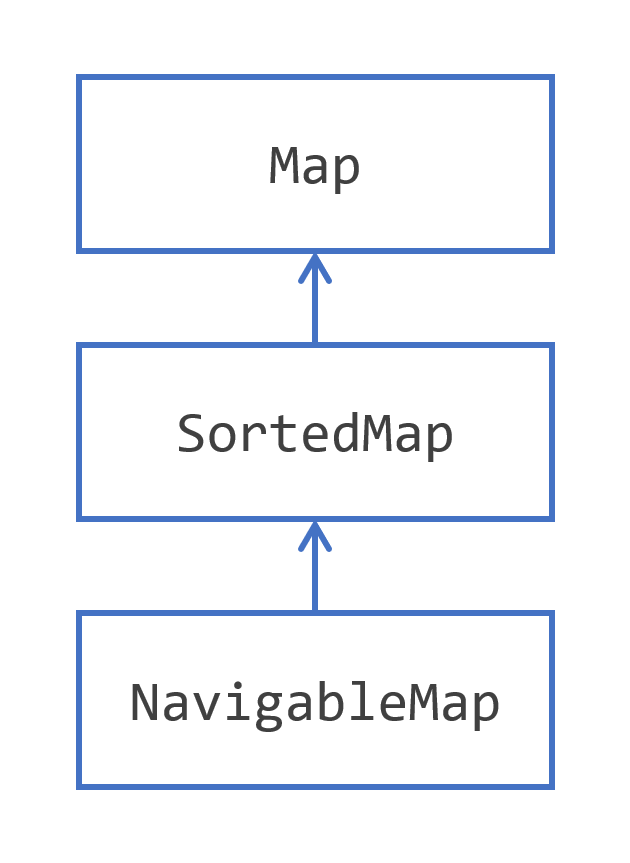
**Introduction to Map Hierarchy**

1. **Introduction to Map Hierarchy**
   * The Collections Framework provides an implementation of the hashmap structure, a fundamental data structure for storing key-value pairs.
2. **How Hashmaps Work**
   * Hashmaps store key-value pairs where:
     + The key represents a value uniquely.
     + The value can be any object, while the key is usually a simple object (string, number, etc.).
     + Keys allow retrieval of their corresponding values.
     + Keys are unique, but values don’t have to be.
     + A key-value pair forms an entry in the hashmap.
3. **Map Interface and Extensions**
   * The **Map interface** implements the key-value pair concept.
   * **SortedMap**: Maintains key-value pairs in sorted order.
   * **NavigableMap**: Extends SortedMap with additional navigation methods.
4. **Map Implementations in JDK**
   * **HashMap**: The most widely used implementation.
   * **LinkedHashMap**: Maintains insertion order of key-value pairs.
   * **IdentityHashMap**: Compares keys by reference (==) instead of equals() and hashCode().
5. **Multimaps**
   * A single key can be associated with multiple values (not directly supported in Java Collections).
   * This behavior can be achieved using maps with lists as values.



## **Using the Convenience Factory Methods for Collections to Create Maps**

1. **Java SE 9 Introduced Convenience Factory Methods for Maps**
   * Immutable maps can be created using Map.of() and Map.ofEntries().
2. **Creating Immutable Maps**
   * **For up to 10 key-value pairs** → Use Map.of():

Map<Integer, String> map = Map.of(1, "one", 2, "two", 3, "three");

* + **For more than 10 key-value pairs** → Use Map.ofEntries():

Map<Integer, String> map = Map.ofEntries(

Map.entry(1, "one"),

Map.entry(2, "two"),

Map.entry(3, "three")

);

1. **Restrictions on These Factory Methods**
   * **Maps and entries are immutable** → No modifications allowed after creation.
   * **Null keys and values are not allowed** → Throws NullPointerException if attempted.
   * **Duplicate keys are not permitted** → Results in an IllegalArgumentException.
2. **Potential Interview Questions**
   * How do you create an immutable map in Java?
   * What are the differences between Map.of() and Map.ofEntries()?
   * Why are Map.of() and Map.ofEntries() considered more efficient than traditional HashMap initialization?
   * What happens if you try to insert null in Map.of()?

**Storing Key/Value Pairs in a Map**

### ****Key Points for Interview Preparation****

1. **Rules for Key-Value Relationships**
   * A **key** can be bound to only **one** value.
   * A **value** can be associated with **multiple keys**.
2. **Consequences for Map Structure**
   * **Keys are unique** → Forms a **Set** structure.
   * **Key-value pairs are unique** → Also follows a **Set** structure.
   * **Values can have duplicates** → Behaves like a **Collection**.
3. **Common Map Operations**
   * **Adding a key-value pair** (put(K, V)) → Fails if the key already exists.
   * **Retrieving a value** (get(K)) using its key.
   * **Removing a key-value pair** (remove(K)).
4. **Set-Like Operations on Maps**
   * **Check if empty** (isEmpty()).
   * **Get size** (size()).
   * **Merge another map** (putAll(Map<K, V>)).
   * **Clear all entries** (clear()).

### ****Potential Interview Questions****

### ****Short Answers:****

1️⃣ **How does the uniqueness of keys affect map storage?**

* Keys must be unique; duplicate keys overwrite previous values.
* Keys are stored using hashing (HashMap) or sorting (TreeMap).

2️⃣ **What happens if you try to insert a duplicate key in a Map?**

* The new value replaces the existing one for that key.
* No exception is thrown.

map.put("A", 10);

map.put("A", 20); // Overwrites 10

System.out.println(map.get("A")); // Output: 20

3️⃣ **How can you retrieve all unique keys and values from a Map?**

* **Keys:** map.keySet() (always unique).
* **Values:** new HashSet<>(map.values()) (removes duplicates).
* **Entries:** map.entrySet() (unique key-value pairs).

4️⃣ **Explain the difference between containsKey(K) and containsValue(V).**

* containsKey(K): Checks if a key exists in the map (efficient in HashMap).
* containsValue(V): Checks if a value exists (slower, requires full scan).

5️⃣ **What is the difference between HashMap, LinkedHashMap, and TreeMap regarding ordering?**

* **HashMap**: No ordering, fast lookups.
* **LinkedHashMap**: Maintains insertion order.
* **TreeMap**: Stores keys in **sorted order** (natural or custom comparator).

**Exploring the Map interface**

### ****Key Points for Interview Preparation****

1. **The Map Interface**
   * Base type that models key-value storage in Java.
   * Includes **Map.Entry<K, V>** to represent key-value pairs.
2. **Choosing Key Types Carefully**
   * **Using mutable keys is discouraged** → Mutating a key changes its hash code, making the entry **unrecoverable**.
   * Example of **a bad key choice**:

class MutableKey {

int id;

MutableKey(int id) { this.id = id; }

public void setId(int newId) { this.id = newId; }

@Override public int hashCode() { return id; }

@Override public boolean equals(Object obj) { return (obj instanceof MutableKey) && this.id == ((MutableKey) obj).id; }

}

Map<MutableKey, String> map = new HashMap<>();

MutableKey key = new MutableKey(1);

map.put(key, "Value1");

key.setId(2); // Key mutated!

System.out.println(map.get(key)); // null! Entry is lost due to hash change.

1. **Map.Entry<K, V> Interface**
   * **Methods in Map.Entry**:
     + getKey() → Returns the key.
     + getValue() → Retrieves the value.
     + setValue(value) → Updates the value.
2. **Entries Are Views on the Map**
   * Modifying an entry’s value updates the original map.
   * **Keys cannot be changed** via Map.Entry to prevent corruption.
   * Example:

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

map.put(1, "One");

for (Map.Entry<Integer, String> entry : map.entrySet()) {

System.out.println(entry.getKey() + " -> " +entry.getValue());

entry.setValue("Updated One"); // Changes reflect in the map

}

System.out.println(map.get(1)); // Output: "Updated One"

### ****Potential Interview Questions****

### ****Short Answers:****

1️⃣ **Why should you avoid using mutable objects as map keys?**

* Mutable objects can change after being added to the map, potentially altering their hash code.
* This can break the map's ability to retrieve the correct value because the key's identity might change, leading to unpredictable behavior.

2️⃣ **How does modifying a Map.Entry affect the original map?**

* Modifying the value of a Map.Entry directly **modifies the original map** because Map.Entry is a **view** of the map.
* Changes to the value (using setValue()) are reflected in the map.

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

map.put("A", 1);

Map.Entry<String, Integer> entry = map.entrySet().iterator().next();

entry.setValue(2);

System.out.println(map.get("A")); // Output: 2

3️⃣ **Can you update a key inside Map.Entry? Why or why not?**

* **No**, you cannot update a key in Map.Entry because the key is immutable once it's added to the map.
* Allowing key modification could corrupt the map, as it may change the hash code or map structure.

4️⃣ **Explain the difference between HashMap, TreeMap, and LinkedHashMap in key ordering.**

* **HashMap**: **No specific order** of keys (unordered).
* **TreeMap**: Keys are stored in **sorted order** (natural or based on a custom comparator).
* **LinkedHashMap**: Keys are stored in **insertion order** (preserves the order in which entries were added).