40.317 Lecture 9

Dr. Jon Freeman

16 June 2020

Slido Event #S140

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Agenda

- OO and Design Patterns: a brief revisit
- Graphical User Interfaces (“GUIs”)
- GUIs in Python 3
- Service-Oriented Architecture
- Is an SO design always a good design?

OO and Design Patterns, revisited

The “One True Class Hierarchy” for your problem domain sounds like a helpful thing to have, and rewarding to create.

Could its creation and use lead to problems, now or in the future?

OO and Design Patterns, revisited

The more volatile the problem domain, the more an OTCH can be a hindrance.

Consider instead the “Properties Pattern,” described by Steve Yegge in a 2008 post.

It is not to be confused with the Prototype Pattern, which is merely about speeding up object construction by cloning.

The Prototype Pattern, briefly

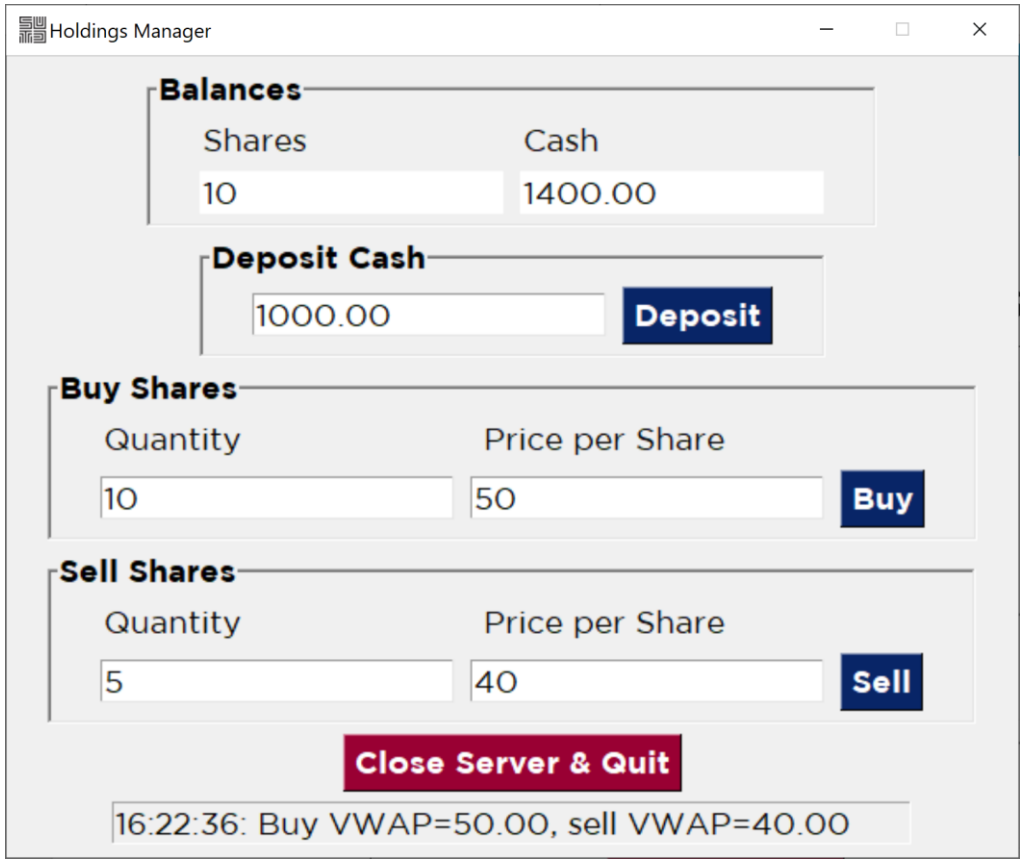
An object A contains:

- a reference to a similar object B which serves as A's prototype;
- a set of <name, value> pairs which either *mask* or *supplement* the <name, value> pairs in B.

- The relationship between A and B is not **IS-A**, and not **HAS-A**, but rather **IS-LIKE**.
  - **IS-LIKE** can of course co-exist with other relationships.

GUIs

Why are we studying this topic?
We will use a demo to answer this question!



In-Class Discussion

Does this GUI make you think differently about our example application?

If so,

- How?
- Why do you suppose that is?

In-Class Discussion, continued

What changes do you suppose I needed to make to the server in order to create this GUI?

Some of the input validations could be done in the client alone, which would be more efficient. Should they?

A Story of a GUI's Importance

A story from my work experiences about how important a GUI can be.

- The underlying cognitive bias at work in this story is known as the false consensus effect.
  - You can use one cognitive bias – in this case, the IKEA effect – to counteract another!

Two Laws of UI Design

From Jef Raskin in his book *The Humane Interface*:

- **First Law**: A computer shall not harm your work or, through inactivity, allow your work to come to harm.
- **Second Law**: A computer shall not waste your time or require you to do more work than is strictly necessary.

Principles of UI Design

According to Larry Constantine and Lucy Lockwood, there are six principles which involve:

- Structure
- Simplicity
- Visibility
- Feedback
- Tolerance
- Reuse

Six UI Design Principles, continued

- Structure:
  - Group related vs. unrelated things
  - Make similar things resemble one another, and differentiate dissimilar things
  - In general, adopt a consistent approach users can recognise
- Simplicity:
  - Make common tasks easy
  - Communicate clearly in the user's own language
  - Provide shortcuts which are meaningfully related to longer procedures

Six UI Design Principles, continued

- Visibility:
  - Only *required* options and content should be visible
  - Don’t overwhelm with alternatives or confuse with unnecessary information

- Feedback: in concise, unambiguous language, keep users informed of
  - actions or interpretations
  - changes of state or condition
  - relevant errors or exceptions

Six UI Design Principles, continued

- Tolerance:
  - Allow undoing and redoing if possible
  - Permit varied inputs, and varied sequences of inputs
  - Interpret all reasonable actions

- Reuse:
  - Reuse both internal and external components and behaviours
  - Decreases the need for users to re-think and remember

In-Class Discussion

Think of at least one example of good UI design, and at least one example of poor UI design, from any of the technologies you use routinely, such as:

- Android or iOS
- Windows or MacOS
- Apps such as iTunes, Netflix, and so on
- Popular websites

GUIs in Python 3

For this class we want a GUI toolkit which

- is actively maintained,
- is simple to install, and
- has a fairly low learning curve.

This year I recommend PySimpleGUI, https://pysimplegui.readthedocs.io/en/latest

It's a friendly wrapper around tkinter, the standard Python interface to the Tk toolkit.

Installing PySimpleGUI

Simply run

```
pip install pysimplegui
```

The documentation is not concise, but many clear example programs are provided.

You will probably need to read the source code at some point.

Why PySimpleGUI?

- Actively maintained
- Written in pure Python
  - Readable, well organised code base
- Minimal dependencies – just tkinter
- Simple installation, i.e. no C compilation
- Cross-platform
- Very “Pythonic” interface
- Provides many useful GUI elements

PySimpleGUI Demos

There are many to choose from, including:

- Demo_Timer.py
- Klondike_Solitaire.py

PySimpleGUI Caveats

- There is no interface for assigning a callback function to a button click.
  - Instead you have to maintain and use a dictionary of your own which maps button names (“keys”) to such functions.
- In a multi-threaded app:
  - You can only call PySimpleGUI routines from the *main* thread.
  - When your program finishes you must also del[ete] the window after closing it.

Service-Oriented Architecture (SOA)

Why are we studying this topic?

It's the approach we have been recommending, described in full generality.

Implemented well, it promotes:

- encapsulation
- code reuse
- (rapid) scalability
- alignment of business and technology

Definition of SOA: First Attempt

"A set of components which can be invoked, and whose interface descriptions can be published and discovered."

Definition of SOA: Second Attempt

"The policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service consumer. Services can be invoked, published and discovered, and are abstracted away from the implementation using a single, standards-based form of interface." (CBDi Forum)

Properties of a Good SOA Design

- Technology neutral
- Standardised
  - I.e. implemented using standard protocols
- Consumable
  - Automated discovery and use are possible
- Reusable
- Abstracted
- Published
- Formal
  - A contract both sides must adhere to
- Relevant

Service Catalogs

A logical culmination of SOA is a catalog of services users can explore, to discover the services they need by themselves.

Existing service catalogs are company-specific, or at most industry-specific.

Amazon provides a service catalog creation service.

SOA vs. “Web Services”

Web services are defined in a technical way and have to respect certain protocols:

- In general, they send and receive XML documents via HTTP(S).
- They can be discovered via WSDL (Web Services Definition Language).

A “Web service” *uses certain standards* to *expose a capability*. It need not have a true service orientation.

SOA vs. “Microservices”

SOA and MSA are the same set of standards used at different layers of an enterprise. Hence, MSA is a subset of SOA.

Microservices are autonomous and fine-grained. *They are building blocks for application developers.*

Typical use cases for microservices:

- Authentication and authorisation
- Auditing and logging

SOA vs. Good Design

Is a service-oriented design always a good design? Why or why not?

Let's read through this article and then re-consider this question.

SOA vs. Good Design, continued

Designing an SOA involves inventing a set of boundaries.

What's inside each boundary must be self-contained, and must remain so over time.

- How many of us are naturally good at inventing boundaries?
- How does anyone learn how to do this, or learn how to do it better?

Thank you

A BETTER WORLD BY DESIGN.

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN