A Variable Is a Labeled Box

Declaration, initialization, and assignment are three different moves. Java refuses to let you read a local variable you never set -- the same policy a good QA lead has for untrusted test data.

Automation Foundations · ashishautomationlabs.store

Chapter 1 used variables with a promise to explain them. Time to keep it.

Start with the working picture: **a variable is a labeled storage box.** The line

```
int testCasesPlanned = 480;
```

does three things at once, and each has a name worth learning, because compiler errors use these names:

- **Declaration** -- `int testCasesPlanned` creates the box and fixes two facts forever:

its type (`int`: whole numbers) and its name (`testCasesPlanned`).

- **Initialization** -- `= 480` puts the first value in.
- **Assignment (later)** -- `testCasesPlanned = 495;` replaces the contents. The old value

is not saved anywhere; a variable is a box, not a logbook. The type is not written again: the box already exists.

The type is a contract

Try to put the text `"lots"` into `testCasesPlanned`, and the compiler raises a defect against you on the spot -- _incompatible types_ -- before the program ever runs.

Notice how important _before it runs_ is. A whole category of bugs that troubles some other languages (wrong kind of data arriving at runtime, in production, at 2 a.m.) is caught in Java at compile time, at your desk. That is **static typing**, and it is one of the reasons companies chose Java for automation: in a suite of thousands of tests, you want the most defects caught by the cheapest reviewer -- and no reviewer is cheaper than the compiler.

Uninitialized means untrusted

Predict this one:

```
public class Uninitialized {
    public static void main(String[] args) {
        int retryCount;
```

```
        System.out.println(retryCount);
    }
}
```

A reasonable guess: it prints `0`, or some random junk. Actual:

```
Uninitialized.java:4: error: variable retryCount might not have been initialized
```

It does not even compile. Java refuses to let you read a local variable that was never given a value. A value you never set is a value you cannot trust. Java would rather stop the build than let you test with garbage data. You have met QA leads with this exact policy.

Two rules about names

Convention: variable names use `camelCase` -- first word small, each next word capitalized: `passRate`, `defectId`, `maxRetryCount`. The compiler does not care; every Java human does, and code that breaks this rule looks unprofessional in review.

Craft: name the meaning, not the type. `int d = 3;` compiles fine and tells the next reader nothing. `int maxLoginAttempts = 3;` is documentation that never becomes outdated.

You have rejected test cases with steps like "click the button" for the same reason -- which button? Unclear instructions are defects, and names are instructions to future readers, including future you.

You can declare a box, put a value in it, replace that value, and trust the compiler to reject the wrong kind of contents. The next lesson is the short catalog: which types you will actually use every day in automation, and four questions that choose for you.