principles: explanatory text

when writing informal text, don’t
› belabor the obvious
› repeat what’s already said (better) in code or models

more is not better!
› loses reader’s attention
› makes it hard to focus on what matters
\[ x = x + 1 \] // add one to the variable \( x \)

Notes:

- Each shopper has one password, username and shopping cart.
- Each Storeowner has one password, username, and shopping catalog.
- Users can have the same password, two catalogs can have the same items, and two carts can have the same items.
- Each cart/catalog can have multiple items.
- Each store item has one price, name, and quantity.
- There may be more than one store item with the same price, name or quantity.
principles: style in models

use the features of the modeling language
› don’t invent syntax on the fly
› follow conventions

example: object models
› use boxes, not ovals
› label the relation arrows, with relation names (not “has”)
› use lower case for relations, capitalize sets
Each shopper has one password, username and shopping cart.

Each storeowner has one password, username, and shopping catalog.

Users can have the same password, two catalogs can have the same items, and two carts can have the same items.

Each cart/catalog can have multiple items.

Each store item has one price, name, and quantity.

There may be more than one store item with the same price, name or quantity.

no relation names

Store Items?
principles: object models

the object model defines
› the state of the system; what’s stored  
› so use it!

basic questions
› what (information is stored)  
› when (transient or persistent)  
› where (client or server)

conceptual model
› key concepts, generalizations

invariants
› uniqueness properties especially
- Each shopper has one password, username, and shopping cart.
- Each storeowner has one password, username, and shopping catalog.
- Users can have the same password, two catalogs can have the same items, and two carts can have the same items.
- Each cart/catalog can have multiple items.
- Each store item has one price, name, and quantity.
- There may be more than one store item with the same price, name, or quantity.

**basic questions**
- what: shopping catalog? quantity of store item (in stock? on order)?
- when: shopping cart persists or one per session?
- where: cart in cookie or database?
Each shopper has one password, username, and shopping cart.

Each storeowner has one password, username, and shopping catalog.

Users can have the same password, two catalogs can have the same items, and two carts can have the same items.

Each cart/catalog can have multiple items.

Each store item has one price, name, and quantity.

There may be more than one store item with the same price, name, or quantity.

Conceptual model:
- Key concepts: what is a store item? CartItem = CatalogItem?
- Generalization: no shoppers also storeowners?
• Each shopper has one password, username and shopping cart.
• Each Storeowner has one password, username, and shopping catalog.
• Users can have the same password, two catalogs can have the same items, and two carts can have the same items.
• Each cart/catalog can have multiple items.
• Each store item has one price, name, and quantity.
• There may be more than one store item with the same price, name or quantity.

**invariants**

› usernames unique?
› shared shopping catalogs? shared carts?
› item in multiple catalogs? with same quantity?
principles: Rails model classes

a model class defines a set of
› objects = database tuples

so
› if name of class of User, expect many users
› if name of class is Catalog, expect many catalogs
class Catalog < ActiveRecord::Base
  attr_accessible :description, :name, :price, :quantity
end

what should this class be called?
› product?
principles: model/code correspond

why build on object model?
› get problem straight
› invent key concepts
› *make implementation easier*

so code should be based on object model
› correspondence of names
› correspondence of structure

but separation of concerns
› model concern: what’s stored and why
› code concern: efficiency, navigation, encapsulation

so derive code by simple decisions
› which sets become model classes?
› which relations become attributes? associations?
class Cart < ActiveRecord::Base
    attr_accessible :amount, :item, :user
end

class Catalog < ActiveRecord::Base
    attr_accessible :description, :name, :price, :quantity
end

class Shared < ActiveRecord::Base
    attr_accessible :creator, :list, :price, :upvotes
end

class User < ActiveRecord::Base
    attr_accessible :email, :password, :password_confirmation
    has_secure_password
    validates_presence_of :password, :on => :create
end

relationship to object model?
› seems to have started again from scratch?
issues

› what do “items” relations store? better names + comments
› storekeeper owns cart? where are users?
› item quantities?
› when is checks out updated? look to operations
class Item < ActiveRecord::Base
  attr_accessible :description, :name, :price
end

class OrderIt < ActiveRecord::Base
  attr_accessible :item_number, :order_id
end

class CartItem < ActiveRecord::Base
  attr_accessible :item_number, :store_number
end

class CartsController < ApplicationController
  # DELETE Cart Item object
  # Increment the Item object quantity
  def destroy_item
    @cart_item = CartItem.find_by_id(params[:id])
    @item = Item.find_by_id(@cart_item.item_number)
    @item.increment_quantity
    @cart_item.destroy
  end

  class Cart < ActiveRecord::Base
    # requires: customer who is checking out cart
    # modifies: cart items, adding each to a new order belonging to the customer, deletes cart items from customer cart
    def final_checkout(customer)
      @order = customer.orders.new()
      @order.update_attributes(customer_id: @customer, id: self.id)
      self.cart_items.each do |item|
        @item = @order.order_its.new()
        @item.update_attributes(item_number: item.item_number)
        item.destroy
      end
    end
  end
end