Template Method
Template Method Intent

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

(Behavioral)
What are the characteristics of a “template”?
The **Template Method** pattern defines the structure or steps of an algorithm.

**PartyPlanner**
- `planParty()`
- `selectDate()`
- `sendInvites()`
- `pickActivities()`
- `selectMenu()`
- `buyRefreshments()`
- `buyFood()`
- `cleanHouse()`
- `partyHard()`

**Let's plan a party!**

**WeddingPartyPlanner**
- `selectDate()`
- `sendInvites()`
- `pickActivities()`
- `selectMenu()`
- `buyRefreshments()`
- `buyFood()`
- `cleanHouse()`
- `partyHard()`

**FratPartyPlanner**
- `selectDate()`
- `sendInvites()`
- `pickActivities()`
- `selectMenu()`
- `buyRefreshments()`
- `buyFood()`
- `cleanHouse()`
- `partyHard()`

**WinterBallPlanner**
- `selectDate()`
- `sendInvites()`
- `pickActivities()`
- `selectMenu()`
- `buyRefreshments()`
- `buyFood()`
- `cleanHouse()`
- `partyHard()`
Template Method has one of the simplest structures of all the classic GoF patterns.

- **AbstractClass** always implements the TemplateMethod().
  - *It defines the algorithm structure*
  - *ConcreteClass'es should not be able to override it*

- **Subclasses usually use functionality in the parent class**
  - *Here the parent class via calls defined in TemplateMethod() uses functionality implemented in the ConcreteClass*
  - *Hollywood Principle – don't call us, we'll call you*
  - *Some primitive operations could have default or non-overridable functionality in AbstractClass*
It is interesting to compare the Template Method pattern with the Strategy pattern.

- What are connections between Strategy and Template Method?
- How are Strategy and Template Method similar? Different?