Recitation 6: Enums and Collections

Recitation TA Names Here

Old-fashioned, error prone

```java
public class PlayingCard {
  // 1 Hearts, 2 Spades, 3 Clubs, 4 Diamonds
  private int suit;

  // 1 Ace, 2, …, 10, 11 Jack, 12 Queen, 13 King
  private int value;
  // …
}
```

Don’t program like this! Frought with danger. Have to use integers, e.g. `if (c.suit == 1) // …`
User may forget what 1 means and make a mistake.

Better, but still problematic

```
public class PlayingCard {
  public static final int Hearts= 1;
  public static final int Spades= 2;
  // …
  private int suit;
  private int value;
  // …
}
```

Well, still relying on integers, and user isn’t forced to use names, can still use integers. (Professionals won’t, beginners will)

Declare an enum, in a new file Suit.java:

```
public enum Suit {SPADES, CLUBS, DIAMONDS, HEARTS};
```

- New `enum` keyword
- Can use any access modifier
- **Enumerate** over all possible values
- A enum is a subclass of `java.lang.Enum`

```
public class Card {
     Suit suit;
     …
```

```
Then, user writes:

    if (c.suit == Suit.SPADES)
```

Enums: Tidbits

An enum’s constructor is **private**

The ONLY objects of class Suit that can be created are:

`Suit.SPADES, Suit.CLUBS, Suit.DIAMONDS, and Suit.HEARTS.`

```
public enum Suit {SPADES, CLUBS, DIAMONDS, HEARTS};
```

Enums: Tidbits

- `Suit.values()` returns a `Suit[]` of the possible constants
- `.ordinal()` returns the position in the list of constants (i.e. the order declared)
- Implement `Comparable` using the declaration order
- `.toString()` returns the name of the constant

Enums: Switch Statement

```
Suit s= Suit.SPADES;
switch(s) {
      case SPADES:
      case CLUBS:
            color= “black”;
      break;
      case HEARTS:
      case DIAMONDS:
            color= “red”;
      break;
```

Cases **fall-through** until reach a break statement!

Collections: Overview

- Different implementations to do (generally) the same thing
  - Store data about a group of information
- Each has benefits and drawbacks for each use case

Lists (ArrayList, LinkedList, …)	Stacks
Sets (and sorted sets)	Queues
Bags (multi-set: sets with repeated values)	Maps (and sorted maps) [like dictionaries]

Collections: ArrayLists

- **Indexed:** identify each element by a number
  `0..list.size() - 1`
- **Ordered** (due to indexing)
- **Dynamic Memory Allocation**
  - An ArrayList doubles in size if it gets too big

Useful Methods to Know	`.add(element)`
`.get(index)`	`.contains(element)`
`.remove(index)`	`.size()`

Aside: ArrayLists vs. Arrays

- Both are indexed and ordered
- Syntax differences:
    - `list.get(2)` vs. `arr[2]` when getting an element
    - `list.add(element)` vs. `arr[index] = element` for adding an element
- Dynamic Memory Allocation: arrays have **fixed amount of space**
- Know the max number of elements in the list? Use an array.
- Otherwise, use an `ArrayList`

Aside: ArrayLists vs. Arrays

If you want to maintain a list of values in an array, you need **TWO** variables:

1.) the array and 2.) its size

```
int[] b= …;
int numEles= 0;
// b[0..numEles-1] = 0
// Add 5 to the list
b[n]= 5; numEles++;
// b[0..numEles-1] = 1
```

An ArrayList maintains the size for you

```
ArrayList<Double> b= …;
//num elements in list
b.size()

//Add 5.0 to the list
b.add(5.0)
```

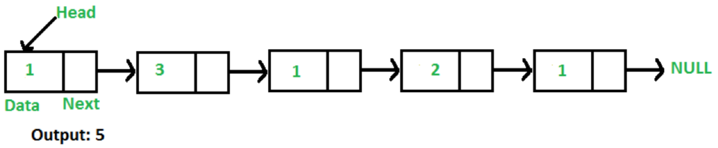
Collections: HashSets

- **Unordered** and **unindexed**
- **No duplicate elements**
  - Adding duplicates to a set does nothing
- **Very fast** for adding, removing, and contains operations!

You will learn all about hash sets later in the course! For now, just use HashSet as a nice implementation of a set

Useful Methods to Know	`.add(element)`
`.contains(element)`	`.remove(element)`
`.size()`	`.isEmpty()`

Collections: LinkedLists



- **Ordered,** but **not quite indexed** like an `ArrayList`
- Start at the head or tail and traverse through the List
- You implement this in A3!

Useful Methods to Know	`.add(element)`
`.get(index)`	`.remove()`
`.size()`	`.prepend(element)`

Collections: Stacks

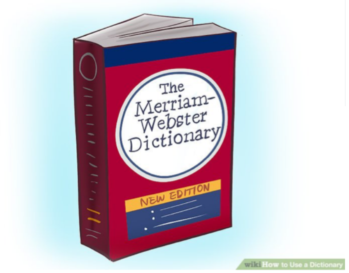
- **Ordered,** but **not indexed**
- **Last in, first out** ordering (LIFO)
- Add to the top, remove from the top

Useful Methods to Know	`.push(element)`
`.pop()`	`.empty()`
`.peek()`	

Collections: Queues

- **Ordered,** but **not indexed**
- **First in, first out** ordering (FIFO)
- Add to the top, remove from the bottom

Useful Methods to Know	`.add(element)`
`.poll()`	`.isEmpty()`
`.peek()`	

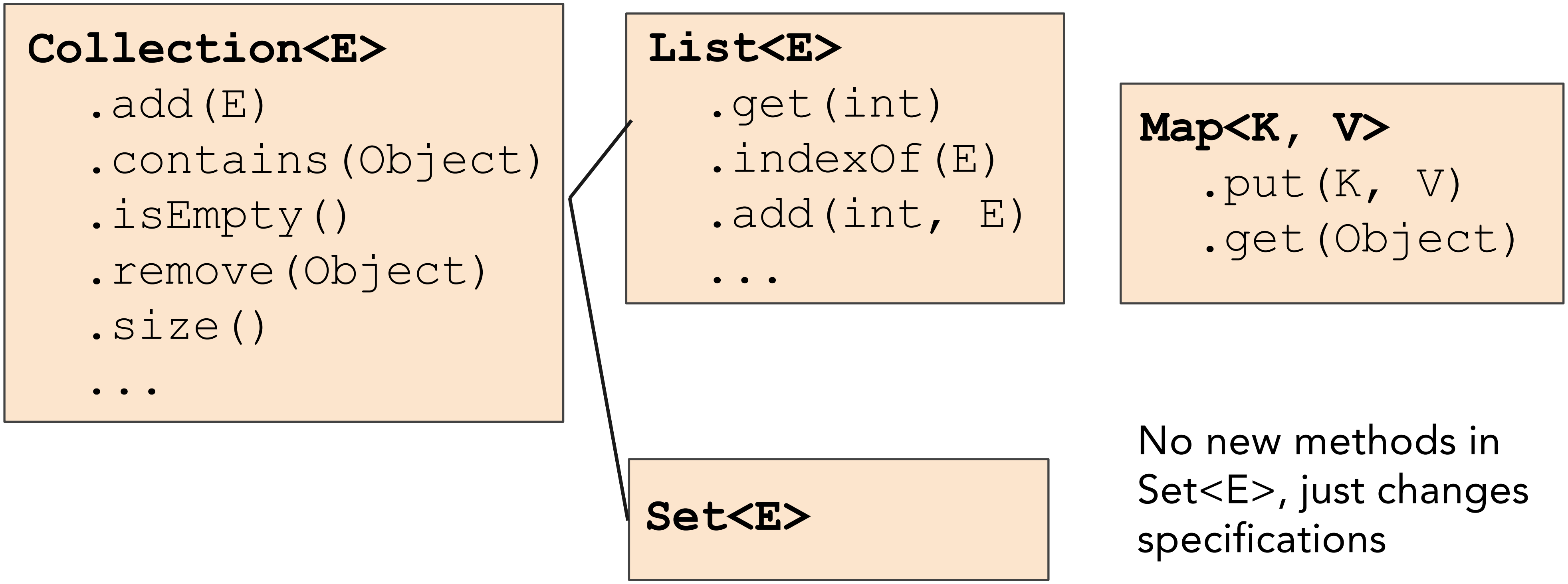


Collections: HashMap

- **Indexed** by **keys**, ordering depends on implementation
- **Key-value pairs (in dictionary: word-meaning pairs)**
- Like a **dictionary** in Python

Useful Methods to Know	`.put(key, value)`
`.get(key)`	`.containsKey(key)`
`.keySet()`	`.remove(key)`

Important Interfaces & Classes



```
Collection<E>
   .add(E)
   .contains(Object)
   .isEmpty()
   .remove(Object)
   .size()
   ...
```

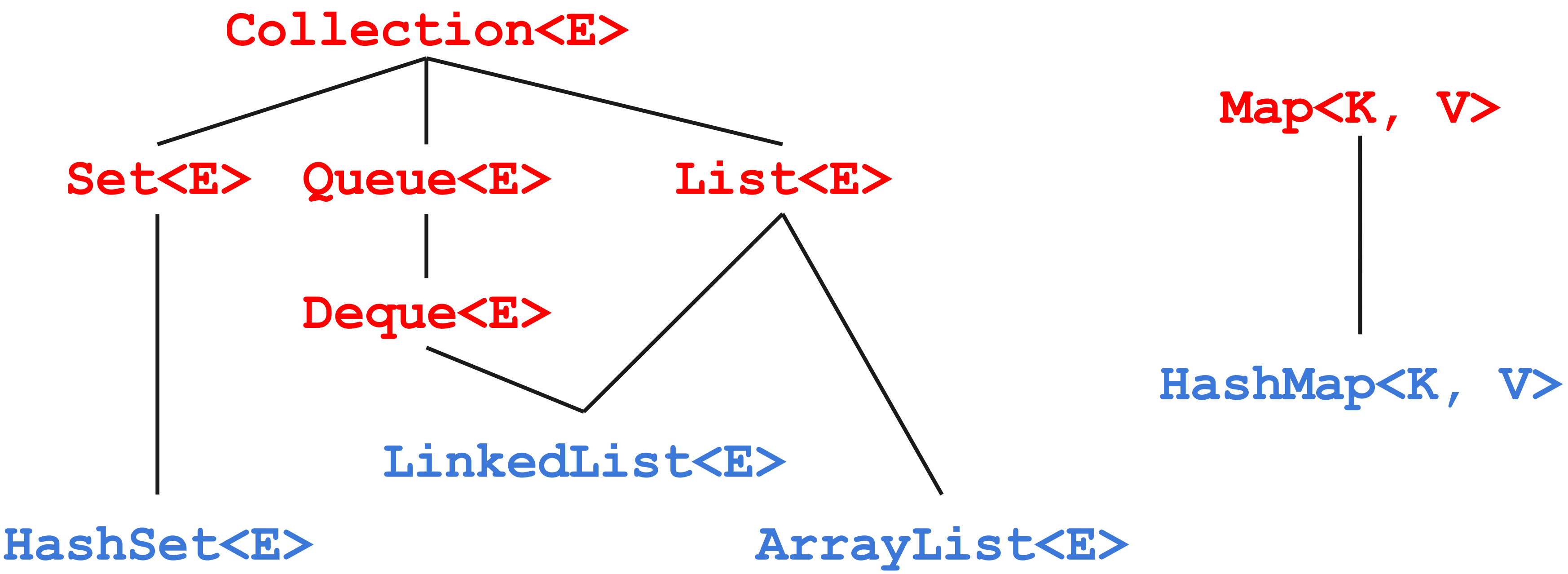
```
List<E>
   .get(int)
   .indexOf(E)
   .add(int, E)
   ...
```

```
Set<E>
```

```
Map<K, V>
   .put(K, V)
   .get(Object)
```

No new methods in Set<E>, just changes specifications

Important Interfaces & Classes



Iterating Without Indices: For-each Loop

```
HashSet<E> set= new HashSet<E>();
// .. store values in the set ..

// for (EleType varName : Collection) { ... }
for (E element : set) {
      // process each element
      System.out.println(element);
}
```

Collection Problems & Practice

1. Remove duplicates from an array
2. Find all negative numbers in an array
3. Create a random note
4. Implement a Stack with a max API
5. Braces parsing

Remove Duplicates

```
/**
 * [removeDups] removes all duplicates from
 * an array of integers.
 */
public static Integer[] removeDups(int[] arr) {
      // TODO: Implement me!
}
```

Find Negative Numbers

```
/**
 * [findNegNums] finds all negative numbers
 * in an array and returns those integers
 */
public static Integer[] findNegNums(int[] arr) {
      // TODO: Implement me!
}
```

Create Ransom Note

```java
/**
 * [isRansomNote] is true if you can use the
 * letters in the magazine to create a ransom
 * note.
 */
public static boolean isRansomNote(String note,
String magazine) {
      // TODO: Implement me!
}
```

Stack with Max() function in O(1) time

```
/**
 * MaxStack has normal Stack functionality, but
 * also includes a .max() function that returns
 * the max value in the stack in constant time.
 */
public class MaxStack {
      // TODO: Implement me!
}
```

Braces Parsing

```
/**
 * [isValidParen] is true the format of square
 * and parenthesis are oriented correctly.
 * Ex: “(())” -> true, “([)]” -> false,
 *     “(()” -> false, “)(“ -> false
 */
public static boolean isValidParen(String str) {
      // TODO: Implement me!
}
```