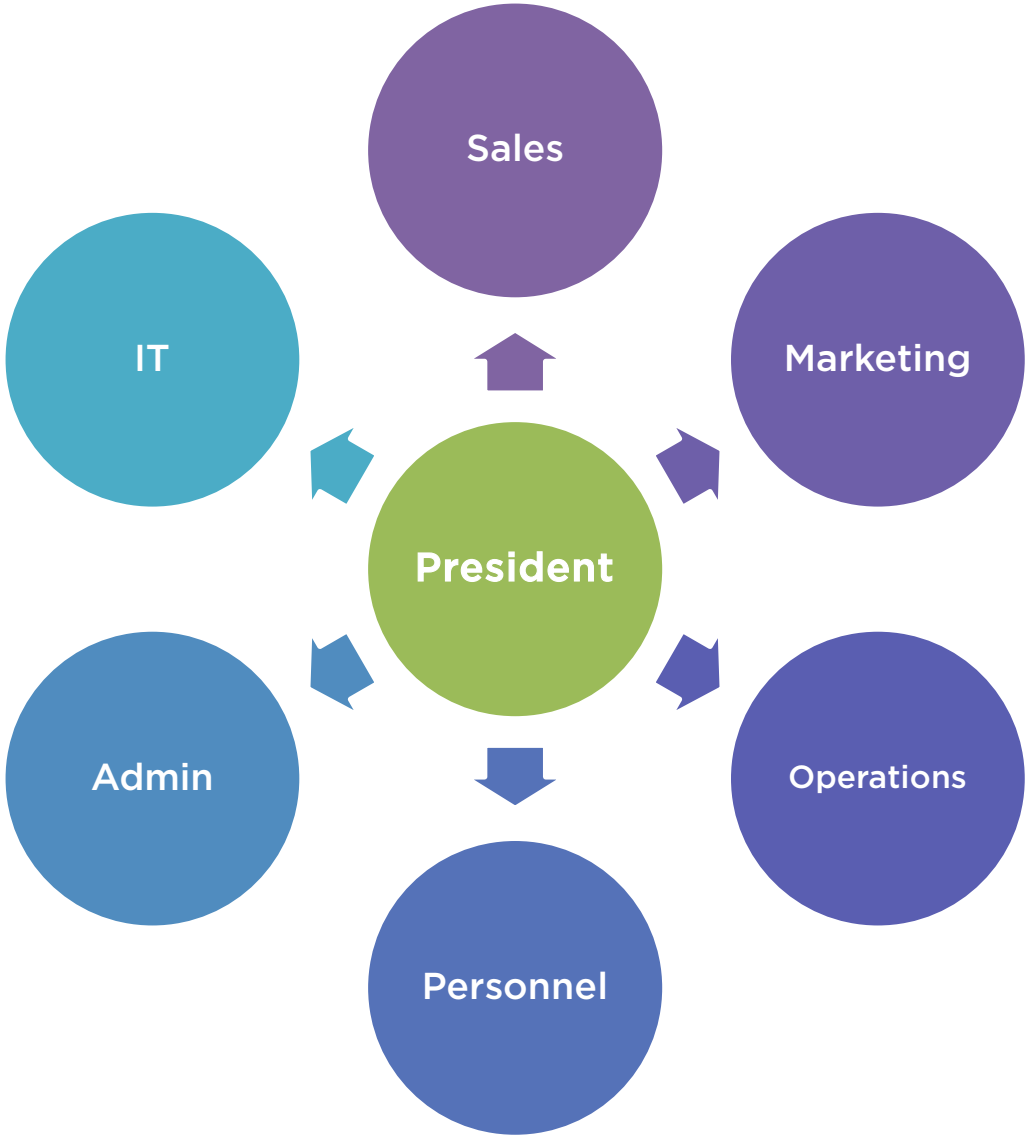
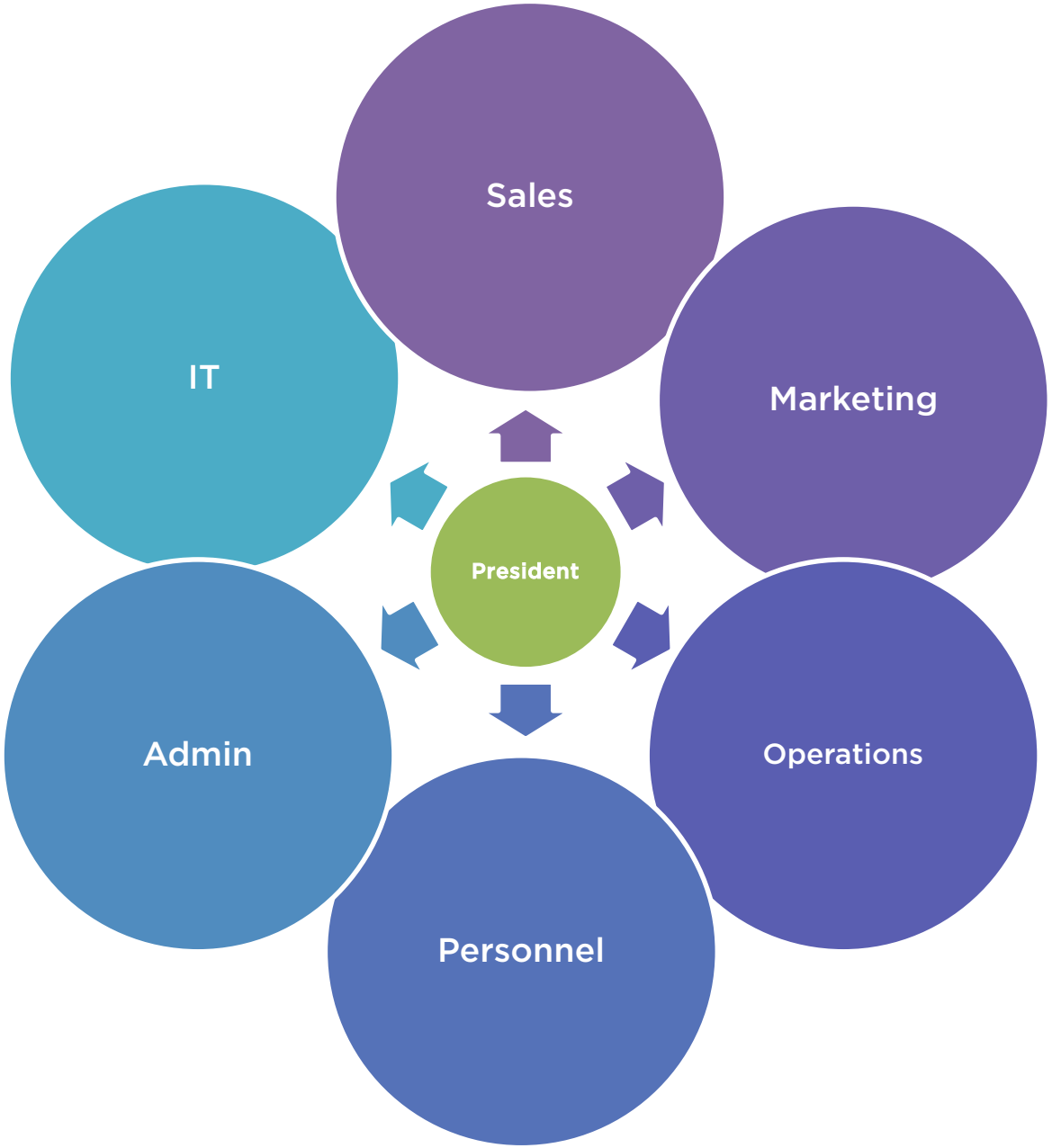
Separation of Responsibilities

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/


Sales
IT
Marketing
President
Admin
Operations
Personnel


Sales
Marketing
Operations
Personnel
Admin
IT
President


Name
Email Address
Postal Address
Validate
Retrieve
Save
Customer

Object-Oriented Programming (OOP)

Identifying classes

- Represents business entities
- Defines properties (data)
- Defines methods (actions/behavior)

Separating responsibilities

- Minimizes coupling
- Maximizes cohesion
- Simplifies maintenance
- Improves testability

Establishing relationships

Leveraging reuse

Minimizing Coupling

Customer	Product	Order	Order Item
•Name •Email address •Home address •Work address •Validate() •Retrieve() •Save()	•Product name •Description •Current price •Validate() •Retrieve() •Save()	•Customer •Order date •Shipping address •Order items •Validate() •Retrieve() •Save()	•Product •Quantity •Purchase price •Validate() •Retrieve() •Save()

Data Access Layer

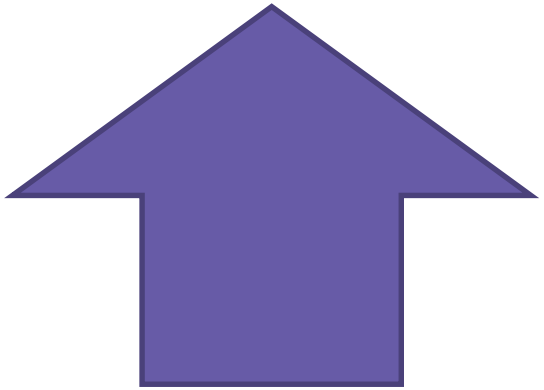
Maximizing Cohesion

Customer	Product	Order	Order Item
•Name •Email address •Home address •Work address •Validate() •Retrieve() •Save()	•Product name •Description •Current price •Validate() •Retrieve() •Save()	•Customer •Order date •Shipping address •Order items •Validate() •Retrieve() •Save()	•Product •Quantity •Purchase price •Validate() •Retrieve() •Save()

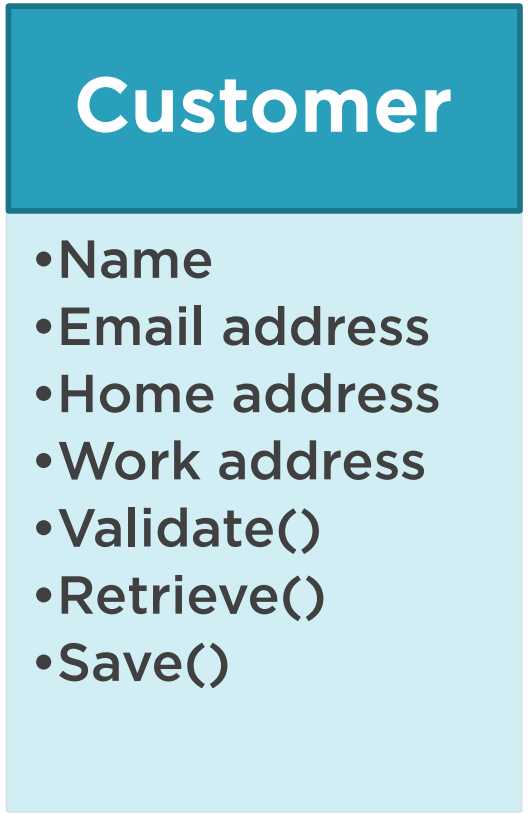
Coupling and Cohesion

Low Coupling

High Cohesion



Separating Responsibilities



Separating Responsibilities

Customer

- Name
- Email address
- Home address
- Work address
- Validate()
- Retrieve()
- Save()

Address

- Street line 1
- Street line 2
- City
- State/Province
- Postal Code
- Country
- Address type
- Validate()

Separating Responsibilities

Customer Repository

- Retrieve()
- Save()

Customer

- Name
- Email address
- Home address
- Work address
- Validate()

Address

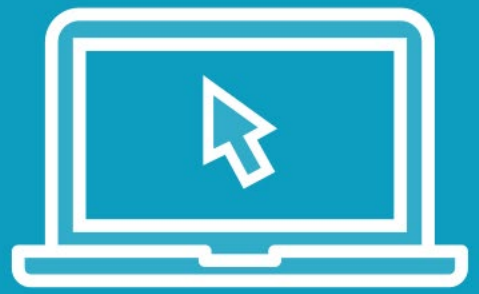
- Street line 1
- Street line 2
- City
- State/Province
- Postal Code
- Country
- Address type
- Validate()

Separating Responsibilities

Customer	Product	Order	Order Item
•Name •Email address •Home address •Work address •Validate()	•Product name •Description •Current price •Validate()	•Customer •Order date •Shipping addr. •Order items •Validate()	•Product •Quantity •Purchase price •Validate()

Customer Repository	Product Repository	Order Repository	Address
•Retrieve() •Save()	•Retrieve() •Save()	•Retrieve() •Save()	•Street line 1 + 2 •City •State/Province •Postal Code •Country •Address type •Validate()

Demo

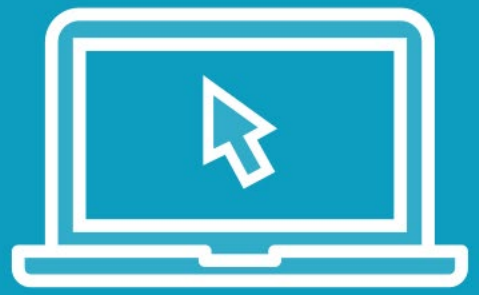


Building the Address class

Address

- Street line 1
- Street line 2
- City
- State/Province
- Postal Code
- Country
- Address type
- Validate

Demo

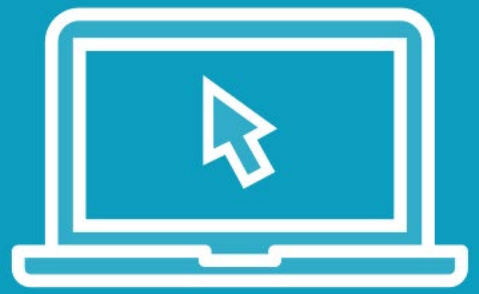


Building the Customer Repository Class

Customer Repository

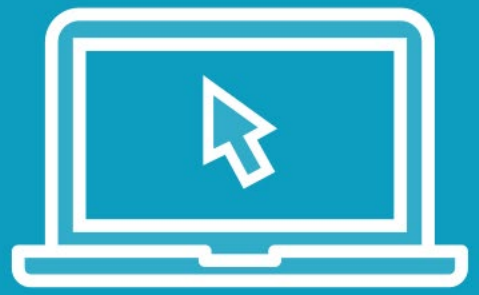
- Retrieve()
- Save()

Demo



Testing the Customer Repository Class

Demo



Building the remaining repository classes

Product Repository	Order Repository
•Retrieve() •Save()	•Retrieve() •Save()

Separating Responsibilities

Customer	Product	Order	Order Item
•Name •Email address •Home address •Work address •Validate()	•Product name •Description •Current price •Validate()	•Customer •Order date •Shipping addr. •Order items •Validate()	•Product •Quantity •Purchase price •Validate()

Customer Repository	Product Repository	Order Repository	Address
•Retrieve() •Save()	•Retrieve() •Save()	•Retrieve() •Save()	•Street line 1 + 2 •City •State/Province •Postal Code •Country •Validate()

Evaluate Coupling

What: Dependence on other classes or external resources

How: Extract dependencies into their own classes

Why: Easier to test and maintain

Example: Move the responsibility for accessing the data store to a repository class

Evaluate Cohesion

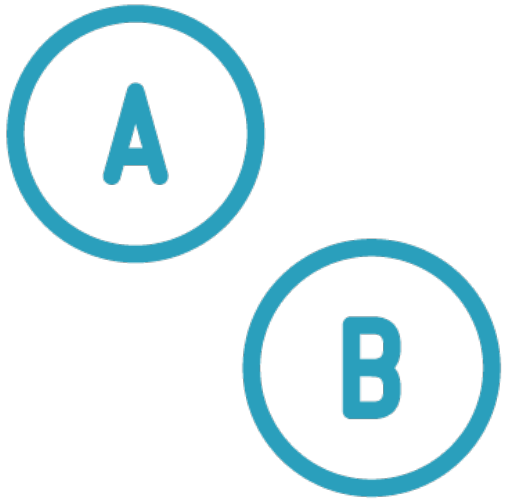
What: Class members should relate to the class purpose

How: Extract unrelated members into their own classes

Why: Easier to understand, test and maintain

Example: Move the responsibility for managing addresses into a separate class

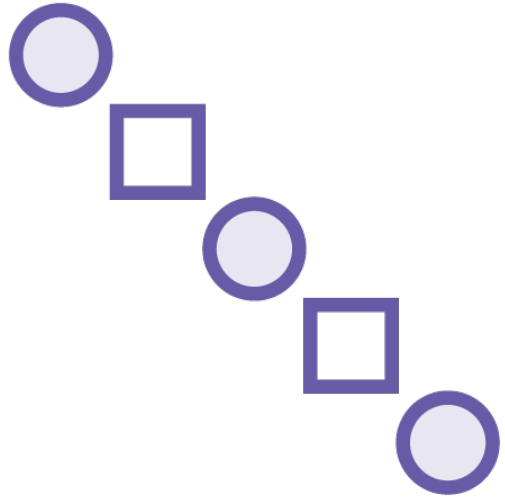
Additional Concepts



Separation of concerns

YAGNI

You aren't going to need it



Design patterns

Object-Oriented Programming (OOP)

Identifying classes

- Represents business entities
- Defines properties (data)
- Defines methods (actions/behavior)

Separating responsibilities

- Minimizes coupling
- Maximizes cohesion
- Simplifies maintenance
- Improves testability

Establishing relationships

Leveraging reuse