Lecture 3



Java PROGRAMMING (2)

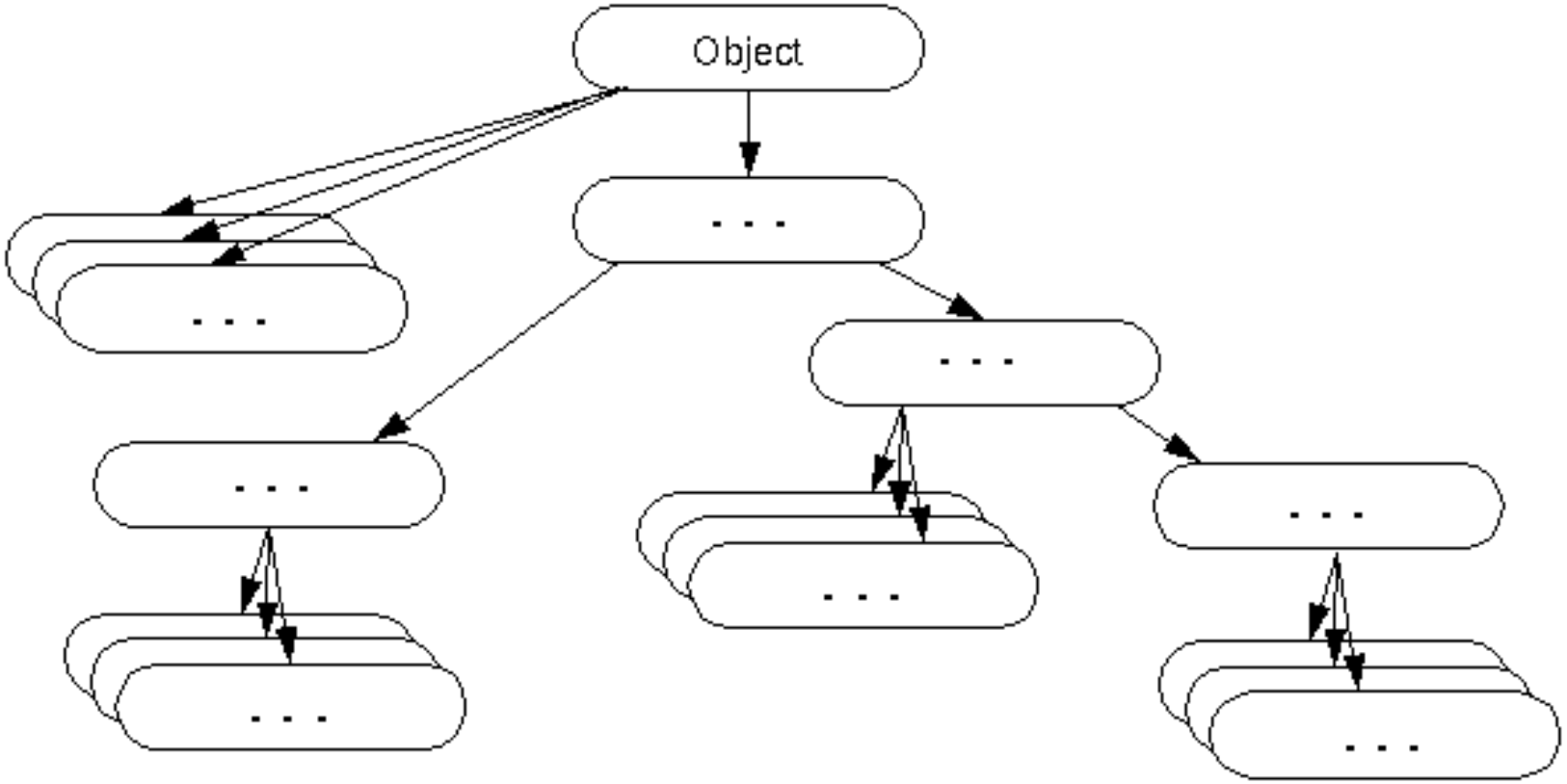
BY: DR. MOHAMMED ABDELFATTAH

Lecture Objectives

- Object Class
- Wrapper Classes
  - Objects
  - Methods
  - Converting
- Date and Time
- Enums
  - Inside Classes
  - With Switch
  - Looping
  - Verses Classes
- ArrayList
  - Add Items
  - Access Items
  - Edit Items
  - Remove Items
  - Clear Items
  - Size
  - Types
  - Looping
  - Sorting
- List Interface

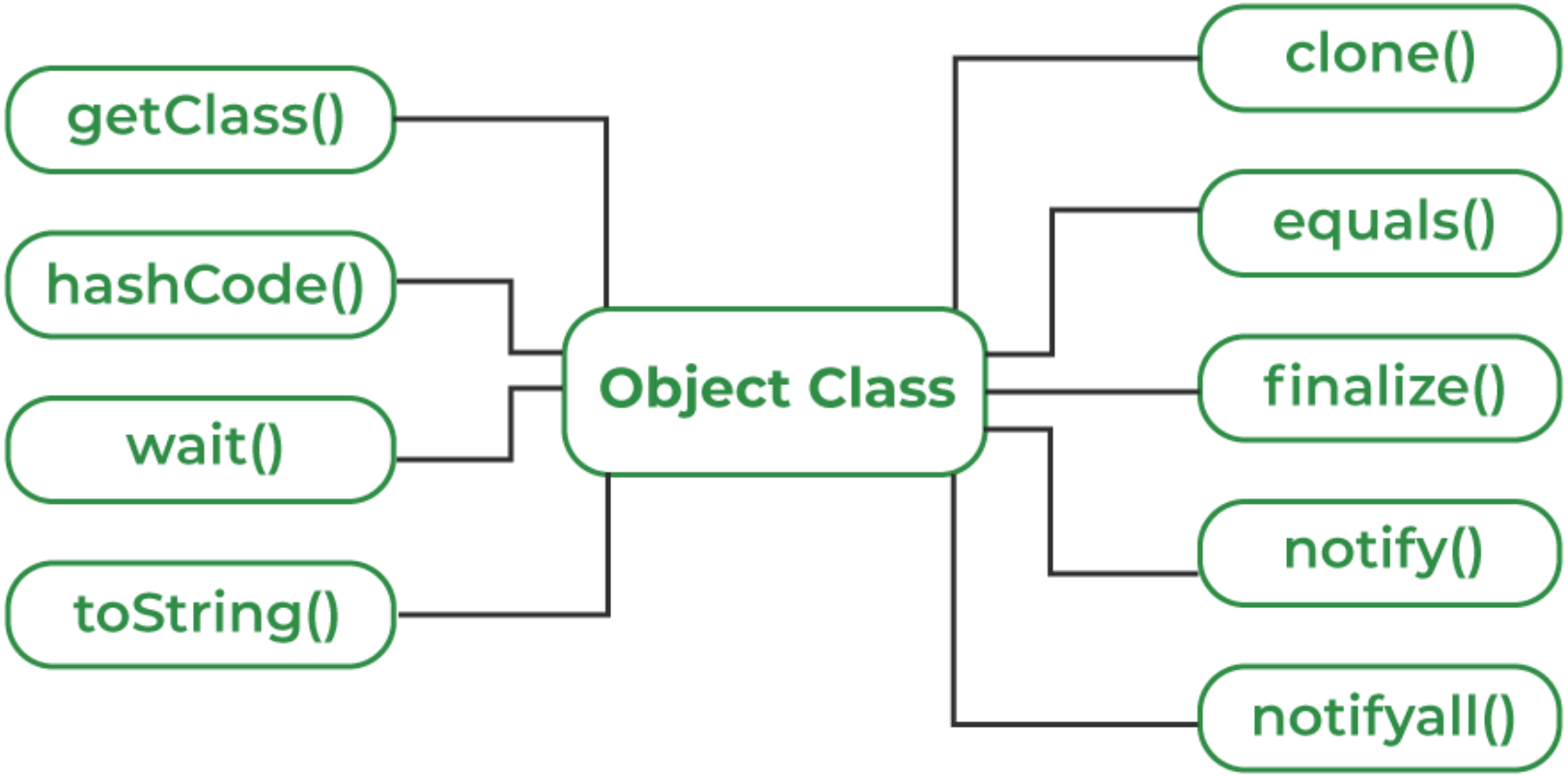
Object Class

All classes inherit methods from the following classes: **java.lang.Object**



Object Class' Methods

The object class has many methods, such as:



Object Class' Methods Description

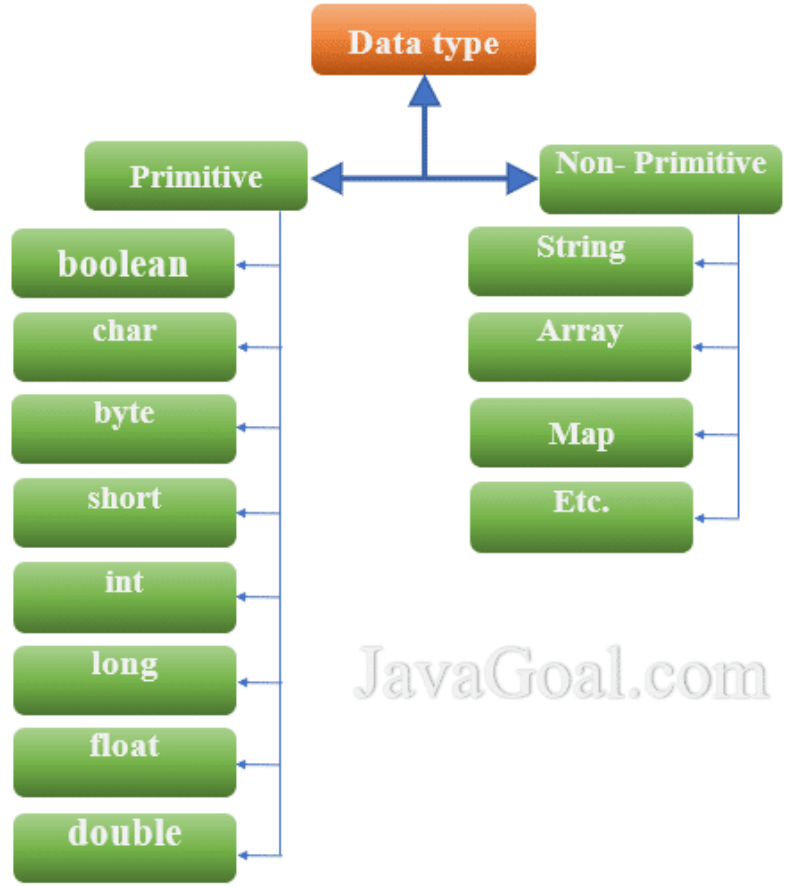
The Object class provides many methods, such as:

Method	Description
public int hashCode()	returns the hashcode number for this object.
public boolean equals(Object obj)	compares the given object to this object.
protected Object clone() throws CloneNotSupportedException	creates and returns the exact copy (clone) of this object.
public String toString()	returns the string representation of this object.

Wrapper Classes

Wrapper classes provide a way to use primitive data types (int, boolean, etc..) as objects.

Sometimes you must use wrapper classes, for example when working with **Collection** objects, such as ArrayList, where **primitive types cannot be used** (the list can only store objects).




```
// Invalid
ArrayList<int> myNumbers = new ArrayList<int>();

// Valid
ArrayList<Integer> myNumbers = new ArrayList<Integer>();
```

Wrapper Classes

The table below shows the primitive type and the equivalent wrapper class.

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

Creating Wrapper Objects

To create a wrapper object, use the wrapper class instead of the primitive type. To get the value, you can just print the object.

```
public class Main {
  public static void main(String[] args) {
    Integer myInt = 5;
    Double myDouble = 5.99;
    Character myChar = 'A';

    System.out.println(myInt);
    System.out.println(myDouble);
    System.out.println(myChar);
  }
}
```

```
//Output

5

5.99

A
```

Wrapper Objects' Methods

Since you're now working with objects, you can use certain methods to get information about the specific object. For example, the following methods are used to get the value associated with the corresponding wrapper object, as:

- **intValue(),**
- **byteValue(),**
- **shortValue(),**
- **longValue(),**
- **floatValue(),**
- **doubleValue(),**
- **charValue(),**
- **booleanValue().**

Example

```
public class Main {
  public static void main(String[] args) {
    Integer myInt = 5;
    Double myDouble = 5.99;
    Character myChar = 'A’;

    System.out.println(myInt.intValue());
    System.out.println(myDouble.doubleValue());
    System.out.println(myChar.charValue());
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    Integer myInt = 5;
    Double myDouble = 5.99;
    Character myChar = 'A’;

    System.out.println(myInt);
    System.out.println(myDouble);
    System.out.println(myChar);
  }
}
```

```
//Output
5
5.99
A
```

Convert Wrapper Objects

Another useful method is the toString() method, which is used to **convert wrapper objects** to strings.

In the following example, we convert an Integer to a String, and use the length() method of the String class to output the length of the "string“.

```
public class Main {
  public static void main(String[] args) {
    Integer myInt = 100;
    String myString = myInt.toString();

    System.out.println("My number as a string: "+ myString);
    System.out.println("String length: "+myString.length());
  }
}
```

//Output

My number as a string: 100

String length: 3

Date and Time

Java does not have a built-in Date class, but we can import the java.time package to work with the date and time API. The package includes many date and time classes.

Class	Description
LocalDate	Represents a date (year, month, day (yyyy-MM-dd))
LocalTime	Represents a time (hour, minute, second and nanoseconds (HH-mm-ss-ns))
LocalDateTime	Represents both a date and a time (yyyy-MM-dd-HH-mm-ss-ns)
DateTimeFormatter	Formatter for displaying and parsing date-time objects

Display Current Date

To display the current date, import the java.time.LocalDate class, and use its now() method.

```
import java.time.LocalDate; // import the LocalDate class


public class Main {
  public static void main(String[] args) {
    LocalDate myObj = LocalDate.now(); // Create a date object
    System.out.println(myObj); // Display the current date
  }
}
```

//Output

2024-04-23

Display Current Time

To display the current time (hour, minute, second, and nanoseconds), import the java.time.LocalTime class, and use its now() method.

```
import java.time.LocalTime; // import the LocalTime class


public class Main {
  public static void main(String[] args) {
    LocalTime myObj = LocalTime.now();
    System.out.println(myObj);
  }
}
```

//Output

18:21:59.301319200

Display Current Date and Time

To display the current date and time, import the java.time.LocalDateTime class, and use its now() method.

```
// import the LocalDateTime class
import java.time.LocalDateTime;


public class Main {
  public static void main(String[] args) {
    LocalDateTime myObj = LocalDateTime.now();
    System.out.println(myObj);
  }
}
```

```
//Output

2024-04-23T18:24:32.677091800
```

Formatting Date and Time

The "T" in the example above is used to separate the date from the time. You can use the DateTimeFormatter class with the ofPattern() method in the same package to format or parse date-time objects. The following example will remove both the "T" and nanoseconds from the date-time.

```
import java.time.LocalDateTime; // Import the LocalDateTime class
import java.time.format.DateTimeFormatter; // Import the DateTimeFormatter class

public class Main {
  public static void main(String[] args) {
    LocalDateTime myDateObj = LocalDateTime.now();
    System.out.println("Before formatting: " + myDateObj);
    DateTimeFormatter myFormatObj = DateTimeFormatter.ofPattern("dd-MM-yyyy HH:mm:ss");

    String formattedDate = myDateObj.format(myFormatObj);
    System.out.println("After formatting: " + formattedDate);
  }
}
```

//Output
Before formatting: 2024-04-23T18:27:58.498864200
After formatting: 23-04-2024 18:27:58

Formatting Date and Time

```java
import java.time.Instant;

// Import the Date package
import java.util.Date;

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

        // create a date of current time
        Date date = Date.from(Instant.now());

        // print the date instance
        System.out.println("Date: " + date.toString());
    }
}
```

//Output

Date: Tue Apr 23 23:22:26 EET 2024

Formatting Date and Time

The ofPattern() method accepts all sorts of values, if you want to display the date and time in a different format.

Value	Example
yyyy-MM-dd	"1988-09-29"
dd/MM/yyyy	"29/09/1988"
dd-MMM-yyyy	"29-Sep-1988"
E, MMM dd yyyy	"Thu, Sep 29 1988"

Enums

An enum is a special "class" that represents a group of constants (unchangeable variables, like final variables).

To create an enum, use the enum keyword (instead of class or interface), and separate the constants with a comma. Note that they should be in uppercase letters.

```
enum Level
{
  LOW,
  MEDIUM,
  HIGH
}
```

```
Level myVar = Level.MEDIUM;
```

Note Please: Enum is short for "**enumerations**", which means "specifically listed".

Enum Inside Classes

You can also have an enum inside a class.

```
public class Main {

  enum Level {
    LOW,
    MEDIUM,
    HIGH
  }

  public static void main(String[] args) {

    Level myVar = Level.MEDIUM;
    System.out.println(myVar);
  }
}
```

```
//Output

MEDIUM
```

Enum With Switch Statements

Enums are often used in switch statements to check for corresponding values.

```
enum Level {LOW, MEDIUM,HIGH}

public class Main {
  public static void main(String[] args) {
    Level myVar = Level.MEDIUM;

    switch(myVar) {
      case LOW:
        System.out.println("Low level");
        break;
      case MEDIUM:
         System.out.println("Medium level");
        break;
      case HIGH:
        System.out.println("High level");
        break;
    }
  }
}
```

//Output

Medium level

Example 2

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

    System.out.println(Mobile.Motorola);  //print an Enum

    Mobile mobile = Mobile.Samsung;
    if(mobile == Mobile.Samsung) {          //Usage in IF statment
       System.out.println("Matched");
    }

    switch(mobile) {                         //Usage in Swith statment
       case Samsung:
          System.out.println("Samsung");
          break;
       case Nokia:
          System.out.println("Nokia");
          break;
       case Motorola:
          System.out.println("Motorola");
    }
 }
}
enum Mobile {Samsung, Nokia,Motorola}
```

```
//Output

Motorola

Matched

Samsung
```

Loop Through Enums

The enum type has a values() method, which returns an array of all enum constants. This method is useful when you want to loop through the constants of an enum.

```
enum Level
{
  LOW,
  MEDIUM,
  HIGH
}
```

```
for (Level myVar : Level.values())
{
  System.out.println(myVar);
}
```

```
//Output
LOW
MEDIUM
HIGH
```

Enums Verses Classes

- An enum can, just like a class, have attributes and methods.
- The only difference is that enum constants are public, static and final (unchangeable - cannot be overridden).
- An enum **cannot** be used to create objects, and it **cannot** extend other **classes** (but it **can** implement **interfaces**).

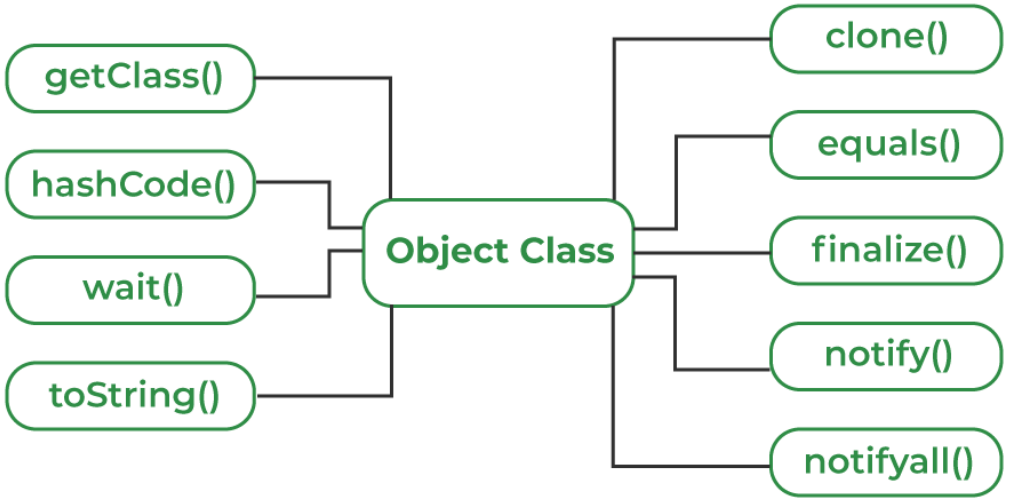
Why And When To Use Enums?

- Use enums when you have **values** that you know aren't going to change, like month days, days, colors, deck of cards, etc.

Enum Class Declaration

Following is the declaration for **java.lang.Enum** class

```
public abstract class Enum<E extends Enum<E>>
    extends Object
    implements Comparable<E>, Serializable
```



This class inherits methods from the following classes: **java.lang.Object**

Example



If we print the string representation of the enum using the enum directly, using toString(), or using the name() method, it will print the same string as declared.

```
enum WEEKDAY { MONDAY, TUESDAY, WEDNESDAY, THRUSDAY, FRIDAY, SATURDAY, SUNDAY }


System.out.println(WEEKDAY.MONDAY);
System.out.println(WEEKDAY.MONDAY.toString());
System.out.println(WEEKDAY.MONDAY.name());
```

```
//Output
MONDAY
MONDAY
MONDAY
```

Overriding Object Class' Methods: toString()




```
enum WEEKDAY
{
  // enum value constants
  MONDAY, TUESDAY, WEDNESDAY, THRUSDAY, FRIDAY, SATURDAY, SUNDAY;

  // override the toString() method for custom description
  @Override
  public String toString()
  {

     return switch(this){
             case MONDAY: yield "Day 1";
             case TUESDAY:yield "Day 2";
             case WEDNESDAY:yield "Day 3";
             case THRUSDAY:yield "Day 4";
             case FRIDAY:yield "Day 5";
             case SATURDAY:yield "DAY 6";
             case SUNDAY: yield "Day 7";
             };
  }
}
```

```
public class Main {
  public static void main(String[] args) {
    // invoke toString() internally
    System.out.println(WEEKDAY.MONDAY);
    // invoke toString explicitly
    System.out.println(WEEKDAY.TUESDAY.toString());
    // invoke name() method to get the default name
    System.out.println(WEEKDAY.WEDNESDAY.name());
 }
}
```

//Output
Day 1
Day 2
WEDNESDAY

ArrayList

The ArrayList class is a resizable array, which can be found in the java.util package.

The difference between a built-in array and an ArrayList in Java, is that 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 you want. The syntax is also slightly different.

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

ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList object
```

Remember: Array **indexes** start with 0: [0] is the **first element**. [1] is the **second element**, etc.

Add Items

The ArrayList class has many useful methods. For example, to add elements to the ArrayList, use the **add()** method.

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
  }
}
```

//Output

[Volvo, BMW, Ford, Mazda]

Access Items

To access an element in the ArrayList, use the **get()** method and refer to the index number.

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
    System.out.println(cars.get(0));
  }
}
```

```
//Output

[Volvo, BMW, Ford, Mazda]

Volvo
```

Edit Items

To modify an element, use the **set()** method and refer to the index number.

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);

    cars.set(0, "Opel");
    System.out.println(cars);
  }
}
```

```
//Output

[Volvo, BMW, Ford, Mazda]

[Opel, BMW, Ford, Mazda]
```

Remove Item: Using Index

To remove an element, use the **remove()** method and refer to the index number.

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);

    cars.remove(0);
    System.out.println(cars);
  }
}
```

```
//Output

[Volvo, BMW, Ford, Mazda]

[BMW, Ford, Mazda]
```

Remove Item: Using Value

To remove an element, use the **remove()** method and refer to the value of the item.

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);

    cars.remove("Mazda");
    System.out.println(cars);
  }
}
```

//Output

[Volvo, BMW, Ford, Mazda]

[Volvo, BMW, Ford]

Remove All Items

To remove all the elements in the ArrayList, use the **clear()** method:

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);

    cars.clear();
    System.out.println(cars);
  }
}
```

```
//Output

[Volvo, BMW, Ford, Mazda]

[]
```

ArrayList Size

To find out how many elements an ArrayList have, use the **size** method.

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);

    System.out.println(cars.size());
  }
}
```

//Output

[Volvo, BMW, Ford, Mazda]

4

Looping Through ArrayLists

Loop through the elements of an ArrayList with a for loop, and use the **size()** method to specify how many times the loop should run.

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");

    for (int i = 0; i < cars.size(); i++) {
      System.out.println(cars.get(i));
    }
  }
}
```

```
//Output

Volvo

BMW

Ford

Mazda
```

Looping Through ArrayLists

You can also loop through an ArrayList with the **for-each** loop:

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");

    for (String i : cars) {
      System.out.println(i);
    }
  }
}
```

```
//Output

Volvo

BMW

Ford

Mazda
```

Other ArrayList Types

Elements in an ArrayList are actually objects. In the examples above, we created elements (objects) of type "String". Remember that a String in Java is an object (not a primitive type). To use other types, such as int, you must specify an equivalent **wrapper class**: Integer. For other primitive types, use: Boolean for boolean, Character for char, Double for double, etc.. Create an ArrayList to store numbers (add elements of type Integer):

```
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(10);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(25);
    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
}
```

```
//Output

10

15

20

25
```

Sort ArrayLists' Items

Another useful class in the **java.util** package is the Collections class, which include the sort() method for sorting lists **alphabetically** or **numerically**.

```
import java.util.ArrayList;
import java.util.Collections;  // Import the Collections class

public class Main {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");

    Collections.sort(cars);  // Sort cars
    for (String i : cars) {
      System.out.println(i);
    }
  }
}
```

```
//Output

BMW

Ford

Mazda

Volvo
```

Example 2

```
import java.util.ArrayList;
import java.util.Collections;  // Import the Collections class

public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(33);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(34);
    myNumbers.add(8);
    myNumbers.add(12);

    Collections.sort(myNumbers);  // Sort myNumbers

    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
}
```

```
//Output
8
12
15
20
33
34
```

List Interface



The List interface extends Collection and declares the behavior of a collection that stores a sequence of elements.

- Elements can be **inserted or accessed** by their position in the list, using a zero-based index.
- A list may contain **duplicate** elements.
- In addition to the methods defined by Collection, List defines some of its own.
- Several of the list methods will throw an UnsupportedOperationException if the collection cannot be modified, and a ClassCastException is generated when one object is incompatible with another.

List Interface

```java
import java.util.*;

public class Main {
  public static void main(String[] args) {
    // Creating two lists using List interface
    List < Integer > list1 = new ArrayList < Integer > ();
    List < Integer > list2 = new ArrayList < Integer > ();
    // Adding elements to list 1
    list1.add(0, 10);
    list1.add(1, 20);
    // Printing list 1
    System.out.println("list1 : " + list1);
    // Adding elements to list 2
    list2.add(10);
    list2.add(20);
    list2.add(30);
    // Adding all elements of list 2 in list 1



  }
}
```

```java
    list1.addAll(1, list2);
    System.out.println("list1 : " + list1);
    // Printing list 2
    System.out.println("list2 : " + list2);
    // Removes an element from list 1 (from index 1)
    list1.remove(1);
    // Printing list 1
    System.out.println("list1 after removing an element: " + list1);
    // get() method
    System.out.println("list1 using get() : " + list1.get(2));
    // Replacing element
    list1.set(0, 50);
    // Printing the list 1
    System.out.println("list1 : " + list1);
```

//Output

list1 : [10, 20]
list1 : [10, 10, 20, 30, 20]
list2 : [10, 20, 30]
list1 after removing an element: [10, 20, 30, 20]
list1 using get() : 30
list1 : [50, 20, 30, 20]

Thanks

References:
https://www.w3schools.com & https://www.tutorialspoint.com & https://www.javatpoint.com