

# Using API in Java



EECS1022:  
Programming for Mobile Computing  
Winter 2018

CHEN-WEI WANG

# Learning Outcomes

---

Understand:

- Self-Exploration of Java API
- Method Signature
- Parameters vs. Arguments
- Non-Static Methods and Collection Library
- Static Methods and Math Library

# Application Programming Interface (API)

- 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 myList;
    ...
}
```

- 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!**

# 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 **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 **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 **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!

# Signature of a Method

**Signature** 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 *signatures* and descriptions of methods.

## Example Method Signatures: Math Class

- The class Math contains methods for performing basic numeric operations such as the elementary exponential, logarithm, square root, and trigonometric functions.

| Modifier and Type | Method and Description   |
|-------------------|--|
| static double     | <b>abs</b> (double a)<br>Returns the absolute value of a double value. |
| static float      | <b>abs</b> (float a)<br>Returns the absolute value of a float value.   |
| static int        | <b>abs</b> (int a)<br>Returns the absolute value of an int value.      |
| static long       | <b>abs</b> (long a)<br>Returns the absolute value of a long value.     |

- Method Overloading**: multiple methods sharing the *same name*, but with *distinct lists* of parameters (e.g., abs method).
- The abs method being static allows us to write

```
Math.abs(-2.5).
```

# Case Study: Guessing a Number

**Problem:** Your program:

- *internally* and *randomly* sets a number between 0 and 100
- *repeatedly* asks the user to enter a guess, and hints if they got it, or should try something smaller or larger
- once the user got it and still wishes to continue, *repeat* the game with a different number

**Hints:**

```
static double      random()
```

Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.

```
(int) Math.random() * 100
```

or

```
(int) (Math.random() * 100)
```

??



# Example Method Signatures: ArrayList Class

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

|         |   |
|---------|---|
| int     | <b>size()</b><br>Returns the number of elements in this list.   |
| boolean | <b>add(E e)</b><br>Appends the specified element to the end of this list.   |
| void    | <b>add(int index, E element)</b><br>Inserts the specified element at the specified position in this list.   |
| boolean | <b>contains(Object o)</b><br>Returns true if this list contains the specified element.  |
| E       | <b>remove(int index)</b><br>Removes the element at the specified position in this list.   |
| boolean | <b>remove(Object o)</b><br>Removes the first occurrence of the specified element from this list, if it is present.  |
| int     | <b>indexOf(Object o)</b><br>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       | <b>get(int index)</b><br>Returns the element at the specified position in this list.  |

# 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 }
```

See **Java Data Types** (3.3.1) – (3.3.2) in **Classes and Objects** for another example on ArrayList.

# Example Method Signatures: HashTable Class

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

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

# 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", "E");
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 }12 of 13
```

# Index (1)

---

**Learning Outcomes**

**Application Programming Interface (API)**

**Classes vs. Methods**

**Parameters vs. Arguments**

**Signature of a Method**

**Example Method Signatures: Math Class**

**Case Study: Guessing a Number**

**Example Method Signatures: ArrayList Class**

**Case Study: Using an ArrayList**

**Example Method Signatures: HashTable Class**

**Case Study: Using a HashTable**