Java's Built-In Data Structures

ArrayLists

An ArrayList is a built-in data structure that uses a dynamic array to store its elements.
In order to use this data structure, you must import java.util.ArrayList at the top of your program.

```
// Create an ArrayList that stores
Strings:
ArrayList<String> months = new
ArrayList<String>();

// Add values to the end of the
ArrayList:
months.add("January");
months.add("March");

// Add values to a specified index in
the ArrayList:
months.add(1, "February");

// Access a value from the ArrayList:
System.out.println(months.get(0));  //
Prints: January

// Remove a value from the ArrayList:
months.remove("February");
months.remove(0); // Removes the
zeroeth element
```

LinkedLists

A LinkedList is a built-in data structure that uses a doubly linked list to store its elements.
In order to use this data structure, you must import java.util.LinkedList at the top of your program.

```
// Create a LinkedList that stores
Strings:
LinkedList<String> days = new
LinkedList<String>();

// Add values to the end of the
LinkedList:
days.add("Monday");
```

```
days.add("Tuesday");
```

codecademy

```
// Add values to a specified index in
the LinkedList:
days.add(0, "Sunday");

// Access a value from the LinkedList:
System.out.println(days.get(0)); //
Prints: Sunday

// Remove a value from the LinkedList:
days.remove("March");
days.remove(1); // Removes the element
at index 1
days.remove(); // Removes the first
element in the LinkedList
```

HashMaps

A `HashMap` is a built-in data structure that stores a collection of key-value pairs. Each key acts as a unique identifier for its associated value. In order to use this data structure, import the `HashMap` class at the top of the program.

```
// Create a HashMap with String keys
and Integer values:
HashMap<String, Integer>
fruitInventory = new HashMap<>();

// Add key-value pairs to HashMap:
fruitInventory.put("apples", 5);
fruitInventory.put("strawberries", 7);
fruitInventory.put("oranges", 4);

// Remove a key-value pair from
HashMap:
fruitInventory.remove("apples");

// Access a value:
System.out.println(fruitInventory.get(
"oranges")); // Prints: 4

// Find HashMap size:
System.out.println(fruitInventory.size
()); // Prints: 2
```

```
// Iterate over a HashMap
for (String key :
fruitInventory.keySet()) {
  System.out.println("Key: " + key +
", Value: " +
fruitInventory.get(key));
}
/* Prints:
Key: strawberries, Value: 7
Key: oranges, Value: 4
*/
```

Sets

A `Set` is a built-in data structure that stores an unordered collection of unique values. A `Set` can only store reference type values.

There are multiple implementations of a `Set`: the `HashSet`, the `TreeSet`, and the `LinkedHashSet`. While these classes share the similarities of storing unique objects and having access to the same methods, there are a few differences between the three that impact the order items appear within the structure as well as the runtime of some of the methods.

In order to utilize a `Set` and its associated operations, import its class at the top of the program.

```
// Create a HashSet of Strings:
Set<String> shapes = new
HashSet<String>();

// Add items to a Set:
shapes.add("square");
shapes.add("triangle");
shapes.add("circle");

// Remove a value:
shapes.remove("square");

// Check for a value:
System.out.println(shapes.contains("ci
rcle")); // Prints: true

// Finding the size of a Set:
System.out.println(shapes.size());  //
Prints: 2

// Iterate through a Set:
for (String item: shapes) {
  System.out.println(item);
}
/* Prints:
triangle
circle
```

```
*/
```

codecademy

Print Share