Sets and Maps

Arrays and lists are straightforward for storing a collection of objects. In this activity, you'll gain experience with two other kinds of collections. Sets and maps are quite useful for implementing a wide variety of algorithms.

Manager:		Recorder:	
Presenter:		Reflector:	

Content Learning Objectives

After completing this activity, students should be able to:

- Summarize methods in the `Set` interface.
- Summarize methods in the `Map` interface
- Explain differences between sets and maps.

Process Skill Goals

During the activity, students should make progress toward:

- Discuss results while running code interactively. (Oral Communication)

Facilitation Notes

When reporting out Model 1, point out there are no get/set methods. Discuss example uses of sets (e.g., eliminating duplicates, testing membership, comparing differences).

When reporting out Model 2, compare maps with sets. For example, maps have a `get` method, but sets have `contains`. Discuss example uses of maps (e.g., counting words in a document).

Key questions: #5, #11, #13

Model 1 Set of Strings

Type each line of code below in *JShell*, ***one at a time***, and record the results. You only need to record the output to the right of the "==>" symbol. For example, if *JShell* outputs `$8 ==> true`, then just write `true`. If an error occurs, record the error message.

Java code	Shell output
`Set<String> names = new Set<>();`	java.util.Set is abstract; cannot be instantiated
`Set<String> names = new HashSet<>();`	[]
`names.add("WAS")`	true
`names.add("BAL")`	true
`names.add("PHI")`	true
`names`	[PHI, WAS, BAL]
`names.contains("DEN")`	false
`names.add("DEN")`	true
`names.contains("DEN")`	true
`names.contains("den")`	false
`names.add("DEN")`	false
`names.add(123)`	int cannot be converted to java.lang.String
`names.size()`	4
`names`	[PHI, WAS, DEN, BAL]
`names.remove("WAS")`	true
`names.remove("IND")`	false
`names`	[PHI, DEN, BAL]
`names.isEmpty()`	false
`names.clear()`	
`names.size()`	0
`names.isEmpty()`	true

Questions (20 min)

Start time:

1. For the collection above:

a) What is the interface name? Set

b) What is the class name? HashSet

c) What is the variable name? names

d) What is the element type? String

2. Based on the shell output, describe what the following methods return:

a) `add` true if this set did not already contain the specified element

b) `remove` true if this set contained the specified element

3. Consider the contents of `names` just before `"WAS"` was removed.

a) What was the size of `names` at this point? 4

b) How many times was the `add` method called? 6

c) Explain why these two numbers are different.

DEN was added a second time, but it was already in the set.
123 could not be added, because its type didn't match.

4. Continuing the previous question:

a) In what order were the strings added to the set? WAS, BAL, PHI, DEN

b) In what order were they displayed in the output? PHI, WAS, DEN, BAL

c) Why do you think the two orders are different?

Sets have no defined order

5. In your own words, summarize what a `Set` is in Java. Give an example from everyday life.

The `java.util.Set` interface models the mathematical notion of a *set*, i.e., a collection that has no duplicates. For example, you could have the set of all computer science majors.

6. In discrete mathematics, sets have three basic operations:

- Union $(S \cup T)$: all elements in S or T (or both)
- Intersection $(S \cap T)$: elements in both S and T
- Difference $(S - T)$: elements in S but not in T

Based on the documentation for `java.util.Set`, which methods implement these operations?

```
set1.addAll(set2);    // Union
set1.retainAll(set2); // Intersection
set1.removeAll(set2); // Difference
```

Model 2 Map of Team Names

The following abbreviations are for National Football League (NFL) teams:

ATL	Atlanta Falcons
DEN	Denver Broncos
IND	Indianapolis Colts
MIA	Miami Dolphins
SEA	Seattle Seahawks

Complete the table below using *JShell* (the same way you did for Model 1).

Java code	Shell output
`Map<String, String> teams;`	null
`teams = new Map<>();`	java.util.Map is abstract; cannot be instantiated
`teams = new HashMap<>();`	{}
`teams.isEmpty()`	true
`teams.put("MIA", "Miami Dolphins")`	null
`teams.put("MIA", "Miami")`	"Miami Dolphins"
`teams.size()`	1
`teams`	{MIA=Miami}
`teams.put("ATL", "Atlanta")`	null
`teams.put("SEA", "Seattle")`	null
`teams`	{MIA=Miami, ATL=Atlanta, SEA=Seattle}
`teams.containsKey("ATL")`	true
`teams.containsKey("DEN")`	false
`teams.containsValue("Miami")`	true
`teams.containsValue("Dolphins")`	false
`teams.get("SEA")`	"Seattle"
`teams.get("IND")`	null
`teams.get(0)`	null
`teams.remove("MIA")`	"Miami"
`teams.remove("MIA")`	null
`teams`	{ATL=Atlanta, SEA=Seattle}
`teams.keySet()`	[ATL, SEA]
`teams.values()`	[Atlanta, Seattle]

Questions (25 min)

Start time:

7. For the collection above:

a) What is the interface? Map

b) What is the class? HashMap

c) What type of keys? String

d) What type of values? String

8. Based on the shell output, describe what the following methods return:

a) `put` The previous value associated with the key, or null if not mapped.

b) `get` The value to which the specified key is mapped, or null if none.

9. What type of object does the `keySet` method return? Describe its contents.

In this example, it returns a `Set<String>` containing all the abbreviations.

10. What type of object does the `values` method return? Describe its contents.

In this example, it returns a `Collection<String>` containing all the team names.

11. In your own words, summarize what a `Map` is in Java. Give an example from everyday life.

An object that "maps" keys with values. A map cannot contain duplicate keys; each key can map to at most one value. For example, you could maps English words to their definitions.

12. Why did `teams.get(0)` return null, even though there were values in the map?

You cannot use "indexes" to access values in a map; only keys. There is no value mapped to the key of 0. Besides, keys in the `teams` map need to be strings.

13. Write Java code that defines a map named `dow` that represents the seven days of the week as follows: Sun=1, Mon=2, Tue=3, etc. Run your code in *JShell* to make sure it works.

```
Map<String, Integer> dow = new HashMap<>();
dow.put("Sun", 1);
dow.put("Mon", 2);
dow.put("Tue", 3);
dow.put("Wed", 4);
dow.put("Thu", 5);
dow.put("Fri", 6);
dow.put("Sat", 7);
```

14. Print the `dow` variable in *JShell*. What do you notice about the order of its contents?

The contents appear to be listed in a random order:

```
{Thu=5, Tue=3, Wed=4, Sat=7, Fri=6, Sun=1, Mon=2}
```