison

Random Access

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;

public class ListDemo {
    public static void main(String[] args) {
        Integer[] a = new Integer[50000];

        //LinkedList
        LinkedList<Integer> linkedList = new LinkedList<>(Arrays.asList(a));
        int totalCnt = 100000;

        long started = System.nanoTime();
        for (int k = 0; k< totalCnt; k++) {
            linkedList.get(25000);
        }
        long time = System.nanoTime();
        long timeTaken = time - started;
        System.out.println("time taken for LinkedList:" + timeTaken/1000000.0 + "ms");

        // ArrayList
        ArrayList<Integer> arrayList = new ArrayList<>(Arrays.asList(a));

        started = System.nanoTime();
        for (int k = 0; k< totalCnt; k++) {
            arrayList.get(25000);
        }
        time = System.nanoTime();
        timeTaken = time - started;
        System.out.println("time taken for ArrayList:" + timeTaken/1000000.0 + "ms");
        }
    }
```

Performance Comparison

Random Access

- time taken for LinkedList:4283.996375ms
- time taken for ArrayList:1.295042ms

Why is the LinkedList slower?

Performance Comparison

Insert element at the starting index

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;

public class ListDemo {
    public static void main(String[] args) {
        Integer[] a = new Integer[50000];

    //LinkedList
    LinkedList<Integer> linkedList = new LinkedList<>(Arrays.asList(a));
    int totalCnt = 100000;

    long started = System.nanoTime();
    for (int k = 0; k< totalCnt; k++) {
        linkedList.add(0, 9); // insert element 9 at index 0
    }
    long time = System.nanoTime();
    long timeTaken = time - started;
    System.out.println("time taken for LinkedList:" + timeTaken/1000000.0 + "ms");

    // ArrayList
    ArrayList<Integer> arrayList = new ArrayList<>(Arrays.asList(a));

    started = System.nanoTime();
    for (int k = 0; k< totalCnt; k++) {
        arrayList.add(0, 9); // insert element 9 at index 0
    }
    time = System.nanoTime();
    timeTaken = time - started;
    System.out.println("time taken for ArrayList:" + timeTaken/1000000.0 + "ms");
    }
}
```

Performance: Insertion at index 0

- time taken for LinkedList:5.082417ms
- time taken for ArrayList:1015.342292ms

Why is the ArrayList slower?

Generics

- generics enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods

Generics

Look at the code below.

```
public class Generic {
    public static void main(String[] args) {
        Container b = new Container();
        b.set(1.4);
        System.out.println(b.get());

    }
}
class Container {
    private double object;
    public void set(double object) {
        this.object = object;
    }
    public double get() {
        return this.object;
    }
}
```

What if I want my container to be able to contain integer, or any other object?

Generics

Must change the highlighted code!

```
public class Generic {
    public static void main(String[] args) {
        Container b = new Container();
        b.set(1.4);
        System.out.println(b.get());

    }
}
class Container {
    private double object;
    public void set(double object) {
        this.object = object;
    }
    public double get() {
        return this.object;
    }
}
```

Generics

Can we change it to Object?

```
public class Generic {
    public static void main(String[] args) {
        Container b = new Container();
        b.set(1.4);
        System.out.println(b.get());

    }
}
class Container {
    private Object object;
    public void set(Object object) {
        this.object = object;
    }
    public Object get() {
        return this.object;
    }
}
```

Generics

It might be an issue later during runtime (look at the example below. The code can be compiled but it will raise a runtime error)
Java is designed to prevent bugs and error before execution. Generics can be used to safe-guard your code.

```
public class Generic {
    public static void main(String[] args) {
        Container b = new Container();
        b.set("Hello World");
        Double s = (Double) b.get();
        System.out.println(s);
    }
}

class Container {
    private Object object;
    public void set(Object object) {
        this.object = object;
    }
    public Object get() {
        return this.object;
    }
}
```

Generics

You can use **generics** to take type as arguments. Possible runtime errors are prevented

```
public class Generic {
    public static void main(String[] args) {
        Container<String> c1 = new Container<>();
        c1.set("Hello World");
        String s = c1.get();

        Container<Double> c2 = new Container<>();
        c2.set(12.2);
        Double d = c2.get();

        System.out.println(s);
        System.out.println(d);
    }
}

class Container<T> {
    private T something;
    public void set(T something) {
        this.something= something;
    }
    public T get() {
        return this.something;}
}
```

Generics

- We have utilized generics previously when using ArrayList or LinkedList object. We need to define the object type that we want the List to contain.

```
ArrayList<String> w1 = new ArrayList<String>();

ArrayList<Integer> w2 = new ArrayList<Integer>();


ArrayList<String> w3 = new ArrayList<>(); // JVM can infer the type
```

Generics

- Using generics allows for error detection at compile time rather than runtime, i.e., static checking

```
ArrayList<String> l = new ArrayList<String>();
l.add("hello");
l.add("bye");
l.add("haha");
l.add(2);          // error detect at compile time
```

Iterating List Using Iterator

Besides using for loop, for-each loop, and while loop, we can use **Iterator** class to loop through List object

One main benefit of using iterator is that we can safely modify the list during the looping.

Iterating List Using Iterator

```java
import java.util.ArrayList;
import java.util.Iterator;

public class IteratingListDemo {
    public static void main(String[] args) {
        ArrayList<Integer> s = new ArrayList<>();
        s.add(10);
        s.add(30);
        s.add(50);
        s.add(70);

        for (Iterator<Integer> iter = s.iterator(); iter.hasNext(); ) {
            Integer val = iter.next();
            System.out.println(val + "");
            if (val > 50) {
                iter.remove();
            }
        }

        System.out.println("ArrayList after looping: " + s);
    }
}
```

Output:

10

30

50

70

ArrayList after looping: [10, 30, 50]

Can you modify code above using while instead?

How to modify list using for or while loop?