Coupling / Cohesion

By Rick Mercer with help from
Object-Oriented Design Heuristics, Arthur Riel
**Coupling**

- **Coupling**: A measure of how strongly one class is connected to, has knowledge of, or relies upon other classes
  - Low coupling: the class is not dependent on many other classes—good
  - High Coupling: class is dependent on many others—bad
    - A change to one class forces changes in others
    - More difficult to understand a type in isolation
    - Harder to reuse because it depends on others

 Assign responsibilities so coupling remains low
Which is the better design?

- Should a Point of Sale Terminal (POST) object construct an instance of Payment, then add the payment to the sale
  - POST would be coupled to both Payment and Sale in the following Collaboration diagram
  
  - With this design, POST needs to know 2 classes

Example from Craig Larmans *Applying UML and Patterns*
An alternative design

- Try associating Payment with Sale
  - Sale is coupled to Payment in either design
  - But low coupling favors this 2nd design

In practice coupling can’t be considered in isolation from other design guidelines such as Expert and High Cohesion
Try to keep coupling low

- Common forms of coupling
  - Class A has an instance of Class B
  - Class A send a message to an instance of Class B
  - Class A is a subclass of Class B (inheritance)
  - Class A implements interface I

- Coupling does occur, in any design

- A dangerous case of high coupling:
  - Allow Class A uses instance variable of Class B
  - Consider low coupling important when evaluating design
High Cohesion, Good

- **Cohesion**
  - Synonyms: consistency, pulling together
  - A measure of how strongly related and focused the responsibilities of a class are

- **Assign responsibilities for high cohesion**
  - High cohesion is good
  - Low cohesion is bad
    - hard to understand
    - hard to reuse
    - hard to maintain
    - fragile; constantly affected by change
High Cohesion

- High cohesion is when parts of a module are grouped because they all contribute to a single well-defined task of the module.

- Examples
  - PlayList manages the songs in the queue
  - BankAccount has no blast off method
  - Any well structured class where the methods are related, String for example
High vs. Low Cohesion

- High functional cohesion exists when the elements of a class "all work together to provide some well-bounded behavior"  
  
  Grady Booch

- Low cohesion
  
  - One class is responsible for things in different functional areas. Examples:
    - Model arranges Views, Listeners make the Rules
  
  - One class has sole responsibility of many tasks
    - Jukebox coordinate activities and determines if a song can play, plays audio files, maintains collections, manages the play list, ...
High Cohesion

- A class has moderate responsibilities in one functional area and collaborates with other objects to fulfill tasks:
  - e.g. PokerDealer coordinates but gets help from
    - a deck of cards that can shuffle and deal cards
    - the player that can figure out what to do next and knows what he or she can bet
    - a view to show poker hands, pot, cards, animations

- Small number of methods, highly related functionality, doesn't do too much
Benefits of High Cohesion

- Design is clearer and more easily understood
- Maintenance (bug fixes, enhancements, changing business rules) is easier
  - The fewer classes you touch the better Adele Goldberg
- High cohesion can help you attain lower coupling