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

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 > or Map <K, V>
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(“HY234-3234")); // 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: Review

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!**

- 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.

Review: Map<T, T>

A **map** is a collection that utilizes Key Value Pairs, allowing ***values*** to be assigned and then located using *user-defined* ***keys***.

- Collection of Keys and a Collection of Values that are organized such that the value can be retrieved using the key.
- Indexed by the key rather than order, allowing for very fast retrieval of a specific value.

Map Keys

1. Can be any reference type
2. Must be unique
3. Cannot be null
4. Stored as a Set

Map Values

1. Can be any reference type
2. Can have duplicates
3. Can be null

Creating a Map<T, T>

Map<T, T> variable = new HashMap<T, T>();

Map - Map Interface

<**T**, T> - the first Type is the Data Type of the Key

<T, **T**> - the second Type is the Data Type of the Value

HashMap - the implementation class to instantiate

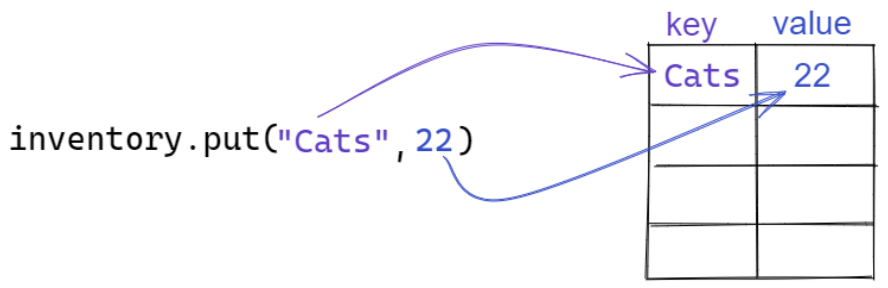
The Data Type of the Key and the Value are not related, and do not need to be the same.

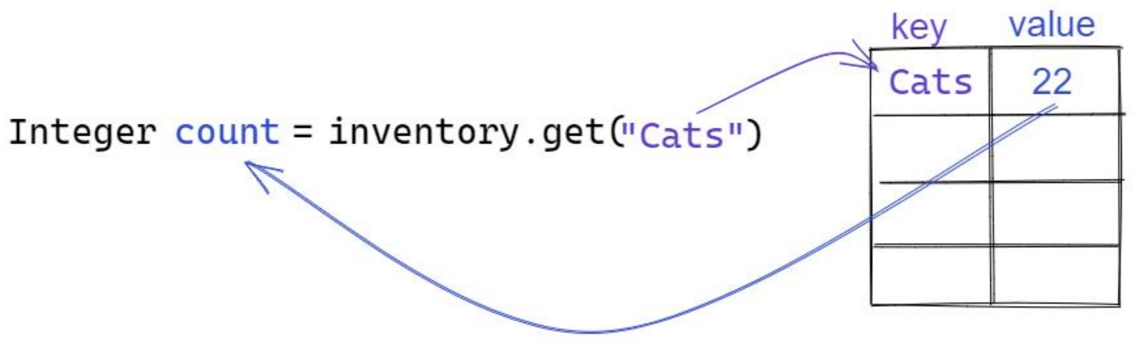
<Integer, Double>
<Integer, House>
<String, String>
<Character, Boolean>

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

Add and Getting Map Elements

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





Map Methods

.put(key, value)	***Adds or Updates*** the value in the Map. If the key does not exist it adds the key and the value.
.get(key)	Returns the value associated with the given key. If the key does not exist null is returned.
.remove(key)	Removes a key/value pair from the map. If the key exists the value is returned, otherwise null is returned
.containsKey(key)	Returns true if the key exists in the map
.containsValue(value)	Returns true if the value exists in the map
.keySet()	Returns all the keys in the map as a Set<T> collection
.entrySet()	Returns all Key/Value pairs as Entry<T, T> objects

Looping over a Map with entrySet()

entrySet() returns the key value pairs in the map as a Set<Entry<T, T><, which can be used in a foreach loop.

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

for ( Entry<String,Integer> nextEntry : inventory.entrySet() ) {

    String key = nextEntry.getKey();
    Integer value = nextEntry.getValue();

    }
```

Map Order

How a map orders data is dependent on the implementation class used.

HashMap does not maintain order.

```
Map<T, T> map = new HashMap<T, T>();
```

What Loop to use

Loop	**Reason**
for	Need an index or count. Need to be able to move through a Collection or Array in an arbitrary manner Used for Collections, Arrays, or to loop a set number of times
forEach	Need to loop from the first item to the last of an Array or Collection Don’t need an index or count Only Used with Collections or Arrays
While	Have a boolean condition that determines when the loop should stop…