

# **CS 417**

## **Design Patterns**

### **Design patterns: Template and Strategy**

Dr. Chad Williams  
Central Connecticut State University

# Agenda

- Announcements
  - Midterm next Wednesday
  - Review session next class
- Design pattern: Template Method
- Design pattern: Strategy

# Abstract classes

## Purpose:

- Provide some common implementation
- Deferring other common functionality to child classes (abstract)
  - Place holder for different implementations of common behavior
  - Methods referred to as *hooks*, as different implementations may hang on common place holder

# Template Methods

- Template pattern performs common functionality deferring details to subclasses
  - Within abstract class, methods that contain calls to hooks are called *template methods*
  - Template method defines structure of how common behavior is carried out across all subclasses

# Design pattern: Template method

- **Category:** Behavioral design pattern
- **Intent:** To define the skeleton of an algorithm in a method, deferring some steps to subclasses, thus allowing subclasses to redefine certain steps of the algorithm

# Design Pattern: Template method (cont.)

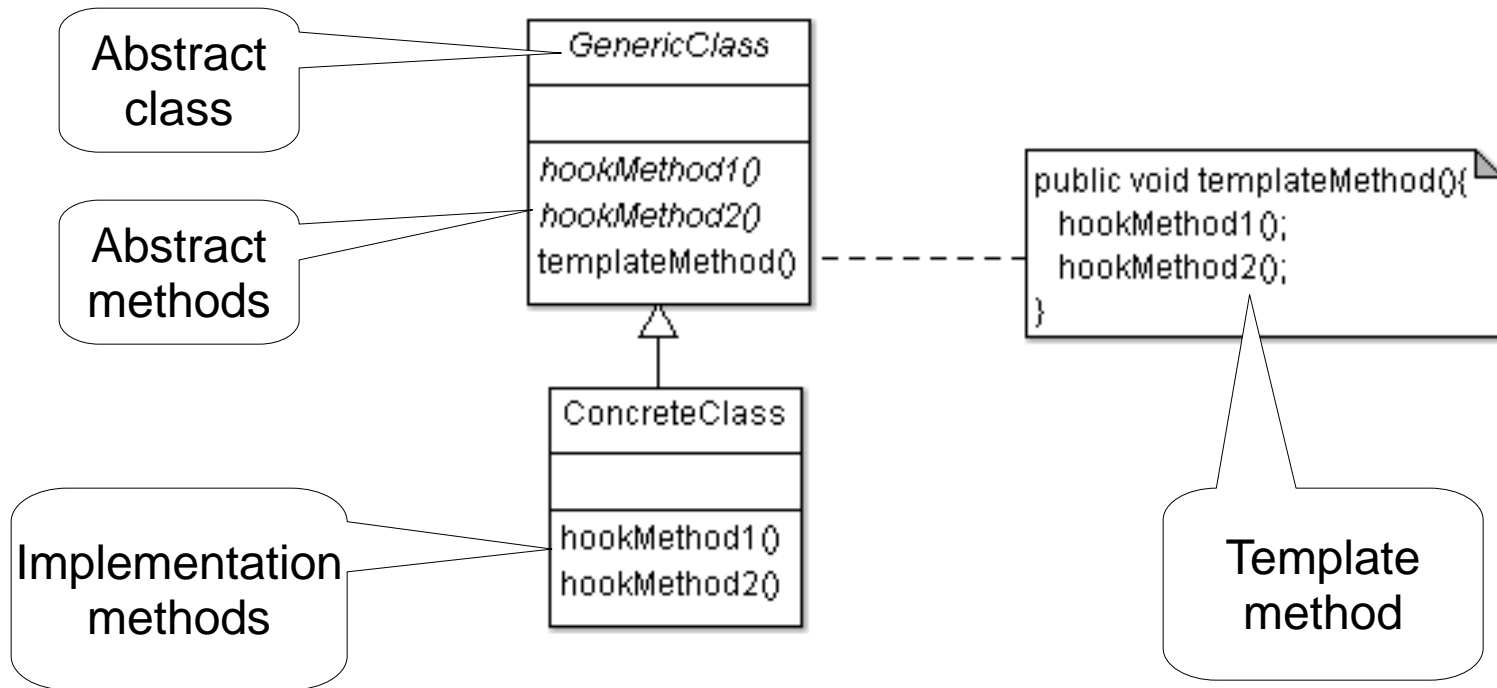
- **Applicability:** The Template Pattern should be used:
  - Implement invariant parts of algorithm leaving implementation to subclasses
  - Refactor and localize common behavior of subclasses to avoid duplication

# Design Pattern: Template method (cont.)

- **Participants:**

- **GenericClass** - defines abstract hook methods that subclasses must override to implement steps of algorithm(s). Defines functionality through calls to hook methods.
- **ConcreteClass** – implements the hook methods to carry out subclass specific steps of the algorithm(s) defined in the template method

# Template method UML

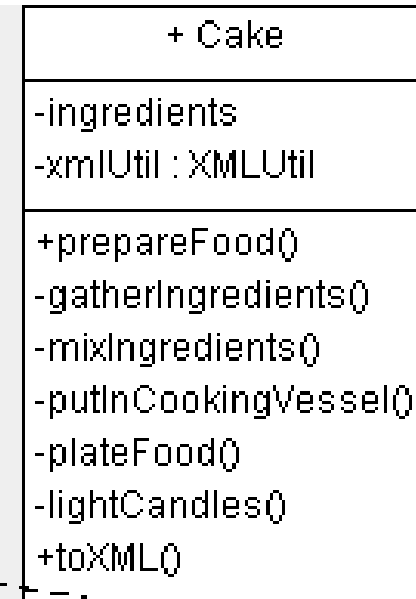
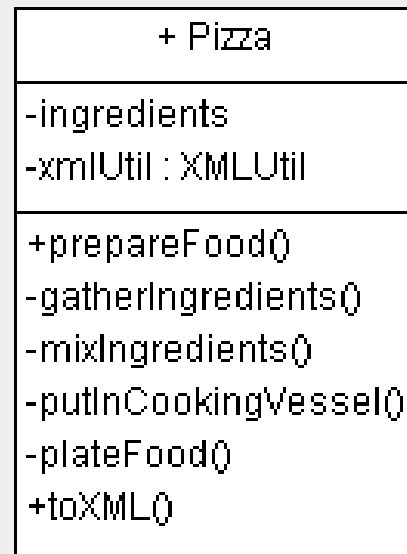




# Template method cont.

- Hook methods do not have to be abstract
- Generic class can define a default implementation thus allowing the subclasses to implement their own version

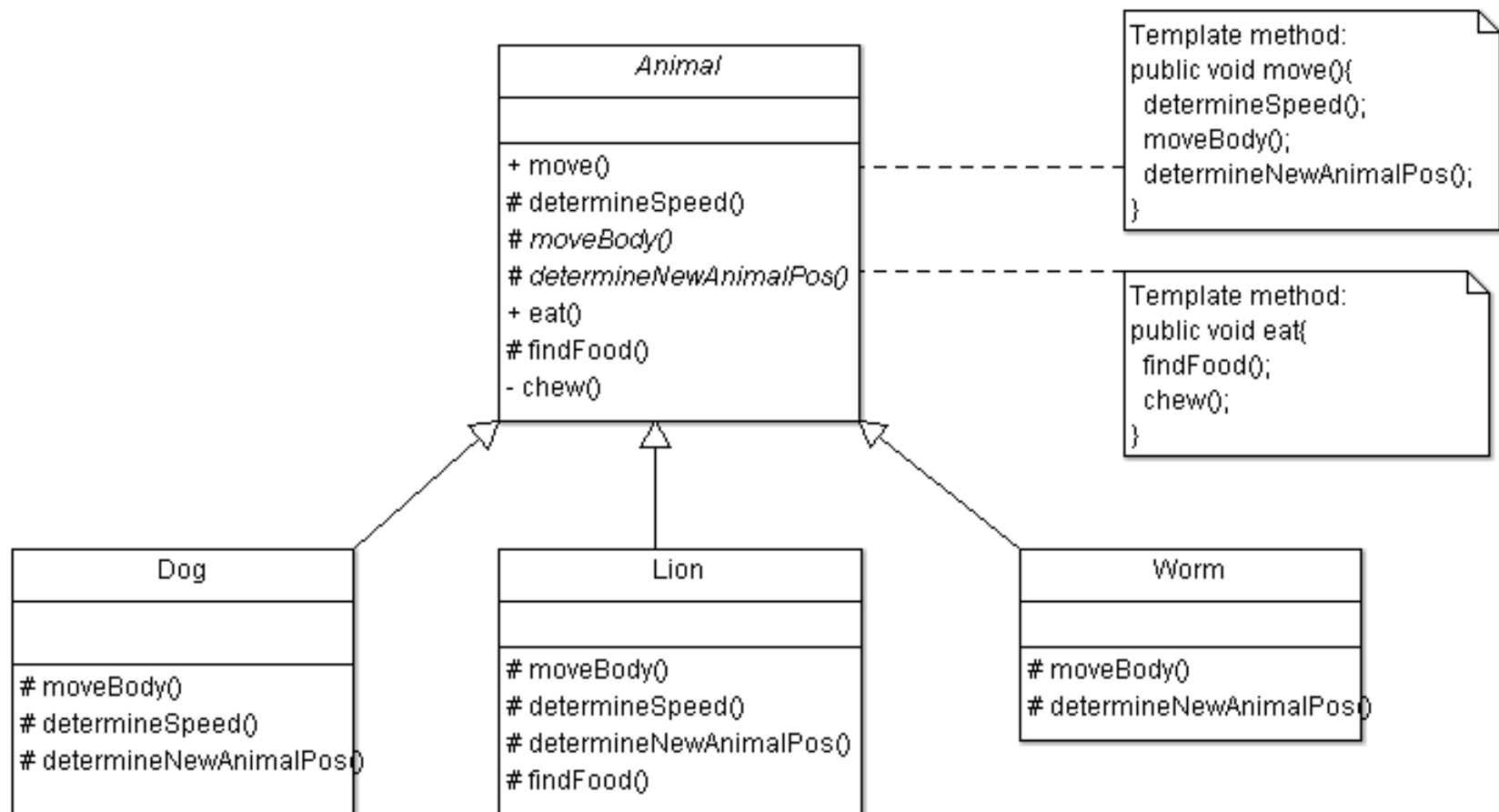
# Pizza, Cake, and Food revisited



gatherIngredients();  
mixIngredients();  
putInCookingVessel();  
cook(); //different implementation details  
plateFood();

gatherIngredients();  
mixIngredients();  
putInCookingVessel();  
cook(); //different implementation details  
plateFood();  
lightCandles();

# Template pattern example



# Group work - Template pattern

Draw UML including methods, be able to explain methods visibility and placement

- Web browsers – IE, Firefox, Chrome

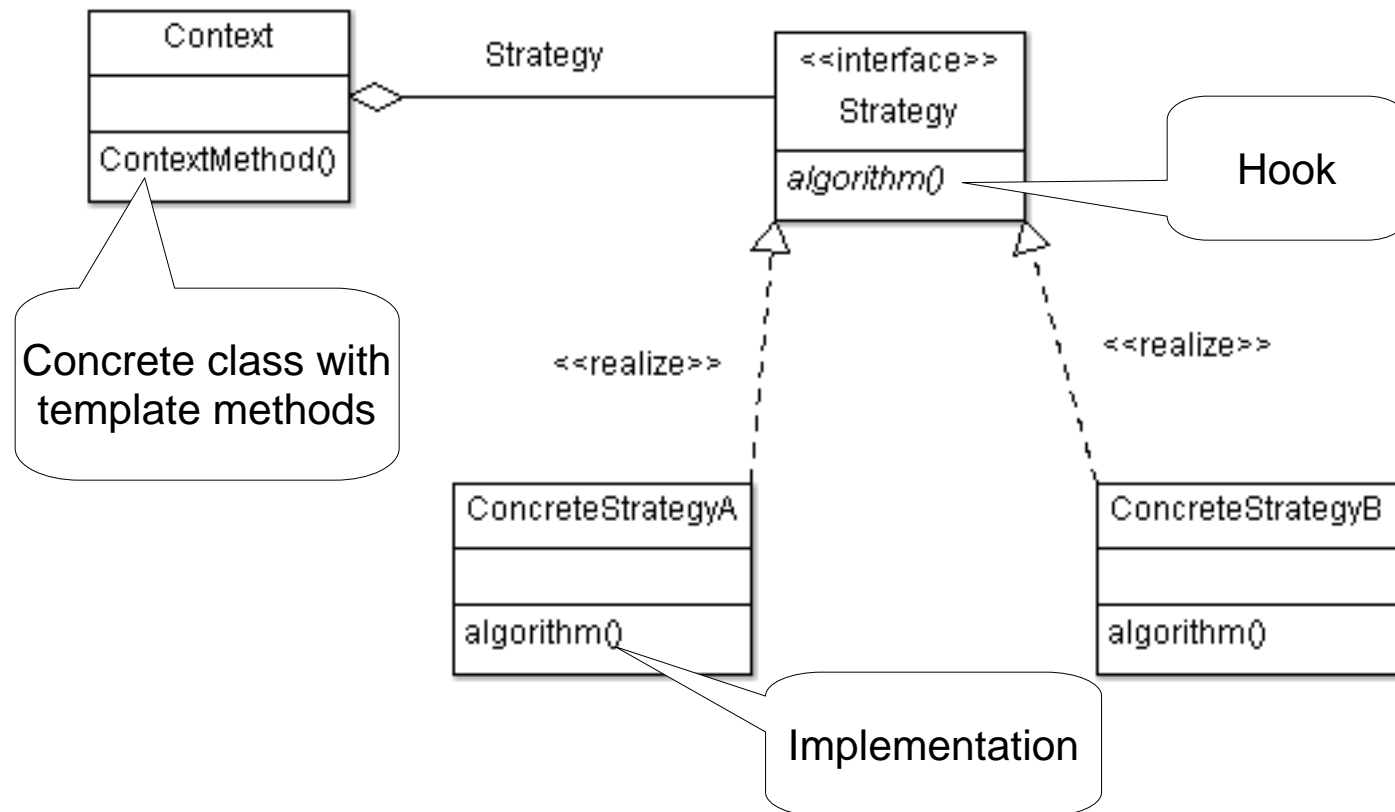
Functionality to support:

- **Visit a page** involves retrieving a page, parsing HTML, accepting cookies, rendering page
- **Back a page**
- Browser differences (hypothetical)
  - Chrome checks cache on visiting a page
  - Parsing common to Firefox and Chrome
  - Accepting cookies different on all browsers
  - Rendering page different

# Design pattern: Strategy

- Participants:
  - **Strategy** - which declares an interface common to all supported algorithms
  - **ConcreteStrategy** - implements the Strategy interface
  - **Context** – maintains references to one or more Strategy objects

# Strategy UML



# Strategy pattern cont.

- Similar to the Template pattern except hook and template method reside in different classes
  - Strategy interface has hook methods
  - Context class has template methods that call the hook methods

# Design pattern: Strategy pattern

- **Category** – Behavior design pattern
- **Intent** – Define a family of algorithms, encapsulate each one, and make them interchangeable



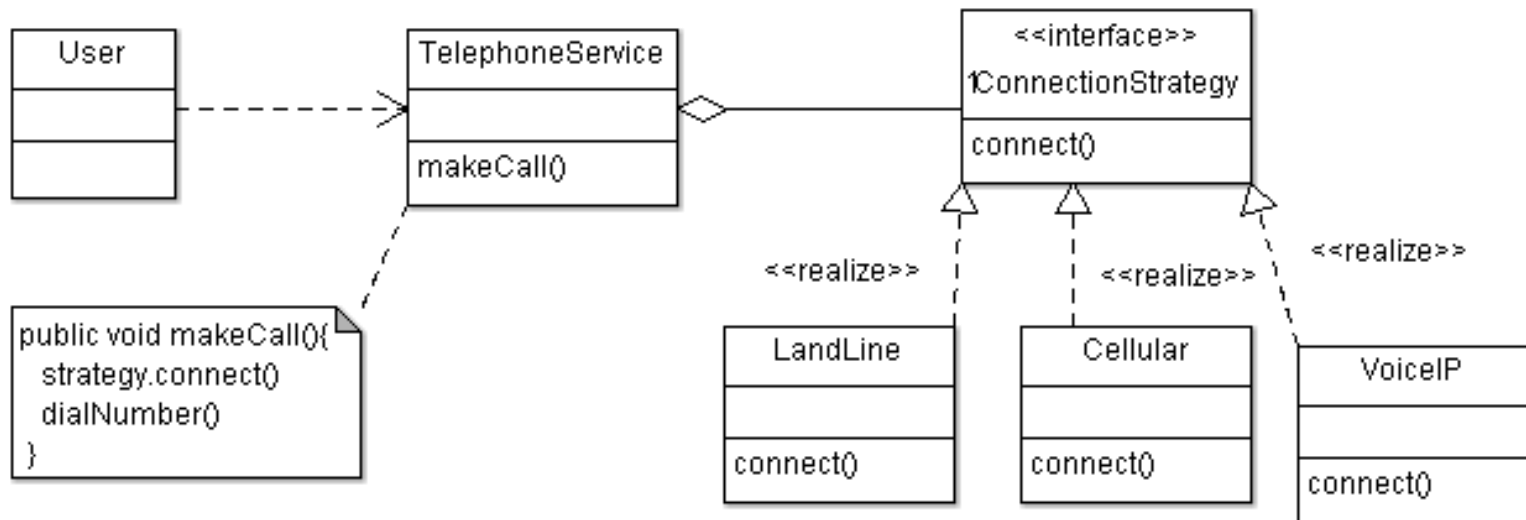
# Design pattern: Strategy cont.

- **Applicability** – pattern should be used when:
  - Many related classes that differ only in their behavior
  - Different variants of an algorithm needed
  - The algorithm uses data that clients should not know about
  - Form of generalization when a class defines many behaviors in multiple conditional statements and methods

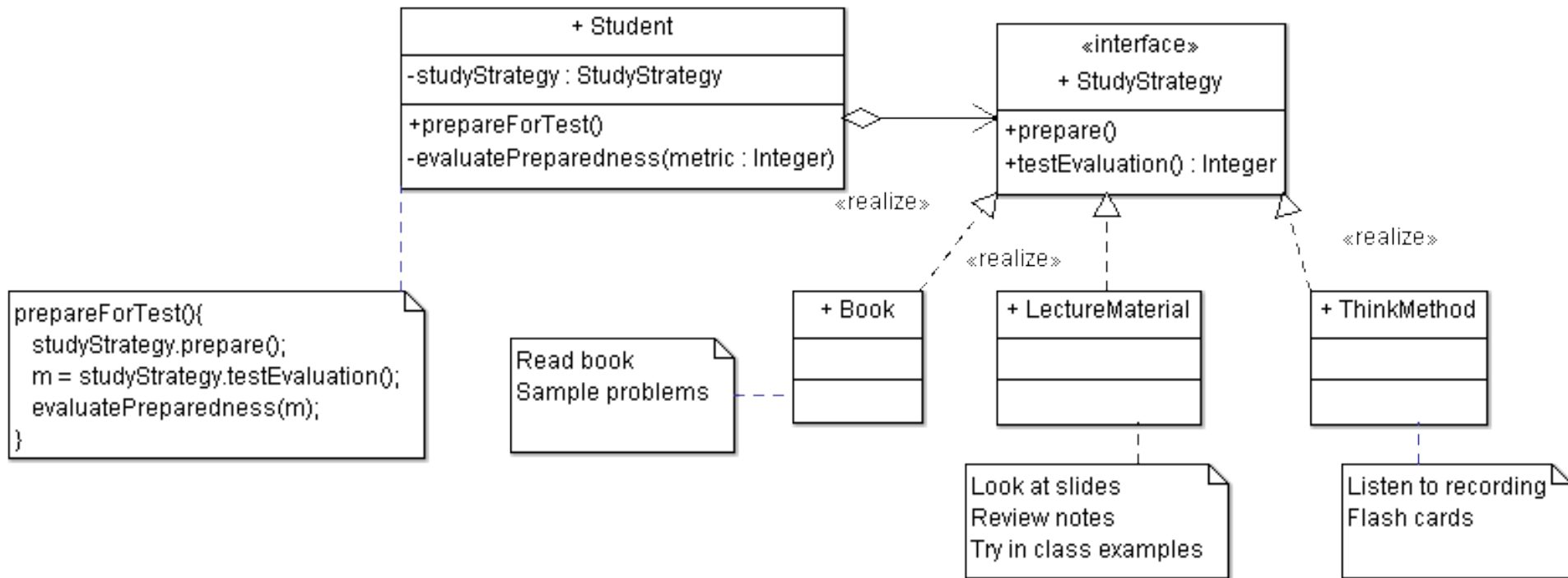
# Abstract coupling

- *Abstract coupling* refers to the way clients couple with their service providers
- With abstract coupling a client accesses services through an interface or abstract class without knowing concrete class that provides service
- Enhances extensibility and reuseability

# Example telephone



# Study strategy



# Group work - Strategy

Draw the UML including methods for a traveler application

- The application has a traveler that chooses a destination and a routing service decides how the person gets from point A to point B
  - Walk
  - Drive
  - Bus
- The steps of travel are “go to transportation”(access), wait for transportation, travel, “exit and travel to end point”(egress)