

Using Java Collections



EECS2030 B: Advanced
Object Oriented Programming
Fall 2019

CHEN-WEI WANG

Learning Outcomes

Understand:

- Method Header
- Parameters vs. Arguments
- Self-Exploration of Java API

Application Programming Interface (API)

- Each time before you start solving a problem:
 - As a **beginner**, crucial to implement **everything** by yourself.
 - As you get more **experienced**, first check to see if it is already solved by one of the library classes or methods.

Rule of the Thumb: DO NOT REINVENT THE WHEEL!

- An **Application Programming Interface (API)** is a collection of **programming facilities** for **reuse** and building your applications.
- Java API contains a library of **classes** (e.g., Math, ArrayList, HashMap) and **methods** (e.g., sqrt, add, remove):

<https://docs.oracle.com/javase/8/docs/api/>

- To use a library class, put a corresponding `import` statement:

```
import java.util.ArrayList;
class MyClass {
    ArrayList<String> myList;
    ... /* call methods on myList */
}
```

Classes vs. Methods

- A **method** is a **named** block of code **reusable** by its name.
e.g., As a user of the `sqrt` method (from the `Math` class):
 - Implementation code of `sqrt` is **hidden** from you.
 - You only need to know how to **call** it in order to use it.
- A **non-static method** must be called using a **context object**.
e.g., Illegal to call `ArrayList.add("Suyeon")`. Instead:

```
ArrayList<String> list = new ArrayList<String>();  
list.add("Suyeon")
```
- A **static method** can be called using the **name of its class**.
e.g., By calling `Math.sqrt(1.44)`, you are essentially **reusing** a block of code, **hidden** from you, that will be executed and calculate the square root of the input value you supply (i.e., 1.44).
- A **class** contains a collection of **related** methods.
e.g., The `Math` **class** supports **methods** related to more advanced mathematical computations beyond the simple arithmetical operations we have seen so far (i.e., +, -, *, /, and %).

Parameters vs. Arguments

- **Parameters** of a *method* are its *input variables* that you read from the API page.
e.g., `double pow(double a, double b)` has:
 - two parameters `a` and `b`, both of type `double`
 - one output/return value of type `double`
- **Arguments** of a *method* are the specific *input values* that you supply/pass in order to use it.
e.g., To use the `pow` method to calculate 3.4^5 , we call it by writing `Math.pow(3.4, 5)`.
- **Argument values** must conform to the corresponding *parameter types*.
e.g., `Math.pow("three point four", "5")` is an invalid call!

Header of a Method

Header of a *method* informs users of the *intended usage*:

- *Name* of method
- List of *inputs* (a.k.a. *parameters*) and their types
- Type of the *output* (a.k.a. *return type*)
 - Methods with the `void` return type are mutators.
 - Methods with non-`void` return types are accessors.

e.g. In Java API, the **Method Summary** section lists *headers* and descriptions of methods.

Example Method Headers: ArrayList Class

An ArrayList acts like a “resizable” array (indices start with 0).

int	size() Returns the number of elements in this list.
boolean	add(E e) Appends the specified element to the end of this list.
void	add(int index, E element) Inserts the specified element at the specified position in this list.
boolean	contains(Object o) Returns true if this list contains the specified element.
E	remove(int index) Removes the element at the specified position in this list.
boolean	remove(Object o) Removes the first occurrence of the specified element from this list, if it is present.
int	indexOf(Object o) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
E	get(int index) Returns the element at the specified position in this list.

Generic Parameters: ArrayList Class (1)

- Consider the API of `ArrayList`:

```
1 class ArrayList<E> {  
2     boolean add(E e)  
3     E remove(int index)  
4     E get(int index)  
5 }
```

- **L1** *declares* a generic parameter `E`, denoting the type of values stored in the array list.
- All other occurrences of `E` at **L2**, **L3**, and **L4** refer to whatever `E` is *instantiated* by some caller.
- A caller of `ArrayList` may *instantiate* `E` to any known class:

```
1 ArrayList<String> list1 = new ArrayList<String> ();  
2 ArrayList<Point> list2 = new ArrayList<Point> ();
```


Generic Parameters: ArrayList Class (2)

A caller of `ArrayList` may *instantiate* `E` to any known class:

```
1 ArrayList<String> list1 = new ArrayList<String> ();  
2 ArrayList<Point> list2 = new ArrayList<Point> ();
```

- o **L1** instantiate `E` to `String`, as if the following class was declared:

```
class ArrayList {  
    boolean add(String e)  
    String remove(int index)  
    String get(int index)  
}
```

- o **L2** instantiate `E` to `Point`, as if the following class was declared:

```
class ArrayList {  
    boolean add(Point e)  
    Point remove(int index)  
    Point get(int index)  
}
```

Case Study: Using an ArrayList

```
1 import java.util.ArrayList;
2 public class ArrayListTester {
3     public static void main(String[] args) {
4         ArrayList<String> list = new ArrayList<String>();
5         println(list.size());
6         println(list.contains("A"));
7         println(list.indexOf("A"));
8         list.add("A");
9         list.add("B");
10        println(list.contains("A")); println(list.contains("B")); println(list.contains("C"));
11        println(list.indexOf("A")); println(list.indexOf("B")); println(list.indexOf("C"));
12        list.add(1, "C");
13        println(list.contains("A")); println(list.contains("B")); println(list.contains("C"));
14        println(list.indexOf("A")); println(list.indexOf("B")); println(list.indexOf("C"));
15        list.remove("C");
16        println(list.contains("A")); println(list.contains("B")); println(list.contains("C"));
17        println(list.indexOf("A")); println(list.indexOf("B")); println(list.indexOf("C"));
18
19        for(int i = 0; i < list.size(); i ++ ) {
20            println(list.get(i));
21        }
22    }
23 }
```

Example Method Headers: HashTable Class

A HashTable acts like a two-column table of (searchable) keys and values.

int	size() Returns the number of keys in this hashtable.
boolean	containsKey(Object key) Tests if the specified object is a key in this hashtable.
boolean	containsValue(Object value) Returns true if this hashtable maps one or more keys to this value.
V	get(Object key) Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
V	put(K key, V value) Maps the specified key to the specified value in this hashtable.
V	remove(Object key) Removes the key (and its corresponding value) from this hashtable.

Generic Parameters: Hashtable Class (1)

- Consider the API of Hashtable:

```
1 class Hashtable<K, V> {  
2     V put(K key, V value)  
3     V get(Object key)  
4 }
```

- **L1 declares** two generic parameters K and V , denoting types of keys and values stored in the hash table.
 - All other occurrences of K and V at **L2**, **L3**, and **L4** refer to whatever K and V are *instantiated* by some caller.
- A caller of `ArrayList` may *instantiate* E to any known class:

```
1 Hashtable<String, Integer> t1 = new Hashtable<String, Integer>();  
2 Hashtable<Integer, String> t2 = new Hashtable<Integer, String>();
```

Generic Parameters: Hashtable Class (2)

A caller of `Hashtable` may *instantiate* `K` and `V` to any known classes:

```
1 Hashtable<String, Integer> t1 = new Hashtable<String, Integer>();
2 Hashtable<Integer, String> t2 = new Hashtable<Integer, String>();
```

- o **L1** instantiate `K` and `V` to, respectively, `String` and `Integer`, as if the following class was declared:

```
class Hashtable {
    Integer put(String key, Integer value)
    Integer get(Object key)
}
```

- o **L2** instantiate `K` and `V` to, respectively, `Integer` and `String`, as if the following class was declared:

```
class Hashtable {
    String put(Integer key, String value)
    String get(Object key)
}
```

Case Study: Using a HashTable

```
1 import java.util.Hashtable;
2 public class HashTableTester {
3     public static void main(String[] args) {
4         Hashtable<String, String> grades = new Hashtable<String, String>();
5         System.out.println("Size of table: " + grades.size());
6         System.out.println("Key Alan exists: " + grades.containsKey("Alan"));
7         System.out.println("Value B+ exists: " + grades.containsValue("B+"));
8         grades.put("Alan", "A");
9         grades.put("Mark", "B+");
10        grades.put("Tom", "C");
11        System.out.println("Size of table: " + grades.size());
12        System.out.println("Key Alan exists: " + grades.containsKey("Alan"));
13        System.out.println("Key Mark exists: " + grades.containsKey("Mark"));
14        System.out.println("Key Tom exists: " + grades.containsKey("Tom"));
15        System.out.println("Key Simon exists: " + grades.containsKey("Simon"));
16        System.out.println("Value A exists: " + grades.containsValue("A"));
17        System.out.println("Value B+ exists: " + grades.containsValue("B+"));
18        System.out.println("Value C exists: " + grades.containsValue("C"));
19        System.out.println("Value A+ exists: " + grades.containsValue("A+"));
20        System.out.println("Value of existing key Alan: " + grades.get("Alan"));
21        System.out.println("Value of existing key Mark: " + grades.get("Mark"));
22        System.out.println("Value of existing key Tom: " + grades.get("Tom"));
23        System.out.println("Value of non-existing key Simon: " + grades.get("Simon"));
24        grades.put("Mark", "F");
25        System.out.println("Value of existing key Mark: " + grades.get("Mark"));
26        grades.remove("Alan");
27        System.out.println("Key Alan exists: " + grades.containsKey("Alan"));
28        System.out.println("Value of non-existing key Alan: " + grades.get("Alan"));
29    }
30 } 14 of 16
```

- Use of ArrayList:

https://www.youtube.com/watch?v=SJjzM2DKA3M&index=2&list=PL5dxAmCmjv_4rOxjfTfIxNp42vO8SnT8n

- Use of HashMap:

https://www.youtube.com/watch?v=_PV7dP5aIMg&list=PL5dxAmCmjv_4rOxjfTfIxNp42vO8SnT8n&index=3

- iPad Notes:

<https://www.eecs.yorku.ca/~jackie/teaching/tutorials/notes/Tutorial%20on%20Java%20Collections.pdf>

Index (1)

Learning Outcomes

Application Programming Interface (API)

Classes vs. Methods

Parameters vs. Arguments

Header of a Method

Example Method Headers: ArrayList Class

Generic Parameters: ArrayList Class (1)

Generic Parameters: ArrayList Class (2)

Case Study: Using an ArrayList

Example Method Headers: Hashtable Class

Generic Parameters: Hashtable Class (1)

Generic Parameters: Hashtable Class (2)

Case Study: Using a Hashtable

Tutorial Videos