HW 0B: A Java Crash Course (Part 2)

Introduction

In this assignment, we will continue going through basic Java syntax, with a focus on *using* the basic data structures. This assignment will relate the basic Java data structures to Python. We strongly recommend completing this material as soon as possible (soon after Lecture 2), as early assignments will use Java data structures.

This assignment is structured assuming that you have taken a course that used Python (e.g. CS 61A, CS 88, Data 8), or have equivalent experience, and assuming that you have completed HW 0A.

Goals and Outcomes

In this HW, you will complete a few Java exercises, and translate your existing programming knowledge to Java. By the end of this HW you will:

- Understand how fundamental data structures in Python translate to Java
- Be able to read short Java snippets that use data structures
- Be able to implement programs in Java that use data structures

Language Constructs

Types

In Java, there are two kinds of types: primitive types and reference types. Primitive types are lowercase, and we named the ones that we care about in Part A: `boolean`, `int`, `char`, `double`. Pretty much every other type is a reference type, such as `String`. If a type starts with a capital letter, it is likely a reference type.

You will learn more about the distinction between primitive and reference types in Lecture 4, but for this homework, you will need to know that each primitive has a corresponding reference type (`Boolean`, `Integer`, `Character`, `Double`). If you are using "generics" to declare a data structure, you *must* use the reference type. You can seamlessly convert between a primitive type and its reference type.

`null`

Java also has `null`, which is the approximate equivalent of `None` in Python. Any reference type can be assigned a value of `null`. If we try to access an instance member or call an instance method from a value of `null`, we will see an error called a `NullPointerException`.

Arrays (fixed-size)

Java arrays are a lot like Python lists. However, Java arrays are *fixed-size*, so we can't add or remove elements (that is, no `append`, `remove`, etc.).

Python	Java

`zeroedLst = [0, 0, 0]`
`lst = [4, 7, 10]`
`lst[0] = 5`
`print(lst[0])`
`print(lst)`
`print(len(lst))`
	`int[] zeroedArray = new int[3];`
	`int[] array = {4, 7, 10};`
	`array[0] = 5;`
	`System.out.println(array[0]);`
	`System.out.println(Arrays.toString(array));`
	`System.out.println(array.length);`

- In `new int[3]`, the `int` is the type in the array; and `3` is the length. With this syntax, all elements take on their “default value”. For `int`, this is 0.
- Arrays do not print nicely, for reasons beyond the scope of HW 0. To print an array, you can call `Arrays.toString(array)`.
- Arrays do not have a length *method*. It is an *instance variable*, so it does not have parentheses.
- Java does not support *negative indexing* or *slicing*.

Foreach Loop

Python	Java

`lst = [1, 2, 3]`
`for i in lst:`
`    print(i)`
	`int[] array = {1, 2, 3};`
	`for (int i : array) {`
	`    System.out.println(i);`
	`}`

- Notice the type declaration of the iterating variable, as well as the usage of `:` instead of `in`.
- We can also use this syntax on certain other types, such as `List`s and `Set`s.

Lists (resizable)

Python	Java

`lst = []`
`lst.append("zero")`
`lst.append("one")`
`lst[0] = "zed"`
	`List<String> lst = new ArrayList<>();`
	`lst.add("zero");`
	`lst.add("one");`
	`lst.set(0, "zed");`

Python	Java
`print(l[0])`	`System.out.println(lst.get(0));`
`print(len(l))`	`System.out.println(lst.size());`
`if "one" in lst:`	`if (lst.contains("one")) {`
`    print("one in lst")`	`    System.out.println("one in lst");`
	`}`
`for elem in lst:`	`for (String elem : lst) {`
`    print(elem)`	`    System.out.println(elem);`
	`}`

- Java has the `List` interface. We largely use the `ArrayList` implementation.
- The `List` interface is *parameterized* by the type it holds, using the angle brackets `<` and `>`.
- `List`s, again, do not support slicing or negative indexing.

Sets

Python	Java
`s = set()`	`Set<Integer> set = new HashSet<>();`
`s.add(1)`	`set.add(1);`
`s.add(1)`	`set.add(1);`
`s.add(2)`	`set.add(2);`
`s.remove(2)`	`set.remove(2);`
`print(len(s))`	`System.out.println(set.size());`
`if 1 in s:`	`if (set.contains(1)) {`
`    print("1 in s")`	`    System.out.println("1 in set");`
	`}`
`for elem in s:`	`for (int elem : set) {`
`    print(elem)`	`    System.out.println(elem);`
	`}`

- Java has the `Set` interface. There are two main implementations: `TreeSet`, and `HashSet`. `TreeSet` keeps its elements in “sorted” order, and is “fast.” In contrast, `HashSet` does not have a defined “order”, but is (usually) **really** “fast.”
- A `Set` canot contain duplicate items. If we try to add a duplicate item, it simply does nothing.

Dictionaries / Maps

Python

```python
d = {}
d["hello"] = "hi"
d["hello"] = "goodbye"
print(d["hello"])
print(len(d))
if "hello" in d:
    print("\"hello\" in d")

for key in d.keys():
    print(key)
```

Java

```java
Map<String, String> map = new HashMap<>();
map.put("hello", "hi");
map.put("hello", "goodbye");
System.out.println(map.get("hello"));
System.out.println(map.size());
if (map.containsKey("hello")) {
    System.out.println("\"hello\" in map");
}
for (String key : map.keySet()) {
    System.out.println(key);
}
```

- Java has the `Map` interface. There are two main implementations: TreeMap, and HashMap. Similarly to sets, `TreeMap` keeps its keys sorted and is fast; `HashMap` has no defined order and is (usually) **really** fast.
- In the angled brackets, we have the “key type” first, followed by the “value type”.
- `Map`s cannot directly be used with the `:` for loop. Typically, we call `keySet` to iterate over a set of the keys.

Classes

Python

```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distanceTo(self, other):
        return math.sqrt(
            (self.x - other.x) ** 2 +
            (self.y - other.y) ** 2
        )

    def translate(self, dx, dy):
        self.x += dx
        self.y += dy
```

Java

```java
public class Point {
    public int x;
    public int y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public Point() {
        this(0, 0);
    }
    public double distanceTo(Point other) {
        return Math.sqrt(
            Math.pow(this.x - other.x, 2) +
            Math.pow(this.y - other.y, 2)
        )
    }
```

Python	Java

	public void translate(int dx, int dy) {
	        this.x += dx;
	        this.y += dy;
	    }
	}

We can use these classes as follows:

Python	Java

p1 = Point(5, 9)
p2 = Point(-3, 3)
print(f"Point 1: ({p1.x}, {p1.y})")
print("Distance:", p1.distanceTo(p2))
p1.translate(2, 2)
print(f"Point 1: ({p1.x}, {p1.y})")
	Point p1 = new Point(5, 9);
	Point p2 = new Point(-3, 3);
	System.out.println("Point 1: (" + p1.x
	    + ", " + p1.y + ")");
	System.out.println("Distance: "
	    + p1.distanceTo(p2));
	p1.translate(2, 2);
	System.out.println("Point 1: (" + p1.x
	    + ", " + p1.y + ")");

Programs

Let’s look at some Java programs that use data structures and classes. Here are some simple ones that you might find yourself referring to if you forget how to do something.

Index of Minimum of a List of Numbers

Python	Java

def min_index(numbers):
    # Assume len(numbers) >= 1
    m = numbers[0]
    idx = 0
    for i in range(len(numbers)):
        if numbers[i] < m:
            m = numbers[i]
	public static int minIndex(int[] numbers) {
	    // Assume numbers.length >= 1
	    int m = numbers[0];
	    int idx = 0;
	    for (int i = 0; i < numbers.length; i++) {
	        if (numbers[i] < m) {
	            m = numbers[i];
	            idx = i;
	        }

Python	Java

`        idx = i`
`    return idx`
	`    }`
	`    return idx;`
	`}`

Exceptions

Lastly, let’s look at how we can throw exceptions in Java compared to Python with previous example.

Python

```
def minIndex(numbers):
    if len(numbers) == 0:
        raise Exception("There are no elements in the list!")
    m = numbers[0]
    idx = 0

    ...

    return m
```

Java

```
public static int minIndex(int[] numbers) {
    if (numbers.length == 0) {
        throw new Exception("There are no elements in the array!")
    }
    int m = numbers[0];
    int idx = 0;

    ...

    return m;
}
```

Programming Exercise

In order to get you more familiar with Java syntax and testing, there are couple of exercises for you to solve! After you complete the functions, we have provided couple tests for you to test. Although we have provided tests, you are welcomed to write your own too! Writing tests is not only crucial for this class but it is one of the most important skills to have in general. It reinforces our understanding of what specific method is supposed to do and allows us to catch edge cases! You will have more exercises for testing but we want you to get exposed early on.

While completing the assignment, you may need to use different data structures like `ArrayList` and `TreeMap`. In order to import these classes, if you hover over wherever you are using the data structures, IntelliJ will give you option to import it or you can do it manually by adding:

```
import java.util.ArrayList;
import java.util.TreeMap;
```

ListExercises

`ListExercises.java` has 4 different methods for you to complete:

- `sum`: This method takes a list `List<Integer> L` and returns the total sum of the elements in that list. If the list is empty, it method should return 0.
- `evens`: This method takes a list `List<Integer> L` and returns a *new* list containing the even numbers of the given list. If there are no even elements, it should return an empty list.
- `common`: This method takes two lists `List<Integer> L1`, `List<Integer> L2` and returns a *new* list containing the common item of the two given lists. If there are no common items, it should return an empty list.
- `countOccurrencesOfC`: This method takes a list and a character `List<String> words`, `char c` and returns the number of occurrences of the given character in a list of strings. If the character does not occur in any of the words, it should return 0.

For this part, you can import `ArrayList`.

MapExercises

`MapExercises.java` has 3 different methods for you to complete:

- `letterToNum`: This method returns a map from every lower case letter to the number corresponding to that letter starting with ‘a’ is 1.
- `squares`: This method takes a list `List<Integer> nums` and returns a map from the integers in the list to their squares. If the given list is empty, it should return an

empty map.

- `countWords`: This method takes a list `List<String> words` and returns a map of the counts of all words that appear in a list of words. If the given list is empty, it should return an empty map.

For this part, you can import `TreeMap`.

Deliverables

- `ListExercises.java`
- `MapExercises.java`

For this assignment, you need to complete the methods in `ListExerises` and `MapExercises`. Make sure you test before you submit it to Gradescope. Although we do not have a submission limit for this specific assignment, in the future it is encouraged to use existing tests and write your own tests to see if your methods work before submitting your code to the autograder.

Last built: 2023-10-26 18:40 UTC