implementing generalization

Daniel Jackson
generalization
option A: two classes

approach
› undo the generalization
› replicate associations

class User < ActiveRecord::Base
  has_many :posts
  has_many :comments
end
class Post < ActiveRecord::Base
  belongs_to :user
  has_many :comments
end
class Comment < ActiveRecord::Base
  belongs_to :user
  belongs_to :post
end
option B: one class

approach
› merge two classes into one
› some fields will become optional

class User < ActiveRecord::Base
  has_many :entries
end
class Entry < ActiveRecord::Base
  belongs_to :user
  has_many :comments :class_name => "Entry"
end
class CreateEntries < ActiveRecord::Migration
  def up
    create_table :entries do |t|
      t.boolean :is_post
      t.references :user
      t.text :content
    end
  end
end
option C: single table inheritance

approach
- single table, but three model classes
- special column in table assigns object to class
- Rails will automatically handle mapping from classes to table

```ruby
rails generate scaffold Entry type:string

class Entry < ActiveRecord::Base
  belongs_to :user
end
class Post < Entry
class Comment < Entry
```

for more info
- http://www.martinfowler.com/eaaCatalog/singleTableInheritance.html
option D: polymorphic association

consider relation content: Entry -> Content

approach
  › two classes, but one association carrying type of target

```ruby
class Content < ActiveRecord::Base
  belongs_to :entry, :polymorphic => true
end
class Post < ActiveRecord::Base
  has_one :content, :as => :entry
end
class Comment < ActiveRecord::Base
  has_one :content, :as => :entry
end
class CreateContents < ActiveRecord::Migration
  def change
    create_table :contents do |t|
      t.string :text
      t.references :entry, :polymorphic => true
    end
  end
end
```