Object-Oriented Programming

Cohesion

Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

Cohesion

Degree of connectivity among the elements of a single module and in object-oriented design, a single class/object

Internal measure on an individual class/object/module/method

Why is Cohesion Important?

- Easier to comprehend because it has a single purpose

- Increase reuse since the class is usable in multiple ways in the same or other projects

- Reduce maintenance costs for fixes and enhancements

- Simplify testing

- Allow for replacement

Cohesion Goal

Maximize internal interaction (intramural) among subelements

- Methods share data members (fields) with other methods

- Data members are used in combination

- The more advanced the features of a class, the more this occurs

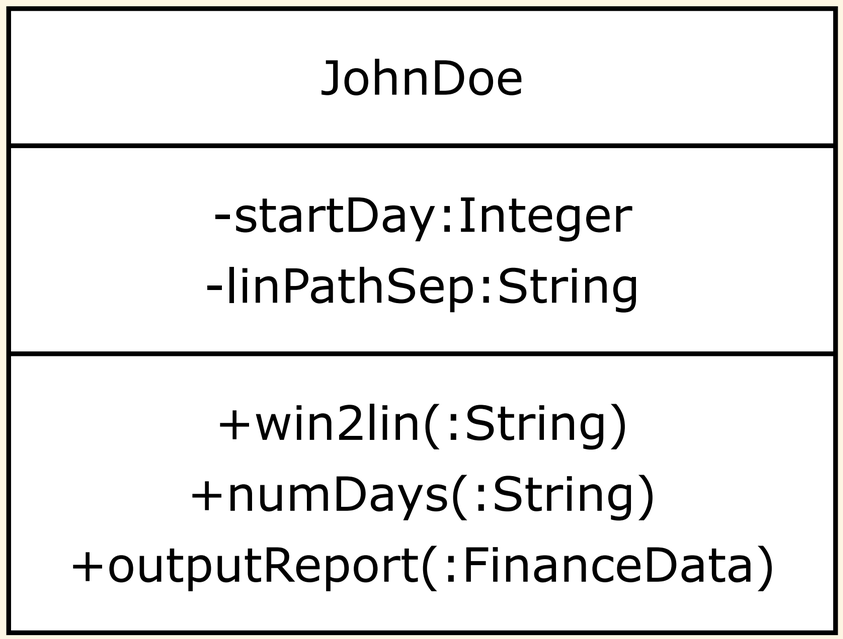
Types of Cohesion

- *Informational* (best)
- *Functional* (best)
- *Sequential* (better)
- *Communicational* (better)
- *Procedural*
- *Temporal*
- *Logical*
- *Coincidental* (worst)

Applicability

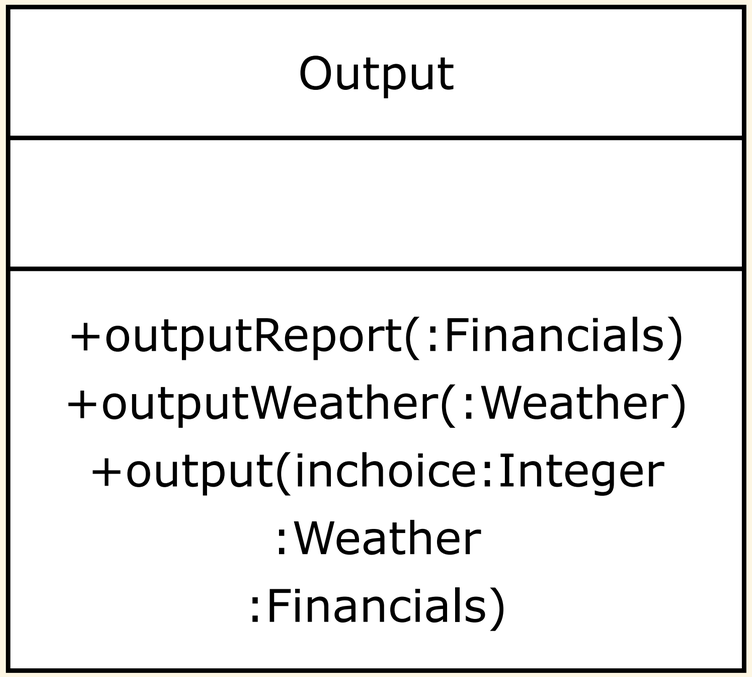
- *system*
- *subsystem/directory*
- *class/file*
- *method/function*
- *section of code*

Coincidental Cohesion



- Performs multiple, completely unrelated actions
- In the example, why are all these methods in this one class? Because John Doe worked on them
- Often based on factors outside of the design: personnel, company organization, history, avoidance of small modules
- No reusability, very difficult to maintain or enhance

Logical Cohesion



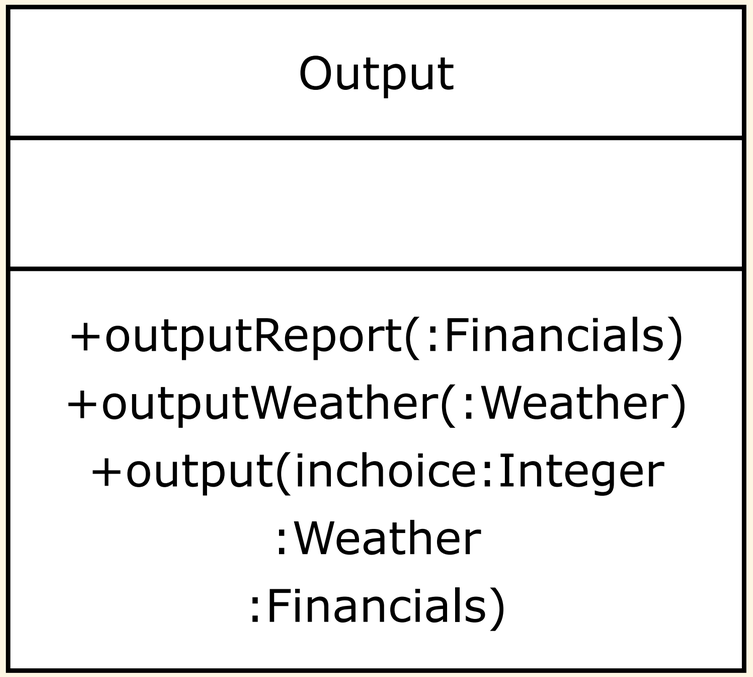
- Performs a series of related actions, one of which is selected by the calling module

- In the example, why are all these methods in this one class? Because they all do output

- Parts have a *logical association*, but not the primary logical association

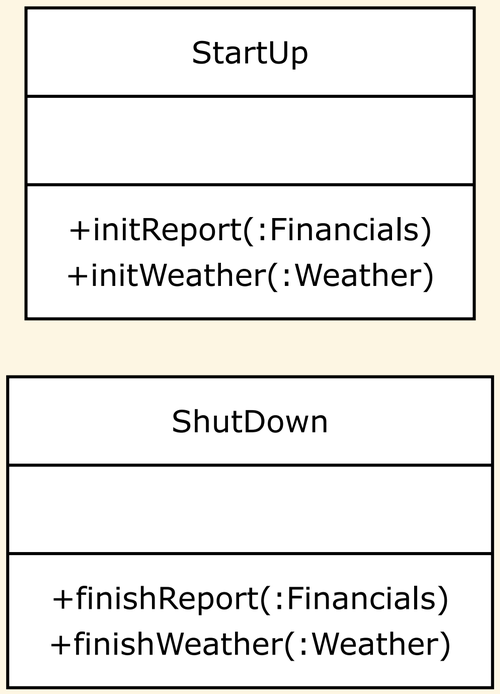
- Primary logical association is based on the highest level of abstraction

Logical Cohesion (cont)



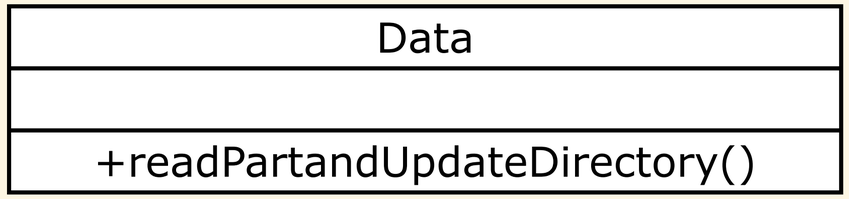
- Often includes both high and low-level actions (in terms of abstraction) in the same class
- Often includes unused parameters for specific uses
- Interface is difficult to understand. Many unrelated actions
- In OO, we put methods near the abstract concept that they work on

Temporal Cohesion



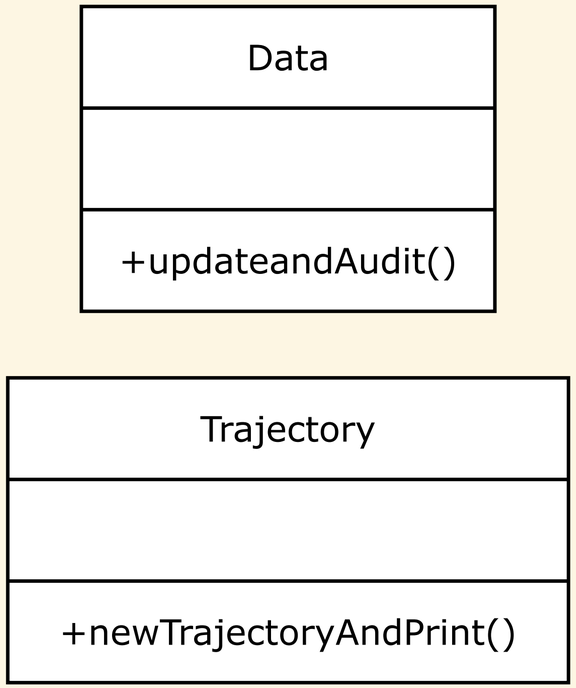
- Perform a series of actions that are related by time occurrence
- In the example, why are all these methods in these classes? Because they all occur at startup or shutdown time
- Addition of subsystems may require additions to multiple modules
- Degrades to Logical Cohesion if the time of action changes
- In OO, we build time occurrence actions into the class and externally control the lifetime

Procedural Cohesion



- Action based on the ordering of steps, related by usage in ordering
- In the example, why does this one method contain both reading a part and updating a directory? Because they both always occur in this order
- Changes to the ordering of steps or purpose of steps require changing the element abstraction, e.g., we add another step
- Situations where this particular sequence applies are often limited
- Each meth cohe

Communicational Cohesion



- Each method performs several operations on its data
- Actions are related but still not completely separated
- Element (e.g., the example methods) cannot be reused
- Maybe the highest cohesion possible/desirable

Sequential Cohesion

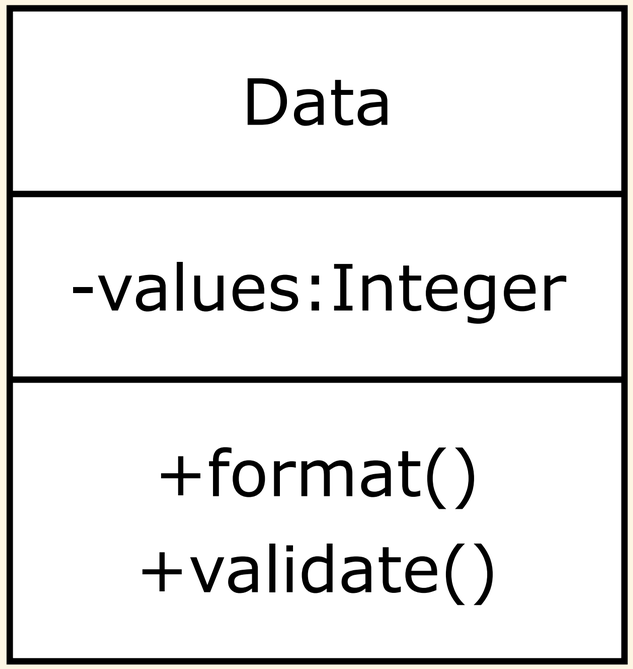
```
std::unique_ptr<Shape> shapeFactory(std::istream& input) {

    // create new shape
    std::unique_ptr<Shape> shape(new Shape());

    // input shape data
    // ...

    // format shape data
    // ...

    // validate shape data
    // ...

    return shape;
}
```

- We place actions together because they are performed in order, each part on the result of the last

- In the example, why are all these operations in this one method? Because the output of each operation is an input to the next operation

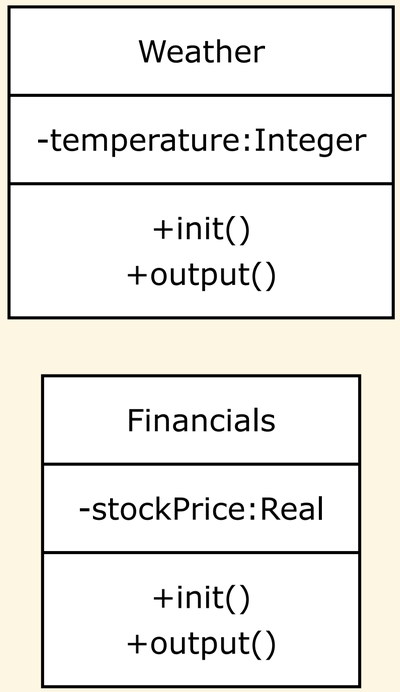
- Requires a crisp abstraction that then becomes the name of the method

Functional Cohesion



- Element (methods) that performs a single action or achieves a single goal
- In the example, why do we have these separate methods? Because each does a well-defined action, indicated by a precise method name
- Maintenance involves the entire element
- High reuse because the element is entirely independent (in its actions) of other elements
- Superior, easily replaced for performance, testing, platform changes, etc.

Informational Cohesion



- Performs several actions
- In the example, why are all these methods in each class? Because they are organized according to the primary abstractions of `Weather` and `Financials`
- Each action has a separate entry point and independent code
- All actions are performed on a shared data structure
- Superior, truly Object-Oriented

Types of Cohesion

- *Informational* (best)
- *Functional* (best)
- *Sequential* (better)
- *Communicational* (better)
- *Procedural*
- *Temporal*
- *Logical*
- *Coincidental* (worst)