A Timeless Way of Building: patterns

- Design patterns had their genesis in Architecture, Christopher Alexander, *The Timeless Way of Building, Notes on the Synthesis of Form*
  - Not universally accepted by architects, but design patterns acknowledged as fundamental to OO programming
  - Adherence to form in explanations isn't as important as understanding the patterns
- "A whole is created by putting together parts. The parts come first: and the form of the whole comes second"
  - "It is impossible to form anything which has the character of nature by adding preformed parts"
  - Create flexible/reusable classes, not modules (do you want to live in a modular home?)

From simple to complex

- "Design [is a] sequence of acts of complexification; structure is injected into the whole by operating on the whole and crinkling it, not by adding little parts to one another. In the process of differentiation, the whole gives birth to its parts: The form of the whole, and its parts, come into being simultaneously. The image of the differentiating process is the growth of an embryo."
  - Start simple, and grow the software in a natural way
  - Inject information as needed, don't start off with the final concept as a full-blown entity: too much information is overload

- Identify patterns, start with those that create context for other patterns, implement by adding things piecemeal.

What is encapsulation

- Why do we have public/private methods/instance variables?
  - What's the purpose of information hiding?
- Why do some subclasses override methods and others don't?
  - Encapsulate what is different in behavior
- Why do we hide subclasses (e.g., via a factory)?
  - TTTButton vs. OogaButton vs. Jbutton
- Why do we hide classes?
  - Does PuzzleGui need to know about TTTPlayer?

What about Inheritance?

(see Design Patterns Explained)

- Find what varies and encapsulate it
  - In an instance variable, in a class, in a hierarchy
  - What is a factory class/hierarchy for?

- Use composition/delegation rather than inheritance
  - Has-a/uses-a can be better than is-a
  - Consider the WallHuggingRectangleBall class proliferation problem

- Design to interfaces/concepts, not to implementations
  - When do we use abstract class vs interface?