Generics: Type Safety Before Run Time

List<String> rejects a wrong-typed value at compile time. The raw List it replaced accepts anything and fails as a ClassCastException at 3 a.m. instead.

Automation Foundations · ashishautomationlabs.store

Your suite takes ninety minutes. The team switches on parallel execution to bring it under twenty, and the trouble starts that night. Tests fail that have not changed in months. A different set fails each run. Run them one at a time and every one passes. The team calls them flaky and adds retries, and the retries hide the problem for a while. Nothing is flaky. The suite has a real defect that only appears when more than one test runs at a time, and its cause is a Core Java idea rather than a Selenium one. This chapter is that idea, plus the one class that fixes it.

It also finishes a piece of unfinished business. You have written angle brackets since Chapter 5 -- `List<String>`, `Map<String, Integer>`, `Optional<String>` -- and been told what they do without being told what they are. That is generics, and it comes first.

`List<String>` means "a list that holds Strings." The angle brackets carry a **type parameter**, and the compiler uses it to check your work. Watch what happens without it. Java still allows the old form, called a **raw type**:

```
List<String> typed = new ArrayList<>();
typed.add("LoginTest");
String name = typed.get(0);
List raw = new ArrayList();
raw.add("CartTest");
raw.add(42);
String bad = (String) raw.get(1);

typed  : LoginTest (no cast needed)
raw    : [CartTest, 42]
raw    : ClassCastException at RUN time
```

The typed list gave back a String directly -- no cast, and no way to put a number into it in the first place. The raw list accepted a String and a number happily, and the failure arrived at run time as a `ClassCastException`. That is the whole argument for generics: the compiler catches the mistake instead of your test run. A suite that fails at compile time costs you a minute. One that fails at 3 a.m. in a pipeline costs you a morning.

The compiler is not silent about the raw version either. Compiling that code produces two unchecked warnings, which are Java saying it can no longer guarantee what goes into that list. Warnings of that kind are worth reading rather than ignoring.

Generics protect you when you use a `List` or a `Map`. The same mechanism is available for methods and classes you write yourself -- which is where the angle brackets stop being someone else's syntax and become yours.