Perfect. Continuing in **the SAME PDF-style format, depth, and simplicity**.

📘 JavaScript Map & Set — Internal & Practical View

(Why they exist • How they work • Memory • Traps)

🔹 1. Why `Map` and `Set` were introduced

Problem with Objects & Arrays

- Objects:
  - Keys are **strings/symbols only**
  - Key order is unreliable for logic
- Arrays:
  - Searching is slow (`O(n)`)
  - Duplicates allowed

👉 `Map` and `Set` solve these problems.

🔎 Internal Behavior

- `Map` and `Set` are **specialized data structures**
- Optimized for:
  - fast lookup
  - predictable behavior
- Stored in **heap memory**

✅ Key Takeaway

`Map` and `Set` exist for **performance and correctness**.

⚠️ Interview Trap

They are NOT replacements for objects/arrays—use case matters.

🔹 2. What is a `Map`?

Definition

A **Map** stores **key–value pairs**, where:

- Keys can be **any type**
- Order is **preserved**
- Lookup is fast

Syntax

```js
```

```
const map = new Map();
```

Example

```js
const map = new Map();

map.set("name", "Anoop");
map.set(1, "one");
map.set(true, "yes");
```

🔎 Internal Behavior

- Keys are stored **by reference**
- No string coercion like objects
- Uses hashing internally

```js
map.get(1);    // "one"
map.get("1");  // undefined
```

✅ Key Takeaway

Map keys are **type-safe**.

⚠️ Interview Trap

`map["key"]` does NOT work.

🔹 3. Object vs Map (VERY IMPORTANT)

Feature	Object	Map
Key type	string/symbol	any type
Order	Not guaranteed	Preserved
Size	Manual	`.size`
Performance	Slower for frequent ops	Optimized

```js
map.size; // number of entries
```

🔹 4. Map Methods (Core)

```js
map.set(key, value);
map.get(key);
map.has(key);
map.delete(key);
map.clear();
```

🔎 **Internal Behavior**

- `set` → inserts or updates
- `get` → reference-based lookup
- `has` → constant-time check

🔹 5. Iterating over a Map

```js
for (let [key, value] of map) {
  console.log(key, value);
}
```

```js
map.forEach((value, key) => {
  console.log(key, value);
});
```

🔎 **Internal Behavior**

- Iteration follows **insertion order**
- Map is **iterable by default**

✅ **Key Takeaway**

Maps are designed for iteration.

🔹 6. Object Keys in Map (POWER FEATURE)

```js
const objKey = { id: 1 };

const map = new Map();
map.set(objKey, "data");
map.get(objKey); // "data"
```

🔎 Internal Behavior

- Object reference used as key
- No serialization or string conversion

⚠️ Interview Trap

```js
map.get({ id: 1 }); // undefined
```

Different object → different reference.

🔹 7. What is a `Set`?

Definition

A **Set** stores **unique values only**.

- No duplicates
- Values can be any type
- Order preserved

Syntax

```js
const set = new Set();
```

Example

```js
const set = new Set();

set.add(1);
set.add(1);
set.add(2);

set.size; // 2
```

🔎 Internal Behavior

- Uses SameValueZero comparison
- `NaN` is considered equal to `NaN`

```js
new Set([NaN, NaN]).size; // 1
```

✅ **Key Takeaway**

Set automatically removes duplicates.

⚠️ **Interview Trap**

Objects inside Set are compared by reference.

🔹 8. Set vs Array

Feature	**Array**	**Set**
Duplicates	Allowed	❌ Not allowed
Search	O(n)	O(1)
Order	Preserved	Preserved

```js
set.has(2); // fast
```

🔹 9. Removing Duplicates Using Set (Common Pattern)

```js
const arr = [1, 2, 2, 3];

const unique = [...new Set(arr)];
```

🔎 **Internal Behavior**

- Array → Set removes duplicates
- Spread converts Set back to Array

✅ **Key Takeaway**

Set is best for uniqueness.

🔹 10. WeakMap & WeakSet (Conceptual)

WeakMap

- Keys must be **objects**
- Weakly referenced
- Garbage collectible

WeakSet

- Stores objects only
- Weak references

🔎 Internal Behavior

- No iteration
- Used for memory-sensitive cases

⚠️ Interview Trap

WeakMap / WeakSet are NOT iterable.

🧠 FINAL SUMMARY (EXAM + INTERVIEW)

- `Map` → key–value store with any key type
- `Set` → unique values only
- Both preserve insertion order
- Faster than objects/arrays for lookups
- Keys/values are reference-based
- WeakMap/WeakSet help avoid memory leaks

⏭️ NEXT (Same Format)

Reply with **one word**:

- `spread-rest`
- `array-vs-object`
- `object-methods`
- `storage`
- `weakmap-weakset-deep`

I'll continue in **exact same simplified PDF-style**.