Lec 7: Linked Lists

Sam McCauley

February 23, 2026

Admin

- In-person lab on Wednesday: Linked Lists
- Practice quiz out; uses material from today and Wednesday
- Today: wrap up ArrayLists; discuss memory in Java; intro to Linked Lists

Let’s fill in (finish filling in) the ArrayListInt class together

Important Takeaway

- When we have the same code twice, should *factor it out* into a new method

Important Takeaway

- When we have the same code twice, should *factor it out* into a new method
    - Does improve code length (not a priority)

Important Takeaway

- When we have the same code twice, should *factor it out* into a new method
    - Does improve code length (not a priority)
    - More importantly: improves organization; makes corrections easier!

Important Takeaway

- When we have the same code twice, should *factor it out* into a new method
    - Does improve code length (not a priority)
    - More importantly: improves organization; makes corrections easier!
-

Memory in Java

Memory

- Your computer stores a large number of 0s and 1s, called “bits”

Memory

- Your computer stores a large number of 0s and 1s, called “bits”
- Organized into sets of 8, called “bytes”

Memory

- Your computer stores a large number of 0s and 1s, called “bits”
- Organized into sets of 8, called “bytes”
- Your computer stores all data this way

Storing Primitive Types

- Primitive type: something like `int`, `char`, `double`, or `boolean`

Storing Primitive Types

- Primitive type: something like `int`, `char`, `double`, or `boolean`
    - `String` is not a primitive type! We'll come back to this

Storing Primitive Types

- Primitive type: something like `int`, `char`, `double`, or `boolean`
    - `String` is not a primitive type! We'll come back to this
- When you declare a variable, Java allocates bits to store its value.

Storing Primitive Types

- Primitive type: something like `int`, `char`, `double`, or `boolean`
  - `String` is not a primitive type! We'll come back to this
- When you declare a variable, Java allocates bits to store its value.
- Example: an `int` is 32 bits, and stores the binary representation of the number

```
1 int x = 2;
```

Declaring and Initializing Variables

- Declaring the variable creates the “box”: it assigns those 32 bits to our variable

Declaring and Initializing Variables

- Declaring the variable creates the “box”: it assigns those 32 bits to our variable

- Initializing the variable sets it to be the value we want

Declaring and Initializing Variables

- Declaring the variable creates the “box”: it assigns those 32 bits to our variable
- Initializing the variable sets it to be the value we want
- Need both to have a useable variable!

Storing Instance Variables

- When instance variables are primitive types, works largely the same way

Storing Instance Variables

- When instance variables are primitive types, works largely the same way
- Each instance variable is assigned the correct number of bits

Storing Instance Variables

- When instance variables are primitive types, works largely the same way
- Each instance variable is assigned the correct number of bits
- Instance variables of the same object are stored together

Storing Instance Variables Example

Consider our `Move` class from Lab 2:

```java
public class Move {
    private int position;
    private int numSpaces;
}
```

Let's say `position` is 3 and `numSpaces` is 2.

Default Values

- Instance variables in Java do not actually need to be initialized; they are automatically set to default values

Default Values

- Instance variables in Java do not actually need to be initialized; they are automatically set to default values

- Example: `int`s are set to `0`

Default Values

- Instance variables in Java do not actually need to be initialized; they are automatically set to default values
- Example: `int`s are set to `0`
- You *should not* do this on purpose! Your constructor should set all values of instance variables

Default Values

- Instance variables in Java do not actually need to be initialized; they are automatically set to default values
- Example: `int`s are set to `0`
- You *should not* do this on purpose! Your constructor should set all values of instance variables
- Good to know for debugging, if you do this accidentally

Storing Objects

- Objects are not stored in the same way!

Storing Objects

- Objects are not stored in the same way!
- An object variable actually stores the *address* of the relevant data

Storing Objects

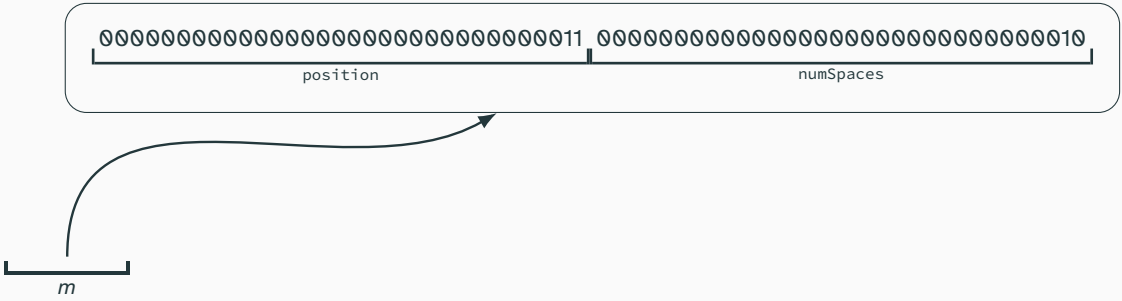
- Objects are not stored in the same way!
- An object variable actually stores the *address* of the relevant data
- This address is called a *reference*. (Sometimes called a pointer in other languages.)

Storing Objects

- Objects are not stored in the same way!
- An object variable actually stores the *address* of the relevant data
- This address is called a *reference*. (Sometimes called a pointer in other languages.)
- Let's draw a picture. Since the reference stores a location, I'll draw an arrow to symbolize where the address stored using the 1s and 0s is pointing to

Storing Objects: Example

```
1  Move m = new Move(3, 2);
```



Declaring and Instantiating an Object

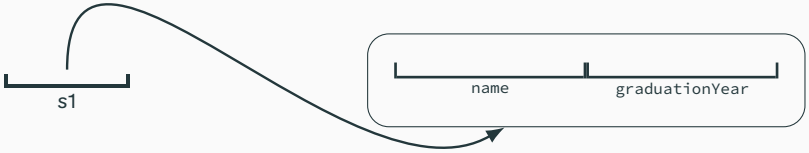
```
1  Move m = new Move(3, 2);
```

- `Move m` creates space (that is to say: assigns bits) to hold the reference
- The `new` keyword allocates the memory, creating the rectangle in the diagram
- The `Move()` constructor is called, which fills in the values of the variables
- Finally, Java creates a reference and stores it in `m`

Simple Example Program with References

Let's say we have a `Student` class with two public variables: `String name` and `int graduationYear`

```
1  Student s1 = new Student();
2  s1.graduationYear = 2026;
3  System.out.println(s1.graduationYear);
```



null Keyword

- `null` is a special reference value, indicating that no address is stored. Default value for object instance variables.

null Keyword

- `null` is a special reference value, indicating that no address is stored. Default value for object instance variables.
- If you try to “follow” (dereference) a null pointer, you will get an error

```
1  Student s1 = null;  //OK
2  s1.graduationYear = 2029; //Null pointer exception
```

null Keyword

- `null` is a special reference value, indicating that no address is stored. Default value for object instance variables.
- If you try to “follow” (dereference) a null pointer, you will get an error

```
1 Student s1 = null;  //OK
2 s1.graduationYear = 2029; //Null pointer exception
```

Can explicitly check to avoid the exception if we want:

```
1 if(s1 != null) {  //do not access s1 if null is stored
2     s1.graduationYear = 2029;
3 }
```

Objects as Instance Variables

- An instance variable can be an object

Objects as Instance Variables

- An instance variable can be an object
- Could be an object from the same class, or another class

Objects as Instance Variables

- An instance variable can be an object
- Could be an object from the same class, or another class
- Takeaway so far: these objects store an address. (Rather than storing an entire copy of the object)

Objects as Instance Variables

- An instance variable can be an object
- Could be an object from the same class, or another class
- Takeaway so far: these objects store an address. (Rather than storing an entire copy of the object)
- Example: let's write a class to keep track of EMTs. Each EMT has a certification number, and a partner.

```
1 public class EMT {
2     public int certNumber;
3     public EMT partner;
4 }
```

EMT Example

```
1 public class EMT {
2     public int certNumber;
3     public EMT partner;
4 }
```

- Let's say that Devin and Ira are partners: Ira is Devin's partner, and Devin is Ira's partner.

EMT Example

```
1 public class EMT {
2     public int certNumber;
3     public EMT partner;
4 }
```

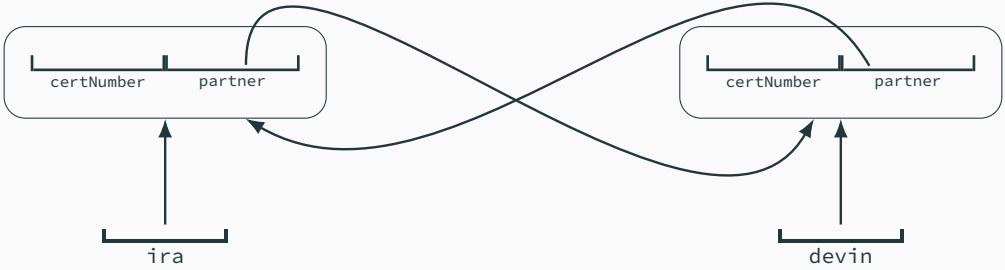
- Let's say that Devin and Ira are partners: Ira is Devin's partner, and Devin is Ira's partner.
- The key insight here is that these two objects *point to* each other.

EMT Example

```
1 public class EMT {
2     public int certNumber;
3     public EMT partner;
4 }
```

- Let's say that Devin and Ira are partners: Ira is Devin's partner, and Devin is Ira's partner.
- The key insight here is that these two objects *point to* each other.
- It's similar to Ira having a piece of paper that has Devin's name on it (and vice-versa)—but instead of a name, what's actually stored is a memory address.

EMT Example



EMT Example: Code

- How can we get the certification number of Ira's partner? (Remember: we made instance variables `public` for this example)

EMT Example: Code

- How can we get the certification number of Ira's partner? (Remember: we made instance variables `public` for this example)

```
1  int irasPartnerCert = ira.partner.certNumber;
```

EMT Example: Code

- How can we get the certification number of Ira's partner? (Remember: we made instance variables `public` for this example)

```
1  int irasPartnerCert = ira.partner.certNumber;
```

- What would this look like with getters named `getParner()` and `getCertNumber()`?

EMT Example: Code

- How can we get the certification number of Ira's partner? (Remember: we made instance variables `public` for this example)

```
1  int irasPartnerCert = ira.partner.certNumber;
```

- What would this look like with getters named `getParner()` and `getCertNumber()`?

```
1  int irasPartnerCert = ira.getPartner().getCertNumber();
```

Linked List Strategy Discussion

Remembering a Sequence

- Let’s say that in our CS 136 class we want to remember a sequence of 15 numbers.

Remembering a Sequence

- Let’s say that in our CS 136 class we want to remember a sequence of 15 numbers.
    - Hard for any one of us to do, but very doable as a group!

Remembering a Sequence

- Let’s say that in our CS 136 class we want to remember a sequence of 15 numbers.
    - Hard for any one of us to do, but very doable as a group!
- We’ll have the student in the first seat memorize the first number; the second seat memorize the second number, and so on.

Remembering a Sequence

- Let's say that in our CS 136 class we want to remember a sequence of 15 numbers.
    - Hard for any one of us to do, but very doable as a group!
- We'll have the student in the first seat memorize the first number; the second seat memorize the second number, and so on.
- If we want to recover the sequence, we just go seat by seat and have the student recite their number.

Remembering a Sequence with Swaps

- What happens if students change seats?

Remembering a Sequence with Swaps

- What happens if students change seats?
- Or maybe we’re on a zoom call and there are no seats?

Remembering a Sequence with Swaps

- What happens if students change seats?
- Or maybe we're on a zoom call and there are no seats?
- How can we slightly extend our strategy to be resilient to this problem?

Extended Strategy

- Each student remembers two things:

Extended Strategy

- Each student remembers two things:
    - Their number, and

Extended Strategy

- Each student remembers two things:
  - Their number, and
  - the next student in the sequence

Extended Strategy

- Each student remembers two things:
  - Their number, and
  - the next student in the sequence
- Now the seats don't matter! I just need to remember the *first student*

Extended Strategy

- Each student remembers two things:
  - Their number, and
  - the next student in the sequence
- Now the seats don't matter! I just need to remember the *first student*
- I ask them for their number and the next student. Then I can go to that student, ask for their number and the next student, and so on

Extended Strategy

- Each student remembers two things:
  - Their number, and
  - the next student in the sequence
- Now the seats don't matter! I just need to remember the *first student*
- I ask them for their number and the next student. Then I can go to that student, ask for their number and the next student, and so on
- I'll stop when a student says that there is no further student

Linked List Strategy to Store a List of ints

- Linked List can store a sequence of `ints` (we'll generalize it to all types of data later in the week)

Linked List Strategy to Store a List of ints

- Linked List can store a sequence of `ints` (we'll generalize it to all types of data later in the week)
- Can perform exactly the same methods as our `IntArrayList` implementation!

Linked List Strategy to Store a List of ints

- Linked List can store a sequence of `ints` (we'll generalize it to all types of data later in the week)
- Can perform exactly the same methods as our `IntArrayList` implementation!
- Differences in *performance*: linked lists will be faster for some methods, and slower for others

Linked List of Ints

Int Linked List Nodes

- In our example, students held the information of the list

Int Linked List Nodes

- In our example, students held the information of the list
- We'll store the information in `Nodes`

Int Linked List Nodes

- In our example, students held the information of the list
- We'll store the information in `Nodes`
    - We'll call them `IntNode` to clarify that these nodes only help us to store lists of `ints`

Int Linked List Nodes

- In our example, students held the information of the list
- We'll store the information in `Nodes`
    - We'll call them `IntNode` to clarify that these nodes only help us to store lists of `ints`
- What should the `IntNode` class look like? Let's code it up together.

Starting the Int Linked List Class

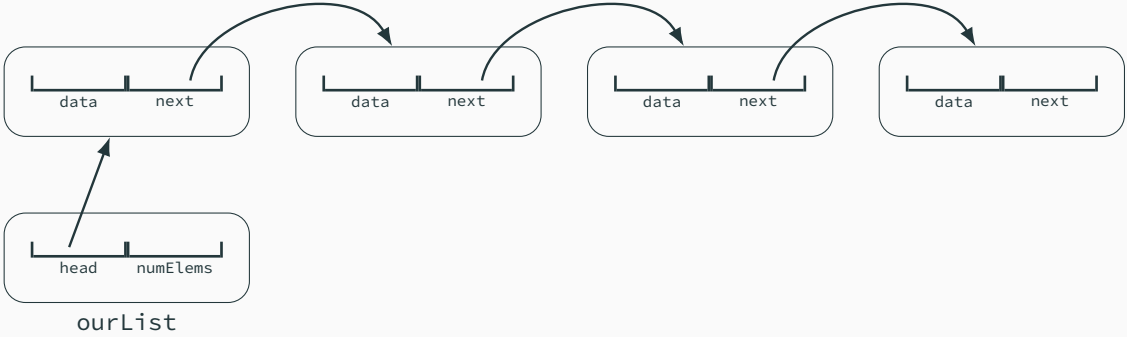
- Our actual list only needs to store a pointer to the first node!

Starting the Int Linked List Class

- Our actual list only needs to store a pointer to the first node!

- We'll also store the number of elements in the list again, like we did for IntArrayList (will be handy)

Overview of Plan



Overview of Plan: With Data

The following list stores: [1, 2, 3, 4]. `ints` are truncated to 4 bits for readability.