

JAVA / LESSON 26 OF 40

Scoreboards and HashMap

Check scores by name, update scoreboard safely

This chapter uses JDK 21 and does not require the preview function. | 90-110 minutes

Make player scoreboard ScoreBoard

There is a new player in the server, and there is no need to add another variable to the scoreboard. Use names as keys to read out scores, and then use clear rules to handle missing points, deductions, sorting, and integer upper limits.

Prerequisites: variables, methods, loops, ArrayList and generics. You don't have to create a player100Score when you let the 100th player join today.

Before you start: It is recommended to complete Lesson 25 first

Read the full lesson and copy the code:

<https://teacher-ma.de/minecraft-java-lesson.php?lesson=26&lang=en>

Long code lines may wrap at the > margin marker. The marker is not part of the code.

Player name is connected to own score

Element	Explanation
key	Player name, unique search identifier
value	Score, repeatable
entry	A key-value mapping

In this lesson

01. Before starting: predict first, then let the program answer
02. keys and values form a lookup table
03. Create HashMap and put data
04. Safety cumulative score
05. Non-existing keys must be handled explicitly
06. entrySet gets the key and value at once
07. Delete players and understand the order
08. Method parameters are oriented to Map contract
09. Ranking list is copied first and then sorted
10. Three types of tests keep scoreboard
11. Challenge Blueprint Player Scoreboard
12. Scoreboard Program
13. Expansion tasks and clearance review
14. Own acceptance record

01 / 14

Before starting: predict first, then let the program answer

This chapter uses JDK 21 and does not require the preview function. The story is borrowed from the block world, and the example runs on the ordinary Java console; if the values and rules are course models, they will be clearly stated.

Each common example is a complete independent file, saved according to the file name marked on the box. Place each example in a different folder to avoid duplicate names of auxiliary classes. When the complete project is divided into sections, they are connected sequentially in the same file.

Terminal command: replace the Chinese placeholder with the actual file name and class name

```
javac -encoding UTF-8 ExampleFile.java
java ExampleClass
```

- Prediction: Write the expected output first, or draw out the changes in references and data.
- Verification: Check line by line after running, and then only change one condition.
- Explanation: Explain whether the results come from Java rules, course contracts, or input data.

"Intentional error" is used to recognize bugs and cannot be mixed with ordinary examples. Use a special temporary directory for file experiments; do not use real archives to test coverage and deletion.

02 / 14

keys and values form a lookup table

Map is like a server scoreboard. The player name is the key and the score is the value; the program quickly finds the corresponding score through the name.

Only one copy of the current value is retained for the same key. Two different players can have exactly the same score.

Player name is connected to own score

Element	Explanation
key	Player name, unique search identifier
value	Score, repeatable
entry	A key-value mapping

First ask who is responsible for searching, and then what to save. Here the player name is used to check the score, so the name is used as the key and the score is used as the value.

Acceptance comparison table

Concept	Scoreboard Example	Rules
key key	Player name Alex	Unique in the same Map
value value	Score 12	Duplication allowed
Mapping entry	Alex corresponds to 12	A set of keys and values

03 / 14

Create HashMap and put data

Write the key and value types when declaring. String, Integer in angle brackets means that only string keys and integer values can be used.

put adds or replaces get key query

Element	Explanation
Map<String,Integer>	Clear key value type
put	The key with the same name replaces the old value
size	Current record number

Import java.util.HashMap; and import java.util.Map; are required. Write Map for variables and HashMap for object creation.

L26Example02.java · Complete independent example

```
import java.util.*;
public class L26Example02 {
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>();
        scores.put("Alex", 12); scores.put("Steve", 12);
        scores.put("Alex", 20);
        System.out.println(scores.get("Alex"));
        System.out.println(scores.size());
    }
}
```

Output

```
20
2
```

Fault experiment: The value type is Integer, and the string "12" is not the integer 12. Errors are intentionally left below, please anticipate them first and then try to fix them.

BadMapType.java · Intentional error example

```
import java.util.*;
public class BadMapType {
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>();
        scores.put("Alex", "12");
    }
}
```

04 / 14

Safety cumulative score

A put with the same name will replace the old value without increasing the number of records. The get of the new key returns null, and direct unboxing and addition will fail. Let's look at simple counting first, and then establish clear contracts for large numbers, null and illegal old records.

Both getOrDefault and merge can handle new players

Element	Explanation
does not exist	getOrDefault gives default 0
null already exists	The default value does not replace it
Safe addition	Convert large numbers to long first and check

The decimal value examples on this page do not contain null; getOrDefault only handles missing keys. Existing nulls, negative scores or cumulative overflows shall be handled according to the contract in Section 7.

L26Example03.java · Complete independent example

```
import java.util.*;
public class L26Example03 {
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>();
        scores.put("Alex", scores.getOrDefault("Alex", 0) + 5);
        scores.merge("Alex", 3, Integer::sum);
        System.out.println(scores.get("Alex"));
    }
}
```

Output

8

This example ensures that there are no null records and the addition of small integers does not overflow. getOrDefault only gives a default value for "key does not exist" and does not replace existing null. The accumulation of large numbers requires long verification in Section 7, and addition cannot be copied directly.

05 / 14

Non-existing keys must be handled explicitly

get returns null for a non-existent key. The scoreboard can choose to display 0, or it can explicitly tell the user that the player cannot be found.

containsKey distinguishes between no players and a score of 0

Element	Explanation
get	Return null for missing key
containsKey	Determine whether the key exists
0 point player	Presence and absence are different

Whether the query method returns 0, null or prompt text should be clearly stated in the method description. The caller should not have to guess.

L26Example04.java · Complete independent example

```
import java.util.*;
public class L26Example04 {
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>();
        scores.put("Alex", 0);
        System.out.println(scores.get("Mia"));
        System.out.println(scores.getOrDefault("Mia", 0));
        System.out.println(scores.containsKey("Alex"));
        System.out.println(scores.containsKey("Mia"));
    }
}
```

Output

```
null
0
true
false
```

06 / 14

entrySet gets the key and value at once

Just traversing keySet and then calling get can also work, but entrySet directly provides complete records, which makes the expression clearer.

Traverse each scoring record

Element	Explanation
entrySet	Get the key and value at the same time
HashMap	No commitment to traversal order
TreeMap copy	Key generation stable display

There is no commitment to the traversal order of HashMap. When testing content, don't mistake the addition order for a fixed output order.

L26Example05.java · Complete independent example

```
import java.util.*;
public class L26Example05 {
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>(Map.of("Steve", 8, "Alex", 12));
        for (var entry : new TreeMap<>(scores).entrySet())
            System.out.println(entry.getKey() + "=" + entry.getValue());
    }
}
```

Output

```
Alex=12
Steve=8
```

07 / 14

Delete players and understand the order

`remove(key)` deletes the entire record and returns the old value; returns null when the key does not exist.
`clear()` will clear the entire table.

remove returns the old value HashMap without queuing

Element	Explanation
<code>remove(key)</code>	Delete and return the old value
<code>clear()</code>	Clear all records
Display order	Need to clearly choose to implement

You need to traverse in order of joining to understand LinkedHashMap; you need to sort by key to understand TreeMap. The challenge for this lesson is not to rely on HashMap order.

L26Example06.java · Complete independent example

```
import java.util.*;
public class L26Example06 {
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>(Map.of("Alex", 12));
        System.out.println(scores.remove("Alex"));
        System.out.println(scores.remove("Alex"));
        scores.clear();
        System.out.println(scores.size());
    }
}
```

Output

```
12
null
0
```

08 / 14

Method parameters are oriented to Map contract

If the method only requires the general operation of Map, the parameters should be written as Map<K,V> instead of being locked as HashMap.

The caller can choose different Map implementations

Element	Explanation
Name check	Reject empty names, remove leading and trailing blanks
long calculation	Prevent int addition from overflowing
Last updated	Data remains unchanged when rejected

Rules for empty Maps, empty names, and negative deductions must be determined in advance. The example rejects empty input and limits the minimum score to 0.

L26Example07.java · Complete independent example

```
import java.util.*;
public class L26Example07 {
    static boolean addScore(Map<String, Integer> scores,
                           String player, int delta) {
        if (scores == null || player == null || player.isBlank()) return false;
        String key = player.strip();
        Integer old = scores.get(key);
        if (scores.containsKey(key) && (old == null || old < 0)) return false;
        int current = old == null ? 0 : old;
        long next = (long) current + delta;
        if (next > Integer.MAX_VALUE) return false;
        scores.put(key, (int) Math.max(0L, next));
        return true;
    }
    static int getScore(Map<String, Integer> scores, String player) {
        if (scores == null || player == null || player.isBlank()) return 0;
        String key = player.strip();
        Integer value = scores.get(key);
        if (!scores.containsKey(key)) return 0;
        if (value == null || value < 0)
            throw new IllegalStateException("Scoring record corrupted: " + key);
        return value;
    }
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>();
        System.out.println(addScore(scores, " Alex ", 5));
        System.out.println(addScore(scores, "Alex", -20));
        System.out.println(getScore(scores, "Alex"));
        scores.put("Alex", Integer.MAX_VALUE);
        System.out.println(addScore(scores, "Alex", 1));
        System.out.println(getScore(scores, "Alex"));
    }
}
```

Output

```
true
true
0
false
2147483647
```

First upgrade the old score to long and then add delta. If it exceeds the upper limit of int, it will be rejected and not written. Clamping below 0 to 0 is the scoring rule; incoming corrupted records that already contain null or negative values are explicitly rejected.

09 / 14

Ranking list is copied first and then sorted

HashMap is suitable for checking scores by name, but is not suitable for expressing the first and second places. When making a ranking list, copy the records to the list and then sort them.

Map is responsible for searching and List is responsible for orderly display

Element	Explanation
Map.entry	Copy unmodifiable value entries
In descending order of scores	Determine ranking
Same score by name	Output is repeatable

Before sorting, use Map.entry to create an independent unmodifiable entry, and the name and score are immutable. Identicals are sorted by name; only the reference to the entrySet is copied, not the numerical snapshot.

L26Example08.java · Complete independent example

```
import java.util.*;
public class L26Example08 {
    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>(Map.of("Steve", 8, "Alex", 8));
        List<Map.Entry<String, Integer>> ranking = new ArrayList<>();
        scores.forEach((name, score) -> ranking.add(Map.entry(name, score)));
        ranking.sort(Map.Entry.<String, Integer>comparingByValue().reversed()
            .thenComparing(Map.Entry.comparingByKey()));
        scores.put("Alex", 99);
        for (var entry : ranking)
            System.out.println(entry.getKey() + "=" + entry.getValue());
    }
}
```

Output

```
Alex=8
Steve=8
```

new ArrayList<>(scores.entrySet()) only copies the entry reference, not an independent numerical snapshot. This example uses Map.entry to create unmodifiable entries, and String and Integer are both immutable; the same entries are sorted by name, and the output can be repeated. Provided the scoreboard does not contain nulls.

10 / 14

Three types of tests keep scoreboard

When testing the extra points method, don't just look at one line of output. Also check if the failed operation secretly changed the Map.

Normal boundary and wrong input must leave evidence

Element	Explanation
Normal bonus points	Old value increased
Minimum 0 points	Point deduction clamping
Failed Check	Equal to snapshot before operation

Often the most important evidence of a failed operation is that the data remains intact. Take a snapshot first, then compare content.

Acceptance comparison table

Scene	Operation	Expected
Normal	Alex plus 5	Score increased by 5
Border	Players with 0 points will be deducted 1	Still 0
Error	Add 5 to empty name	Return false and the data remains unchanged
Error	null Map	Return false without throwing an exception
Query	No player exists	Return 0

L26Example09.java · Complete independent example

```
import java.util.*;

public class L26Example09 {
    static boolean addScore(Map<String, Integer> scores,
                           String player, int delta) {
        if (scores == null || player == null || player.isBlank()) return false;
        String key = player.strip();
        Integer old = scores.get(key);
        if (scores.containsKey(key) && (old == null || old < 0)) return false;
        int current = old == null ? 0 : old;
        long next = (long) current + delta;
        if (next > Integer.MAX_VALUE) return false;
        scores.put(key, (int) Math.max(0L, next));
        return true;
    }

    static int getScore(Map<String, Integer> scores, String player) {
        if (scores == null || player == null || player.isBlank()) return 0;
        String key = player.strip();
        Integer value = scores.get(key);
        if (!scores.containsKey(key)) return 0;
        if (value == null || value < 0)
            throw new IllegalStateException("Scoring record corrupted: " + key);
        return value;
    }

    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>(Map.of("Alex", 12));
        Map<String, Integer> before = new HashMap<>(scores);
        boolean ok = addScore(scores, " ", 5);
        System.out.println(!ok && scores.equals(before));
        System.out.println(getScore(scores, "Mia"));
    }
}
```

Output

```
true
0
```

11 / 14

Challenge Blueprint Player Scoreboard

The challenge program is called ScoreBoard. It should support addition and subtraction of points for new players and old players, safe query and deletion, while not relying on the traversal order of HashMap.

Add deduction, query, delete to form a complete process

Element	Explanation
Add player	Start from 0 when it does not exist
Query missing	This lesson returns 0
Delete player	Mapping completely removed

Challenge Query the results directly by name without using the traversal order of the entire Map as the basis for acceptance.

Acceptance sequence

- Alex adds 5 from 12, and the query results in 17.
- Steve deducted 20 from 8, and the lowest score was clamped to 0.
- Querying missing Mia returns 0, but containsKey is identifiable as missing.
- An empty name operation returns false and the scoreboard remains unchanged.
- Deleting Steve returns the old score of 0, leaving only 1 player.

12 / 14

Scoreboard Program

Create a new ScoreBoard.java, compile and run using Java 21.

Map contract encapsulates secure scoring operations

Element	Explanation
Alex	Add 12 to 5 to get 17
Steve	8 buckle 20 to get 0
Delete Steve	Last 1 player left

Deduct 20 points from Alex continuously and confirm that it will not be lower than 0; then replace Map with TreeMap, and the method does not need to be modified.

ScoreBoard.java · Complete standalone example

```
import java.util.*;

public class ScoreBoard {
    static boolean addScore(Map<String, Integer> scores,
                           String player, int delta) {
        if (scores == null || player == null || player.isBlank()) return false;
        String key = player.strip();
        Integer old = scores.get(key);
        if (scores.containsKey(key) && (old == null || old < 0)) return false;
        int current = old == null ? 0 : old;
        long next = (long) current + delta;
        if (next > Integer.MAX_VALUE) return false;
        scores.put(key, (int) Math.max(0L, next));
        return true;
    }

    static int getScore(Map<String, Integer> scores, String player) {
        if (scores == null || player == null || player.isBlank()) return 0;
        String key = player.strip();
        Integer value = scores.get(key);
        if (!scores.containsKey(key)) return 0;
        if (value == null || value < 0)
            throw new IllegalStateException("Scoring record corrupted: " + key);
        return value;
    }

    public static void main(String[] args) {
        Map<String, Integer> scores = new HashMap<>();
        addScore(scores, "Alex", 12); addScore(scores, "Steve", 8);
        addScore(scores, "Alex", 5); addScore(scores, "Steve", -20);
        System.out.println("Alex=" + getScore(scores, "Alex"));
        System.out.println("Steve=" + getScore(scores, "Steve"));
        System.out.println("Mia=" + getScore(scores, "Mia"));
        System.out.println("Empty name=" + addScore(scores, " ", 5));
        System.out.println("Delete=" + scores.remove("Steve"));
        System.out.println("Number of players=" + scores.size());
    }
}
```

Output

```
Alex=17
Steve=0
Mia=0
Empty name=false
Delete=0
Number of players=1
```

13 / 14

Expansion tasks and clearance review

The job submits code, test table and a fault record at the same time. Explain why a certain field was chosen as a key.

Three-level task expansion server statistics

Element	Explanation
Select key	Explain the search purpose
Definition failed	Missing, empty name, overflow
Verification sequence	Cannot rely on HashMap order

Lesson 27 uses Set to record non-repeating achievements, and then uses Random and probability intervals to design a testable loot table.

Third gear operation

- Wooden pickaxe: Create an item price list and support new additions, price changes and inquiries.
- Iron pickaxe: Count the number of occurrences of a group of dropped objects.
- Diamond Pickaxe: Copy the scoring records to the list and output the top three with the highest scores.

Acceptance comparison table

Self-evaluation	Can	Still need to practice
I can design keys and values	☐	☐
I can safely query for missing keys	☐	☐
I don't rely on HashMap order	☐	☐

14 / 14

Own acceptance record

Beautiful output is the starting point, and being able to explain boundaries is considered mastery. Select one case each from normal, borderline and error, write down your expectations before running, and leave evidence after running.

Leave a basis for judgment

Category	The operation you selected	Check key points
Normal	A common input or a successful action	Is the result consistent line by line?
Border	Range endpoints and adjacent values	Is there one more step or one less thing?
Error	Illegal quantity, missing object or bad data	Whether it is explicitly rejected and whether the status is maintained

- Bug records: triggering conditions, actual phenomena, causes, and minimum repairs.
- Illustrated record: describe the starting point, end point and meaning of each arrow.
- Adaptation record: only change one rule, and modify expectations and tests at the same time.

Technical check: Java SE 21 language specification and standard library. See the language specification for class, interface, type and expression rules; see the corresponding API for the behavior of collections, Random, Files and Stream.

<https://docs.oracle.com/javase/specs/jls/se21/html/index.html>

<https://docs.oracle.com/en/java/javase/21/docs/api/index.html>