



Middlebury

CSCI 201: Data Structures

Fall 2025

Lecture 3R: **Java** Collections

Goals for today:

Goals for today:

- Be able to use `javadoc` and `checkstyle` to create clean, well-documented code.

Goals for today:

- Be able to use `javadoc` and `checkstyle` to create clean, well-documented code.
- Identify the difference between an Abstract Data Type and a Data Structure.

Goals for today:

- Be able to use `javadoc` and `checkstyle` to create clean, well-documented code.
- Identify the difference between an Abstract Data Type and a Data Structure.
- Use an `ArrayList` to store multiple items (of the same type), in which the number of items can increase/decrease based on the needs of your algorithm.

Goals for today:

- Be able to use `javadoc` and `checkstyle` to create clean, well-documented code.
- Identify the difference between an Abstract Data Type and a Data Structure.
- Use an `ArrayList` to store multiple items (of the same type), in which the number of items can increase/decrease based on the needs of your algorithm.
- Use a `HashMap` to store key-value pairs.

What's a **javadoc** comment?

```
1 /**
2  * Retrieves one of the cards in the hand.
3  *
4  * @param cardIndex – index of card in the hand (between 0 and cards.length –
5  *                    1).
6  * @return the card in the hand at the requested index.
7  */
8 public FaceUpCard getCard(int cardIndex) {
9     return cards[cardIndex];
10 }
```

What's a **javadoc** comment?

```
1 /**
2  * Retrieves one of the cards in the hand.
3  *
4  * @param cardIndex - index of card in the hand (between 0 and cards.length -
5  *                    1).
6  * @return the card in the hand at the requested index.
7  */
8 public FaceUpCard getCard(int cardIndex) {
9     return cards[cardIndex];
10 }
```

getCard

```
public FaceUpCard getCard(int cardIndex)
```

Retrieves one of the cards in the hand.

Parameters:

cardIndex - - index of card in the hand (between 0 and cards.length - 1).

Returns:

the card in the hand at the requested index.

What's a javadoc comment?

Constructor Summary

Constructors	
Constructor	Description
<code>FaceUpHand(int numCards)</code>	Create a new FaceUp card game state, which consists of the numCards cards for the game.
<code>FaceUpHand(FaceUpCard[] cards)</code>	Create a new FaceUp card game state, which consists of the specific cards that will be used Used for testing, not for gameplay!

Method Summary

All Methods		Static Methods	Instance Methods	Concrete Methods
Modifier and Type	Method	Description		
int	<code>calculateOptimal()</code>	Determines the optimal number of points in the hand, which is the sum of all the red card points.		
int	<code>faceDownTotal()</code>	Calculate the total score for the cards that are face down		
int	<code>faceUpTotal()</code>	Calculate the total score for the cards that are face up.		
void	<code>flipCard(int cardIndex)</code>	Flips the the card from face-up to face-down or face-down to face-up.		
<code>FaceUpCard</code>	<code>getCard(int cardIndex)</code>	Retrieves one of the cards in the hand.		
static void	<code>main(String[] args)</code>	Test Hand methods		
<code>String</code>	<code>test()</code>			
<code>String</code>	<code>toString()</code>			

Using **checkstyle** as an automated style checker

```
1 public class BadStyle {
2     public static void main(String[] args) {
3         int x = 5, y = 20;
4         String my_name = "Mike Wazowski"; int age = 20;
5         for(int i = 0; i<values.length;i++){
6             if(condition) {
7
8             }
9             else
10            {
11
12            }
13        }
14    }
15 }
```

Using **checkstyle** as an automated style checker

```
1 public class BadStyle {
2     public static void main(String[] args) {
3         int x = 5, y = 20;
4         String my_name = "Mike Wazowski"; int age = 20;
5         for(int i = 0; i<values.length;i++){
6             if(condition) {
7
8             }
9             else
10            {
11
12            }
13        }
14    }
15 }
```

```
1 Starting audit...
2 [WARN] BadStyle.java:1:1: Missing a Javadoc comment. [MissingJavadocType]
3 ...
4 [WARN] BadStyle.java:4:51: Only one statement per line allowed. [OneStatementPerLine]
5 [WARN] BadStyle.java:5:5: 'for' is not followed by whitespace. [WhitespaceAfter]
6 ...
7 Audit done.
```

Using **checkstyle** as an automated style checker

```
1 public class BadStyle {
2     public static void main(String[] args) {
3         int x = 5, y = 20;
4         String my_name = "Mike Wazowski"; int age = 20;
5         for(int i = 0; i<values.length;i++){
6             if(condition) {
7
8             }
9             else
10            {
11
12            }
13        }
14    }
15 }
```

```
1 Starting audit...
2 [WARN] BadStyle.java:1:1: Missing a Javadoc comment. [MissingJavadocType]
3 ...
4 [WARN] BadStyle.java:4:51: Only one statement per line allowed. [OneStatementPerLine]
5 [WARN] BadStyle.java:5:5: 'for' is not followed by whitespace. [WhitespaceAfter]
6 ...
7 Audit done.
```

- **checkstyle** will run automatically on gradescope, and you can run it in vscode!

Using **checkstyle** as an automated style checker

```
1 public class BadStyle {
2     public static void main(String[] args) {
3         int x = 5, y = 20;
4         String my_name = "Mike Wazowski"; int age = 20;
5         for(int i = 0; i<values.length;i++){
6             if(condition) {
7
8             }
9             else
10            {
11
12            }
13        }
14    }
15 }
```

```
1 Starting audit...
2 [WARN] BadStyle.java:1:1: Missing a Javadoc comment. [MissingJavadocType]
3 ...
4 [WARN] BadStyle.java:4:51: Only one statement per line allowed. [OneStatementPerLine]
5 [WARN] BadStyle.java:5:5: 'for' is not followed by whitespace. [WhitespaceAfter]
6 ...
7 Audit done.
```

- **checkstyle** will run automatically on gradescope, and you can run it in vscode!
- We will use the configuration for [Google's java style guide](#).

Motivating dynamic storage

How would you keep track of which Pokémon a player has?



Motivating dynamic storage

How would you keep track of which Pokémon a player has?



How would you organize songs in a music playlist?

#	Title	Album
1	 Blank Space (Taylor's Version) Taylor Swift	1989 (Taylor's Version)
2	 Love Story (Taylor's Version) Taylor Swift	Love Story (Taylor's Version)
3	 I Can Do It With a Broken Heart Taylor Swift	THE TORTURED POETS DEPART...
4	 Cruel Summer Taylor Swift	Lover
5	 I Knew You Were Trouble (Taylor's V... Taylor Swift	Red (Taylor's Version)
6	 Look What You Made Me Do Taylor Swift	reputation

Motivating dynamic storage

How would you keep track of which Pokémon a player has?



How would you organize songs in a music playlist?

#	Title	Album
1	 Blank Space (Taylor's Version) Taylor Swift	1989 (Taylor's Version)
2	 Love Story (Taylor's Version) Taylor Swift	Love Story (Taylor's Version)
3	 I Can Do It With a Broken Heart Taylor Swift	THE TORTURED POETS DEPART...
4	 Cruel Summer Taylor Swift	Lover
5	 I Knew You Were Trouble (Taylor's V... Taylor Swift	Red (Taylor's Version)
6	 Look What You Made Me Do Taylor Swift	reputation

These tasks would be a bit hard to do with fixed-size arrays (directly).

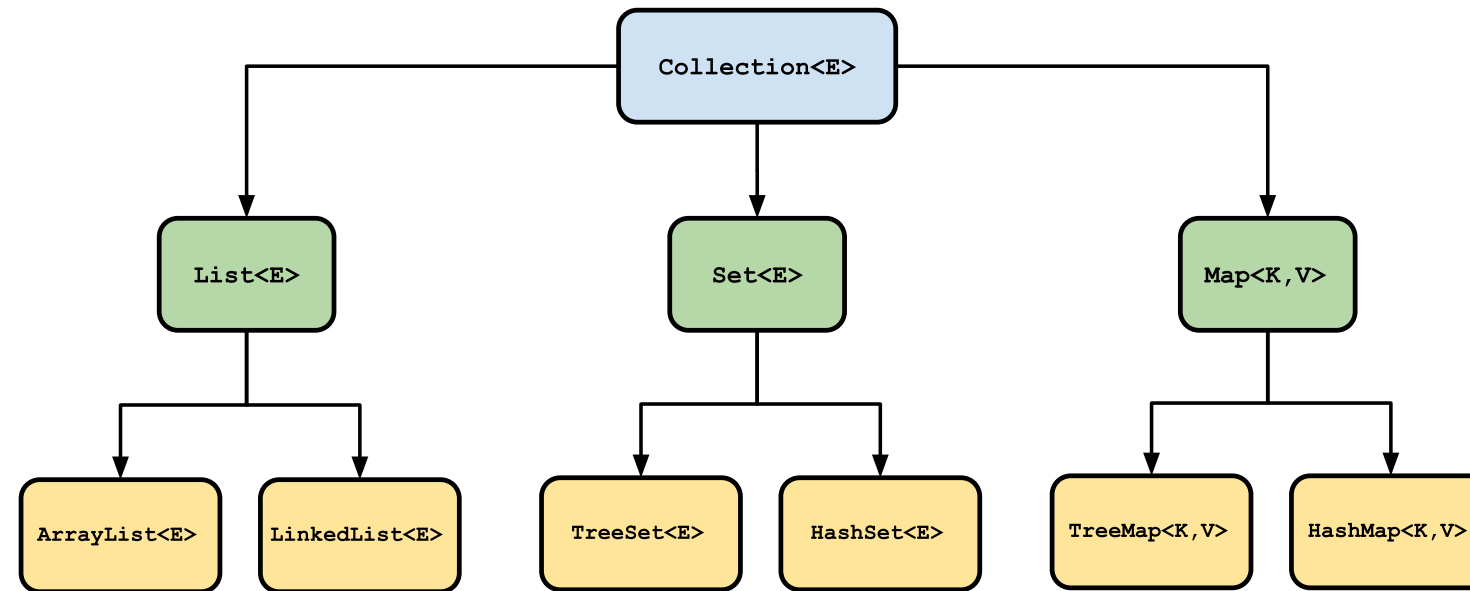
Motivating dynamic storage

Motivating dynamic storage

Imagine we had a utility to keep track of this - what methods would you like?

There are tools (built into **Java**) to help with this.

A collection represents a group of objects, known as its elements.



- **Abstract Data Type (ADT):** formal description of behavior of data type, e.g. a **List** allows accessing item at a specified index (but implementation can vary).
- **Data Structure:** concrete organization/representation of data (implements spec defined by an ADT). Example: **ArrayList**.

Okay, let's take a shot at a **List** ourselves.

We'll design our own implementation of a **List** called **DIYList**.

But first, we should check the **List** spec:

<https://docs.oracle.com/javase/8/docs/api/java/util/List.html>

Okay, let's take a shot at a **List** ourselves.

We'll design our own implementation of a **List** called **DIYList**.

But first, we should check the **List** spec:

<https://docs.oracle.com/javase/8/docs/api/java/util/List.html>

Let's focus on methods to:

1. Construct an empty **DIYList**.
2. **add** an item to a **DIYList**.
3. **remove** an item from a **DIYList**.
4. Retrieve the number of items (**size**) in a **DIYList**.
5. **clear** the items in a **DIYList**.

Okay, let's take a shot at a **List** ourselves.

We'll design our own implementation of a **List** called **DIYList**.

But first, we should check the **List** spec:

<https://docs.oracle.com/javase/8/docs/api/java/util/List.html>

Let's focus on methods to:

1. Construct an empty **DIYList**.
2. **add** an item to a **DIYList**.
3. **remove** an item from a **DIYList**.
4. Retrieve the number of items (**size**) in a **DIYList**.
5. **clear** the items in a **DIYList**.

💡 **Idea:** use a fixed-size array that has enough space (capacity) to hold our items.

If we need more space, just allocate a new larger array and copy items!

Adding (**add**) an item to a **DIYList**.

Without loss of generality, imagine our **DIYList** can only hold **String** items.

```
public class DIYList {  
    int size; // current number of items actually stored  
    String[] items; // capacity is items.length  
  
    ...  
  
    public void add(String item) {  
        // Is there enough space (capacity, i.e. items.length)?  
        // If not, make more space and copy the old items.  
  
        // Place item in items[size] and increment size.  
    }  
}
```

Removing (**remove**) an item from a **DIYList**.

```
public class DIYList {  
    int size; // current number of items actually stored  
    String[] items; // capacity is items.length  
  
    ...  
  
    public void remove(int index) {  
        // if index is beyond the index of the last item, nothing to do -> return  
  
        // replace the item at index with the item at index + 1  
        // replace the item at index + 1 with the item at index + 2  
        // replace the item at index + 2 with the item at index + 3  
        // ... until all items after the index have been shifted left  
  
        // decrement size (because we removed an item)  
    }  
}
```

Clearing (**clear**) the items in a **DIYList**.

```
public class DIYList {  
    int size; // current number of items actually stored  
    String[] items; // capacity is items.length  
  
    ...  
  
    public void clear() {  
        // set all items to null  
  
        // set size to 0  
    }  
}
```

The truth is that we just implemented an **ArrayList**!

Main idea of an **ArrayList**:

- Internally use an array to hold the items.
- This array needs to have enough space (capacity) to hold the items.
- To add an item:
 - First check if there is enough space.
If not, make a new array with more space and copy the old items into this array.
 - Add the new item to the next empty slot.
- How should we increase the capacity?

The truth is that we just implemented an **ArrayList**!

Main idea of an **ArrayList**:

- Internally use an array to hold the items.
- This array needs to have enough space (capacity) to hold the items.
- To add an item:
 - First check if there is enough space.
If not, make a new array with more space and copy the old items into this array.
 - Add the new item to the next empty slot.
- How should we increase the capacity?

<https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#ensureCapacity-int->

The truth is that we just implemented an **ArrayList**!

Main idea of an **ArrayList**:

- Internally use an array to hold the items.
- This array needs to have enough space (capacity) to hold the items.
- To add an item:
 - First check if there is enough space.
If not, make a new array with more space and copy the old items into this array.
 - Add the new item to the next empty slot.
- How should we increase the capacity?

<https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#ensureCapacity-int->

Each **ArrayList** instance has a *capacity*. The capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added to an **ArrayList**, its capacity grows automatically. The details of the growth policy are not specified beyond the fact that adding an element has constant amortized time cost.

An application can increase the capacity of an **ArrayList** instance before adding a large number of elements using the **ensureCapacity** operation. This may reduce the amount of incremental reallocation.

Let's practice with **ArrayLists**.

Note that **Java Collections** only work with reference types
We cannot directly use primitive types.

How do we use them with **int**, **double**, **char** etc.?

Luckily, there are wrapper classes called **Integer**, **Double**, **Character**, etc.

```
import java.util.ArrayList; // don't forget this!
// import java.util.*; // can also be convenient to import everything in java.util

public class ArrayListExamples {
    public static void main(String[] args) {
        // initialize empty array list
        ArrayList<Character> list = new ArrayList<>();

        list.add('R');
        list.add('A');
        list.add('I');
        list.add('N');

        System.out.println("list size = " + list.size()); // 4

        for (int i = 0; i < list.size(); i++) {
            System.out.println("Item[" + i + "]: " + list.get(i));
        }

        list.remove(2); // remove item at index 2 (i.e. the 'I')
        System.out.println("list size = " + list.size()); // 3
        System.out.println(list);
    }
}
```

HashMap: another useful collection to know about.

- **HashMap<K, V>**: stores key-value pairs (keys have type **K** and values have type **V**).
- **Useful methods**: **containsKey** (checks if a key of type **K** exists), **put** (inserts a key-value pair), **get** (retrieves the value associated with some key), **keySet** (retrieves a **Set** of all keys).

```
1 import java.util.HashMap;
2 // import java.util.*; // <-- this can be more convenient!
3
4 public class HashMapExample {
5     public static void main(String[] args) {
6         String lyric = "and i love vermont, but it's the season of the sticks";
7
8         HashMap<Character, Integer> frequency = new HashMap<>();
9         for (int i = 0; i < lyric.length(); i++) {
10             char c = lyric.charAt(i);
11             if (frequency.containsKey(c)) {
12                 frequency.put(c, frequency.get(c) + 1);
13             } else {
14                 frequency.put(c, 0);
15             }
16         }
17
18         Set<Character> characters = frequency.keySet();
19         for (Character c : characters) {
20             System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
21         }
22     }
23 }
```

HashMap: another useful collection to know about.

- **HashMap<K, V>**: stores key-value pairs (keys have type **K** and values have type **V**).
- **Useful methods**: **containsKey** (checks if a key of type **K** exists), **put** (inserts a key-value pair), **get** (retrieves the value associated with some key), **keySet** (retrieves a **Set** of all keys).

```
1 import java.util.HashMap;
2 // import java.util.*; // <-- this can be more convenient!
3
4 public class HashMapExample {
5     public static void main(String[] args) {
6         String lyric = "and i love vermont, but it's the season of the sticks";
7
8         HashMap<Character, Integer> frequency = new HashMap<>();
9         for (int i = 0; i < lyric.length(); i++) {
10             char c = lyric.charAt(i);
11             if (frequency.containsKey(c)) {
12                 frequency.put(c, frequency.get(c) + 1);
13             } else {
14                 frequency.put(c, 0);
15             }
16         }
17
18         Set<Character> characters = frequency.keySet();
19         for (Character c : characters) {
20             System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
21         }
22     }
23 }
```

HashMap: another useful collection to know about.

- **HashMap<K, V>**: stores key-value pairs (keys have type **K** and values have type **V**).
- **Useful methods**: **containsKey** (checks if a key of type **K** exists), **put** (inserts a key-value pair), **get** (retrieves the value associated with some key), **keySet** (retrieves a **Set** of all keys).

```
1 import java.util.HashMap;
2 // import java.util.*; // <-- this can be more convenient!
3
4 public class HashMapExample {
5     public static void main(String[] args) {
6         String lyric = "and i love vermont, but it's the season of the sticks";
7
8         HashMap<Character, Integer> frequency = new HashMap<>();
9         for (int i = 0; i < lyric.length(); i++) {
10             char c = lyric.charAt(i);
11             if (frequency.containsKey(c)) {
12                 frequency.put(c, frequency.get(c) + 1);
13             } else {
14                 frequency.put(c, 0);
15             }
16         }
17
18         Set<Character> characters = frequency.keySet();
19         for (Character c : characters) {
20             System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
21         }
22     }
23 }
```

HashMap: another useful collection to know about.

- **HashMap<K, V>**: stores key-value pairs (keys have type **K** and values have type **V**).
- **Useful methods**: **containsKey** (checks if a key of type **K** exists), **put** (inserts a key-value pair), **get** (retrieves the value associated with some key), **keySet** (retrieves a **Set** of all keys).

```
1 import java.util.HashMap;
2 // import java.util.*; // <-- this can be more convenient!
3
4 public class HashMapExample {
5     public static void main(String[] args) {
6         String lyric = "and i love vermont, but it's the season of the sticks";
7
8         HashMap<Character, Integer> frequency = new HashMap<>();
9         for (int i = 0; i < lyric.length(); i++) {
10             char c = lyric.charAt(i);
11             if (frequency.containsKey(c)) {
12                 frequency.put(c, frequency.get(c) + 1);
13             } else {
14                 frequency.put(c, 0);
15             }
16         }
17
18         Set<Character> characters = frequency.keySet();
19         for (Character c : characters) {
20             System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
21         }
22     }
23 }
```

HashMap: another useful collection to know about.

- **HashMap<K, V>**: stores key-value pairs (keys have type **K** and values have type **V**).
- **Useful methods**: **containsKey** (checks if a key of type **K** exists), **put** (inserts a key-value pair), **get** (retrieves the value associated with some key), **keySet** (retrieves a **Set** of all keys).

```
1 import java.util.HashMap;
2 // import java.util.*; // <-- this can be more convenient!
3
4 public class HashMapExample {
5     public static void main(String[] args) {
6         String lyric = "and i love vermont, but it's the season of the sticks";
7
8         HashMap<Character, Integer> frequency = new HashMap<>();
9         for (int i = 0; i < lyric.length(); i++) {
10             char c = lyric.charAt(i);
11             if (frequency.containsKey(c)) {
12                 frequency.put(c, frequency.get(c) + 1);
13             } else {
14                 frequency.put(c, 0);
15             }
16         }
17
18         Set<Character> characters = frequency.keySet();
19         for (Character c : characters) {
20             System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
21         }
22     }
23 }
```

HashMap: another useful collection to know about.

- **HashMap<K, V>**: stores key-value pairs (keys have type **K** and values have type **V**).
- **Useful methods**: **containsKey** (checks if a key of type **K** exists), **put** (inserts a key-value pair), **get** (retrieves the value associated with some key), **keySet** (retrieves a **Set** of all keys).

```
1 import java.util.HashMap;
2 // import java.util.*; // <-- this can be more convenient!
3
4 public class HashMapExample {
5     public static void main(String[] args) {
6         String lyric = "and i love vermont, but it's the season of the sticks";
7
8         HashMap<Character, Integer> frequency = new HashMap<>();
9         for (int i = 0; i < lyric.length(); i++) {
10             char c = lyric.charAt(i);
11             if (frequency.containsKey(c)) {
12                 frequency.put(c, frequency.get(c) + 1);
13             } else {
14                 frequency.put(c, 0);
15             }
16         }
17
18         Set<Character> characters = frequency.keySet();
19         for (Character c : characters) {
20             System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
21         }
22     }
23 }
```

HashMap: another useful collection to know about.

- **HashMap<K, V>**: stores key-value pairs (keys have type **K** and values have type **V**).
- **Useful methods**: **containsKey** (checks if a key of type **K** exists), **put** (inserts a key-value pair), **get** (retrieves the value associated with some key), **keySet** (retrieves a **Set** of all keys).

```
1 import java.util.HashMap;
2 // import java.util.*; // <-- this can be more convenient!
3
4 public class HashMapExample {
5     public static void main(String[] args) {
6         String lyric = "and i love vermont, but it's the season of the sticks";
7
8         HashMap<Character, Integer> frequency = new HashMap<>();
9         for (int i = 0; i < lyric.length(); i++) {
10             char c = lyric.charAt(i);
11             if (frequency.containsKey(c)) {
12                 frequency.put(c, frequency.get(c) + 1);
13             } else {
14                 frequency.put(c, 0);
15             }
16         }
17
18         Set<Character> characters = frequency.keySet();
19         for (Character c : characters) {
20             System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
21         }
22     }
23 }
```

HashMap: another useful collection to know about.

- **HashMap<K, V>**: stores key-value pairs (keys have type **K** and values have type **V**).
- **Useful methods**: **containsKey** (checks if a key of type **K** exists), **put** (inserts a key-value pair), **get** (retrieves the value associated with some key), **keySet** (retrieves a **Set** of all keys).

```
1 import java.util.HashMap;
2 // import java.util.*; // <-- this can be more convenient!
3
4 public class HashMapExample {
5     public static void main(String[] args) {
6         String lyric = "and i love vermont, but it's the season of the sticks";
7
8         HashMap<Character, Integer> frequency = new HashMap<>();
9         for (int i = 0; i < lyric.length(); i++) {
10             char c = lyric.charAt(i);
11             if (frequency.containsKey(c)) {
12                 frequency.put(c, frequency.get(c) + 1);
13             } else {
14                 frequency.put(c, 0);
15             }
16         }
17
18         Set<Character> characters = frequency.keySet();
19         for (Character c : characters) {
20             System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
21         }
22     }
23 }
```

HashMap: another useful collection to know about.

- **HashMap<K, V>**: stores key-value pairs (keys have type **K** and values have type **V**).
- **Useful methods**: **containsKey** (checks if a key of type **K** exists), **put** (inserts a key-value pair), **get** (retrieves the value associated with some key), **keySet** (retrieves a **Set** of all keys).

```
1 import java.util.HashMap;
2 // import java.util.*; // <-- this can be more convenient!
3
4 public class HashMapExample {
5     public static void main(String[] args) {
6         String lyric = "and i love vermont, but it's the season of the sticks";
7
8         HashMap<Character, Integer> frequency = new HashMap<>();
9         for (int i = 0; i < lyric.length(); i++) {
10             char c = lyric.charAt(i);
11             if (frequency.containsKey(c)) {
12                 frequency.put(c, frequency.get(c) + 1);
13             } else {
14                 frequency.put(c, 0);
15             }
16         }
17
18         Set<Character> characters = frequency.keySet();
19         for (Character c : characters) {
20             System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
21         }
22     }
23 }
```

HashMap: another useful collection to know about.

- **HashMap<K, V>**: stores key-value pairs (keys have type **K** and values have type **V**).
- **Useful methods**: **containsKey** (checks if a key of type **K** exists), **put** (inserts a key-value pair), **get** (retrieves the value associated with some key), **keySet** (retrieves a **Set** of all keys).

```
1 import java.util.HashMap;
2 // import java.util.*; // <-- this can be more convenient!
3
4 public class HashMapExample {
5     public static void main(String[] args) {
6         String lyric = "and i love vermont, but it's the season of the sticks";
7
8         HashMap<Character, Integer> frequency = new HashMap<>();
9         for (int i = 0; i < lyric.length(); i++) {
10             char c = lyric.charAt(i);
11             if (frequency.containsKey(c)) {
12                 frequency.put(c, frequency.get(c) + 1);
13             } else {
14                 frequency.put(c, 0);
15             }
16         }
17
18         Set<Character> characters = frequency.keySet();
19         for (Character c : characters) {
20             System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
21         }
22     }
23 }
```

HashMap: another useful collection to know about.

- **HashMap<K, V>**: stores key-value pairs (keys have type **K** and values have type **V**).
- **Useful methods**: **containsKey** (checks if a key of type **K** exists), **put** (inserts a key-value pair), **get** (retrieves the value associated with some key), **keySet** (retrieves a **Set** of all keys).

```
1 import java.util.HashMap;
2 // import java.util.*; // <-- this can be more convenient!
3
4 public class HashMapExample {
5     public static void main(String[] args) {
6         String lyric = "and i love vermont, but it's the season of the sticks";
7
8         HashMap<Character, Integer> frequency = new HashMap<>();
9         for (int i = 0; i < lyric.length(); i++) {
10             char c = lyric.charAt(i);
11             if (frequency.containsKey(c)) {
12                 frequency.put(c, frequency.get(c) + 1);
13             } else {
14                 frequency.put(c, 0);
15             }
16         }
17
18         Set<Character> characters = frequency.keySet();
19         for (Character c : characters) {
20             System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
21         }
22     }
23 }
```

HashMap: another useful collection to know about.

- **HashMap<K, V>**: stores key-value pairs (keys have type **K** and values have type **V**).
- **Useful methods**: **containsKey** (checks if a key of type **K** exists), **put** (inserts a key-value pair), **get** (retrieves the value associated with some key), **keySet** (retrieves a **Set** of all keys).

```
1 import java.util.HashMap;
2 // import java.util.*; // <-- this can be more convenient!
3
4 public class HashMapExample {
5     public static void main(String[] args) {
6         String lyric = "and i love vermont, but it's the season of the sticks";
7
8         HashMap<Character, Integer> frequency = new HashMap<>();
9         for (int i = 0; i < lyric.length(); i++) {
10             char c = lyric.charAt(i);
11             if (frequency.containsKey(c)) {
12                 frequency.put(c, frequency.get(c) + 1);
13             } else {
14                 frequency.put(c, 0);
15             }
16         }
17
18         Set<Character> characters = frequency.keySet();
19         for (Character c : characters) {
20             System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
21         }
22     }
23 }
```

HashMap: another useful collection to know about.

- **HashMap<K, V>**: stores key-value pairs (keys have type **K** and values have type **V**).
- **Useful methods**: **containsKey** (checks if a key of type **K** exists), **put** (inserts a key-value pair), **get** (retrieves the value associated with some key), **keySet** (retrieves a **Set** of all keys).

```
1 import java.util.HashMap;
2 // import java.util.*; // <-- this can be more convenient!
3
4 public class HashMapExample {
5     public static void main(String[] args) {
6         String lyric = "and i love vermont, but it's the season of the sticks";
7
8         HashMap<Character, Integer> frequency = new HashMap<>();
9         for (int i = 0; i < lyric.length(); i++) {
10             char c = lyric.charAt(i);
11             if (frequency.containsKey(c)) {
12                 frequency.put(c, frequency.get(c) + 1);
13             } else {
14                 frequency.put(c, 0);
15             }
16         }
17
18         Set<Character> characters = frequency.keySet();
19         for (Character c : characters) {
20             System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
21         }
22     }
23 }
```

Find The Bug! 

Here it is!

```
1 import java.util.HashMap;
2 // import java.util.*; // <-- this can be more convenient!
3
4 public class HashMapExample {
5     public static void main(String[] args) {
6         String lyric = "and i love vermont, but it's the season of the sticks";
7
8         HashMap<Character, Integer> frequency = new HashMap<>();
9         for (int i = 0; i < lyric.length(); i++) {
10             char c = lyric.charAt(i);
11             if (frequency.containsKey(c)) {
12                 frequency.put(c, frequency.get(c) + 1);
13             } else {
14                 frequency.put(c, 1); // first time we encounter a character, it counts, so insert 1 not 0
15             }
16         }
17
18         Set<Character> characters = frequency.keySet();
19         for (Character c : characters) {
20             System.out.println("Character " + c + " appears " + frequency.get(c) + " times");
21         }
22     }
23 }
```

See you Friday!

- We'll practice using **ArrayLists** in Friday's lab.
- **HashMaps** were just introduced now in case you find them helpful to solve problems (we'll talk about the underlying data structures later in the semester).
- **HashSets** can also be useful if you want to store an unordered set of items.
- Homework 3 will be released later today!
- Reminder that Smith ([go/smith](#)) has office hours throughout the week and the 201 Course Assistants have drop-in hours in the late afternoons/evenings ([go/cshelp](#)).