

Introduction to Maps in Java

CPSC 2150

Maps

- Maps are a special type of collection
- Maps store items in Key Value Pairs
- A List, or an array just stores an Object
- A Key Value Pair is a set of two *related* objects.
 - The key object uniquely identifies the value object
- Maps are Generic, so they take in 2 data types as arguments
 - The first data type is for the Key
 - The second data type is for the Value
 - They don't have to be the same data type
 - They can be anything that extends Object
 - Everything that is not a primitive
 - Use our wrapper classes to replace primitives

Key Value Pairs

- Key Value pairs help us relate data together, and retrieve data easily.
- With an array, if we want to get a value in the array, we have to have the index
- Our Keys replace our index, and they can be any Object
- We can not have duplicate Keys in our Map, they must be unique.
- Examples:
 - Map <Student, Double> grades
 - The key is Student, and the value is a Double that holds their grade
 - Map <String, Driver> drivers
 - The key is a String holding the drivers license number, and the value is the Driver with that license number

Declaring a Map

- `Map<Object key, Object value>` is an interface with many implementations
 - `HashMap` is a common implementation
- Need to provide datatypes for the **Key** and **Values**
 - Any non-primitive will work
 - Even another type of collection!
- `Map<Integer, String> myMap = new HashMap<Integer, String>();`
- `Map<Integer, String> myMap = new HashMap<>();`
 - Allowable in Java 8 (or higher)
- Note: In following examples, I will use `K` as the key data type and `V` as the value data type
 - `Map<K, V> myMap;`

Map interface

- `public V put(K key, V value)`
 - Adds the key value pair to our map
 - Returns the value (can be ignored)
 - Key must not already exist in the map!
- `public V get(K key)`
 - Returns the value associated with the key
- `public V remove(K key)`
 - Removes the key value pair associated with key
- `public int size()`
 - Returns the number of key value pairs in the map

Map Interface

- **public** boolean **isEmpty**()
 - Returns true **iff** the map contains no key-value pairs
- **public** boolean **containsKey**(K key)
 - Returns true **iff** the map contains the key `key`
 - Calls `K.equals()`
- **public** boolean **containsValue**(V value)
 - Returns true **iff** the map contains the value `value`
 - Calls `V.equals()`

Map.Entry

- Map contains a sub-interface called `Entry`
- `Entry` is the data type used to hold a Key-Value pair in one object
- Has two methods
 - `public Object getKey()`
 - Returns the key as an `Object`
 - `public Object getValue()`
 - Returns the value as an `Object`
- Since it returns an object, and not `K` and `V`, we need to cast to our data type.
- It's helpful for looping through a `Map`
- `Map.entrySet()` returns the set of all `Map.Entry` objects in the `Map`

Looping through a map

- We can use our `for (val : array)` syntax to loop through a map

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

```
...
```

```
for (Map.Entry<Integer, String> m: myMap.entrySet()) {  
    // m holds our current key value pair  
    System.out.println(m.getKey() + "," + m.getValue());  
}
```

The End