Lecture 13 In-Class Example

Collections Framework: HashMap - Gaming Leaderboard System

📄 Files

- `Lecture13.java`
  - Shows how HashMap solves the leaderboard score lookup problem

---

🎯 Topic Overview

Building a gaming leaderboard system that demonstrates HashMap concepts:

1. **HashMap for Player Scores** - Tracking player names mapped to their scores
2. **Fast Score Lookup** - Instantly find a player's score by their name
3. **Dynamic Updates** - Update scores when players complete new achievements
4. **Leaderboard Management** - Add/remove players and display rankings

This example shows why HashMap is perfect for lookup-based systems where we need:

- Fast retrieval by name (player name → score)
- Dynamic updates as scores change
- Efficient tracking of relationships between keys and values

Real-world examples using similar patterns:

- Gaming: Player name → Score (this example!)
- Music: Song title → Play count
- E-commerce: Product SKU → Current price
- Social Media: Username → Follower count
- Banking: Account number → Balance
- Weather: City → Temperature

📚 Learning Goals

HashMap Basics

- ✅ Create a HashMap using generic syntax: `HashMap<KeyType, ValueType> name = new HashMap<>();`
- ✅ Understand HashMap stores key-value pairs with automatic key uniqueness
- ✅ Recognize that HashMap uses hashing internally for fast lookups
- ✅ Know that HashMap provides O(1) average lookup time by key

HashMap Methods - Adding & Updating

- ✅ Use `.put(K key, V value)` to add new entries

- ✅ Understand that `.put()` with existing key updates the value
- ✅ Know that `.put()` returns the old value (or null if new key)
- ✅ Recognize this automatic update behavior (no need to check first)

HashMap Methods - Retrieval & Checking

- ✅ Use `.get(K key)` to retrieve value by key
- ✅ Understand that `.get()` returns null if key doesn't exist
- ✅ Use `.containsKey(K key)` to safely check before getting
- ✅ Use `.containsValue(V value)` to search for values

HashMap Methods - Removal & Status

- ✅ Use `.remove(K key)` to delete entries
- ✅ Understand `.remove()` returns the removed value
- ✅ Use `.size()` to get number of entries
- ✅ Use `.isEmpty()` to check if map is empty
- ✅ Use `.clear()` to remove all entries

HashMap Iteration Techniques

- ✅ Iterate using enhanced for-each with `.entrySet()` (for both key and value)
- ✅ Iterate using enhanced for-each with `.keySet()` (for keys only)
- ✅ Iterate using enhanced for-each with `.values()` (for values only)
- ✅ Understand order of iteration is unpredictable (different from ArrayList)

HashMap Key Concepts

- ✅ Keys must be unique (adding duplicate key overwrites value)
- ✅ Values can be duplicated (multiple players can have same score)
- ✅ HashMap ordering is unpredictable (use LinkedHashMap for insertion order)
- ✅ HashMap is not thread-safe (use ConcurrentHashMap for multi-threading)

Real-World Application

- ✅ Build a leaderboard system with multiple operations
- ✅ Perform lookups, updates, and statistics on player scores
- ✅ Understand when HashMap is the right choice over ArrayList
- ✅ Recognize patterns of key-value relationships in real applications

---

🔑 Key Concepts Demonstrated

HashMap Concepts

Concept	Example in Code	Why It Matters
Key-Value Pairs	`playerScores.put("Alice", 1500)`	Maps player name to their score

Concept	Example in Code	Why It Matters
Unique Keys	Adding same key twice overwrites	Prevents duplicate player entries
Duplicate Values	Multiple players can have 1500 points	Scores can repeat, names cannot
Fast Lookup	`.get("Alice")` instantly finds 1500	O(1) average - no loop needed
Dynamic Updates	`.put("Alice", 2000)` updates score	Easy to modify existing values
Safe Checking	`.containsKey()` before `.get()`	Prevents null pointer issues
Create HashMap	`HashMap<String, Integer> map = new HashMap<>();`	Syntax for instantiation

💡 Real-World Application

This example simulates a video game leaderboard where:

HashMap for Player Scores:

- **Fast Lookup** - Instantly check any player's current score
- **Dynamic Updates** - Update scores as players play new levels
- **Add/Remove Players** - Add new players or remove inactive ones
- **Statistics** - Find highest/lowest scores, calculate averages
- **Key Uniqueness** - Each player appears only once (no duplicate entries)

Why HashMap?

- **Alternative 1 - ArrayList**: Would require looping through all players to find "Alice" (slow!)
- **Alternative 2 - Array**: Fixed size, hard to grow when adding new players
- **HashMap wins**: Instant lookup by player name, dynamic sizing, efficient updates

Real-world leaderboards:

- Xbox Live: Gamertag → Achievement Points
- Steam: Player ID → Total Playtime Hours
- Fortnite: Account → Win Count
- Elo Rating Systems: Player → Rating Score
- Tournament Systems: Team → Match Record

⚡ Why HashMap for Leaderboards?

Why We Use HashMap for This Problem

Problem	Solution	Why HashMap

Problem	Solution	Why HashMap
"What's Alice's score?"	O(1) lookup by name	Instant retrieval without looping
"Update Alice's score"	`.put()` overwrites automatically	No manual checking needed
"Remove inactive players"	`.remove()` by key	Easy deletion by player name
"Does player exist?"	`.containsKey()` check	Fast membership test
"Show all players"	`.entrySet()` iteration	Access all key-value pairs

Key Insight

- **HashMap = Fast lookup dictionary** - Like looking up a person's phone number in a contact list
- **Not ArrayList** = Would need to search through entire list every time
- **Not Array** = Fixed size, wastes space, hard to grow
- **HashMap** = Perfect for key-value lookups where order doesn't matter

---

📊 HashMap vs ArrayList

Feature	HashMap	ArrayList
Lookup by Value	O(1) average by key	O(n) - must loop through
Lookup Syntax	`map.get(playerName)`	Must loop: `for (Player p : list)`
Updates	`.put()` instantly updates	Must find first, then update
Order	Unpredictable iteration	Maintains insertion order
Duplicate Keys	Prevented (overwrites)	Multiple entries allowed
Best For	Key-value relationships	Ordered sequences
Use When	Need fast lookup by key	Need ordered list access by index
Example Use	Player → Score	List of all players in order

🔍 HashMap vs Other Structures

Feature	HashMap	HashSet	TreeMap
Stores	Key-value pairs	Unique values only	Key-value (sorted)
Lookup	O(1) by key	O(1) by value	O(log n) by key
Ordering	Unpredictable	Unpredictable	Sorted by key
Duplicates	Keys unique, values can repeat	All values unique	Keys unique
Best For	Fast lookups	Unique membership	Sorted access
Leaderboard	✓ Perfect (player → score)	✗ Can't map names	✓ But sorted only

---

📋 Common Patterns

Creating HashMap

```
// Generic syntax: HashMap<KeyType, ValueType>
HashMap<String, Integer> playerScores = new HashMap<>();
```

Adding Entries

```
// .put(key, value) - adds new or updates existing
playerScores.put("Alice", 1500);
playerScores.put("Bob", 2000);

// Returns old value, or null if new key
Integer oldScore = playerScores.put("Alice", 1800);  // Returns 1500
```

Retrieving Values

```
// .get(key) - returns value or null
Integer aliceScore = playerScores.get("Alice");  // Returns 1800
Integer unknownScore = playerScores.get("Charlie");  // Returns null

// Safe retrieval with check
if (playerScores.containsKey("Alice")) {
    Integer score = playerScores.get("Alice");
    System.out.println(score);
}
```

Checking Existence

```
// .containsKey(key) - true/false for key
if (playerScores.containsKey("Alice")) { ... }

// .containsValue(value) - true/false for value
if (playerScores.containsValue(1500)) { ... }
```

Removing Entries

```
// .remove(key) - removes and returns value
Integer removedScore = playerScores.remove("Alice");  // Returns 1800

// Check if removed
if (removedScore != null) {
```

```
    System.out.println("Alice had score: " + removedScore);
}
```

Iterating - Entry (Key & Value)

```
// Use .entrySet() to get both key and value
for (Map.Entry<String, Integer> entry : playerScores.entrySet()) {
    String playerName = entry.getKey();
    Integer score = entry.getValue();
    System.out.println(playerName + ": " + score);
}
```

Iterating - Keys Only

```
// Use .keySet() to iterate over just keys
for (String playerName : playerScores.keySet()) {
    System.out.println(playerName);
}
```

Iterating - Values Only

```
// Use .values() to iterate over just values
for (Integer score : playerScores.values()) {
    System.out.println("Score: " + score);
}
```

Size and Empty Checks

```
int totalPlayers = playerScores.size();
if (playerScores.isEmpty()) {
    System.out.println("No players!");
}
```

❌ Common Mistakes

❌ Wrong	✅ Fix
`HashMap<String> map = new HashMap<>();`	Need two types: `<String, Integer>`

❌ Wrong	✅ Fix
`map.get("Alice")` without null check	Check `.containsKey()` first or check `!= null`
`.put()` returns index	`.put()` returns old value, not index
Expecting HashMap to maintain order	Use LinkedHashMap if order matters
`map.add()` - wrong method	HashMap uses `.put()`, not `.add()`
Modifying HashMap while iterating	Collect keys first: `List<String> keys = new ArrayList<>(map.keySet())`
`for (String entry : map)`	Use `.entrySet()`, `.keySet()`, or `.values()`
`.get()` on non-existent key crashes	`.get()` returns null - check before using value
Using ArrayList when need fast lookup	HashMap O(1) vs ArrayList O(n)

📌 Key Takeaways

1. **HashMap = Dictionary** - Like a phone book where name → phone number
2. **Keys are Unique** - Each key can only appear once (duplicates update)
3. **Values can Repeat** - Multiple players can have same score
4. **Fast Lookup** - O(1) average time to find by key
5. **Dynamic Size** - Automatically grows when adding entries
6. **Unordered** - Iteration order is unpredictable (use LinkedHashMap for ordered)
7. **Always Check** - Use `.containsKey()` before `.get()` to avoid null