SRP

SINGLE RESPONSIBILITY PRINCIPLE

- Every class should have a single responsibility.
- Another way to view this is that **a class should only have one reason to change**.
- But who causes the change? An actor.

  Actor: a user of the program or a stakeholder, or a group of such people.

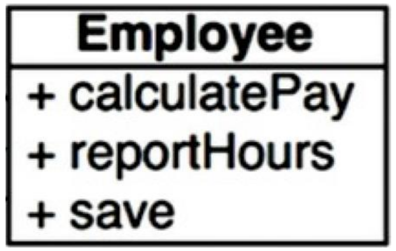
SINGLE RESPONSIBILITY PRINCIPLE

SRP

“This principle is about people. … When you write a software module, you want to make sure that when changes are requested, **those changes can only originate from a single person, or rather, a single tightly coupled group of people representing a single narrowly defined business function**. You want to **isolate your modules from the complexities of the organization as a whole**, and design your systems such that **each module is responsible (responds to) the needs of just that one business function**.” [Uncle Bob, The Single Responsibility Principle]

A STORY OF THREE ACTORS

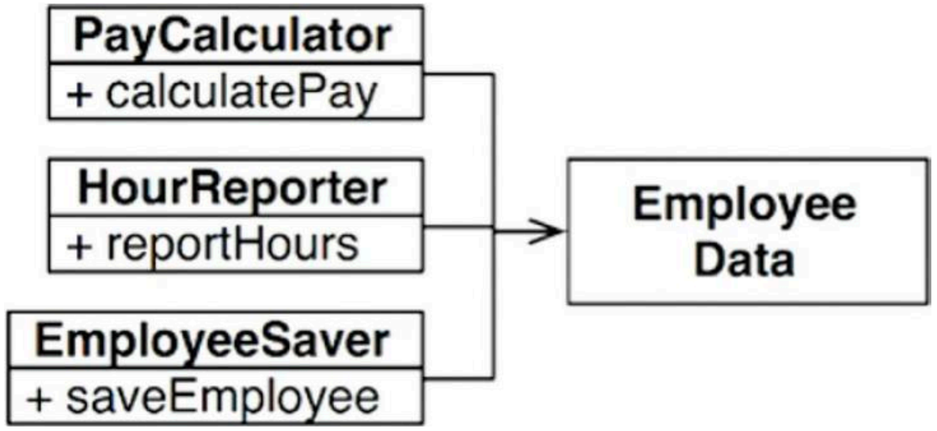
- Domain: an `Employee` class from a payroll application.
  - `calculatePay`: accounting department (CFO)
  - `reportHours`: human resources department (COO)
  - `save`: database administrators (CTO)



- Suppose methods `calculatePay` and `reportHours` share a helper method to calculate `regularHours` (and avoid duplicate code).
- CFO decides to change how non-overtime hours are calculated and a developer makes the change.
- The COO doesn't know about this. What happens?

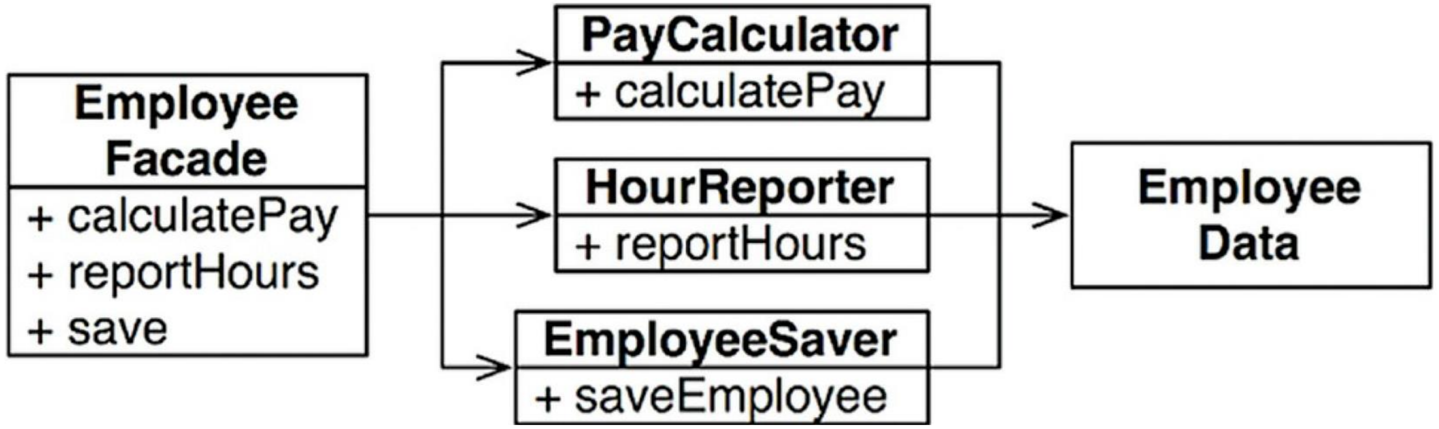
CAUSE OF PROBLEM AND SOLUTION

- Cause of Problem: code is “owned” by more than one actor
- Solution: adhere to the Single Responsibility Principle
  - Factor out the data storage into an `EmployeeData` class.
  - Create three separate classes, one for each actor.



FAÇADE DESIGN PATTERN

- Downside of solution: need to keep track of three objects, not one.
- Solution: create a façade ("the front of a building").
  - Very little code in the façade. Delegates to the three classes.



- We'll talk about the Façade design pattern and many more throughout the term.