Lec 11: static

Sam McCauley

March 4, 2026

Admin

- Midterm Friday
  - Practice midterm posted on Glow; hard copies also available from TAs
  - Review session in lab
- Page listing methods of data structures we've seen will also be on the midterm

Let’s go over removing the last element of an `ArrayList`

equals() Method

equals() method

- In Java, every time you compare objects you should use the `equals()` method

- Every `Object` has one

```
1  public boolean equals(Object other) {
```

How `equals()` is Used

- Used to see if two objects are equivalent
- Example: in `indexOf()` and `contains()` of `ArrayList.java`
- We did this in `LinkedList.java` and `DoublyLinkedList.java` as well

How `equals()` is Used

```java
// find the first occurrence of element and return its index
// if there is no occurrence, return -1
// Note: uses .equals() instead of == to compare objects
public int indexOf(E element) {
    for (int index = 0; index < numElems; index++) {
        if (element.equals(arr[index])) {
            return index;
        }
    }
    return -1;
}
// returns true if element is in the list; false otherwise
public boolean contains(E element) {
    return indexOf(element) != -1;
}
```

- Essentially all Java library data structures use `equals` in the same way

Writing an `equals()` method

- Let’s say we work at a museum
- Want to keep track of works of art
- Each has a title, artist, year, and reference number
- How do we determine if two works of art are equal?

Writing an `equals()` method

```
1  public class WorkOfArt {
2      private String title;
3      private String artist;
4      private int year;
5      private int reference;
6
7      public WorkOfArt() {
8          title = "";
9          artist = "";
10         year = 0;
11         reference = 0;
12     }
13     //assume standard getters and setters
```

What should `equals()` do?

Any suggestions?

1. Check if `title` is the same
2. Check if `reference` is the same

- Are there use cases where the first strategy is better?
- Are there use cases where the second strategy is better?

Checking Title Makes Sense When...

- Let's say the purpose of our `WorkOfArt` class is for users to search for a specific work
- They always search by title
- Then `equals()` lets us search in an `ArrayList<WorkOfArt>` directly

Checking Reference Number Makes Sense When...

- Let's say the purpose of our `WorkOfArt` class is for back-end museum curators to search for more information on a specific piece
- They always search by reference number
- Then `equals()` lets us search in an `ArrayList<WorkOfArt>` directly

Comparing the Strategies

- Takeaway: the `equals()` method depends on *how the class is used;* it's not a matter of right or wrong
- That said, in a vaccuum the second strategy is probably a better default
    - I'd say two "unequal" works of art can have the same name
- Let's implement the second strategy

equals() method parameter

```
1 public boolean equals(Object other) {
```

- Its type is `Object`
- This is because the method is common to all `Objects`
- How do we compare (say) a `WorkOfArt` to an arbitrary `Object`?
- Idea: if the other `Object` is not a `WorkOfArt`, we will always say `false`; otherwise we compare their reference
- Use `instanceof` keyword to check

static

static variables

```
1 public class Shipment {
2     private static int total_num_shipments;
```

- In Java, we have seen that each object of a class has its own copy of every instance variable
- If we use `static`, it is a “static variable” instead of an instance variable
- There is only *one* copy shared across all objects of the class
- Can be accessed just like instance variables
- First question: why?

First Common Use Case

- Keep track of a “global” value (common to all objects)
- Let’s say we run a shipping company. We’ll create a `Shipment` object to keep track of the data from each shipment
- Also want to track the total number of shipments so far. We’ll use a `static` variable

Global Value Example

```
1  public class Shipment{
2      private static int total_num_shipments = 0;
3      private int shipment_ID;
4      private ArrayList<String> items;
5
6      public int getNumShipments() {
7          return total_num_shipments;
8      }
9  }
```

Continuing Global Value Example

- Each time we make a new `Shipment`, we want to update the tracker of how many there have been

- We'll do this in the constructor

```
1 public Shipment(int newID) {
2     shipment_ID = newID;
3     items = new ArrayList<String>();
4     total_num_shipments++;  //we have a new Shipment!
          Increment the counter
5 }
```

Second Usage Example: Storing Constants

- Sometimes we want to store constants common to all objects of the class
- In our `WorkOfArt` class, it's likely useful to store the maximum possible reference number
- If we have `1000` works of art, we don't need `1000` copies of the maximum reference number! We'll use the `static` keyword so there's only one copy

`final` keyword

- We'll also use the `final` keyword in this example

- If a variable is `final`, then it cannot be changed after it is initialized

Second Example in Code

```
1  public class WorkOfArt {
2      private String title;
3      private String artist;
4      private int year;
5      private int reference;
6
7      public static final int min_reference_number = 0;
8      public static final int max_reference_number = 999999;
```

public static variables

- Unlike instance variables, it is OK if static variables are `public`
    - Especially if they are `final`
- We don't need the max reference number to be "hidden" from other classes, and `final` means it cannot be changed
- You may want to make it `private` for the sake of a "clean API": in short, static variables have the same rules as methods in terms of access modifiers

Accessing static variables directly

- Since static varibles don’t need an object, we can access them without using a specific object

- Notation: name of the class, then the dot operator, then the name of the variable

```
1 public static void main(String[] args) {
2     System.out.println(WorkOfArt.max_reference_number); //
          999999
3 }
```

Static Methods

Overview

- Same idea as static variables
    - Not associated with a specific *object*, associated with the class in general
- This is why `main` is `static`: we don't need an object to run the `main` method
- Can call a `static` method with the same notation: using class name then the dot operator then the method name

Static method rules

- You cannot access instance variables from a static method. Why?
- You cannot call a non-static method from a static method. Why?
- It is OK to call a static method from a non-static method.

Static method Examples

- Used when functionality is associated with a specific class, but does not rely on data from a specific object

- Example from last class:

```
1 Integer y = Integer.valueOf(10);
```

Static method Examples

- Primitive type wrapper classes have quite a few staic methods

- Example: this method converts a `String str` into an `int`

```
1 int number = Integer.parseInt(str);
```

Static Method: Second Example

- The `Math` library consists entirely of static methods

- We don't need a `Math` object, or any stored data, to do math! But it's useful to have these methods grouped together.

```
1  double power = Math.pow(8, 2.5); //raise 8 to the 2.5th power
2  double root = Math.sqrt(2);  //store square root of 2 in root
```

Setting Values Early

- This code is a little unusual when you think about it:

```
1 public class ArrayList<E>{
2     private int numElems = 0;
```

Setting Values Early

- So is this (maybe even more so). When is `total_num_shipments` set to `0`?

```
1  public class Shipment{
2      private static int total_num_shipments = 0;
3      private int shipment_ID;
4      private ArrayList<String> items;
```

Setting Values Early

- Long story short:
    - For instance variables, Java sets them at the beginning of the constructor.
    - For static variables, Java sets them when setting up the program

Let’s build a larger class together

Dictionary

Map

- Data structure you may have seen before (often also called a “Dictionary”)
- Associates keys with values
- Operations:
  - `put` new key-value pair (like “add”)
  - `get` the value for a given key

Map Example

- Can be used to store a dictionary!
- Store the definition for each word in the dictionary
- What are the keys and what are the values? What is the type of each? What does `put` do, and what does `get` do?

Map Example 2

- Let's say we have a list of `Student` objects, and want to assign each of them a group
- Can store this as a `Map`
- What are the keys and what are the values? What is the type of each? What does `put` do, and what does `get` do?

Let’s implement the Map together. We’ll call it ListMap since we’ll use a list to store all of the pairs.

Map

- Data structure:
- Associates keys with values
- Operations:
  - `put` new key-value pair (like “add”)
  - `get` the value for a given key