Software Design

Software Engineering
Prof. Maged Elaasar

Learning Objectives

- Principles of software design
- The Gang of Four (GoF) design patterns

Blackjack Requirements

- Card class
  - Rank: A, 2-10, J, Q, K
  - Suit: ♣, ♦, ♥, and ♠
  - Points: 1-11

- The points
  - Ranks **2-10** : points equal to the rank
  - Ranks **J**, **Q**, and **K** : 10 points
  - **Ace** : 11 or 1 points
    - Soft total of Ace-7 : 18 points
    - Hard total of Ace-7 : 8 points

- Shoe vs Deck
  - Desk holds 52 cards
  - Shoe holds multiple decks

BlackJack Implementation

```
public class Card {
   public int rank;
   public String suit;
   public Card(int rank, String suit) {...}
}
```

```
public class Deck {
   public Stack<Card> cards = new Stack();
   public Deck() {
      for (int i : {1, 14})
         for (String j : {♣, ♦, ♥, ♠})
            cards.add(new Card(i, j));
      Collections.shuffle(cards);
   }
   public Card deal() {
      return cards.pop();
   }
   public int[] points(Card card) {
      int rank = card.rank;
      if (rank == 1) return {1, 11};
      else if (rank>2 && rank < 11) return {rank, rank};
      else return {10, 10};
   }
}
```

Not substitutable

Missing Interfaces
(Iterable)

```
public class Shoe {
   public Stack<Card> allCards = new Stack();
   public Shoe() {
      for (int i : n)
            allCards.addAll(new Deck().cards);
      Collections.shuffle(allCards);
   }
   public Card deal() {
      return allCards.pop();
   }
   public void shuffle_burn(n) {
      Collections.shuffle(allCards);
      for (int i: n)
         allCards.remove();
   }
}
```

Mixed responsibilities
(card and points)

Missing responsibilities
(points for a hand)

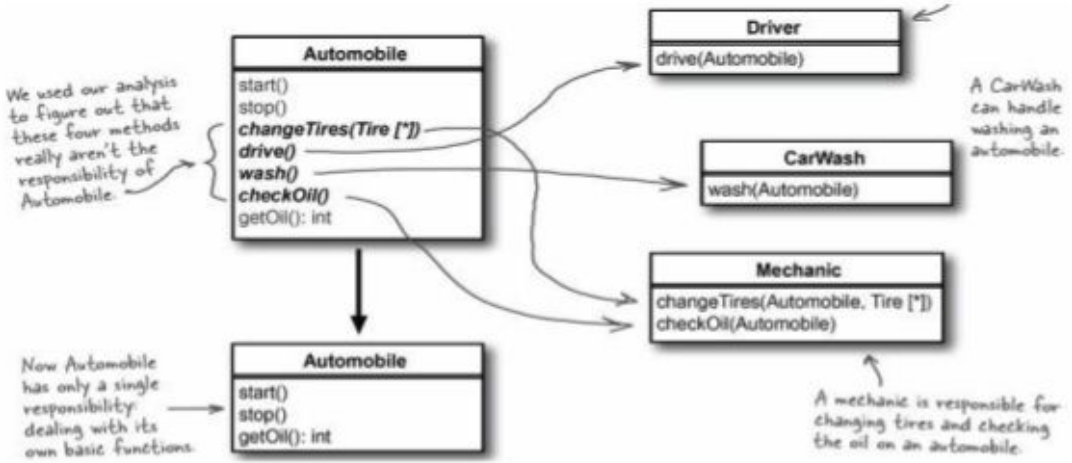
Limited Reuse Potential
(points specific to Blackjack)

SOLID Principles

- **S**ingle Responsibility Principle
- **O**pen-Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

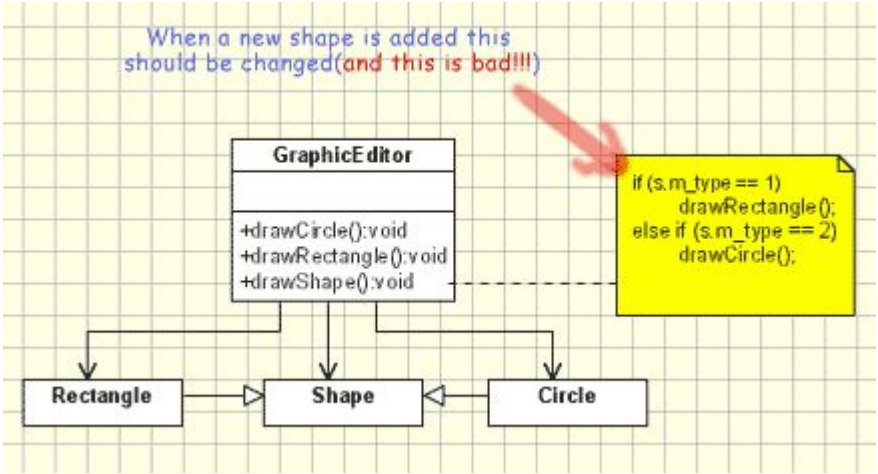
Single Responsibility Principle

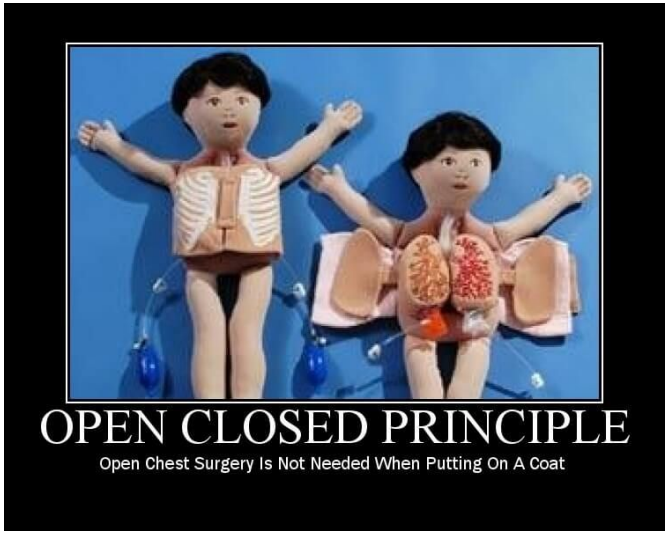
- A class should have responsibility over a single part of the functionality
  - That responsibility should be entirely encapsulated by the class.



Open-Closed Principle

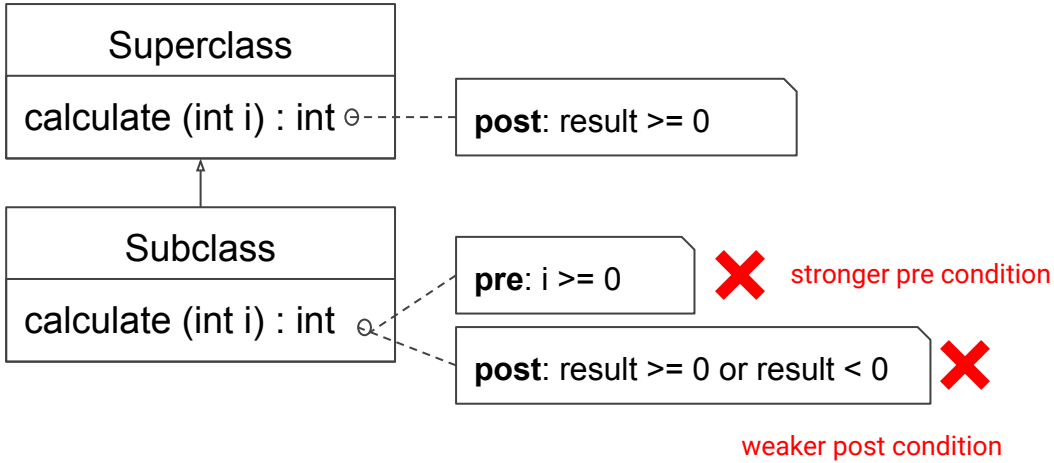
- Entities should be open for extension but closed for modification
  - Classes should be extensible without changing implementation





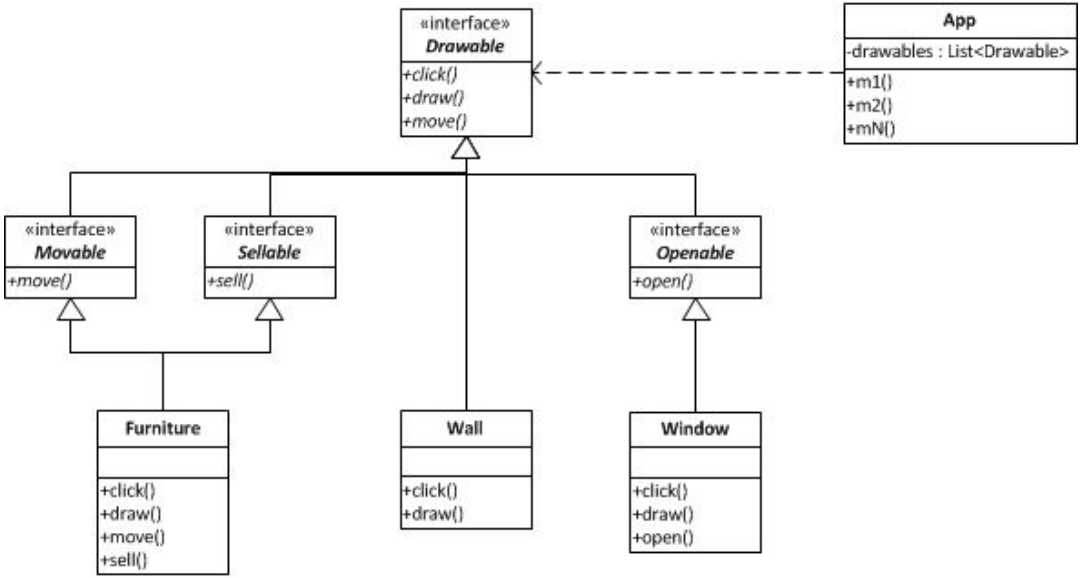
Liskov Substitution Principle

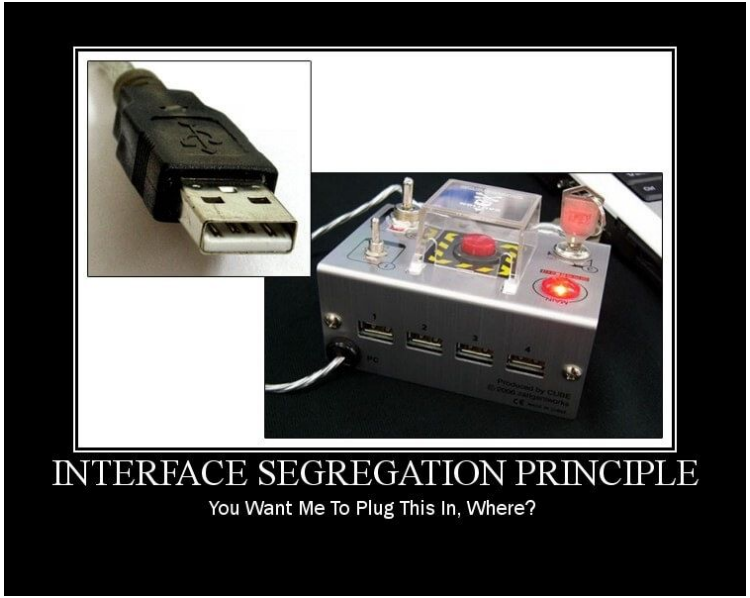
- Subtypes should be substitutable for their supertype without breaking clients
  - only increase visibility of features not decrease it
  - only weaken pre-conditions or strengthen post-conditions
  - only relax input parameter types and strengthen output parameter types
  - only remove but not add exceptions



Interface Segregation Principle

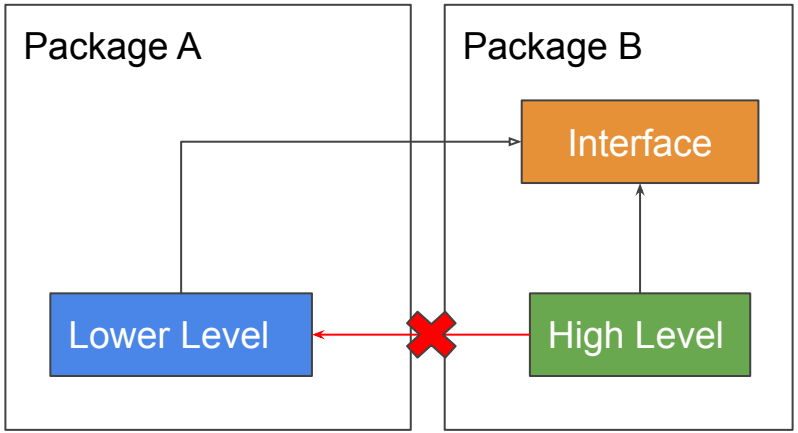
- Clients should not be forced to depend on interfaces they do not use

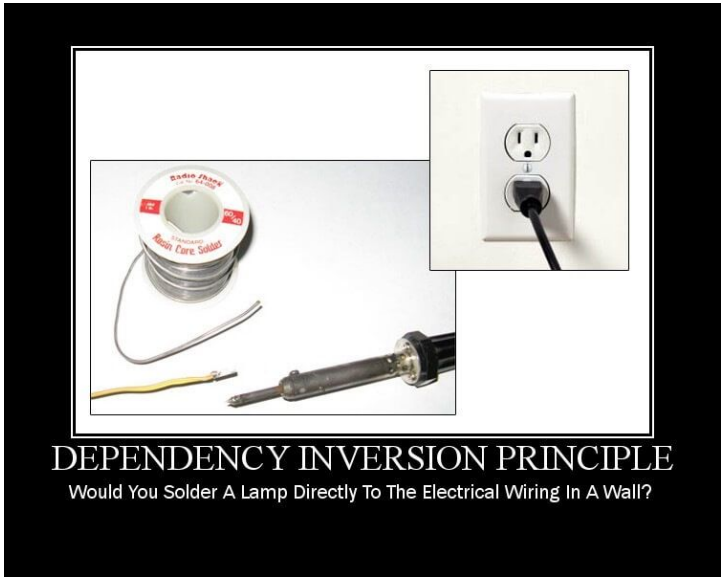




Dependency Inversion Principle

- High-level modules should not depend on low-level modules directly
  - Both should depend on an abstraction (interface)

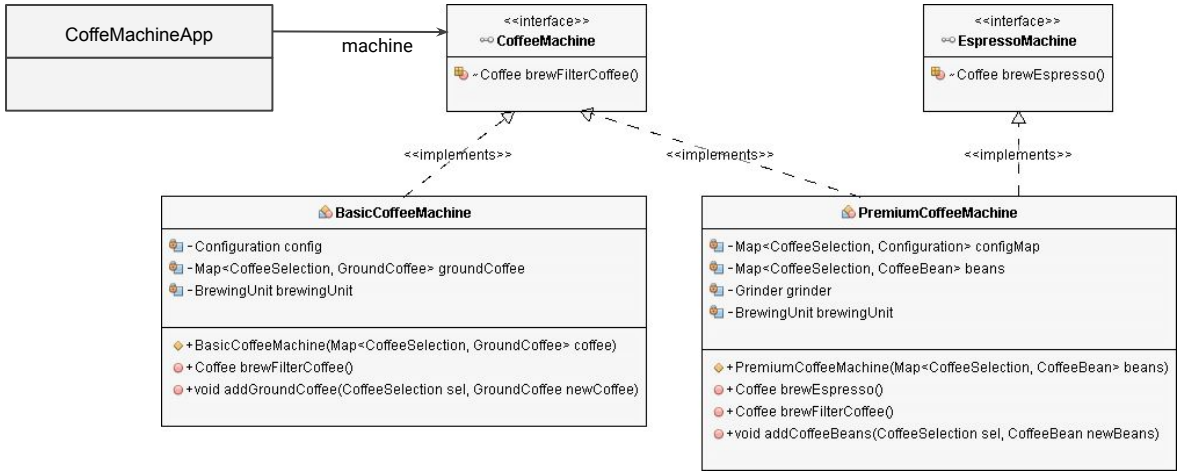




At home design exercise

The design below is for a coffee machine app that automatically brews a fresh cup of coffee in the morning. You can use lots of different coffee machines: rather simple ones that use water and ground coffee to brew filter coffee, and premium ones that include a grinder to freshly grind the required amount of coffee beans and which you can use to brew different kinds of coffee.

Point out how this design adheres to all the **SOLID** principles.



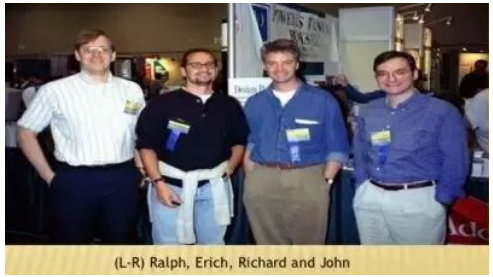
Design Patterns

- Common **solutions** to common design problems
- Common **vocabulary** for discussing system designs
- Reduce system complexity by naming **abstractions**
- Helps with **reorganization** or **refactoring** of class hierarchies
- Tried and tested methods to develop **flexible**, **maintainable** programs

Caveats of Design Patterns

- Design patterns are not a substitute for **thought**
- **Class names** and **directory structures** do not equal good design
- Design patterns have **tradeoffs**
- Design patterns are **realized differently** in each programming language

Gang of Four (GoF) Design Patterns


(L-R) Ralph, Erich, Richard and John

Creational Patterns

1. Abstract Factory
2. Builder
3. Factory Method
4. Prototype
5. Singleton

Structural Patterns

1. Adapter
2. Bridge
3. Composite
4. Decorator
5. Façade
6. Flyweight
7. Proxy

Behavioral Patterns

1. Chain of Responsibility
2. Command
3. Interpreter
4. Iterator
5. Mediator
6. Memento
7. Observer
8. State
9. Strategy
10. Template Method
11. Visitor

Software Design Quiz