Interview Questions: Objects and Classes

Six questions on classes, constructors, encapsulation, this, equals/hashCode, and toString -- the weak answer everyone gives, and the strong one that names the mechanism and cites a real bug you've seen.

Automation Foundations · ashishautomationlabs.store

Q9 · ●●● -- What is the difference between a class and an object?

Weak answer: "A class is a blueprint and an object is an instance of it."

Strong answer: "A class is the definition -- it says what fields and methods something has. An object is an actual thing built from that definition at run time, with its own copy of the fields. The blueprint comparison is right, and the part worth adding is that you can build many objects from one class and each holds its own state. A `LoginPage` class exists once in the code; a hundred test runs create a hundred `LoginPage` objects, each holding its own driver reference."

Follow-up: *"So where do the fields live for each of those objects?"*

Where in this book: Chapter 7.

Q10 · ●●● -- What is a constructor, and why does it have no return type?

Weak answer: "It is called when you create an object and it does not return anything."

Strong answer: "A constructor runs at `new` and its job is to leave the object in a valid, fully initialized state. It has no return type because it is not a normal method that hands something back -- the `new` expression itself produces the object. It has the same name as the class so Java knows what it is. The practical value in a framework is that it makes required data mandatory: if a `TestResult` constructor demands a name and a status, no code can create one without them."

Follow-up: *"What happens if you do not write a constructor at all?"*

Where in this book: Chapter 7.

Q11 · ●●● -- What is encapsulation, and can you give an example from a framework?

Weak answer: "It means making fields private and using getters and setters."

Strong answer: "It means an object controls its own state instead of exposing it. Fields are private, and the outside world goes through methods that can validate, log, or change

implementation later. The clearest automation example is a Page Object: locators are private, and the class exposes behavior like `login(user, password)` rather than the element itself. The test says what it wants done, not how to find things -- so when the page's HTML changes, one class changes and no tests do."

Follow-up: *"Is a class with a public getter and setter for every field really encapsulated?"*

Where in this book: Chapter 7, with the framework shape in Chapter 13.

Q12 · ●●■ -- What does `this` mean?

Weak answer: "It refers to the current object."

Strong answer: "`this` is a reference to the object the method is running on. It matters most in constructors, where the parameter and the field often share a name -- `this.name = name` says the field gets the parameter. Leave `this` off and you assign the parameter to itself, the field stays `null`, and nothing complains. That is a silent bug I have seen: the object builds fine and prints `null` later, far from the cause."

Follow-up: *"What is the output if you forget it?"*

Where in this book: Chapter 7.

Q13 · ●●● -- What are `equals()` and `hashCode()`, and why do they go together?

Weak answer: "`equals` compares objects and `hashCode` returns a number. You should override both."

Strong answer: "`equals` defines what makes two objects of my class the same -- usually a business identity like a defect id. `hashCode` returns a number used by hash-based collections to decide where to store an object. They must agree, because a `HashSet` or `HashMap` first uses `hashCode` to pick a bucket and only calls `equals` within that bucket. Override `equals` alone and two equal objects can land in different buckets, so `equals` is never called between them and the set stores both. I have run exactly that: same class, same `equals`, and the set size is 1 with `hashCode` and 2 without."

Follow-up: *"Write both for a class with two fields."*

Where in this book: Chapter 7 for writing them, Chapter 10 for the proof.

Q14 · ●●■ -- What is `toString()` and why override it?

Weak answer: "It converts an object to a String."

Strong answer: "Every class inherits `toString` from `Object`, and the default prints the class name and a hexadecimal number -- useless in a report. Overriding it means that whenever the

object is printed or concatenated, you get something readable. For a tester that is directly practical: a failure message showing `TestResult[LoginTest, FAILED, 240ms]` can be triaged, while `TestResult@6d06d69c` cannot."

Follow-up: *"Which class does* `toString` *come from if you never wrote one?"*

Where in this book: Chapter 7, with the inheritance explanation in Chapter 11.