CS3231
Object Oriented Programming I

Name: _________________________________ (   ) Date: ________________

Chapter 12: ArrayList

Lesson Objectives

After completing the lesson, the student will be able to

- *create and use ArrayList*
- *understand and use wrapper classes for ArrayList*
- *copying ArrayList*
- *sorting an ArrayList of basic data types*

12.1 ArrayList Class

ArrayList is a class in the standard Java library. It is similar to arrays but more flexible in that it is resizable (i.e. can grow and shrink). In addition, the ArrayList class supplies methods for many common tasks, such as inserting and removing elements.

However there are two main disadvantages of ArrayList over arrays:
- ArrayList are less efficient than arrays.
- The base type of an ArrayList must be class type. Thus if you want an ArrayList of primitive data type, e.g. int, you must simulate this structure with an ArrayList of Integer values, where Integer is the wrapper class whose objects simulate int value. We will discuss about Wrapper class later.

12.1.1 Creating an ArrayList

The ArrayList class is in the package java.util. Thus to use the ArrayList class, we must always import the package.

```
import java.util.*;

//or
import java.util.ArrayList;
```

Constructor	Description
ArrayList()	Constructs an empty list with an initial capacity of ten.
ArrayList(int initialCapacity)	Constructs an empty list with the specified initial capacity.

An ArrayList can be declared and created as such:
```
ArrayList<String> list = new ArrayList<String>();
ArrayList<String> list = new ArrayList<String>(20);
```

12.1.2 Methods in ArrayList Class

Size of ArrayList

int	size()	Returns the number of elements in this list.

list.size() returns size of the list.

Adding element to a list

void	**add**(int index, **E** element)	Inserts the specified element at the specified position in this list.
boolean	**add**(**E** e)	Appends the specified element to the end of this list.

Example1

PROGRAM 12-1

```
import java.util.ArrayList;

public class Example1 {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>(20);
        System.out.printf("size=%d\n", list.size());
        list.add("hello");
        list.add("Ni Hao");
        list.add("Galcia");
        list.add("Aloha");
        System.out.printf("size=%d\n", list.size());
        list.add(2, "malacum");
        System.out.println(list);
    }
}
```

PROGRAM OUTPUT

```
size=0
size=4
[hello, Ni Hao, malacum, Galcia, Aloha]
```

To get objects out of the array list

E	get(int index)	Returns the element at the specified position in this list.

To set an array list element to a new value

E	**set**(int index, **E** element)	Replaces the element at the specified position in this list with the specified element.

The set method can only overwrite existing values. It is different from the add method, which adds a new object to the end of the array list.

To remove an element at a specified position

E	remove(int index)	Removes the element at the specified position in this list.
boolean	remove(Object o)	Removes the first occurrence of the specified element from this list, if it is present.
boolean	removeAll(Collection<?> c)	Removes from this list all of its elements that are contained in the specified collection.

PROGRAM 12-2

```
import java.util.ArrayList;
public class Example2 {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>(20);
        System.out.printf("size=%d\n", list.size());
        list.add("hello");
        list.add("Ni Hao");
        list.add("Galcia");
        list.add("Aloha");
        System.out.printf("size=%d\n", list.size());
        list.add(2, "malacum");
        System.out.println(list);

        System.out.println();
        System.out.printf("Element[3]=%s\n",list.get(3));//return "Galcia"

        list.set(0,"ohayo-gozai-mus");//replace "hello"
        list.remove(2); //remove "malacum

        System.out.println(list);
    }
}
```

Q1: What is the content of names after the following statements?

```
        ArrayList<String> names = new ArrayList<String>();
        names.add("John");
        names.add(0,"Mary");
        names.add("Luke");
        names.remove(1);
```

Refer to the Java API for a full list of methods in ArrayList class.
https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/util/ArrayList.html

Printing an Array List

There are two ways to print the contents in an ArrayList.

- Using the default toString()
- Using a for loop

The syntax of the for-each loop is as follows:

```
for(<data type> <variable name>: <ArrayList name>){
   <statements>;
}
```

Example:

PROGRAM 12-3

```
import java.util.ArrayList;
public class Example3 {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<String>();
        list.add("One");
        list.add("Two");
        list.add("Three");
        list.add("Four");
        list.add("Five");

        System.out.println(list); //the toString() will be invoked

        for(var s: list) //print each element with a for-each loop
            System.out.println(s);


    }
}
```

PROGRAM OUTPUT

```
[One, Two, Three, Four, Five]
One
Two
Three
Four
Five
```

What about an ArrayList of objects?
Consider the following classes:

PROGRAM 12-4A

```
public class Student {
    private String name;
    private String studentID;
    private int yearOfStudy;

    public Student(String name, String  studentID, int yearOfStudy) {
        this.name = name;
        this.studentID = studentID;
        this.yearOfStudy = yearOfStudy;
    }
}
```

PROGRAM 12-4B

```
import java.util.*;
public class Example4 {
    public static void main(String[] args) {
        ArrayList<Student> studentList = new ArrayList<Student>();
        Student s1 = new Student("John","h071111",1);
        Student s2 = new Student("Mary","h082222",2);
        Student s3 = new Student("Tom","h093333",3);
        studentList.add(s1);
        studentList.add(s2);
        studentList.add(s3);

        for(Student s:studentList) {
            System.out.println(s);
        }
    }
}
```

What will be the output printed when PROGRAM 12-4B is executed? Why?

Make relevant changes to the code above such that the following output will be seen:
(h071111, John, 1)
(h082222, Mary, 2)
(h093333, Tom, 3)

What do you think will be seen if we do the following:
`System.out.println(studentList);`

12.2 Wrappers and Auto-Boxing

12.2.1 Wrapper Class

Sometimes, you may want to use a value of a primitive data type but need the value to be an object of a class type (for example, integers to be used in ArrayList).

Wrapper classes provide a class type corresponding to each of the primitive data types.

To convert a value of a primitive type to an “equivalent” value of a class type, you create an object of the corresponding wrapper class using the primitive type value as an argument to the wrapper class constructor.

For example:
```
Integer x = new Integer(42);
```
x is now an object of class Integer, having a value of 42.

This process of going from a value of a primitive type to an object of its wrapper class is sometimes called ***boxing***.

To go in the reverse direction (i.e. from object to primitive type):
```
int y = x.intValue();
```

This is known as ***unboxing***.
The wrapper class contains a list of static methods and instances.

For example, we may find the largest and smallest integer value as such:
```
Integer.MAX_VALUE
Integer.MIN_VALUE
```

The static methods parse and toString are important for converting data from one data type to String and vice versa.

For example:
```
Double.toString(12.43); // converts double value to String
Double.parseDouble(“134.23”); //converts String 123.23 to double value
```

All primitive wrapper classes (Integer, Byte, Long, Float, Double, Character, Boolean and Short) are immutable in Java, so operations like addition and subtraction create a new object and not modify the old.

For a full list of the methods in the respective wrapper classes, refer to the API.

12.2.2 Automatic Boxing and Unboxing

Starting with Java version 5, conversion between primitive types and the corresponding wrapper classes is automatic. This is known as automatic boxing.

The code on the left may be re-written as that shown on the right:

`Integer a = new Integer(47);`
`Double b = new Double(54.23);`
`Character c = new Character('A');`
	`Integer a = 47;`
	`Double b = 54.23;`
	`Character c = 'A';`

Automatic unboxing can also be performed as such:

`int x = a.intValue();`
`double y = b.doubleValue();`
`char z = c.charValue();`
	`int x = a;`
	`double y = b;`
	`char z = c;`

Automatic boxing and unboxing may also be combined:

`Double price = new Double(19.90);`
`price = new Double(price.doubleValue() + 5.12);`
	`Double price = 19.90;`
	`price = price + 5.12;`

12.2.3 ArrayList of Primitive Data Types

Since each of the respective primitive data type can be converted to class type with their wrapper classes, we may declare ArrayList of primitive data type

For example:

```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(65);
list.set(0,42);
```

For example:

PROGRAM 12-5

```
import java.util.ArrayList;
public class Example5 {
    public static void main(String[] args) {
        int sum=0;
        ArrayList<Integer> list = new ArrayList<Integer>();
        list.add(2000);
        list.add(3000);

        for(int i: list) {
            sum+= i;
        }
        System.out.println(sum);
    }
}
```

Q: Modify the above codes so that you can increment every element in the Array List by 100?

12.3 Copying An ArrayList

Consider the following program:

PROGRAM 12-6

```
1   import java.util.ArrayList;
2   
3   public class TestCopy {
4       public static void main(String[] args) {
5   
6           ArrayList<Integer> a = new ArrayList<Integer>();
7           for(int i=0; i<10; i++){
8               a.add(i);
9           }
10  
11          ArrayList<Integer> b = a;
12          ArrayList<Integer> c = (ArrayList<Integer>) a.clone();
13          a.set(0, -99);
14          System.out.println("List a: " + a);
15          System.out.println("List b: " + b);
16          System.out.println("List c: " + c);
17  
18          System.out.println("-----------------------------------" );
19          ArrayList<String> d = new ArrayList<String>();
20          for(int i=0; i<10; i++){
21              d.add((char)(i+65)+"");
22          }
23  
24          ArrayList<String> e = (ArrayList<String>) d.clone();
25          d.set(0, "hello");
26          System.out.println("List d: " + d);
27          System.out.println("List e: " + e);
28      }
29  }
```

PROGRAM OUTPUT

```
List a: [-99, 1, 2, 3, 4, 5, 6, 7, 8, 9]
List b: [-99, 1, 2, 3, 4, 5, 6, 7, 8, 9]
List c: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
-----------------------------------------
List d: [hello, B, C, D, E, F, G, H, I, J]
List e: [A, B, C, D, E, F, G, H, I, J]
```

In line 11, as we've seen with any object, using assignment just makes two variables refer to the same ArrayList object. Hence, any changes made to the elements in the original ArrayList a, will affect that in ArrayList b as see in the Program Output.

ArrayList class has a method called clone() which we can use for copying. In line 12, the clone() method is called to make a copy of ArrayList a. However, note that the clone() method returns a **shallow copy** of the list as stated in the Java API:

`Object`	`clone()`	Returns a shallow copy of this `ArrayList` instance.

As the return type of the clone() method is a generic Object type. We need to typecast the Object back to an ArrayList using the typecast operator.

Note once again that **all primitive wrapper classes (Integer, Byte, Long, Float, Double, Character, Boolean and Short) and String are immutable in Java**, so when **`a.set(0, -99)`** is done in line 13, a new object is created and put in index 0 of ArrayList a. Hence, the changes in ArrayList a, does not affect that in ArrayList c.

Can clone() method then be used to make a copy of an ArrayList of say Student objects?

Consider the following program:

PROGRAM 12-7A

```
1   public class Student {
2       private String name;
3       private String studentID;
4       private int yearOfStudy;
5   
6       public Student(String name, String  studentID, int yearOfStudy) {
7           this.name = name;
8           this.studentID = studentID;
9           this.yearOfStudy = yearOfStudy;
10      }
11  
12      public Student(Student s){
13          this.name = s.name;
14          this.studentID = s.studentID;
15          this.yearOfStudy = s.yearOfStudy;
16      }
17  
18      public void setName(String name){
19          this.name = name;
20      }
21      public String toString(){
22          return "(" + studentID + ", " + name + ", " + yearOfStudy + ")";
23      }
24  }
```

PROGRAM 12-7B

```
1   import java.util.ArrayList;
2   
3   public class TestCopy2 {
4       public static void main(String[] args) {
5           ArrayList<Student> studentList = new ArrayList<Student>();
6           studentList.add(new Student("John","h071111",1));
7           studentList.add(new Student("Mary","h082222",2));
8           studentList.add(new Student("Tom","h093333",3));
9   
10          ArrayList<Student> shallowCopy = (ArrayList<Student>) studentList.clone();
11  
12          ArrayList<Student> deepCopy =  new ArrayList<Student>();
13          for(int i=0; i<studentList.size(); i++){
14              Student s = studentList.get(i);
15              deepCopy.add(new Student(s));
16          }
17  
18          studentList.get(0).setName(">_<"); //update name of first student
19          System.out.println("Original: " + studentList);
20          System.out.println("Shallow Copy: " + shallowCopy);
21          System.out.println("Deep Copy: " + deepCopy);
22      }
23  }
```

PROGRAM OUTPUT

```
Original: [(h071111, >_<, 1), (h082222, Mary, 2), (h093333, Tom, 3)]
Shallow Copy: [(h071111, >_<, 1), (h082222, Mary, 2), (h093333, Tom, 3)]
Deep Copy: [(h071111, John, 1), (h082222, Mary, 2), (h093333, Tom, 3)]
```

In line 10 above, calling clone() method to copy an ArrayList of Student objects will result in a shallow copy. Hence, edits made to `studentList` will affect that in `shallowCopy` list.

In lines 12 to 16, a for loop is used to loop through every element in the studentList. In Line 15, a deep copy of the Student object is added to `deepCopy` list. Hence, edits made to `studentList` will NOT affect that in `deepCopy` list.

12.4 Copying ArrayList to Array

Sometimes it may be useful to copy the contents of the ArrayList into an Array. With JDK 11, you can quickly do this using just one line of code using the method toArray() in ArrayList class.

For example:

PROGRAM 12-8

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

public class ListToArray{
   public static void main(String[] args) {
      ArrayList<String> list = new ArrayList<String>(20);
      list.add("hello");
      list.add("Ni Hao");
      list.add("Galcia");
      list.add("Aloha");
      String[] newarraylist = list.toArray(new String[list.size()]);
      newarraylist[0] = "BYE!";
      System.out.println(list);
      System.out.println(Arrays.toString(newarraylist));

      ArrayList<Student> studentList = new ArrayList<Student>();
      studentList.add(new Student("John","h071111",1));
      studentList.add(new Student("Mary","h082222",2));
      studentList.add(new Student("Tom","h093333",3));
      Student[] newStudentList = studentList.toArray(new Student[studentList.size()]);
      newStudentList[0].setName(">_<");
      System.out.println(studentList);
      System.out.println(Arrays.toString(newStudentList));
   }
}
```

PROGRAM OUTPUT

```
[hello, Ni Hao, Galcia, Aloha]
[BYE!, Ni Hao, Galcia, Aloha]
[(h071111, >_<, 1), (h082222, Mary, 2), (h093333, Tom, 3)]
[(h071111, >_<, 1), (h082222, Mary, 2), (h093333, Tom, 3)]
```

However, once again, please note that a shallow copy is being copied to the array. Hence, editing the attributes of the object via the array will also edit the object in the arraylist.

If you would instead wish to copy contents of an Array to an ArrayList, you can use the addAll() method in Collections class as follows:

```java
String[] array = {"a", "b", "c", "d", "e"};
ArrayList<String> list1 = new ArrayList<String>();
Collections.addAll(list1, array);
System.out.println(list1);
```

12.5 Sorting an ArrayList

Collections API’s utility class Collections provide a handy way to sort an ArrayList in natural ordering provided all elements in the list must implement the Comparable interface.

For example, consider an ArrayList consisting of String elements.

PROGRAM 12-9

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

public class SimpleSorting {
        public static void main(String[] args) {
            ArrayList<String> locationList = new ArrayList<String>();
            locationList.add("California");
            locationList.add("Texas");
            locationList.add("Seattle");
            locationList.add("New Delhi");

            Collections.sort(locationList);

            for (String location : locationList) {
                System.out.println("Location is: " + location);
            }
        }
}
```

PROGRAM OUTPUT

```
Location is: California
Location is: New Delhi
Location is: Seattle
Location is: Texas
```

The same can be applied to an ArrayList of Integer, Double, Character etc.

However, if you wish to use Collections.sort() on ArrayList of custom objects (e.g. a Person class), then the Person class needs to implement the Comparable interface.

We will discuss more on interfaces in the next module.