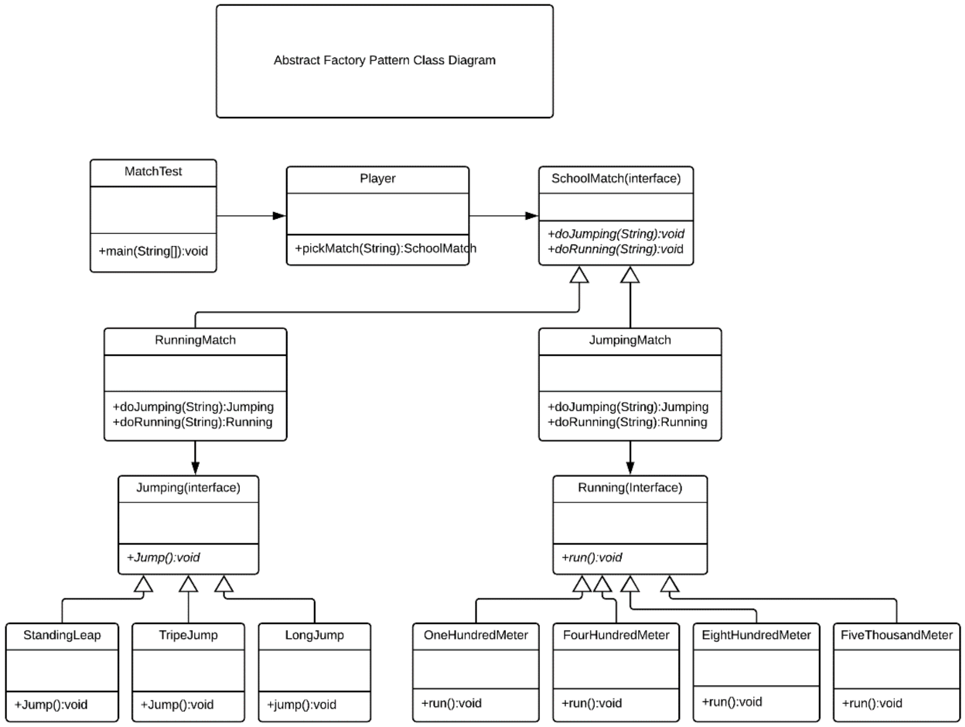
# Design Pattern CATALOG

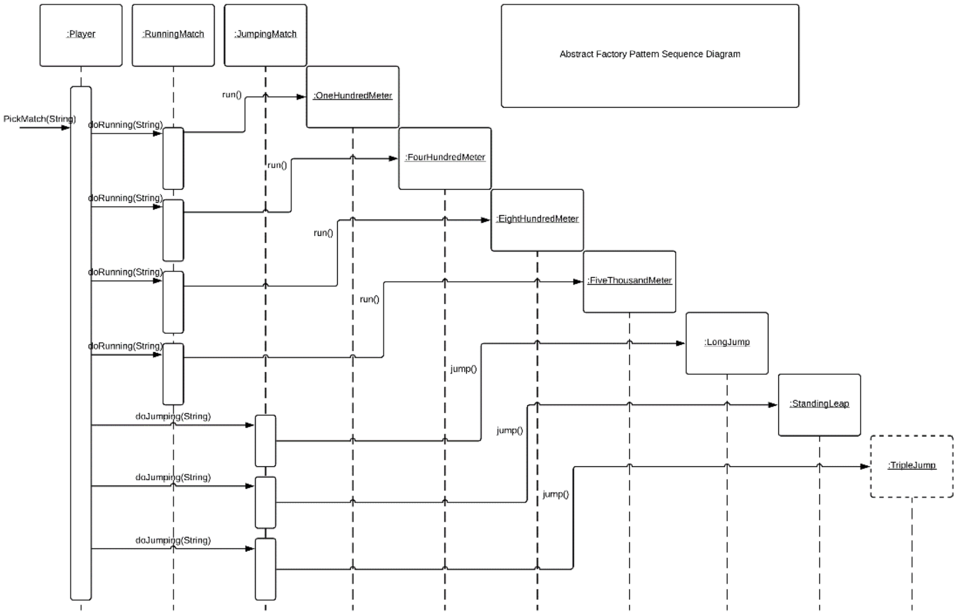
Kaijun he

# Contents

1. Abstract Factory Pattern
   a. Class Diagram



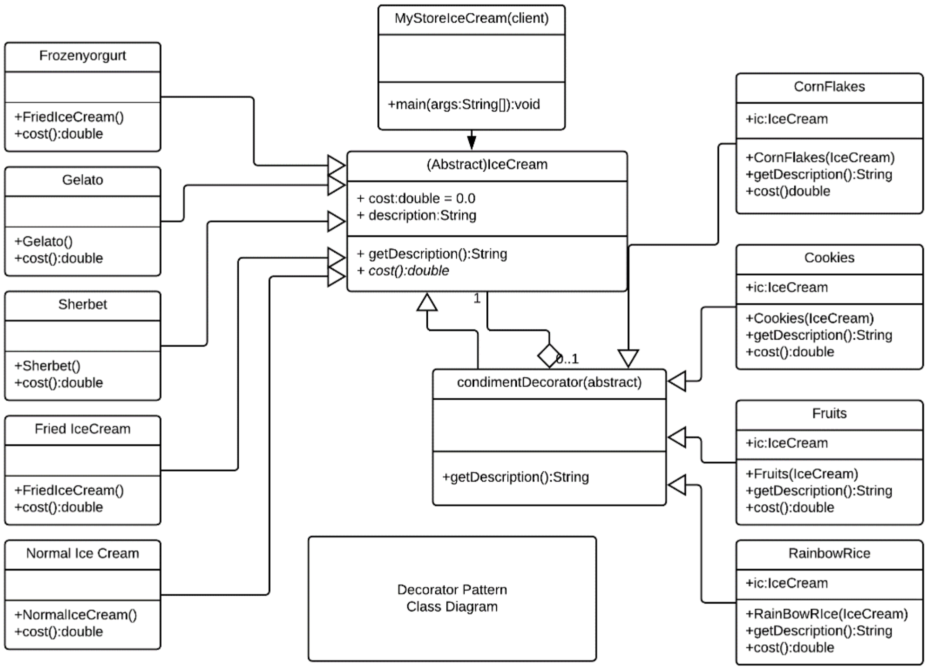
   b. Sequence Diagram



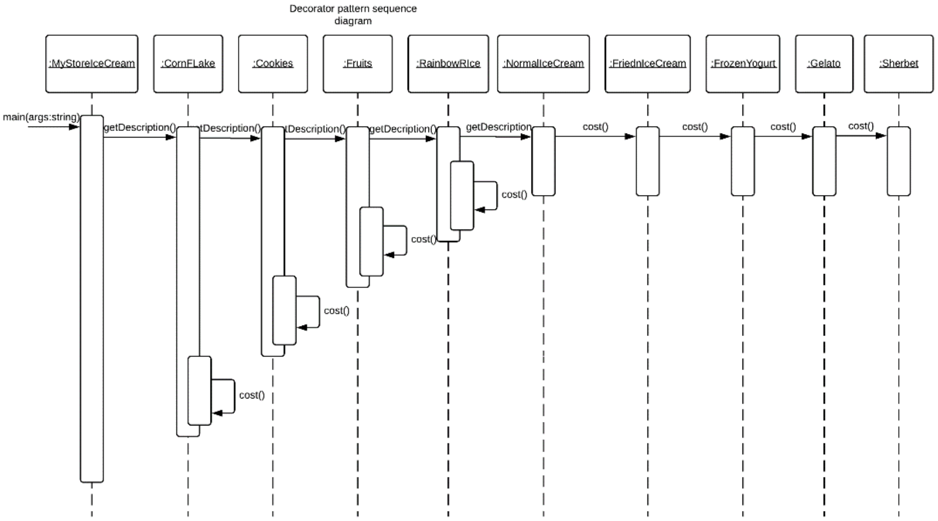
c. Implementation document:
   i. Problem: for this pattern, I try to use abstract factory pattern to solve a school match problem.
   ii. Abstract Factory: SchoolMatch interface
   iii. Abstract Product: Jumping and Running interface
   iv. Concrete Factory: RunningMatch and Jumping Match class in my project which extends SchoolMatch
   v. Concrete Product: StandingLeap, TripeJump,LongJump class implements Jumping interface, OneHundredMeter,FourHundredMeter,EightHundredMeter,FiveThousandMeter classes implements Running Interface.

d. Design Notes: for this abstract factory pattern, at beginning, I'm thinking about to pickup factory and product, and then create two interface since we need to design abstract factory and abstract product, and then add some concrete factories and products into it, after all design a main method to check whether all the products has been created by system out message.

2. Decorator Pattern
   a. Class Diagram



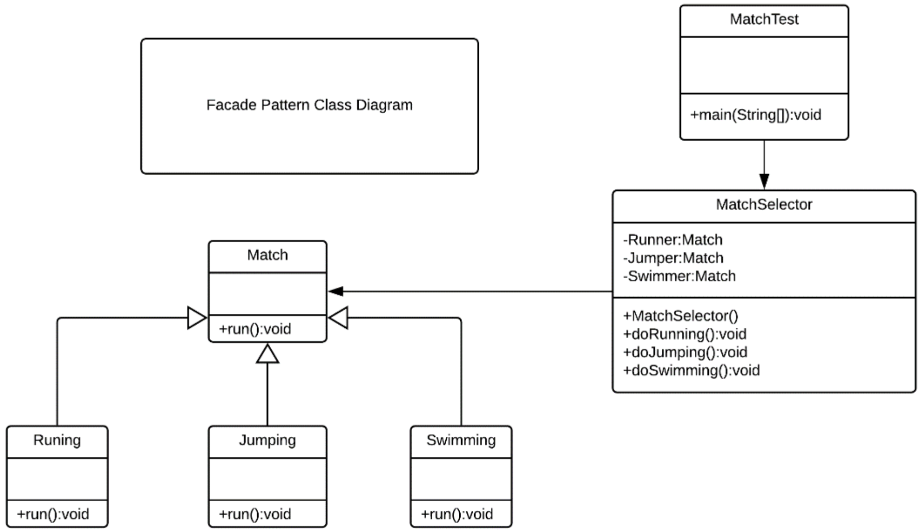
b. Sequence Diagram



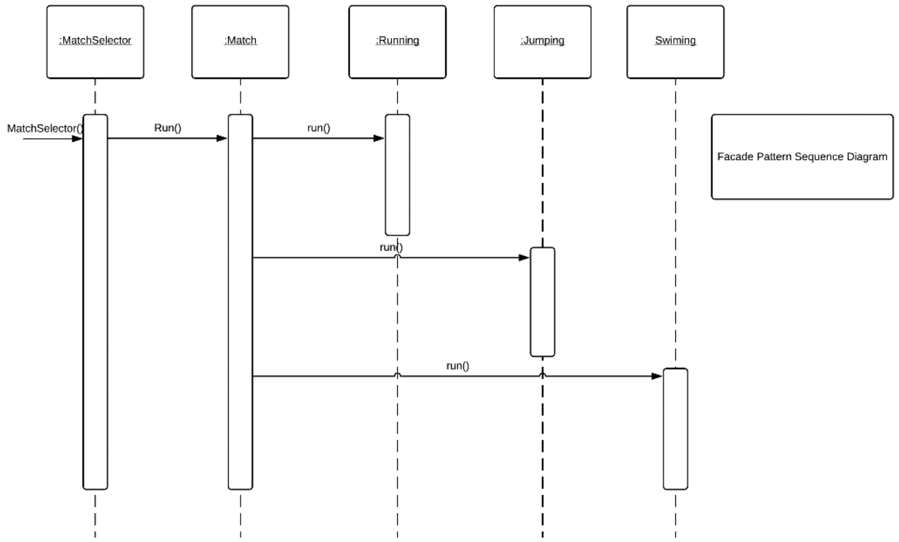
c. Implement document
   i. Component: Abstract class IceCream
   ii. Decorator: abstract class condimentDecorator class
   iii. Concrete component: NormalIceCream, FrozenYogurt,Gelato, Sherbet, FriedIceCream to extends to IceCream abstract class
   iv. Concrete Decorator: CornFlakes, Cookies, Fruits, RainbowRice to implements condimentDecorator abstract class

d. Design Notes: by following the class diagram given in textbook, I'm thinking about what I'm interested topic can be designed as decorator pattern, then I release ice cream will be a good topic, since Ice Cream has lots of types, and then I can add lots of accessories into different ice creams therefore the price will be based on the price of ice cream and addons, users could add more than one item into ice cream, to check my pattern is correct or not. In my main class, I do ice cream to add more than one item like cookies and cornflakes together, and then check the cost total, whether it's matched or not.

3. Façade Pattern
   a. Class Diagram:



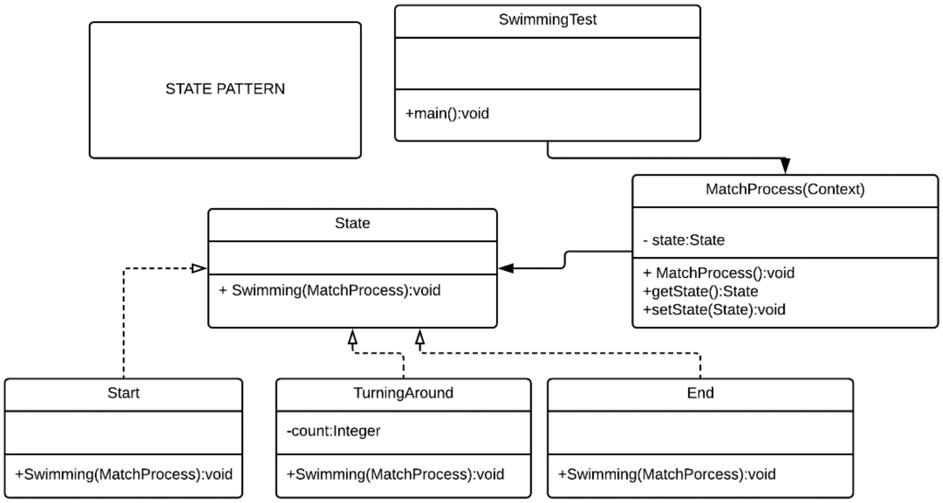
b. Sequence Diagram:



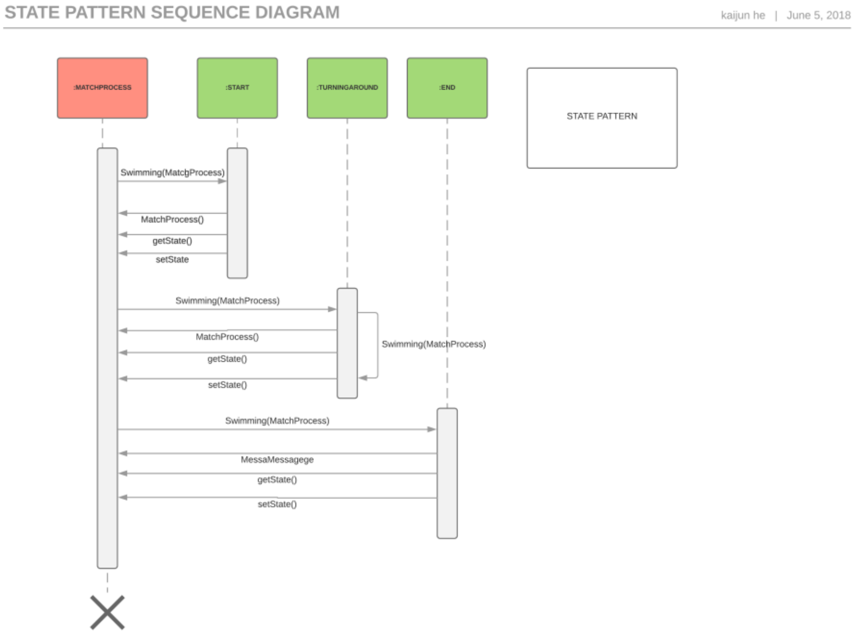
c. Implement documents:
   i. High level interface: Match interface in my project
   ii. Subsystem class: Running, Jumping, Swimming implements Match class
   iii. Façade class: MatchSelector to call other classes.

d. Design Notes: this is a clear pattern. First I design an interface Match and create concrete classes have same methods like Match interface. And then now I need create a façade class which can call those concretes classes, the main part is the façade class, in façade class run the methods of concretes classes.

4. State Pattern
   a. Class Diagram



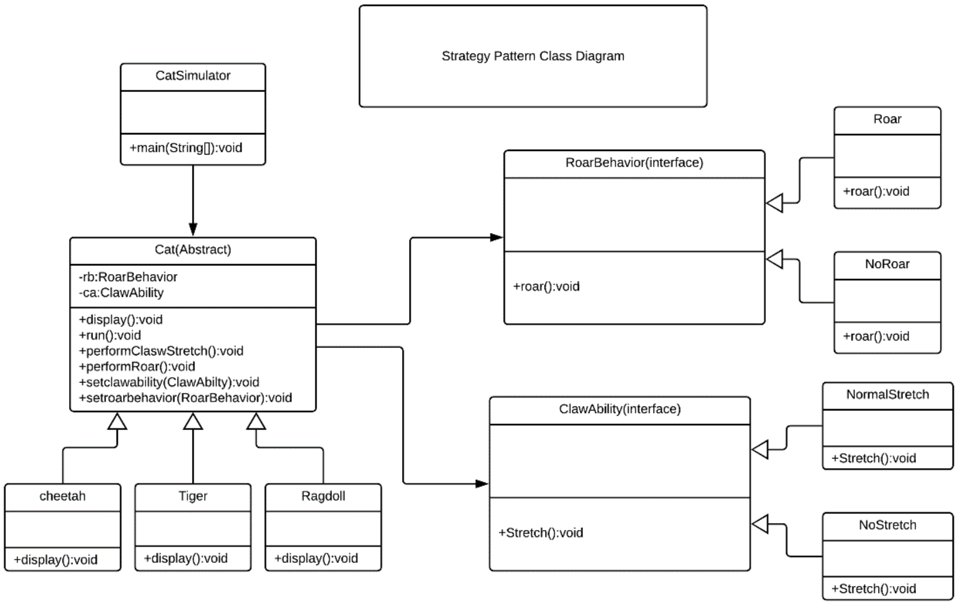
b. Sequence Diagram



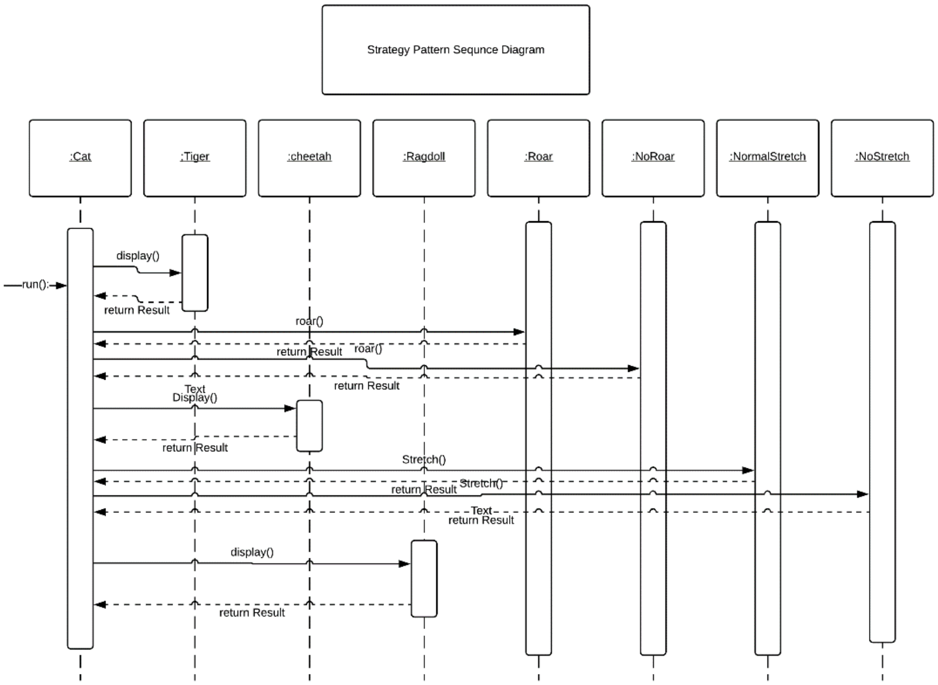
c. Implement document
   i. Context: MatchProcess class to hold and set state
   ii. State: state interface to encapsulate swimming behaviors for the Match Process context
   iii. Concreate State subclasses: Start state, Turnning Around states, End state, which implements the State interface

d. Design Notes: this pattern is used to define some project which have states, for my project 400 meter swimming match, first it will have start state, which is starting of the match , then for 50 meter length pool, when swimmer reach the wall they have to turn around body, so I call this as turning around point, by calculation, swimmer will have to turn 6 times except start and end state.

5. Strategy Pattern
   a. Class Diagram



   b. Sequence Diagram



   c. Implement document
      i. Abstract Strategy: RoarBehavior and ClawAbility interface
      ii. Abstract Context: Cat abstract class, which perform as a cat species.

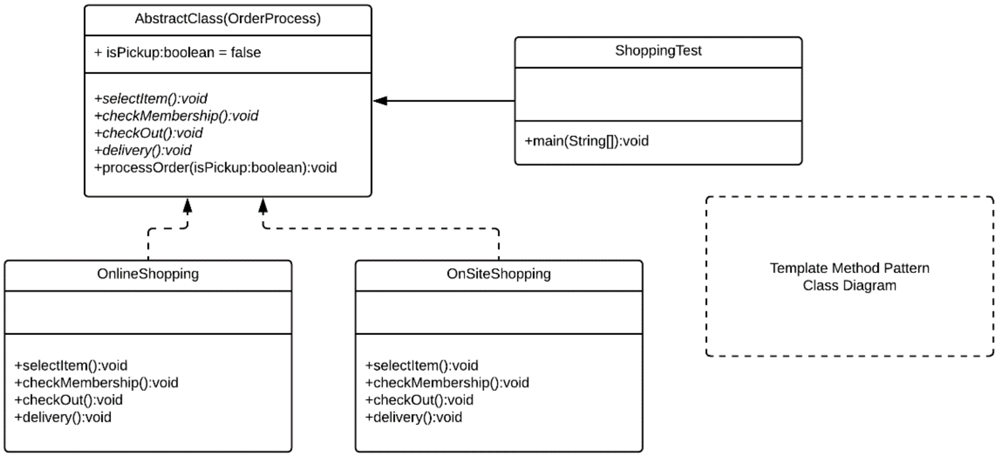
iii. Concreate strategy: Roar and NoRoar class implements RoarBehavior interface. NormalStretch and NoStretch class implements ClawAbility interface

iv. Concreate context: Cheetah, tiger, ragdoll extends Cat abstract class

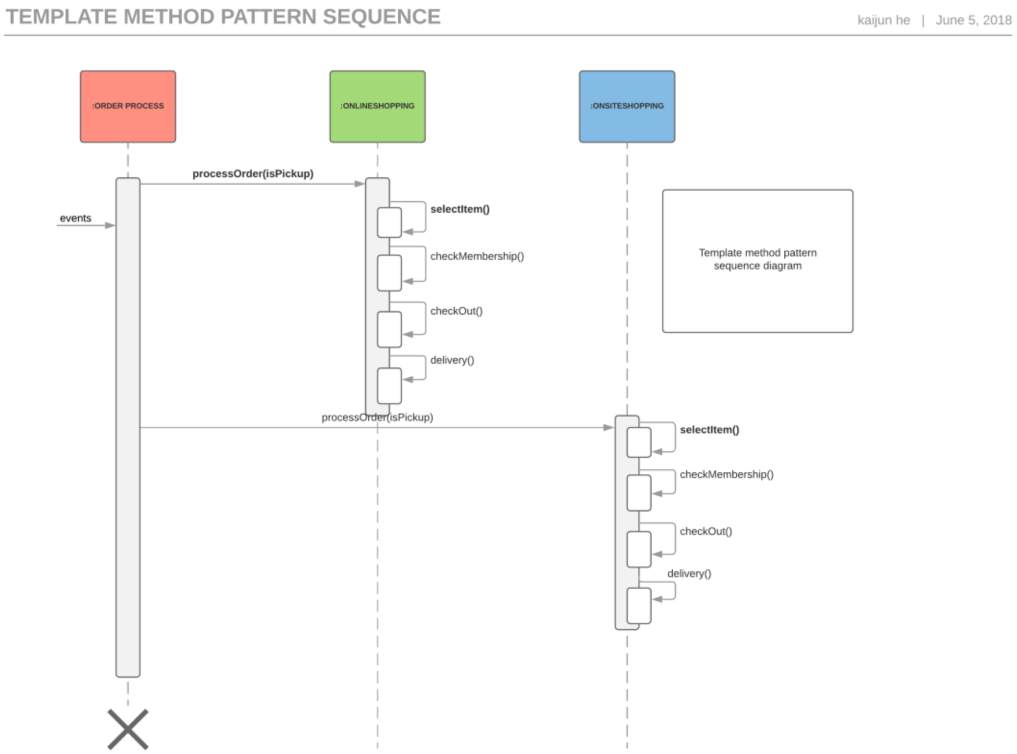
d. Design Notes: for this pattern, I remember a card game I played before which it says only 4 species cat can roar, then I do some research online, find out that cheetah is only one species cat can't stretch their claw, so I pick up tiger which can roar and stretch, ragdoll which is a kind of small cat that can purr and stretch, and last I pick cheetah which only can purr and can't normally stretch.

6. Template Method Pattern

a. Class Diagram:



b. Sequence Diagram



c. Implement documents:
   i. Abstract class: OrderProcess abstract
   ii. Concrete Class: Online Shopping and OnSiteShopping extends the abstract class OrderProcess.

d. Design Notes: this pattern at beginning I think it’s pretty simple, but I’m a little confused when I could use this pattern, suddenly I find out that this template method pattern is used to demonstrate same steps for different target. In my design, I use shopping experience in Costco as my topic, so the process is slectItem(), checkMembership(), checkout() and delivery(), absolutely that the process for onlineShopping and OnsiteShopping are not quite same, but the procedure can be demonstrate as same steps. And in online shopping it can make client to decide to pickup or delivery, so in my abstract class OrderProcess I put this condition. And in my main method text 3 condition, which is online shopping pickup in store, online shopping with shipping, and on site shopping.