Using API in Java

EECS1021:
Object Oriented Programming:
from Sensors to Actuators
Winter 2019

CHEN-WEI WANG

YORK
UNIVERSITÉ
UNIVERSITY

Learning Outcomes

LASSONDE

Understand:

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

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:

```java
import java.util.ArrayList;
class MyClass {
  ArrayList 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 <u>mutators</u>.
  • Methods with non-`void` return types are <u>accessors</u>.

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

Example Method Headers: 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	**abs**(double a) Returns the absolute value of a double value.
static float	**abs**(float a) Returns the absolute value of a float value.
static int	**abs**(int a) Returns the absolute value of an int value.
static long	**abs**(long a) 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)`.

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)
```

??

LASSONDE
SCHOOL OF ENGINEERING

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.

Case Study: Using an ArrayList

```java
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 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.

Case Study: Using a HashTable

```java
import java.util.Hashtable;
public class HashTableTester {
  public static void main(String[] args) {
    Hashtable<String, String> grades = new Hashtable<String, String>();
    System.out.println("Size of table: " + grades.size());
    System.out.println("Key Alan exists: " + grades.containsKey("Alan"));
    System.out.println("Value B+ exists: " + grades.containsValue("B+"));
    grades.put("Alan", "A");
    grades.put("Mark", "B+");
    grades.put("Tom", "C");
    System.out.println("Size of table: " + grades.size());
    System.out.println("Key Alan exists: " + grades.containsKey("Alan"));
    System.out.println("Key Mark exists: " + grades.containsKey("Mark"));
    System.out.println("Key Tom exists: " + grades.containsKey("Tom"));
    System.out.println("Key Simon exists: " + grades.containsKey("Simon"));
    System.out.println("Value A exists: " + grades.containsValue("A"));
    System.out.println("Value B+ exists: " + grades.containsValue("B+"));
    System.out.println("Value C exists: " + grades.containsValue("C"));
    System.out.println("Value A+ exists: " + grades.containsValue("A+"));
    System.out.println("Value of existing key Alan: " + grades.get("Alan"));
    System.out.println("Value of existing key Mark: " + grades.get("Mark"));
    System.out.println("Value of existing key Tom: " + grades.get("Tom"));
    System.out.println("Value of non-existing key Simon: " + grades.get("Simon"));
    grades.put("Mark", "F");
    System.out.println("Value of existing key Mark: " + grades.get("Mark"));
    grades.remove("Alan");
    System.out.println("Key Alan exists: " + grades.containsKey("Alan"));
    System.out.println("Value of non-existing key Alan: " + grades.get("Alan"));
  }
}
```

Index (1)