НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Object-Oriented Programming in the Java language

Part 6. Collections(1/2): Lists.

Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Just before we start…

Generics…

- *Generics* are a language feature introduced to Java in version 1.5. Before generics were introduced in Java, the Object base class was used as an alternative to generics.
- With generics, you write code for one type (say T) that is applicable for all types, instead of writing separate classes for each type.

Example

```
class BoxPrinter<T> {
    private T val;
    public BoxPrinter(T arg) {
        val = arg;
    }
    public String toString() {
        return "[" + val + "]";
    }
}
```

```
BoxPrinter<Integer> value1 =
        new BoxPrinter<Integer>(new Integer(10));
System.out.println(value1);
BoxPrinter<String> value2 =
        new BoxPrinter<String>("Hello world");
System.out.println(value2);
```

Notes for example

1. See the declaration of BoxPrinter:

   ```
   class BoxPrinter<T>
   ```

   You gave the BoxPrinter class a *type placeholder* <T>—the type name T within angle brackets "<" and ">" following the class name.

You can use this type name inside the class to indicate that it is a placeholder for the actual type to be provided later.

Notes for example

2. Inside the class you first use T in field declaration:

```
private T val;
```

You are declaring val of the *generic type*—the actual type will be specified later when you use BoxPrinter.

In main() , you declare a variable of type BoxPrinter for an Integer like this:

```
BoxPrinter<Integer> value1
```

Here, you are specifying that T is of type Integer—identifier T (a placeholder) is replaced with the type Integer.

So, the val inside BoxPrinter becomes Integer because T gets replaced with Integer.

Notes for example

3. Now, here is another place where you use T:

```
public BoxPrinter(T arg) {
    val = arg;
}
```

Similar to the declaration of val with type T, you are saying that the argument for BoxPrinter constructor is of type T.

Later in the main() method, when the constructor is called in new, you specify that T is of type Integer:

```
new BoxPrinter<Integer>(new Integer(10));
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Example

```
class Pair<T1, T2> {
    T1 object1;
    T2 object2;
    Pair(T1 one, T2 two) {
        object1 = one;
        object2 = two;
    }
    public T1 getFirst() {
        return object1;
    }
    public T2 getSecond() {
        return object2;
    }
}
```

```
Pair<Integer, String> worldCup =
        new Pair<Integer, String>(2018, "Russia");
System.out.println("World cup " + worldCup.getFirst() +
        " in " + worldCup.getSecond());
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Diamond Syntax

- To simplify your life, Java 1.7 introduced the diamond syntax, in which the type parameters may be omitted: you can just leave it to the compiler to infer the types from the type declaration. So, the declaration can be simplified as

```
Pair<Integer, String> worldCup =
        new Pair<>(2018, "Russia");
```

Note that it is a common mistake to forget the diamond operator < > in the initialization expression, as in

```
Pair<Integer, String> worldCup = new Pair( one: 2018,  two: "Russia");
```

Unchecked assignment: 'main.Pair' to 'main.Pair<java.lang.Integer,java.lang.String>' more... (Ctrl+F1)

Unchecked call to 'Pair(T1, T2)' as a member of raw type 'main.Pair' more... (Ctrl+F1)

ArrayList

- This lecture covers only one class from the Java Collection API: **ArrayList**. The rest of the classes from the Java Collection API are covered in next one.
- One of the reasons to include this class in the first part could be how frequently this class is used by all Java programmers.
- **ArrayList** is one of the most widely used classes from the Collections framework. It offers the best combination of features offered by an **array** and the **List** data structure.
- The most commonly used operations with a list are: **add** items to a list, **modify** items in a list, **delete** items from a list, and **iterate** over the items

Using of ArrayList

- One frequently asked question by Java developers is, “Why should I bother with an ArrayList when I can already store objects of the same type in an array?”
- The answer lies in the *ease of use of an ArrayList*.
- You can compare an ArrayList with a resizable array. As you know, once it’s created, you can’t increase or decrease the size of an array.
- On the other hand, an ArrayList automatically increases and decreases in size as elements are added to or removed from it.
- Also, unlike arrays, you don’t need to specify an initial size to create an ArrayList.

Important properties of ArrayList

- It implements the interface ***List***.
- It allows ***null*** values to be added to it.
- It implements all list operations (add, modify, and delete values).
- It allows duplicate values to be added to it.
- It maintains its insertion order.
- You can use either ***Iterator*** or ***ListIterator*** to iterate over the items of an ***ArrayList***.
- It supports generics, making it type safe.
  (You have to declare the type of the elements that should be added to an ***ArrayList*** with its declaration.)

Creating an ArrayList

Let's see:

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

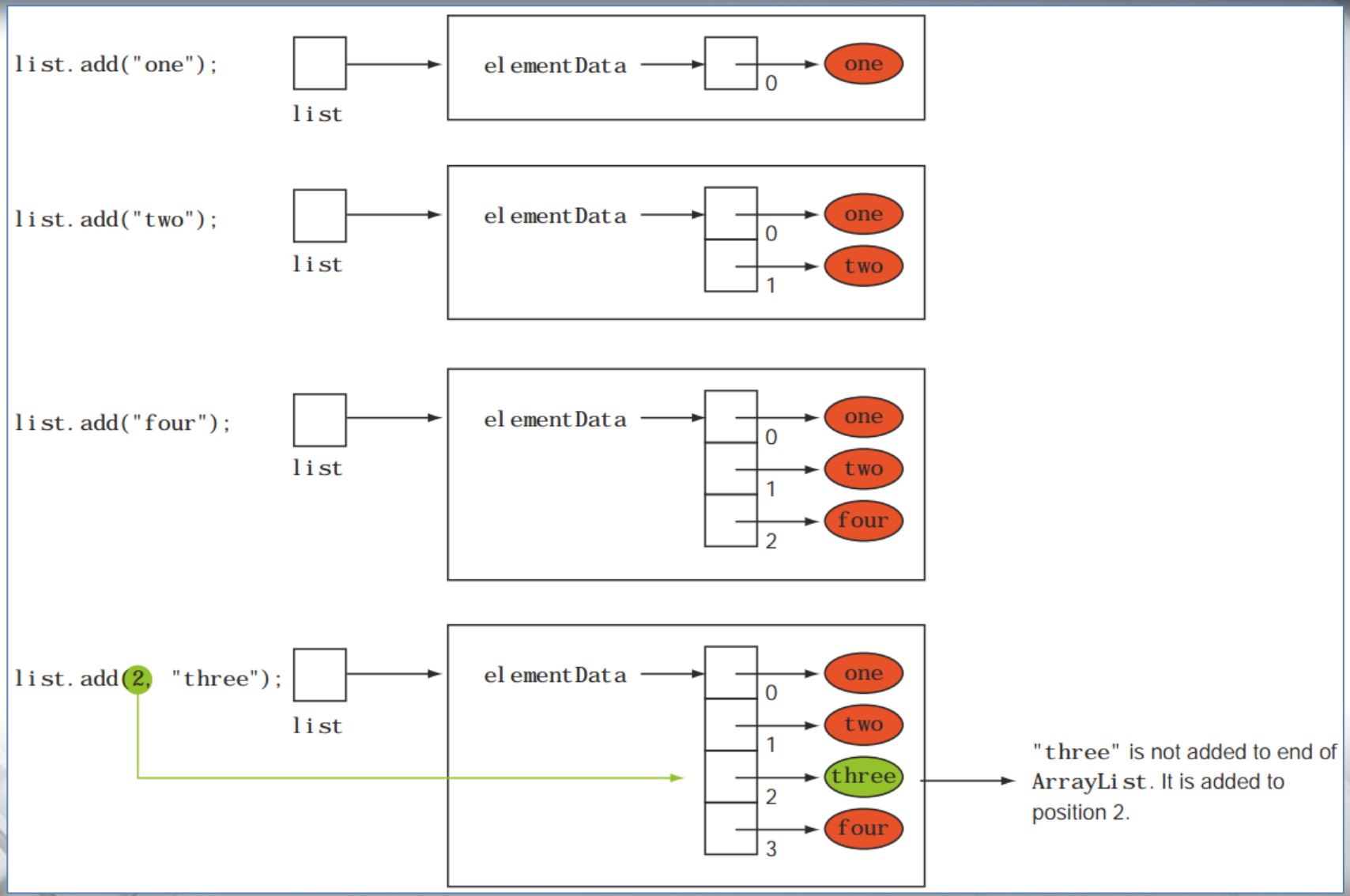
Starting with Java version 7, you can omit the object type on the right side of the equal sign and create an ArrayList as follows:

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

Adding elements to an ArrayList

```
ArrayList<String> list = new ArrayList<>();
list.add("one");
list.add("two");
list.add("four");
list.add(2, "three");
```

Adding elements to an ArrayList



Accessing elements of an ArrayList

Using enhanced for loop:

```
ArrayList<String> myArrList = new ArrayList<>();
myArrList.add("One");
myArrList.add("Two");
myArrList.add("Four");
myArrList.add(2, "Three");
for (String element : myArrList) {
    System.out.println(element);
}
```

Using ListIterator:

```
ListIterator<String> iterator =
        myArrList.listIterator();
while (iterator.hasNext()) {
    System.out.println(iterator.next());
}
```

Modifying the elements of an ArrayList

```
ArrayList<String> myArrList = new ArrayList<>();
myArrList.add("One");
myArrList.add("Two");
myArrList.add("Three");
myArrList.set(1, "One and Half");
for (String element:myArrList) {
    System.out.println(element);
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Deleting the elements of an ArrayList

```
ArrayList<String> myArrList = new ArrayList<>();
String s1 = "One";
String s2 = "Two";
String s3 = "Three";
String s4 = "Four";
myArrList.add(s1);
myArrList.add(s2);
myArrList.add(s3);
myArrList.add(s4);
myArrList.remove(1);
for (String element:myArrList) {
    System.out.println(element);
}
myArrList.remove(s3);
myArrList.remove("Four");
System.out.println();
for (String element : myArrList) {
    System.out.println(element);
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Example

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Other methods of ArrayList

ADDING MULTIPLE ELEMENTS TO AN ARRAYLIST

You can add multiple elements to an ArrayList from another ArrayList or any other class that's a subclass of Collection by using the following overloaded versions of method addAll:

- `addAll(Collection<? extends E> c)`
- `addAll(int index, Collection<? extends E> c)`

Other methods of ArrayList

ADDING MULTIPLE ELEMENTS TO AN ARRAYLIST

```
ArrayList<String> myArrList = new ArrayList<>();
myArrList.add("One");
myArrList.add("Two");
ArrayList<String> yourArrList = new ArrayList<>();
yourArrList.add("Three");
yourArrList.add("Four");
myArrList.addAll(1, yourArrList);
for (String val : myArrList) {
    System.out.println(val);
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Other methods of ArrayList

CLEARING ARRAYLIST ELEMENTS

You can remove all the ArrayList elements by calling clear on it.

```
ArrayList<String> myArrList = new ArrayList<>();
myArrList.add("One");
myArrList.add("Two");
myArrList.clear();
for (String val:myArrList) {
    System.out.println(val);
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Other methods of ArrayList

ACCESSING INDIVIDUAL ARRAYLIST ELEMENTS

- get(int index) — returns the element at the specified position in this list.
- size() — returns the number of elements in this list.
- contains(Object o) — returns true if this list contains the specified element.
- indexOf(Object o) — returns the index of the first occurrence of the specified element in this list, or -1 if this list doesn’t contain the element.
- lastIndexOf(Object o) — returns the index of the last occurrence of the specified element in this list, or -1 if this list doesn’t contain the element.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Other methods of ArrayList

ACCESSING INDIVIDUAL ARRAYLIST ELEMENTS

- Three methods: **contains**, **indexOf**, and **lastIndexOf**—require you to have an unambiguous and strong understanding of how to determine the equality of objects.
- ArrayList stores objects, and these three methods will compare the values that you pass to these methods with all the elements of the ArrayList

Equality of Objects

- By default, objects are considered equal if they are referred to by the same variable (the String class is an exception with its pool of String objects).
- If you want to compare objects by their state (values of the instance variable), override the equals method in that class.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Equality of Objects

The equals method implements an equivalence relation on non-null object references:

- ***It is reflexive**: for any non-null reference value x, x.equals(x) should return true.*
- ***It is symmetric**: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.*
- ***It is transitive**: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.*
- ***It is consistent**: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals() comparisons on the objects is modified.*
- *For any non-null reference value x, x.equals(null) should return false.*

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Other methods of ArrayList

CREATING AN ARRAY FROM AN ARRAYLIST

- You can use the method toArray to return an array containing all the elements in an ArrayList in sequence from the first to the last element.
- Method toArray creates a new array, copies the elements of the ArrayList to it, and then returns it.

Now comes the tricky part. No references to the returned array, which is itself an object, are maintained by the ArrayList. But the references to the individual ArrayList elements are copied to the returned array and are still referred to by the ArrayList

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Example

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Questions?

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Object-Oriented Programming in the Java language

Part 6. Collections(1/2): Lists.

Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua