Inheritance & Polymorphism

COMP2026

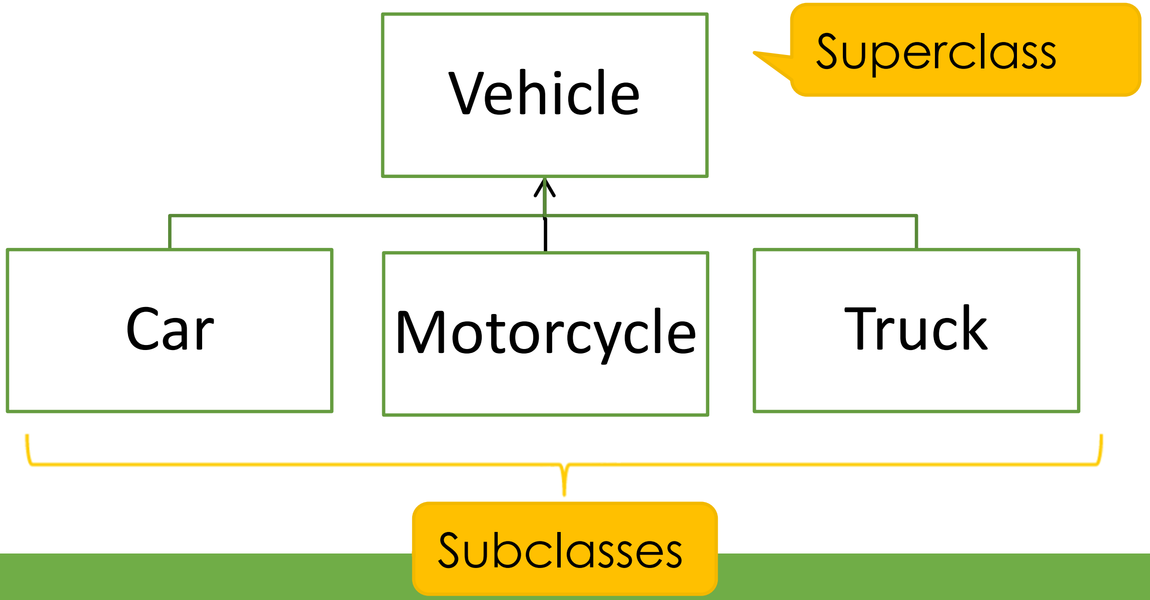
PROBLEM SOLVING USING OBJECT ORIENTED PROGRAMMING

Overview

- Inheritance
  - Writing subclasses
- Polymorphism

Recap: Inheritance

- Inheritance is a relationship between a more general class (superclass) and a more specialized class (subclass)
- Example: Inheritance diagram of vehicles



Recap: Inheritance

- The relation between subclass and superclass is the “is a” relationship
- E.g. Every car is a vehicle
- Cars share the common traits of all vehicles, such as the ability to transport people form one place to another
- We say that the class Car inherits from the class Vehicle
- A subclass inherits data (instance variables)and behavior (methods) from a superclass

Why do we need inheritance?

- Inheritance is actually a way to derive new classes from existing ones
- We can reuse(inherit) or change(override) members in existing classes in order to adapt them to new circumstances
- To derive a new class from an existing one, we use the **extends** keyword

```
public class Car extends Vehicle{
   ...
}
```

The class Car inherits all instance variables and methods from the class Vehicle

Superclass

Example: Shape

```java
public class Shape {

    private String name;

    public Shape() {
        this("");
    }

    public Shape(String name) {
        this.name = name;
    }

    public String getName() {
        return this.name;
    }

    public String toString() {
        return "Shape: " + this.name;
    }
}
```

Example: Rectangle

- Write a class named Rectangle that is a subclass of the given Shape class
- The class should have the following members:
  - A private double field for the width of the rectangle
  - A private double field for the length of the rectangle
  - A constructor that accepts the name, width and length as arguments
  - A default constructor that sets the name to empty string, width and length to 1
  - Implementation of the area() and perimeter() methods
  - Override the toString() method and return a string in the following format:

    Rectangle: *name*

    Width: *width*

    Length: *length*

```
public class Rectangle extends Shape
{   private double width;
    private double length;

    public Rectangle()
    {
        this("", 1, 1);
    }
    public Rectangle(String name, double width, double length)
    {
        super(name);
        this.width = width;
        this.length = length;
    }
    public double area()
    {
        return width * length;
    }
    public double perimeter()
    {
        return (width + length)*2;
    }
    public String toString()
    {
        return "Rectangle: " + super.getName() + "\nWidth: "
               + width + "\nLength: " + length;
    }
}
```

Using this keyword to call another constructor in the same class

Using super keyword to call the constructor in the super class

Consider the following inheritance hierarchy

```
public class Employee{
  private String name;
  private double salary;

  public Employee(String name, double salary){
    this.name = name;
    this.salary = salary;
  }


  public String getName(){
    return name;
  }


  public double getSalary(){
    return salary;
  }

  public String toString(){
    return "Name: " + name +
           "\nSalary: " + salary;
  }
}
```

```
public class Manager extends Employee{

  private double bonus;

  public Manager(String name, double salary, double bonus){
    super(name, salary);
    this.bonus = bonus;
  }

  public double getBonus(){
    return bonus;
  }

  public double getSalary(){
    return super.getSalary() + bonus;
  }

  public String toString(){
    return "Name: " + name +
           "\nBase Salary: " + super.getSalary() +
           "\nBonus: " + bonus;
  }
}
```

Polymorphism

- Under inheritance, we could use a subclass object whenever the superclass object is expected in the program

```
Employee e;  //e is Employee type
e = new Empolyee(...);  //OK
```

```
Employee e;
e = new Manager(...);  //OK, Manager object can be
                       //use as well
```

- However, the converse is not correct:

```
Manager m;
m = new Empolyee(...);  //Error!
```

Polymorphism example

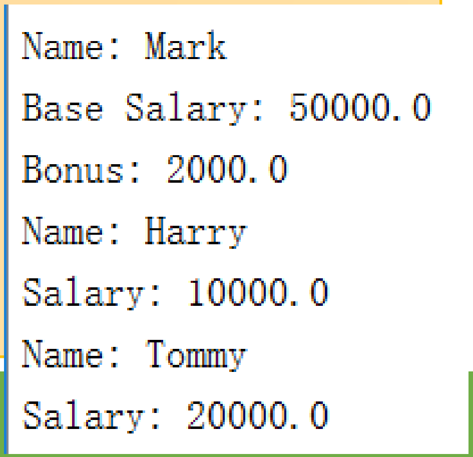
```
import java.util.*;
public class StaffTester {
   public static void main(String[] args)   {

      List<Employee> staffList = new ArrayList<>();

      // fill the staff array with Manager and Employee objects
      staffList.add(new Manager("Mark", 50000, 2000));
      staffList.add(new Employee("Harry", 10000));
      staffList.add(new Employee("Tommy", 20000));

      // print information of all staff
      for (int i = 0; i < staffList.size(); i++){
        System.out.println(staffList.get(i));
      }
   }
}
```

Corresponding toString() method is called base on the **actual object type**



```
Name: Mark
Base Salary: 50000.0
Bonus: 2000.0
Name: Harry
Salary: 10000.0
Name: Tommy
Salary: 20000.0
```

Part A
Discovery Exercises

Type your answer in **XXXXXXXX_lab10.docx**

ArrayList

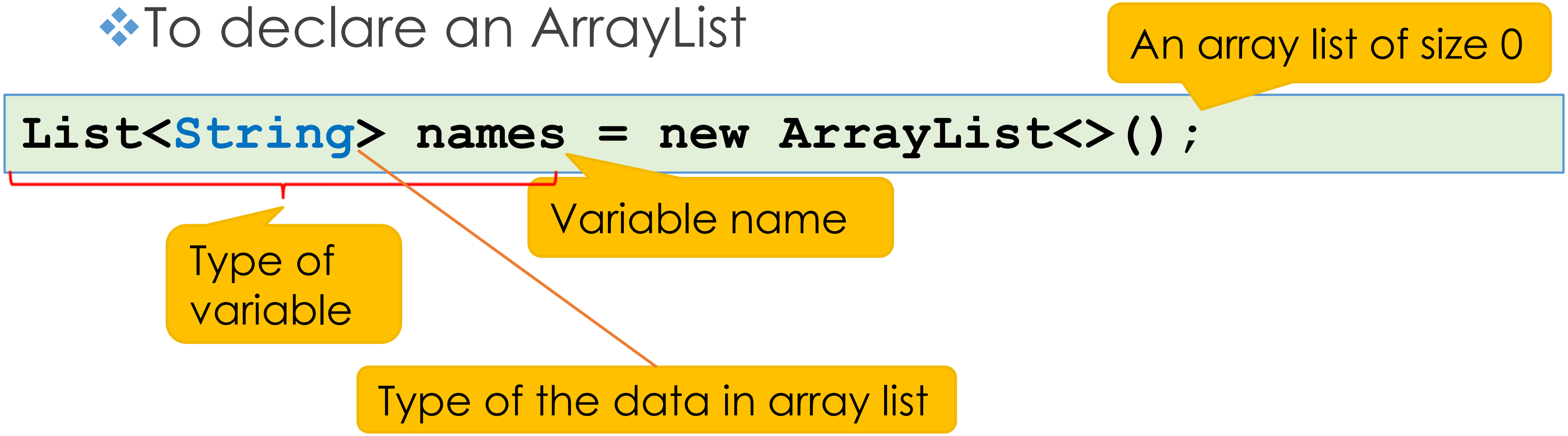
- The ArrayList class is a resizable array, which can be found in the java.util package
- See https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
- The difference between a built-in array and an ArrayList
  - the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one)
  - while elements can be added and removed from an ArrayList whenever we want

Declaring and Using Array Lists

- ArrayList class is contained in the java.util package
- In order to use array lists, we need to import it by the statement

```
import java.util.*;
```

- To declare an ArrayList



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

An array list of size 0

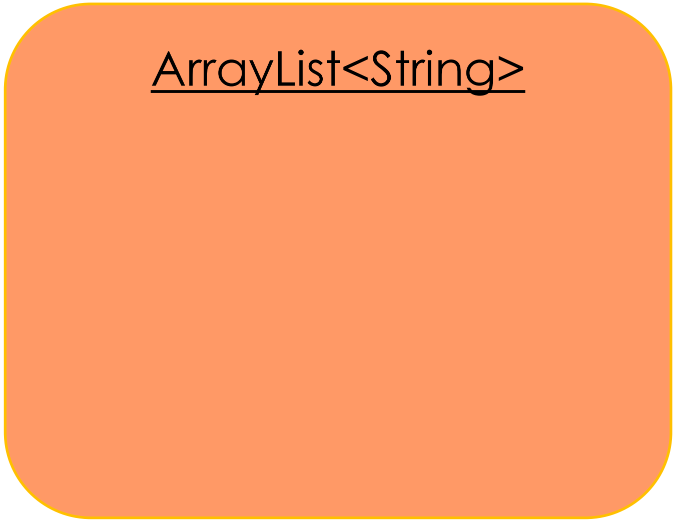
Variable name

Type of variable

Type of the data in array list

Adding element into ArrayList

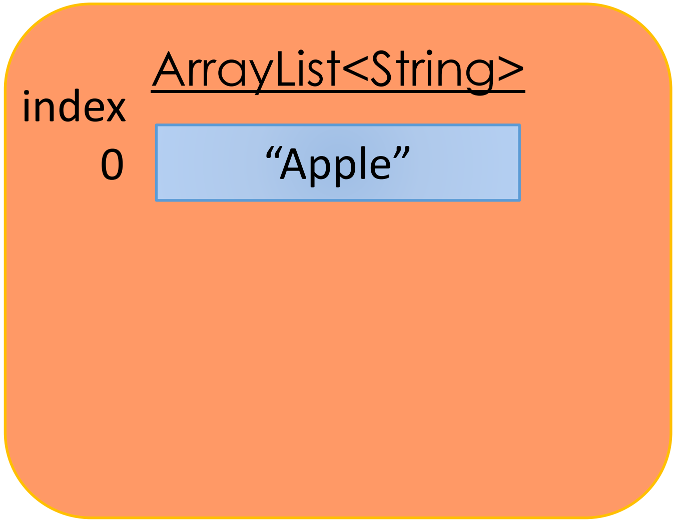
```
List<String> names = new ArrayList<>();
//Now names has size 0
```



ArrayList<String>

Adding element into ArrayList

```
List<String> names = new ArrayList<>();
//Now names has size 0
names.add("Apple"); //Now names has size 1
```



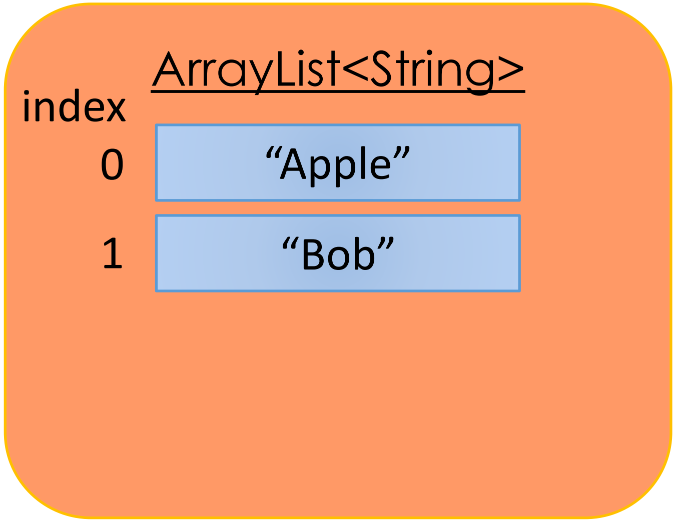
ArrayList<String>

index

0 “Apple”

Adding element into ArrayList

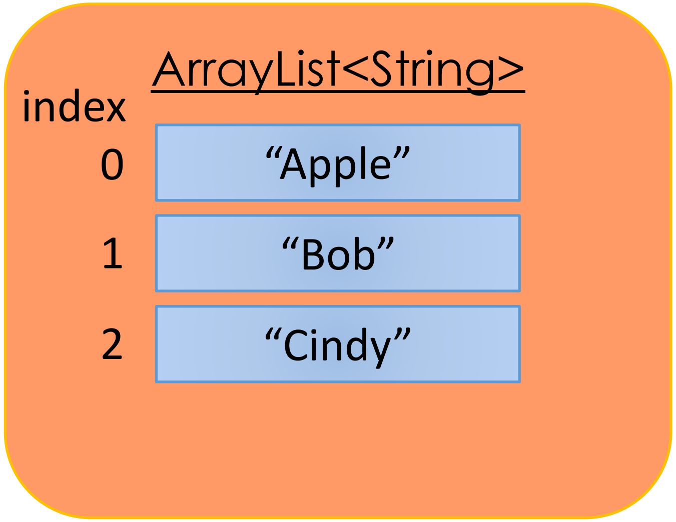
```
List<String> names = new ArrayList<>();
//Now names has size 0
names.add("Apple"); //Now names has size 1
names.add("Bob"); //Now names has size 2
```



index	ArrayList<String>
0	“Apple”
1	“Bob”

Adding element into ArrayList

```
List<String> names = new ArrayList<>();
//Now names has size 0
names.add("Apple"); //Now names has size 1
names.add("Bob"); //Now names has size 2
names.add("Cindy"); //Now names has size 3
```

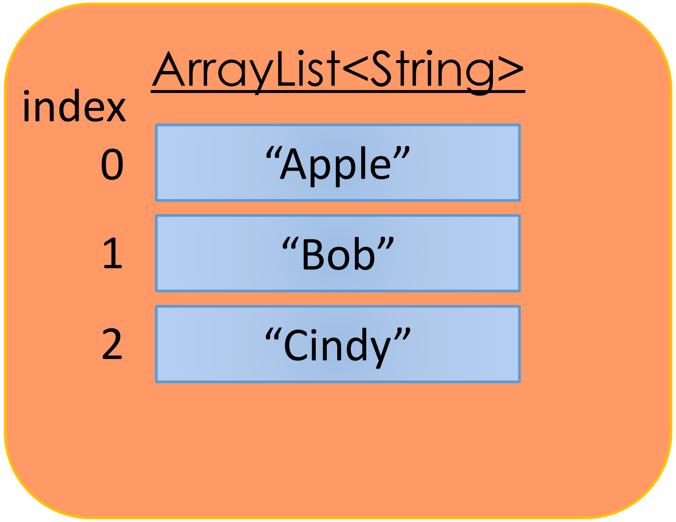


ArrayList<String>

index	
0	“Apple”
1	“Bob”
2	“Cindy”

Adding element into specific location of ArrayList

```
...//Now names has size 3
```



ArrayList<String>

index

0 “Apple”

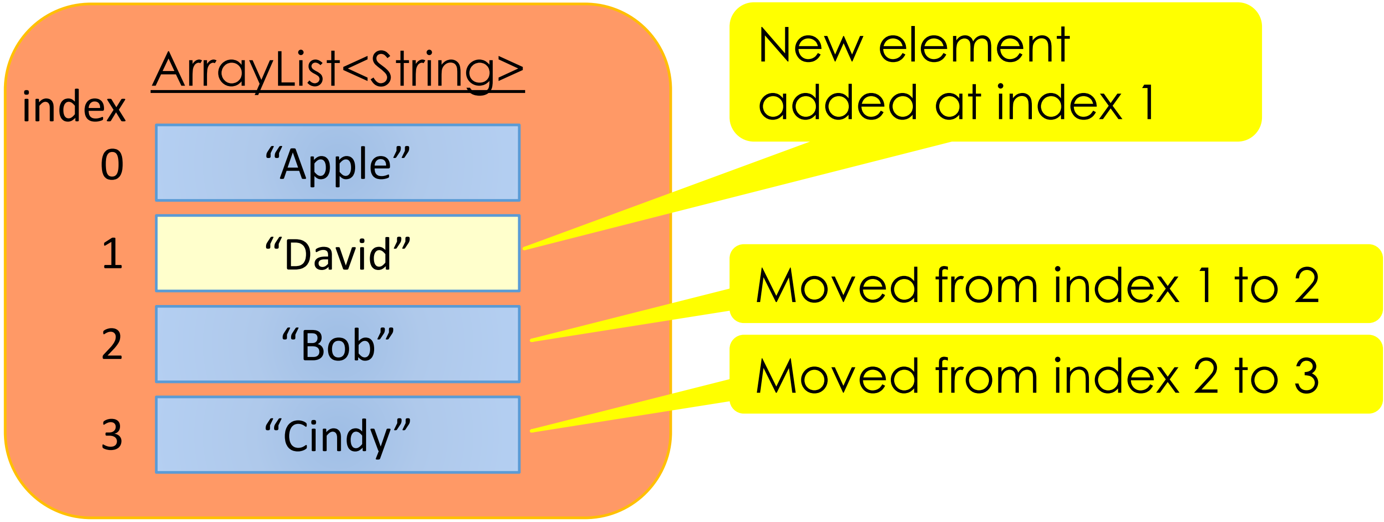
1 “Bob”

2 “Cindy”

Adding element into specific location of ArrayList

```
...//Now names has size 3
names.add(1, "David"); //add element at index 1
                       //Now names has size 4
```

index



ArrayList<String>

index	
0	“Apple”
1	“David”
2	“Bob”
3	“Cindy”

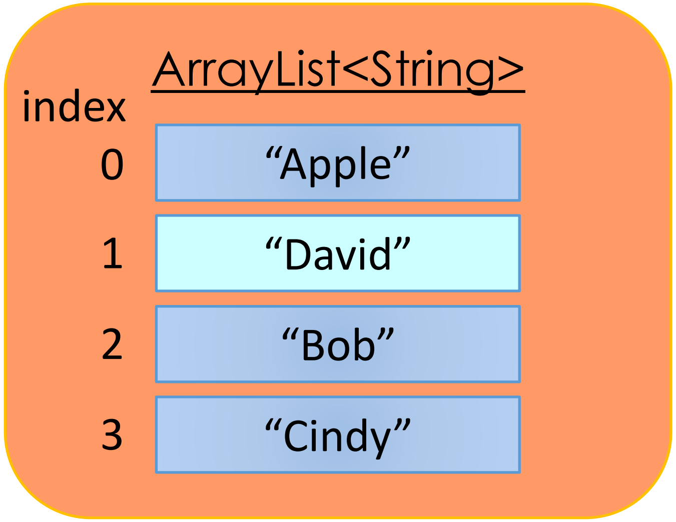
New element added at index 1

Moved from index 1 to 2

Moved from index 2 to 3

Removing element from ArrayList

```
...//Now names has size 4
```



ArrayList<String>

index

0 “Apple”

1 “David”

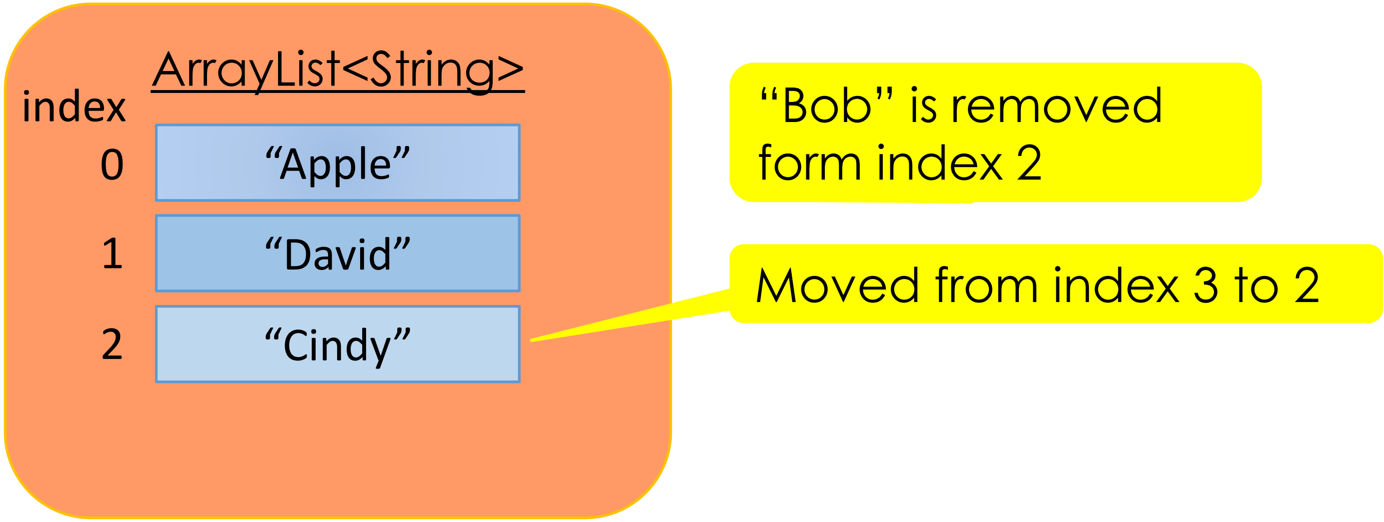
2 “Bob”

3 “Cindy”

Removing element from ArrayList

```
...//Now names has size 4
names.remove(2); //remove element at index 2
                 //Now names has size 3
```

index

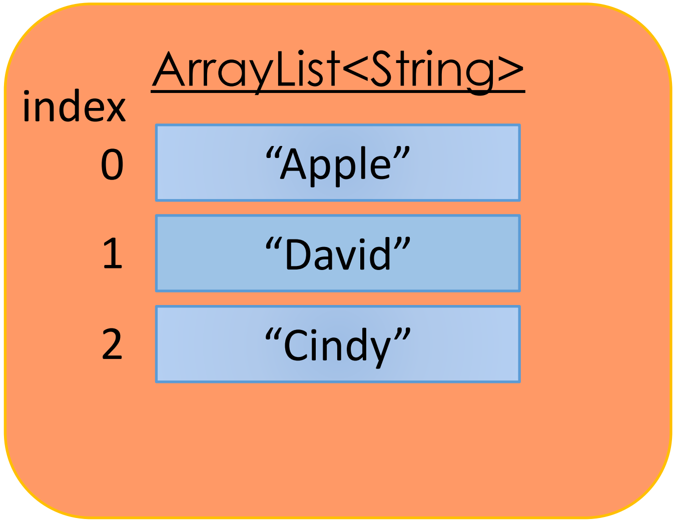


"Bob" is removed form index 2

Moved from index 3 to 2

Replacing an element with a different value

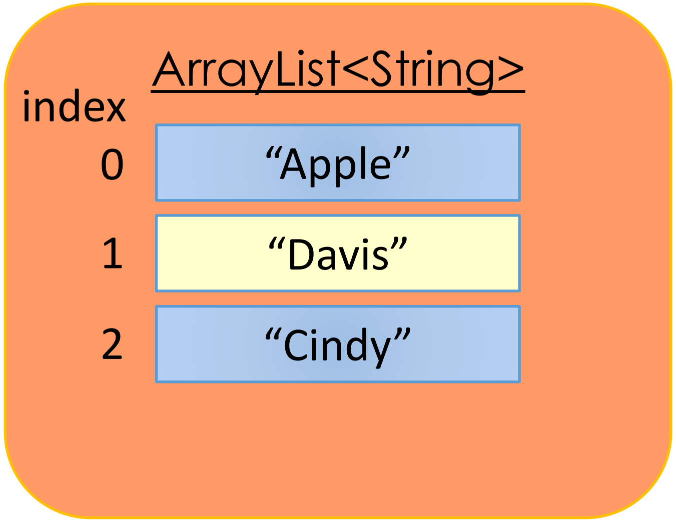
```
...//Now names has size 3
```



Replacing an element with a different value

```
...//Now names has size 3
names.set(1, "Davis"); //replace value at index 1
                       //names has size 3
```

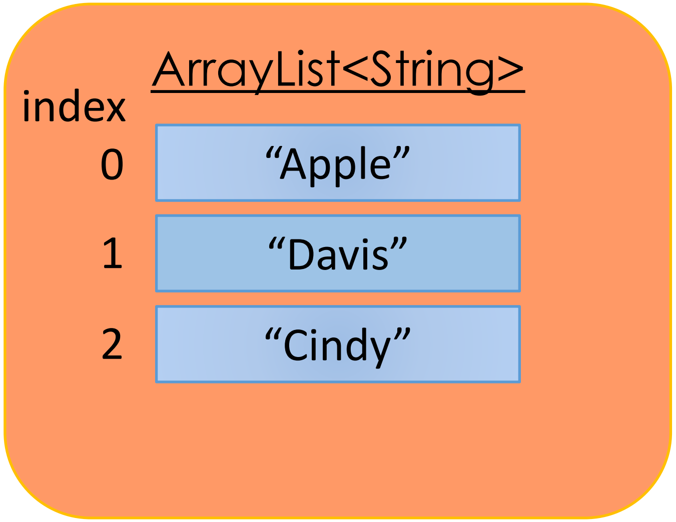
index



index	ArrayList<String>
0	“Apple”
1	“Davis”
2	“Cindy”

“David” is replaced by “Davis”

Getting an element from ArrayList



```
...//get the value at index 2
String name = names.get(2);
System.out.println(name); //prints "Cindy"
```

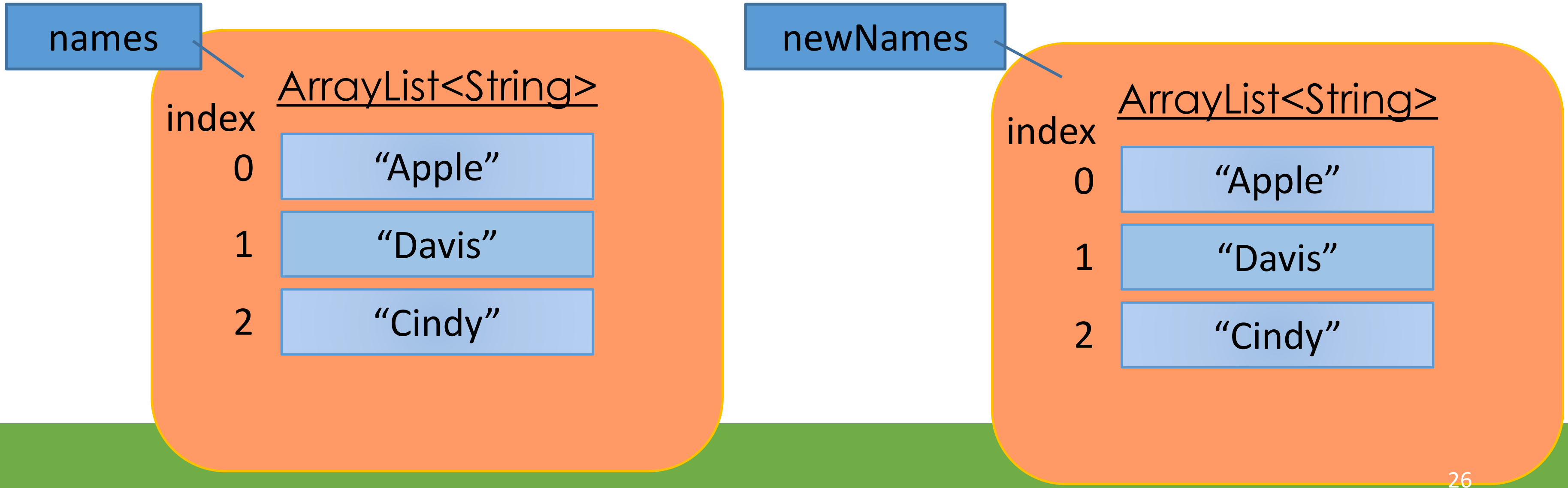
Copying ArrayLists

- To make a copy of an array list, create the copy and pass the original list into it

```
//create another copy of the names array list
List<String> newNames = new ArrayList<>(names);
```

Variable name of the new array list

Original array list



names

ArrayList<String>

index

0 "Apple"

1 "Davis"

2 "Cindy"

newNames

ArrayList<String>

index

0 "Apple"

1 "Davis"

2 "Cindy"

Comparing Array and Array List

Operation	Arrays	Array Lists
Get an element	x = values[3];	x = values.get(3);
Replace an element	values[3] = 12;	values.set(3, 12);
Number of elements	values.length	values.size()
Add an element	-	values.add(36);
Remove an element	-	values.remove(3);

Part B
Programming Exercises

Lab Exercise Submission

- Submit the following to Moodle
  - ***XXXXXXXX*_ lab10.docx**
  - ***XXXXXXXX*_lab10.zip**

*Replace “***XXXXXXXX***” with your **student ID**

Deadline: Before the next Monday noon

References

- Dean, J., & Dean, R. (2008). Introduction to programming with Java: A problem solving approach. Boston: McGraw-Hill.
- Forouzan, B. A., & Gilberg, R. F. (2007). Computer science: A structured programming approach using C (3rd ed.). Boston, MA: Thomson Course Technology.
- Gaddis, T. (2016). Starting out with Java (6th ed.). Pearson.
- Liang, Y. D. (2013). *Introduction to Java programming: Comprehensive version*. (8th ed.). Pearson.
- Schildt, H. (2006). *Java a beginner's guide*. New York: McGraw Hill.
- Wu, C. T. (2010). *An introduction to object-oriented programming with Java*. Boston: McGraw Hill Higher Education
- Xavier, C. (2011). Java programming: A practical approach. New Delhi: Tata McGraw Hill.
- Zakhour, S., Kannan, S., & Gallardo, R. (2013). The Java tutorial: A short course on the basics (5th ed.).
- yet another insignificant Programming Notes. (n.d.). Retrieved from https://www3.ntu.edu.sg/home/ehchua/programming