CSCI 2120:
Software Design & Development II

UNIT 2: Collections Framework & Generics

Map

Overview

1. Introduction
2. Map Interface
3. Map Hierarchy Diagram
4. Map Implementation Classes
5. Map.Entry Interface
6. Map Constructors
7. Map Methods
8. Map Examples

Introduction

A **map in Java** is a container object that stores elements in the form of key and value pairs. A key is a unique element (object) that serves as an “index” in the map.

The element that is associated with a key is called value. A map stores the values associated with keys. In a map, both keys and values must be objects and cannot be primitive types.

A map cannot have duplicate keys. Each key maps to only one value. This type of mapping is called **one-to-one mapping in java**.

Introduction

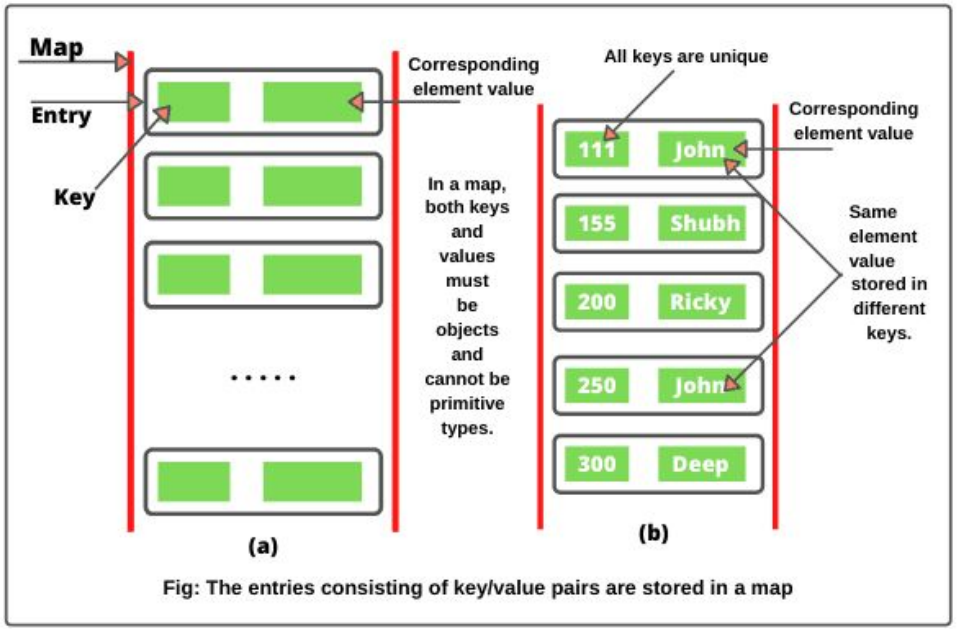


Fig: The entries consisting of key/value pairs are stored in a map

All keys must be unique, but values may be duplicate (i.e. the same value can be stored to several different keys).

A key and its associated value are called an entry that is stored in a map as shown in the figure.

After the entry (key/value pairs) is stored in a map, we can retrieve (get) its value by using its key.

The figure shows a map in which each entry consists of Id number as a key and a name as the value.

Key point: The main difference between maps and sets is that maps contain keys and values, whereas sets contain only keys.

Map interface

Map interface in Java is defined in `java.util.Map` package. It is a part of the Java collections framework but it does not extend the collection interface.

Java map interface is defined like this:

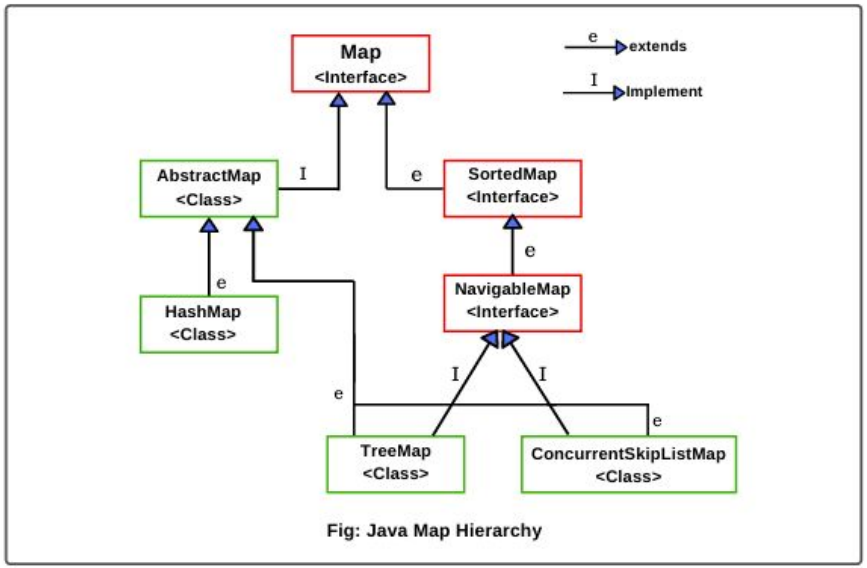
```
public interface Map<K, V> // Map is a generic.
```

In this syntax, K defines the type of keys and V defines the type of values.

For example, a mapping of Integer keys and String values can be represented with a `Map<Integer,String>`. The Map interface provides the methods for fast retrieval, deletion, and updating of the pair through the key.

Map Hierarchy Diagram

The hierarchy diagram of Java Map interface is shown in the below figure.



Let’s understand a brief introduction about interfaces and implementation classes shown in the hierarchy diagram of Java map.

Map Hierarchy Diagram

1. SortedMap Interface: A SortedMap is a subinterface of Map interface that extends Map interface. The entries in the map are maintained in ascending order based on the keys.
SortedMap is generic and is declared like this:

```
interface SortedMap<K, V>
```

The methods defined by SortedMap interface are:
`comparator(), entrySet(), firstKey(), headMap(K toKey), keySet(). lastKey(), etc.`

Map Hierarchy Diagram

2. NavigableMap Interface: NavigableMap interface is a recent addition to the Collections Framework that was added by Java 1.6 version. It extends SortedMap interface.

TreeMap and ConcurrentSkipListMap classes implement the NavigableMap interface. As the name suggest, NavigableMap provides many useful methods that make Map navigation easy.

We can retrieve entries based on the closest match to a given key. NavigableMap is a generic interface that can be declared like this:

```
interface NavigableMap<K, V>
```

The methods defined by NavigableMap interface are:
`ceilingEntry(K key), ceilingKey(K key), descendingKeySet(), descendingMap(), firstEntry(), lastEntry(), etc.`

Map Implementation Classes

The collection class library has several classes that provide implementations of map interface in Java. They are as follows:

1. AbstractMap: It is an abstract class that implements map interface. It is the parent class of all concrete map implementation classes such as HashMap, TreeMap, and LinkedHashMap. It implements all methods in Map interface except entrySet() method.

2. EnumMap: The EnumMap class extends AbstractMap and implements Map interface. It is especially for use with keys of Enum type.

3. HashMap: It is a concrete class that extends AbstractMap. It uses a hash table to store elements. It is mainly used for locating a value, inserting, and deleting an entry.

4. TreeMap: It is a concrete class that extends AbstractMap and implements NavigableMap interface. It used a tree for storing elements. It is used for traversing keys in sorted order. The keys can be sorted using the Comparable interface or Comparator interface.

Map Implementation Classes

5. LinkedHashMap: It is a concrete class that provides implementation of Java map interface. It extends HashMap class with a linked-list implementation that supports the insertion order of entries in the map.

The entries in a HashMap are not ordered, but entries in a LinkedHashMap can be retrieved in the order in which they were inserted into the map.

6. WeakHashMap: The WeakHashMap class extends AbstractMap interface to use a hash table with keys of weak type. Weak keys allow an element in a map to be garbage collected when its key is no longer used anywhere in the program.

7. IdentityHashMap: This class extends AbstractMap and uses reference equality for comparing entries. This class is not used for general purposes.

Map.Entry Interface

The **Map.Entry interface** enables us to work on an entry in the map. An entry of a map is an object of type Map.Entry interface, where Entry is an inner interface of Map interface.

Map.Entry interface is defined in `java.util.Map.Entry` package. Each `Map.Entry` object contains one key/value pair. `Map.Entry` is a generic interface and is declared like this:

```
interface Map.Entry<K, V>
```

In this syntax, K defines the type of keys, and V defines the type of values.

Map.Entry Interface

There are several methods defined by Map.Entry interface in Java. They are as follows:

Method	Description
boolean equals(Object obj)	It is used to check for equality the specified object with the other existing object. It returns true if the specified object obj is a Map.Entry whose key and value are equal to that of the existing object.
K getKey()	It is used to retrieve the key for a map entry. Its return type is key.
V getValue()	It is used to get the value for a map entry. Its return type is value.
int hashCode()	It returns hash code value for a map entry.
void setValue(V value)	This method is used to replace the existing value corresponding to this entry with the specified value and returns the replaced value.

How to create Map Object in Java?

We can create an object of the map using any of its three concrete classes: HashMap, LinkedHashMap, or TreeMap. Here, we will take HashMap class constructor to create a Map in Java.

The general syntax to create a map object is as follows:

```
// It create an empty map.
Map<K, V> map = new HashMap<>();

// It creates a map with initializing elements of m.
Map<K, V> map = new HashMap<>(Map m);

// It creates a map with initialization of initial capacity of HashMap.
Map<K, V> map = new HashMap<>(int initialCapacity);

// It creates a map object with initializing both initial capacity and fill ratio of HashMap.
Map<K, V> map = new HashMap<>(int initialCapacity, float fillRatio);
```

Map Methods in Java

Method	Description
V put(K key, V value)	It is used to add an entry with specified key and value in the map.
void putAll(Map m)	It is used to add all entries from into this map.
V putIfAbsent(K key, V value)	It is used to add specified value with specified key in the map only if it is not already specified.
V remove(Object key)	This method is used to delete an entry for the specified key. It will return null if the key is not in the map.
boolean remove(Object key, Object value)	This method is used to remove the specified value associated with specified key from the map.
Set<K> keySet()	This method returns a set consisting of the keys in the invoking map. It provides a set-view of the keys.

Map Methods in Java

Method	Description
void clear()	This method is used to remove all entries from the map.
V get(Object key)	This method returns the value for the specified key in this map.
int hashCode()	It returns the hash code value for the invoking map.
boolean isEmpty()	This method is used to check whether the map contains any entries. It returns true if the invoking map is empty, otherwise returns false if it contains at least one key.
int size()	The size() method returns the number of entries (number of key/value pairs) in the map.
V replace(K key, V value)	This method is used to replace the specified value for a specified key.

Map Methods in Java

Method	Description
boolean replace(K key, V oldValue, V newValue)	This method is used to replace old value with new value for a specified key.
Collection<V> values()	This method returns a collection view of the values in the map.
boolean containsKey(Object key)	This method is used to check whether the map contains an entry for the specified key. It returns true if the invoking map contains an entry for the specified key. Otherwise, returns false.
boolean containsValue(Object value)	This method is used to check whether the map contains an entry for the specified value. It returns true if the map contains specified value. Otherwise, returns false.
boolean equals(Object obj)	This method is used to compare the specified Object with map. If obj is a map and contains the same entries, it returns true otherwise, false.

Map Examples

Let’s take some example programs where we will perform some useful operations like adding, removing, replacing entry in a map based on the above methods.

Example 1: Adding entries

1. Adding entries: Let’s create a program where we will add entries in map using put() method. We will also check a map is empty or not before adding entries in a map. Look at the source code to understand better.

Example 1: Adding entries

```
import java.util.HashMap;
import java.util.Map;
public class MapTester1 {
   public static void main(String[] args) {
       // Create a map of generic type.
       Map<Integer, String> map = new HashMap<>();

       // Checking map is empty or not.
       boolean isEmpty = map.isEmpty();
       System.out.println("Is Map is empty? " +isEmpty);

       // Adding entries in the map. Call put() method to add entries in map.
       map.put(101, "Red");
       map.put(103, "Green");
       map.put(102, "Yellow");
       map.put(104, "Blue");
       map.put(106, "Pink");

       System.out.println("Entries in Map: " +map);
       int size = map.size();
       System.out.println("No. of entries in Map: " +size);

       // Create another map.
       Map<Integer,String> map2 = new HashMap<>();
       map2.put(115, "Brown");
       map2.put(120, "Purple");
       map2.put(125, "Green");
       map.putAll(map2);
       System.out.println("Entries in updated Map: " +map);
   }
}
```

Example 1: Adding entries

Output:

```
Is Map is empty? true
Entries in Map: {101=Red, 102=Yellow, 103=Green, 104=Blue, 106=Pink}
No. of entries in Map: 5
Entries in updated Map: {115=Brown, 101=Red, 102=Yellow, 103=Green, 104=Blue, 120=Purple, 106=Pink, 125=Green}
```

Example 2: Removing entries

2. Removing entries: Let’s create a program where we will remove entry from a map using remove() method.

Example 2: Removing entries

```
import java.util.HashMap;
import java.util.Map;
public class MapTester2 {
   public static void main(String[] args) {
       // Create a map of generic type.
       Map<Integer, String> map = new HashMap<>();

       // Adding entries in the map.
       map.put(101, "Red");
       map.put(103, "Green");
       map.put(102, "Yellow");
       map.put(104, "Blue");
       map.put(106, "Pink");

       // Removing an entry for the specified key.
       map.remove(104);
       System.out.println("Entries in Map after removing an entry: " +map);

       // Removing specified value associated with the specified key from the map.
       map.remove(106,"Pink");
       System.out.println("Entries in Map after removing Pink: " +map);
       map.clear();
       System.out.println(map);
   }
}
```

Example 2: Removing entries

Output:

```
Entries in Map after removing an entry: {101=Red, 102=Yellow, 103=Green, 106=Pink}
Entries in Map after removing Pink: {101=Red, 102=Yellow, 103=Green}
{}
```

Example 3: Replacing values

3. Replacing value: Let’s make a program where we will replace a specified value for a specified key. Look at the source code.

Example 3: Replacing values

```
import java.util.HashMap;
import java.util.Map;
public class MapTester3 {
   public static void main(String[] args) {
       Map<String, String> map = new HashMap<>();

       map.put("A", "Apple");
       map.put("B", "Boy");
       map.put("C", "Cat");
       map.put("D", "Dog");
       map.put("E", "Elephant");

       System.out.println("Original Entries in Map: " +map);
       // Replacing a specified value for a specified key.
       map.replace("E", "Element");
       System.out.println("Updated Entries in Map after replacing: " +map);

       // Replace old value with a new value.
       map.replace("B", "Boy", "Bucket");
       System.out.println(map);
   }
}
```

Example 3: Replacing values

Output:

```
Orignal Entries in Map: {A=Apple, B=Boy, C=Cat, D=Dog, E=Elephant}
Updated Entries in Map after replacing: {A=Apple, B=Boy, C=Cat, D=Dog, E=Element}
{A=Apple, B=Bucket, C=Cat, D=Dog, E=Element}
```

Example 4: Getting keys and values

4. Getting keys and values: Let’s take an example program where we will get a set-view of keys and values of the invoking map.

Example 4: Getting keys and values

```
import java.util.Collection;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
public class MapTester4 {
   public static void main(String[] args) {
       Map<String, String> map = new HashMap<>();

       map.put("B", "B");
       map.put("D", "H");
       map.put("A", "A");
       map.put("C", "R");
       map.put("E", "T");

       Set<String> keys = map.keySet();
       System.out.println("Set view of keys: " +keys);

       Collection<String> values = map.values();
       System.out.println("Collection view of values: " +values);
   }
}
```

Example 4: Getting keys and values

Output:

```
Set view of keys: [A, B, C, D, E]
Collection view of values: [A, B, R, H, T]
```

Example 5: Getting value for key

5. Getting value for key: Let’s create a program where we will get a value for the specified key in the map. We will also check whether the map contains an entry for the specified key.

Example 5: Getting value for key

```
import java.util.HashMap;
import java.util.Map;
public class MapTester5 {
   public static void main(String[] args) {
       Map<String, String> map = new HashMap<>();

       map.put("V", "Violet");
       map.put("I", "Indigo");
       map.put("B", "Blue");
       map.put("G", "Green");
       map.put("Y", "Yellow");
       map.put("O", "Orange");
       map.put("R", "Red");

       String value = map.get("V"); // It will return a value for specified key in this map.
       System.out.println(value);

       boolean entryKey = map.containsKey("B");
       System.out.println(entryKey);

       boolean entryValue = map.containsValue("Brown");
       System.out.println(entryValue);
   }
}
```

Example 5: Getting value for key

Output:

```
Violet
true
false
```

END