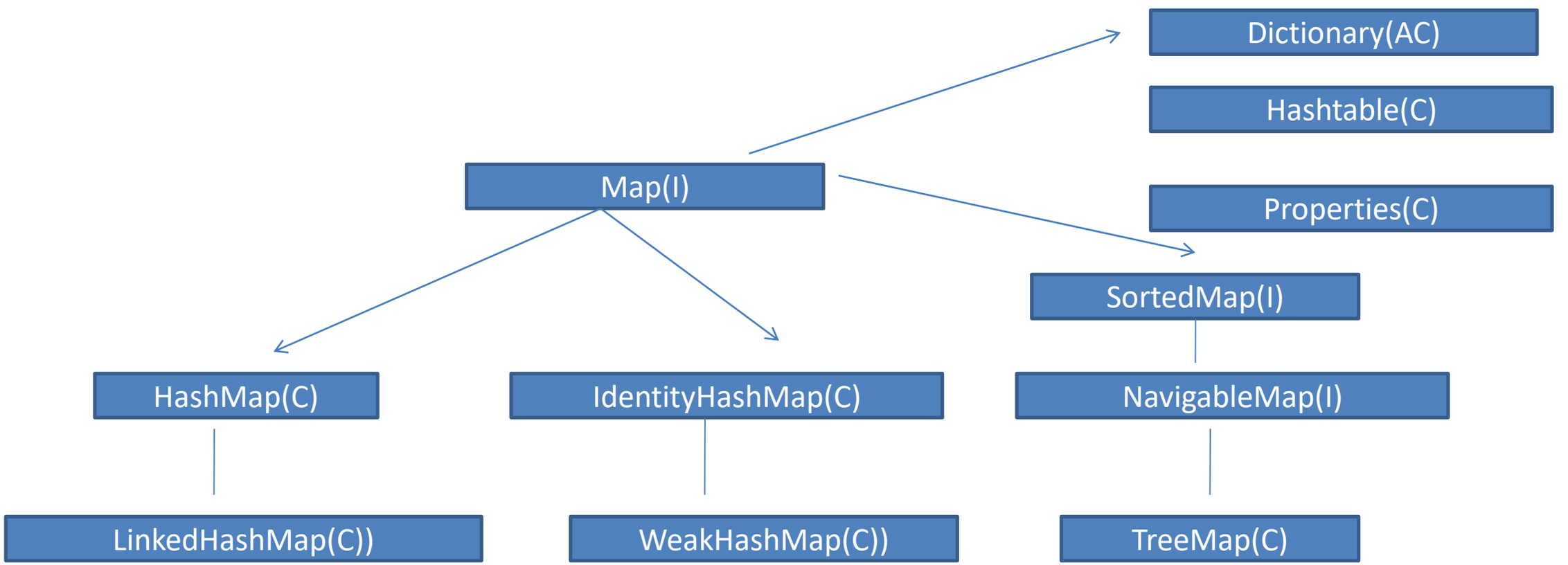
Map:

1. Map is **Not Child Interface** of **Collection**. Hence we cant apply **collections interface methods here.**
2. If we want to represent a group of objects as **key-value pairs** then we should go for **Map Interface.**
3. Duplicate **keys** are not allowed but **values** can be **duplicated**.
4. Each **key-value** pair is known as **one entity.**



HashMap	LinkedHashMap	TreeMap
Duplicate Values Allowed	Duplicate Values Allowed	Duplicate Values Allowed
Null Key Only Once	Null Key Only Once	Null Key not allowed
Null Value Many	Null Value Many	Null Value Many
Duplicate key Overrides the Values	Duplicate key Overrides the Values	Duplicate key Overrides the Values
Sort based on Keys	Ordered Elements	Sort based on Keys
DS HashTable	DS HashTable and LinkedList	Red Black Tree
1.2V	1.4V	1.2V
Non Synchronized	Non Synchronized	Non Synchronized

values()
values() returns a collection view of the **values** contained in the map.

keyset()
returns the set of **keys** contained in the map.

```
//Sort based on keys
Map<Integer, String> hashMap = new HashMap<Integer, String>();
hashMap.put(5, "NameFive");
hashMap.put(3, "NameThree");
hashMap.put(1, "NameOne");
hashMap.put(4, "NameFour");
hashMap.put(2, "NameTwo");
hashMap.put(2, "NameTwoDuplicate"); //Duplicate key Overrides the Values
hashMap.put(8, null); // null allowed multiple times
hashMap.put(7, null); // // null allowed multiple times
hashMap.put(null, "null");
hashMap.put(null, "null"); //Duplicate null key and null value
System.out.println(hashMap); // {null=null, 1=NameOne, 2=NameTwoDuplicate, 3=NameThree, 4=NameFour,
5=NameFive, 6=NameFive, 7=null, 8=null}
```

```java
//size() remove() clear()
Map<Integer, String> hm = new HashMap<Integer, String>();
hm.put(1, "NameOne");
hm.put(2, "NameTwo");
hm.put(3, "NameThree");
hm.put(4, "NameFour");
hm.put(5, null);
// returns the number of entries in the map.
System.out.println(hm.size()); // 5

System.out.println(hm); // {1=NameOne, 2=NameTwo, 3=NameThree, 4=NameFour, 5=null}
hm.remove(3);
hm.remove(2, "Adv Java"); // {1=NameOne, 2=NameTwo, 4=NameFour, 5=null}
System.out.println(hm);
hm.remove(4, "NameFour");
System.out.println(hm); // {1=NameOne, 2=NameTwo, 5=null}
hm.clear();
System.out.println(hm); // {}
```

```java
// values() and keySet():
HashMap<Integer, String> hm = new HashMap<Integer, String>();
hm.put(1, "Core Java");
hm.put(2, "Adv Java");
hm.put(3, "Hibernate");
hm.put(5, "Rest");
hm.put(4, "Spring");

// values(): Returns a Collection view of the values contained in this map.
Collection<String> values = hm.values();
System.out.println("Set of Values: " + values); //Set of Values: [Core Java, Adv Java, Hibernate, Spring, Rest]

// keySet(): Returns a Set view of the keys contained in this map.
Set<Integer> keySet = hm.keySet();
System.out.println("Set of Keys: " + keySet); // Set of Keys: [1, 2, 3, 4, 5]
```

```
//Ordered Elements, Duplicate Values Allowed, Duplicate Keys Overrides the Data
LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>();
map.put(2, "NameTwo");
map.put(2, "NameTwoDuplicate");//Duplicate Keys overrides the Data
map.put(3, "NameThree");
map.put(1, "NameOne");
map.put(4, "NameFour");
map.put(5, "NameFour"); //Duplicate Value Repeated
System.out.println(map); // {2=NameTwoDuplicate, 3=NameThree, 1=NameOne, 4=NameFour, 5=NameFour}
```

```
// Null Key only once and Null Value multiple times
LinkedHashMap<Integer, String> map = new LinkedHashMap<Integer, String>();
map.put(2, "NameTwo");
map.put(3, "NameThree");
map.put(null, "NameOne");
map.put(null, "NameFour"); // Data Overrides
map.put(1, null);
map.put(4, null);
System.out.println(map); // {2=NameTwo, 3=NameThree, null=NameFour, 1=null, 4=null}
```

```
//It will order all the elements in the form of keys
Map<Integer, String> hm = new TreeMap<Integer, String>();
hm.put(3, "NameThree");
hm.put(2, "NameTwo");
hm.put(1, "NameOne");
hm.put(1, "NameOneDuplicate"); // Duplicate Keys Overrides
hm.put(4, "NameFour");
hm.put(5, "NameFive");
hm.put(6, "NameSix"); // Values can be Duplicated
hm.put(7, "NameSix"); // Values can be Duplicated
hm.put(8, null); // Null allowed at values side
hm.put(9, null); // Null allowed at values side
//hm.put(null, "Python"); // java.lang.NullPointerException
System.out.println(hm);
//{1=NameOneDuplicate, 2=NameTwo, 3=NameThree, 4=NameFour, 5=NameFive, 6=NameSix, 7=NameSix,
8=null, 9=null}
```

HashMap and IdentityHashMap:

The difference b/w HashMap and IdentityHashMap is HashMap uses equals() method and IdentityHashMap use == operator to compare key and value

HashMap uses hashCode() were IndentityHashMap uses System.identityHashCode()

We use IdentityHashMap when comparision based on key and values

HashMap and WeakHashMap

HashMap is not allowing for garbage collector once we add elements
WeakHashMap allows garbage collector to destroy elements

```
String s1 = new String("a");
String s2 = new String("a");
HashMap
HashMap<String, String> hashMap = new HashMap<String, String>();
hashMap.put(s1, "A");
hashMap.put(s2, "B"); //Duplicate Key Overrides the Value
System.out.println(hashMap); // {a=B}
System.out.println(s1.equals(s2)); // true
System.out.println(s1.hashCode()); //97
System.out.println(s2.hashCode()); //97

IdentityHashMap
Map<String, String> identityHashMap = new IdentityHashMap<String, String>();
identityHashMap.put(s1, "A");
identityHashMap.put(s2, "B"); //Duplicate Key Value will not Overrides
System.out.println(identityHashMap); // {a=A, a=B}
System.out.println(s1 == s2); // false
System.out.println(System.identityHashCode(s1)); // 212628335
System.out.println(System.identityHashCode(s2)); // 1579572132
```

```
//HashMap
HashMap<String, Integer> hashMap = new HashMap<String, Integer>();
String s1 = new String("a");
hashMap.put(s1, 1);
System.out.println(hashMap); // {a=1}

s1 = null;
System.gc();
System.out.println(hashMap); // {a=1}

//WeakHashMap
WeakHashMap<String, Integer> weakHashMap = new WeakHashMap<>();
String s2 = new String("a");
weakHashMap.put(s2, 1);
System.out.println(weakHashMap); // {a=1}

s2 = null;
System.gc();
System.out.println(weakHashMap); // {}
}
```