What do programmers do when they are hungry?

They grab a byte!

Module 1-8

Collections: Maps and Sets

Objectives

- Identify when to use a Map
- Effectively use objects of the Map collection class
- Understand common Map API operations
- (talk about optimization of code)

Collections

1. Classes that live in a package
    - Packages are a way of organizing code
2. Come from standard library of classes
    - java.util package
3. Already written for you and generic enough to be useful in many situations

Maps: Introduction

Map< T, T >
Maps are used to store key value pairs.

- Examples of key value pairs: dictionary entries (word -> definition), a phone book (name -> phone number), a list of employees (employee number -> employee name)
- Key must be unique, values can be duplicated

Maps

- Unordered collection
  - Allows values to be located using user-defined keys
  - Snack machine
    - Key “a5” gets you a bag of Fritos

Maps: Declaring

Maps follow the following declaration pattern (programming to the Map interface).

```
import java.util.HashMap;
import java.util.Map;

public class MyClass {

        public static void main(String args[]) {

        Map <Integer, String> myMap = new HashMap<>();
        }
}
```

Note the we will need these 2 imports for a hash map.

We are creating a type of Map called a HashMap

We have specified that the key will be an integer and the value will be the String

Note the "**new**" keyword which **instantiates** the map.

Maps: put method

The put method adds an item to the map. The data types must match the declaration.

```
Map <Integer, String> myMap = new
HashMap<>();
myMap.put(10, "Rick");
myMap.put(2, "Beth");
myMap.put(43, "Jerry");
myMap.put(47, "Summer");
myMap.put(15, "Mortimer");
```

The put method call requires two parameters:

- The key
  - In this example it is of data type Integer
- The value
  - In this example it is of data type String
- On the highlighted line, we inserted an entry with a key of 10 and a value of Rick.

Maps: containsKey method

The containsKey method returns a boolean indicating if the key exists.

```
Map <String, String> reservations = new HashMap<>();

reservations.put("HY234-9234", "Rick");
reservations.put("HY234-4235", "Beth");
reservations.put("HY234-3234", "Jerry");

System.out.println(reservations.containsKey("HY234-4235")); // True
System.out.println(reservations.containsKey("AAAI-4235")); // False
System.out.println(reservations.containsKey("Jerry")); // False
```

- The containsKey method requires one parameter, the key you are searching for.
- containsKey returns a boolean

Note that in the last example returns false because it's not a key, it's a value

Maps: containsValue method

The containsValue method returns a boolean indicating if the value is in the Map.

```
Map <String, String> reservations = new HashMap<>();

reservations.put("HY234-9234", "Rick");
reservations.put("HY234-4235", "Beth");
reservations.put("HY234-3234", "Jerry");

System.out.println(reservations.containsValue("Rick")); // True
System.out.println(reservations.containsValue("Betsy")); // False
System.out.println(reservations.containsValue("AA234-1111")); // False
```

- The containsValue method requires one parameter, the value you are searching for.
- containsValue returns a boolean

Note that in the last example returns false because it's not a value

Maps: get method

The get method returns the value associated with a key.

```
Map <String, String> reservations = new HashMap<>();

reservations.put("HY234-9234", "Rick");
reservations.put("HY234-4235", "Beth");
reservations.put("HY234-3234", "Jerry");

String name = reservations.get("HY234-9234");
System.out.println(name); // Prints Rick

String anotherName = reservations.get("AAI93-2345");
System.out.println(name); // Prints null
```

- The get method requires one parameter, the key you are searching for.
- It will return the value associated with the key.
- If keys do not match the parameter provided, it returns a null.

Maps: remove method

The remove method removes an item from the map, given a key value.

```
Map <String, String> reservations = new HashMap<>();

reservations.put("HY234-9234", "Rick");
reservations.put("HY234-4235", "Beth");
reservations.put("HY234-3234", "Jerry");

System.out.println(reservations.get("HY234-3234"));
// Prints Jerry
reservations.remove("HY234-3234");
System.out.println(reservations.get("HY234-3234"));
 // Prints null
```

- The remove method requires one parameter, the key you are searching for.

Maps: size method

The size method lists the size of the map in terms of key value pairs present.

```
Map <String, String> reservations = new HashMap<>();

reservations.put("HY234-9234", "Rick");
reservations.put("HY234-4235", "Beth");
reservations.put("HY234-3234", "Jerry");

System.out.println(reservations.size()); // Prints 3
reservations.remove("HY234-3234");
System.out.println(reservations.size()); // Prints 2
```

- The size method requires no parameters.
- It will return an integer, the number of key value pairs present.

Maps: looping through the pairings

The keySet() method returns a Set of all keys in the Map.

```
Map <String, String> reservations = new HashMap<>();

reservations.put("HY234-9234", "Rick");
reservations.put("HY234-4235", "Beth");
reservations.put("HY234-3234", "Jerry");

Set<String> keys  = reservations.keySet();

for (String reservationNumber: keys) {
       System.out.println(reservationNumber + " is for " +
           reservations.get(reservationNumber);
}
```

- Keys will contain a set of all the keys in the reservations HashMap
- We can use a forEach loop to iterate through to print out the values
- Most efficient way to access a Map

Maps: looping through the pairings

The entrySet() method returns a Set of all map entries.

```
Map <String, String> reservations = new HashMap<>();

reservations.put("HY234-9234", "Rick");
reservations.put("HY234-4235", "Beth");
reservations.put("HY234-3234", "Jerry");

for (Map.Entry<String, String> reservation: reservations.entrySet())
{
       System.out.println(reservation.getKey() + " is for " +
           reservation.getValue());
}
```

- Reservations.entrySet() will contain a set of all the entries in the reservations HashMap
- We can use a forEach loop to iterate through to print out the values

Maps: Some Additional Rules

Maps are used to store key value pairs.

- Do not use primitive types with Maps, use the Wrapper classes instead.
- Make sure there are no duplicate keys. **If a key value pair is entered with a key that already exists, it will overwrite the existing one!**

Let’s Code!

- KeySet returns a set of keys
- EntrySet returns a set of map entries (key, value pairs)

Sets: Introduction

A set is also a collection of data.

- It differs from other collections we've seen so far in that no duplicate elements are allowed.
- It is also **unordered**.

Sets: Declaring

The following pattern is used in declaring a set.

```
import java.util.HashSet;
import java.util.Set;

public class MyClass {

   public static void main(String args[]) {

       Set<Integer> primeNumbersLessThan10 = new HashSet<>();
   }
}
```

Note the we will need these 2 imports for a hash set.

We are creating a type of Set called a HashSet

We have specified that the set will contain only integers.

Note the "**new**" keyword which **instantiates** the set.

Sets: add method

The add method creates a new element in the set.

```
Set<Integer> primeNumbersLessThan10 = new HashSet<>();
primeNumbersLessThan10.add(2);
primeNumbersLessThan10.add(3);
primeNumbersLessThan10.add(5);
```

Only one parameter is required, the data that is being added.

In this example I have specified that this is a set of Integers, so the integers 2, 3, and 5 are being added.

Arrays vs Lists vs Maps vs Sets

- Use **Arrays** when … you know the maximum number of elements, and you know you will primarily be working with primitive data types.
- Use **Lists** when … you want something that works like an array, but you don't know the maximum number of elements.
- Use **Maps** when … you have key value pairs.
- Use **Sets** when … you know your data does not contain repeating elements.

If you know you will be dealing with non primitive data types like **POJO's** (Plain Old Java Objects) you may want to avoid arrays and instead use lists, maps, or sets.

Algorithmic Complexity

- Many different solutions to solve problem
- Sometimes need an efficient solution
  - Scalability – works well for large data set (what we will focus on)
  - Memory needed
- Correctness has to do with whether method solved problem
- Efficiency has to do with how method is defined
- Measure algorithm speed in terms of number of operations performed relative to input size.
- Want to know the worse possible amount of time it could take
  - Big O notation – Big Omicron – Worst case
  - Discussed in terms of input size of N

Execution-time requirements

- How long will it take to run?

```
public boolean isLastElementEven(int[] array){
        return array[array.length - 1] % 2 == 0;
}
```

- What if array is 1 element long?
- 100? 1000?
- O (1)

Execution-time requirements

- How long will it take to run?

```
public boolean doesArrayContain10(int[] array){
   boolean hasATen = false;
   for(int i = 0; i < array.length; i++) {
      if (array[i] == 10){
         return true;
      }
   }
   return false;
}
```

- O (n)
- Worst case is we search every element and cannot find a 10

Execution-time requirements

- How long will it take to run?

```
public boolean doesArrayContainDuplicates(int[] array){
   boolean hasDuplicate = false;
   for(int i = 0; i < array.length; i++) {
     for (int j = 0; i < array.length; j++){ //nested loop
        if (i == j) {
          continue;
         }
         if (array[i] == array[j]){
           hasDuplicate = true;
         }
   }
   return hasDuplicate;
}
```

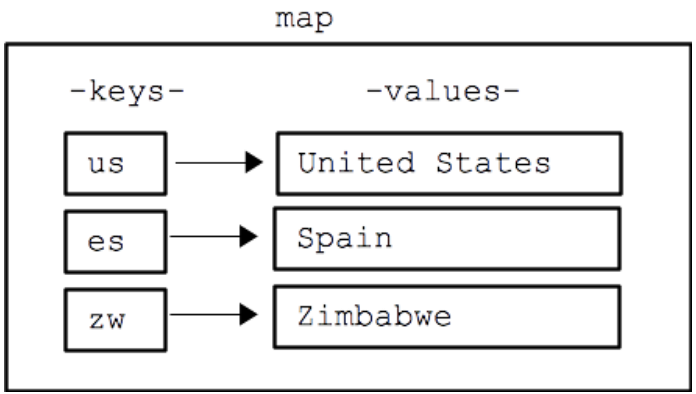
- O (n^2)
- Worst case is we compare each element and there are no duplicates
- Nested loops are always O(n^2)

Real world examples of Complexity

- O(1) - determining if a number is odd or even
- O(log N) - finding a word in the dictionary (using binary search)
- O(N) - reading a book
- O(N log N) - sorting a deck of playing cards (using merge sort)
- O(N^2) - checking if you have everything on your shopping list in your trolley
- O(infinity) - tossing a coin until it lands on heads
-
- O(10^N): trying to break a password by testing every possible combination (assuming numerical password of length N)

Objectives

- Should be able to effectively use objects of the Map collection class

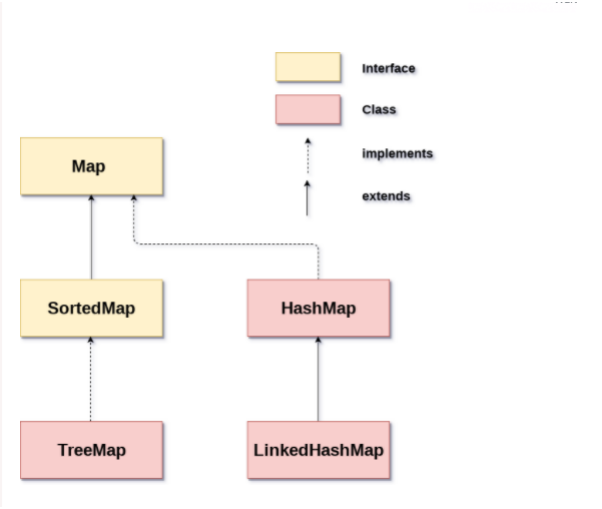



```
java.lang.ClassCastException: com.buam.mapsigns.maps.Renderer
cannot be cast to com.buam.mapsigns.maps.Renderer
        at com.buam.mapsigns.maps.MapSequence.render(MapSequen
ce.java:88) ~[?:?]
```

Java really hates me

Objectives

- Should be able to effectively use objects of the Map collection class
- Should be able to use common Map API operations



Method	Description
V put(Object key, Object value)	It is used to insert an entry in the map.
void putAll(Map map)	It is used to insert the specified map in the map.
V putIfAbsent(K key, V value)	It inserts the specified value with the specified key in the map only if it is not already specified.
V remove(Object key)	It is used to delete an entry for the specified key.
boolean remove(Object key, Object value)	It removes the specified values with the associated specified keys from the map.
Set keySet()	It returns the Set view containing all the keys.
Set<Map.Entry<K,V>> entrySet()	It returns the Set view containing all the keys and values.
void clear()	It is used to reset the map.
V compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	It is used to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
V computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)	It is used to compute its value using the given mapping function, if the specified key is not already associated with a value (or is mapped to null), and enters it into this map unless null.
V computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	It is used to compute a new mapping given the key and its current mapped value if the value for the specified key is present and non-null.
boolean containsValue(Object value)	This method returns true if some value equal to the value exists within the map, else return false.
boolean containsKey(Object key)	This method returns true if some key equal to the key exists within the map, else return false.
boolean equals(Object o)	It is used to compare the specified Object with the Map.
void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for each entry in the map until all entries have been processed or the action throws an exception.

Objectives

- Should be able to effectively use objects of the Map collection class
- Should be able to use common Map API operations
- (talk about optimization of code)

https://www.youtube.com/watch?v=v4cd1O4zkGw