Keys Are Unique, and put Overwrites

There is no 'add a second entry under the same key.' Putting a key that already exists silently replaces its value -- and getOrDefault is the clean way to handle a key that might be missing.

Automation Foundations · ashishautomationlabs.store

There is no "add a second entry under the same key." Putting a key that already exists replaces its value:

```
Map<String, Integer> slaHours = new HashMap<>();
slaHours.put("P1", 4);
slaHours.put("P2", 24);
slaHours.put("P1", 8);
System.out.println("P1 SLA : " + slaHours.get("P1"));
System.out.println("Size   : " + slaHours.size());
int p3 = slaHours.getOrDefault("P3", 72);
System.out.println("P3 SLA : " + p3);
```

```
P1 SLA : 8
Size   : 2
P3 SLA : 72
```

Three `put`s, size two. The second `put("P1", ...)` silently overwrote the first, and no warning was given. That behavior is correct and useful -- it is how you update a value -- but it catches people who expect a Map to accumulate. If you truly want several values under one key, the value itself should be a list.

`getOrDefault(key, fallback)` is the clean way to handle a missing key: it returns the value when present, and your fallback when not. No `null`, no crash.

> *Given `Map<String,String> m = new HashMap<>(); m.put("a","1"); m.put("b","2");*

m.put("a","3"); `-- what does` m.size() `return? What does` m.get("a")` return?

> *What does* `m.get("z")` *return?*

>

> *Answer --* `2`*, because keys are unique and the third* `put` *replaced the first.* `"3"`*, the replacement value.* `null` *-- a missing key is not an error, which is why* `getOrDefault` *or* `containsKey` *matters.*

When do you reach for a Map instead of a list of objects? When the job is lookup by a known key. A list of `TestResult` objects is right for reporting every run in order. A `Map<String, TestResult>` is right when you keep asking "what happened to LoginTest?" If you find yourself looping through a list to find one item by name, you wanted a Map.

One lookup at a time works. The next question is what happens when you need every pair -- and that's where a Map's biggest trap for testers shows up.