OO Design for the Rest of Us

OOD teaches a complicated method, best for large systems.

Here we teach the ten cent version.
Ten Cent Object-Oriented Design

Let’s do this.
Some (more) things we’d like to be true

- I can drive the design from the user stories
- The code just “writes itself”
- My code will be easy to understand by the team
- My code will be SRP and DRY
myCity’s Show Friends on Map Story

As a user
I want to see my nearby friends on the map
So that I know who to contact for a spontaneous meet-up

Given I am logged in
And my location is known
When my friends are online
And my friends’ location is known
And my friends are near me
Then I see them on the map

Rule

Write code so that it sounds like the specification.
I call this “coding to the specification”

Recall that OO theory and SRP dictate objects that:
- represent a “thing”
- respond to messages about itself

This suggests the start of a design process
1. Identify the “things”
2. Identify their messages
3. Assemble into classes
4. Right things call right messages
myCity’s Show Friends on Map Story, II

As a user
I want see my nearby friends on the map
So that I know who to contact for a spontaneous meet-up

Given I am logged in
And my location is known
When my friends are online
And my friends’ location is known
And my friends are near me
Then I see them on the map

1. Identify the “things”
   - me (previous user story)
   - friends, friend
   - map (previous)

2. Identify the messages
   - me.logged-in (previous)
   - friend.location-known
   - friend.location
   - friend.online
   - friend.near(me)
   - friend.image

3. Assemble into classes (TBD)

4. Right things call right messages (TBD)

- Note how messages “sound like” the specification
- Plural objects may be list/set of singular
- Classes for some objects may already exist
  - Look for new messages
- May find repetitive messages
  - e.g., location-known vs. location?
  - maybe location just returns null (or not)
- Implied messages
  - “see them on the map” -> something to see
As a user
I want to see my nearby friends on the map
So that I know who to contact for a spontaneous meet-up

Given I am logged in
And my location is known
When my friends are online
And my friends’ location is known
And my friends are near me
Then I see them on the map

1. Identify the “things”
   - me (previous user story)
   - friends, friend
   - map (previous, built-in)

2. Identify the messages
   - me.logged-in (previous)
   - friend.location-known
   - friend.location
   - friend.online
   - friend.near(me)
   - friend.image

3. Assemble into classes

4. Right things call right messages (TBD)
myCity’s Show Friends on Map Story, IV

**Given** I am logged in  
**And** my location is known  
**When** my friends are online  
**And** my friends’ location is known  
**And** my friends are near me  
**Then** I see them on the map

- This is the hardest part
- What *thing* does something to *itself* so friends get drawn on map?  
  - **Not mentioned in scenario!**  
- **Map?** Predefined in Android  
- **How about a FriendMap?**  
  - A map with added responsibility of drawing my friends  
- Let’s take a look at this

```java
class Friend {  
    Location location(); // null -> not-known  
    boolean online();  
    Bitmap image();  
}

class Friends implements Iterable<Friend> {  
    Friends online();  
    Friends locationKnown(); // -> online?  
}
```

4. **Right things call right messages**
class Friend {
    Location location(); // null -> not-known
    boolean online();
    Bitmap image();
}

class Friends implements Iterable<Friend> {
    Friends online();
    Friends locationKnown(); // -> online?
}
Given I am logged in
And my location is known
When my friends are online
And my friends’ location is known
And my friends are near me
Then I see them on the map

class Friend {
    Location location();
    boolean online();
    Bitmap image();
}

class Friends implements Iterable<Friend> {
    Friends online();
    Friends locationKnown(); // -> online?
}

class NearbyFriendsMap extends GoogleMap {
    private Friends friends;
    private GoogleMap map;
    NearbyFriendsMap(GMap map, Friends friends);
    void showNearbyFriends();
    // reimplemt every GM method so that it re-calls
    // the method on map. Called delegation. E.g.
    Marker addMarker(...) { map.addMarker(...); }
}

- Is a map, plus nearby friend stuff
- Dual responsibilities (Map things, Friend drawing)
- Huge amount of work (reimplement all methods)
- Illegal in Android: GoogleMap is a “final” class!
- SRP implies class that just does drawing friends

4. Right things call right messages
Basic idea is an intermediary whose **Single Responsibility** is *drawing my friends on the map*.

```java
class NearbyFriendsTracker {
    private Friends friends;
    private GoogleMap map;
    NearbyFriendsTracker(GMap map, Friends friends);
    void showNearbyFriends();
}
```

- **Single responsibility**: drawing friends on map!
- Don’t have to bunch of “extra” methods
- Very similar, despite SRP
- Still trying to make code sounds like specification

4. **Right things call right messages**
UML Diagram: Subclass Approach

Friends
- List<friend> friends
- Friends online()
- Friends locationKnown()

Friend
- boolean online()
- Location location()
- Bitmap image()

NearbyFriendsMap
- GoogleMap map
- Friends friends
- showNearbyFriends()

GoogleMap

<interface>
Iterable

*
UML Diagram: Middleman Approach

```
<interface>
  Iterable
  ...

Friends
- List<friend> friends
  ----------------------
  + Friends online()
  + Friends locationKnown()

* 

Friend
+ boolean online()
+ Location location()
+ Bitmap image()

NearbyFriendsTracker
- GoogleMap map
  ----------------------
  + showNearbyFriends()

GoogleMap
  ...
```
Take Aways

- Design method combines a few guidelines
  - find *things* and *messages*
  - pack into SRP classes
  - make code sound like the specification

- Non-trivial, takes experience
  - plural thing implies singular things
  - messages that don’t quite work
  - “Missing” (implied) things and messages

- May take multiple tries on harder parts

- Very common to revise after the fact (refactor)
  - New insights, new features benefit from diff. design
Object-oriented design is intuitive, but subtle
- Java is just a tool, does not guarantee good design
  (Just because I have an expensive camera does not make me a good photographer :)
- Easy to put functionality in wrong place, make classes too big, or make too small

Possible to diagnosis and repair a design before or after the coding (may require both)
- SRP, DRY
- Change in one class affects another (SRP)
- Small change affects multiple classes or methods

Unfortunately, there are many kinds of design mistakes, and unique repairs for them