Object Oriented Analysis & Design
面向对象分析与设计

Lecture_08 通用的职责分配软件原则 GRASP（二）

主讲: 姜宁康 博士

1、GRASP原则六: 多态 Polymorphism

- How to handle alternative behaviors based on type 如何处理依据类型不同而有不同行为的一类需求？
  - 比如，开餐馆
    - 苏州人喜欢甜、四川人喜欢麻、湖南人喜欢辣，咋处理？

1.1 9条GRASP原则

- Information Expert
  - responsibilities should be assigned to objects that contain relevant information
- Creator
  - the creator of an object is usually an object that contains, or aggregates it
- High Cohesion
  - responsibilities of a certain class must be highly related
- Low Coupling
  - interdependency between classes should remain low
- Controller
  - class which handles external system events
- **Polymorphism 多态原则**
- **Indirection 间接原则**
- **Pure Fabrication 纯虚构原则**
- **Protected Variations 隔离变化**

1.2 Iteration 2 More Requirement

- 第一次迭代结束时，完成了
  - 当前软件功能测试：单元测试、用户可接受测试、负载测试、可使用性测试等
  - 必须有客户加入，并给出反馈 Customers engaged in and feedback
  - 把基线稳定下来，发布内部版本 stabilized baseline internal release
- 第二次迭代时，要考虑加入新的功能
  - 需求变化、业务规则细化、考虑更多的用例
  - 制定本次迭代的计划活动等
  - Ex, Monopoly game
    - When a player lands on the Go square, the player receives $200
    - When a player lands on the Go-To-Jail square, they move to the Jail square
    - When a player lands on the Income-Tax square, the player pays the minimum of $200 or 10% of their worth

1.2 Iteration 2 More Requirement

- 例如POS系统，第二次迭代时增加另外的需求
  - 支持多种第三方服务的接口 Support for variations in third-party external services
    - 计算税费、信用卡授权认证等
  - 复杂的定价机制 Complex pricing rules
  - 可插拔的业务规则 Pluggable business rules
  - GUI窗口在信息发生变化时得到更新 GUI window updates when information changes
- 这些功能点，可能属于前轮迭代同样的用例，但更多的是讨论非功能性需求
- 这些需求对领域模型的影响较小
- 同样一项功能，原来一种处理方法就可以，现在需要适应多种处理方法，设计方案该如何支持?
  - 比如付费：现金、储蓄卡、信用卡、支付宝、微信

1.3 GRASP rule6: Polymorphism(多态)

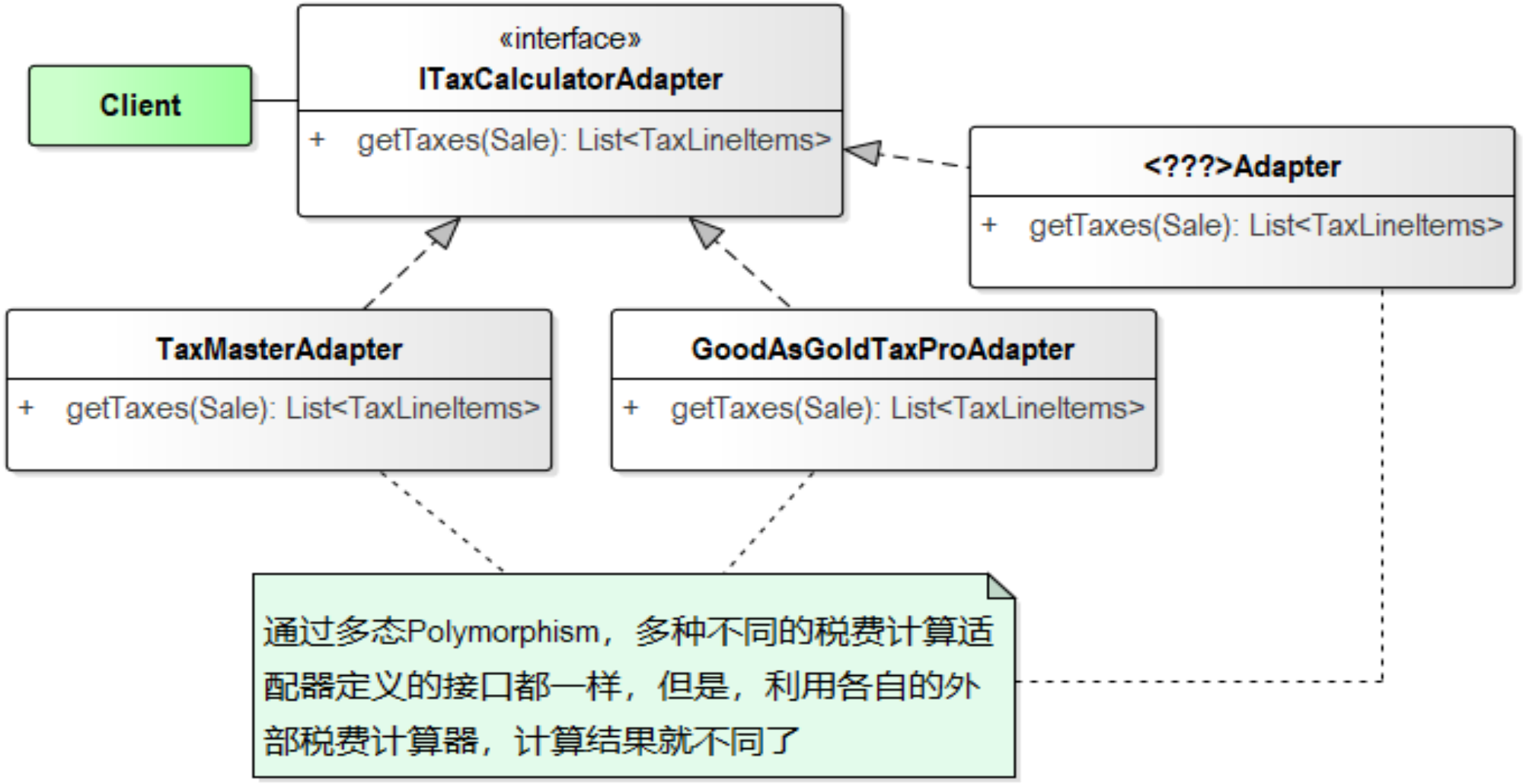
- **Name: Polymorphism(多态)**
- **Problem:**
  - 如何处理依据类型不同而有不同行为的一类需求？How to handle alternative behaviors based on type? How to create pluggable software components?
- **Solution:**
  - 使用多态操作为依据类型变化的行为 进行职责分配 When related alternatives or behaviors vary by type (class), assign responsibility for the behavior using polymorphic operations to the types for which the behavior varies
- **Corollary(推论):**
  - 不要去测试对象的类型或者条件逻辑，并以此选择相应的行为 Do not test for the type of an object and use conditional logic to perform varying alternatives based on type
  - 即，不要使用条件逻辑，而是为不同的类定义相同名字的方法 That is, don't use conditional logic, but assign the same name to services (methods) in different classes
  - 不同的类实现了相同的接口、或者有一个共同的父类（继承）The different classes usually implement a common interface or are related in an implementation hierarchy with a common superclass

1.4 多态案例1: 适应多种不同的税费计算器

问：哪个对象应当负责处理这些变化的外部税费计算器接口？ What objects should be responsible for handling these varying external tax calculator interfaces?

答: 多态机制可以适配不同的外部税费计算器（实现接口的方法）
Polymorphism in adapting to different external tax calculators

推论：很容易增加新的税费计算器，对现有的实现代码影响很小



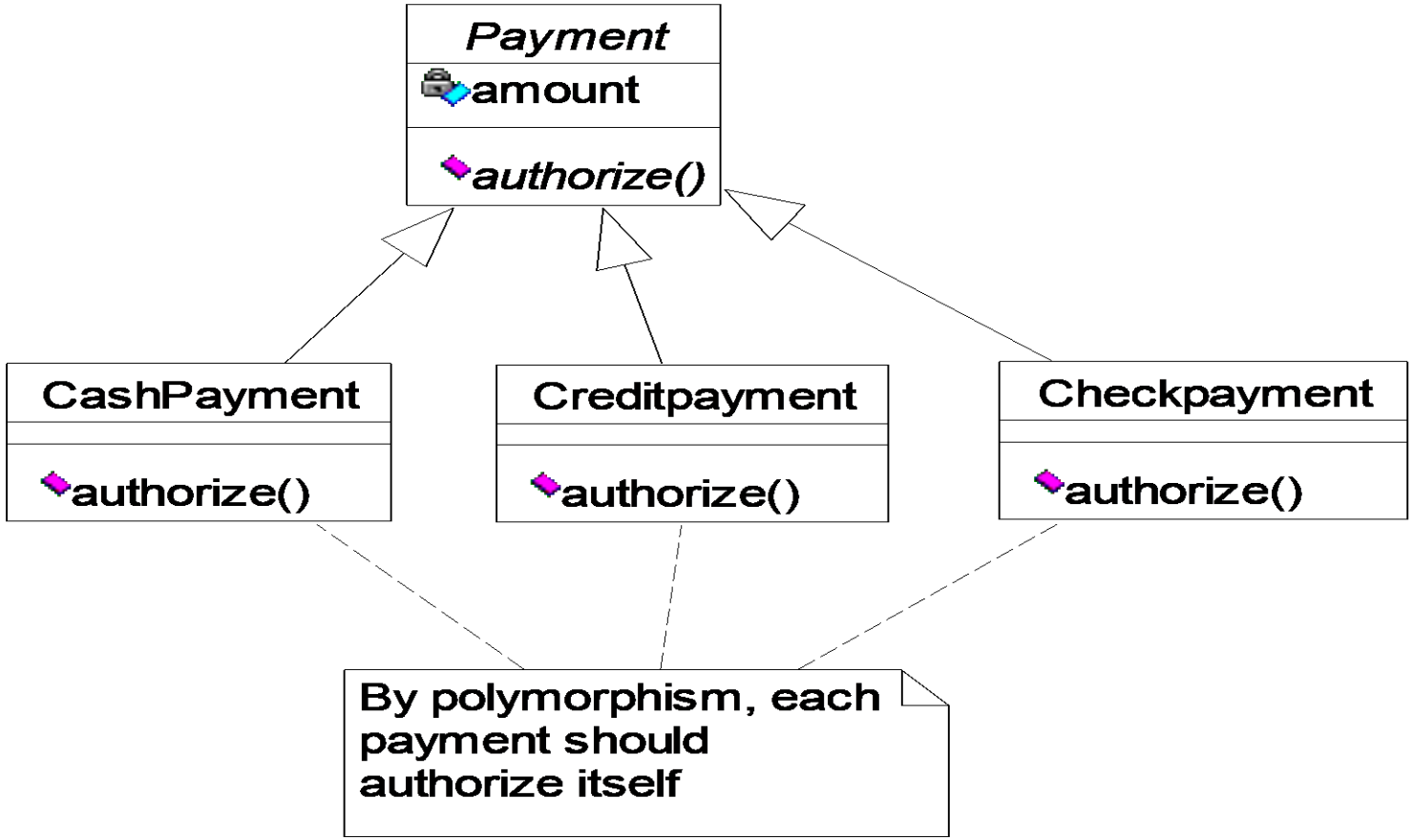
1.4 多态案例2: 不同的授权认证

在POS应用，哪个对象应当负责认证不同的支付方式? In the POS application, who should be responsible for authorising different kinds of payments?

现金 、信用卡 、支票 、支付宝 、微信

答: 多态机制可以适配不同的支付方式（继承的方法覆盖 override)

Polymorphism in adapting to different external tax calculators

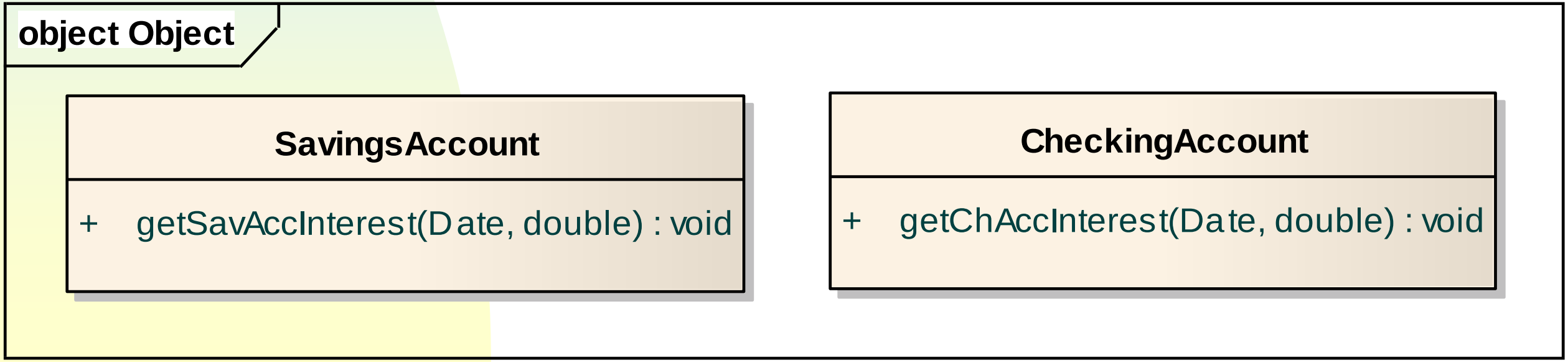


1.4 多态案例3

- 一个银行应用，有两种类型的账户: 支票账户 CheckingAccount 和储蓄账户 SavingsAccount
  - 计算一段时间每个账户的利息 Suppose that you want to evaluate for each account the interest accumulated over a period
  - 不同类别的账户，其利息计算方法是不同的，有各自的利率 The implementation of evaluating interest is different for each account type. It uses a different set of interest rates

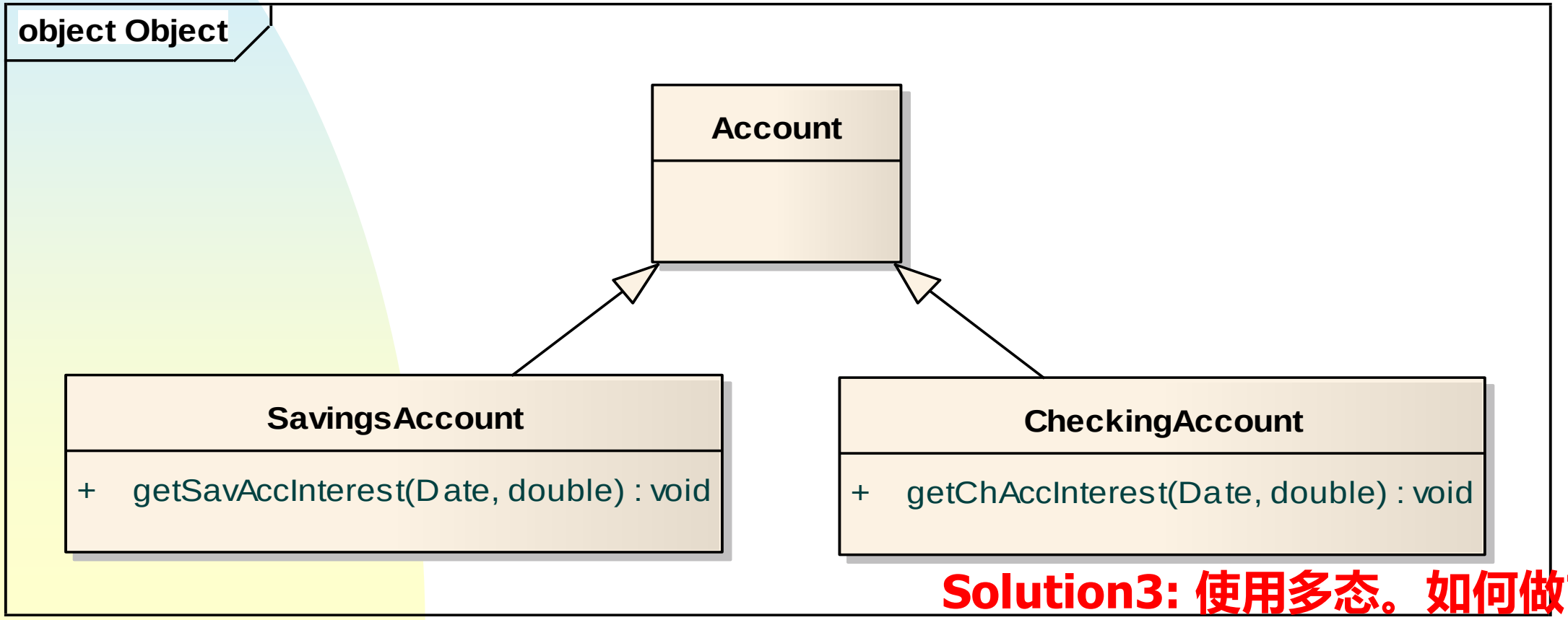
1.4 多态案例3

- **Solution 1: Class SavingsAccount with method getSavAccInterest(startDate, endDate), and class CheckingAccount with method getChAccInterest(startDate, endDate)**
  - **First go through objects of class SavingsAccount and invoke their method getSavAccInterest(…)**
  - **Then go through objects of class CheckingAccount and invoke their method getChAccInterest(…)**



1.4 多态案例3

- **Solution 2: Same as Solution 1, but SavingsAccount and CheckingAccount have the same superclass Account**
  - **Go through all objects of class Account. If an object is of subclass SavingsAccount, invoke getSavAccInterest(…), else invoke getChAccInterest(…)**



Solution3: 使用多态。如何做?

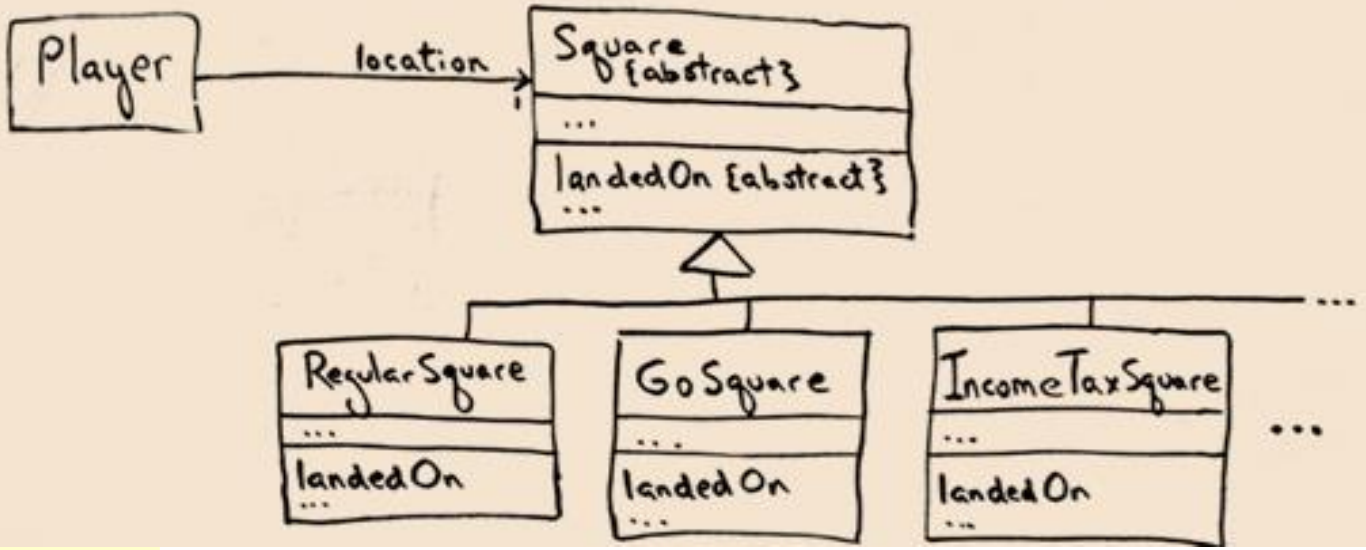
1.4 多态案例4

- **Monopoly Problem: How to Design for Different Square Actions?**

```
// bad design
SWITCH ON square.type
        CASE GoSquare: player receives $200
        CASE IncomeTaxSquare: player pays tax
...
```

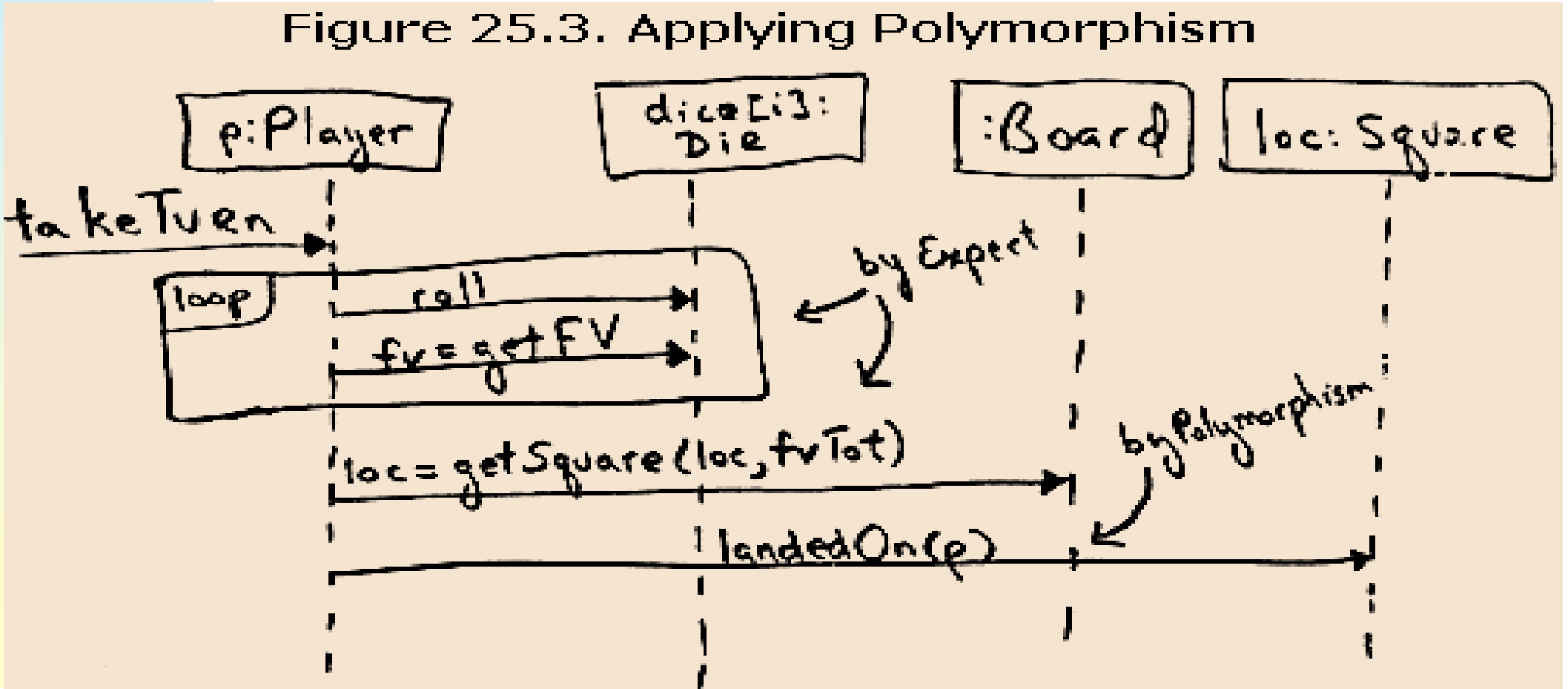
- **Solution（better design）: Applying Polymorphism rule**
  - **to create a polymorphic operation for each type for which the behavior varies**
  - **Landedon()**

Figure 25.2. Applying Polymorphism to the Monopoly problem



1.4 多态案例4

- 当Player登陆到Square上的时候，有哪个对象通知Square呢? What object should send the landedOn message to the square that a player lands on?
  - by the principles of Low Coupling and by Expert: Player



1.4 多态案例4

- consider each of the polymorphic cases
  - 1) GoSquare: increase $200

Figure 25.4. The *GoSquare* case.

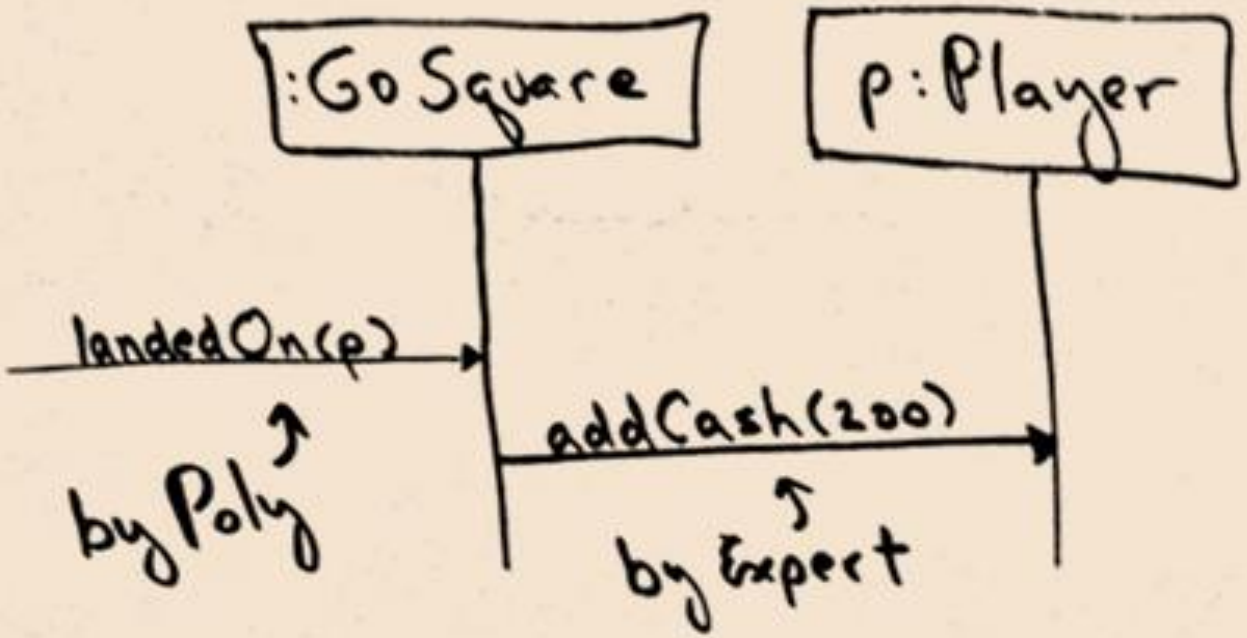
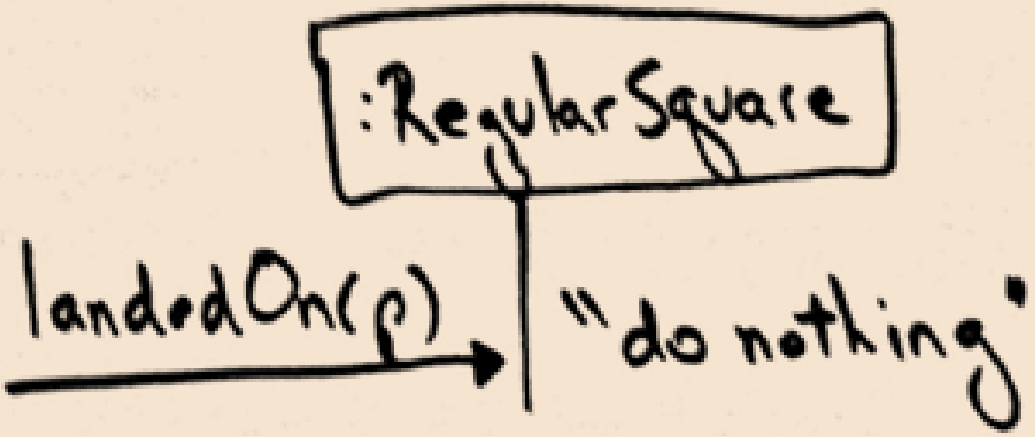


Figure 25.5. The *RegularSquare* case.

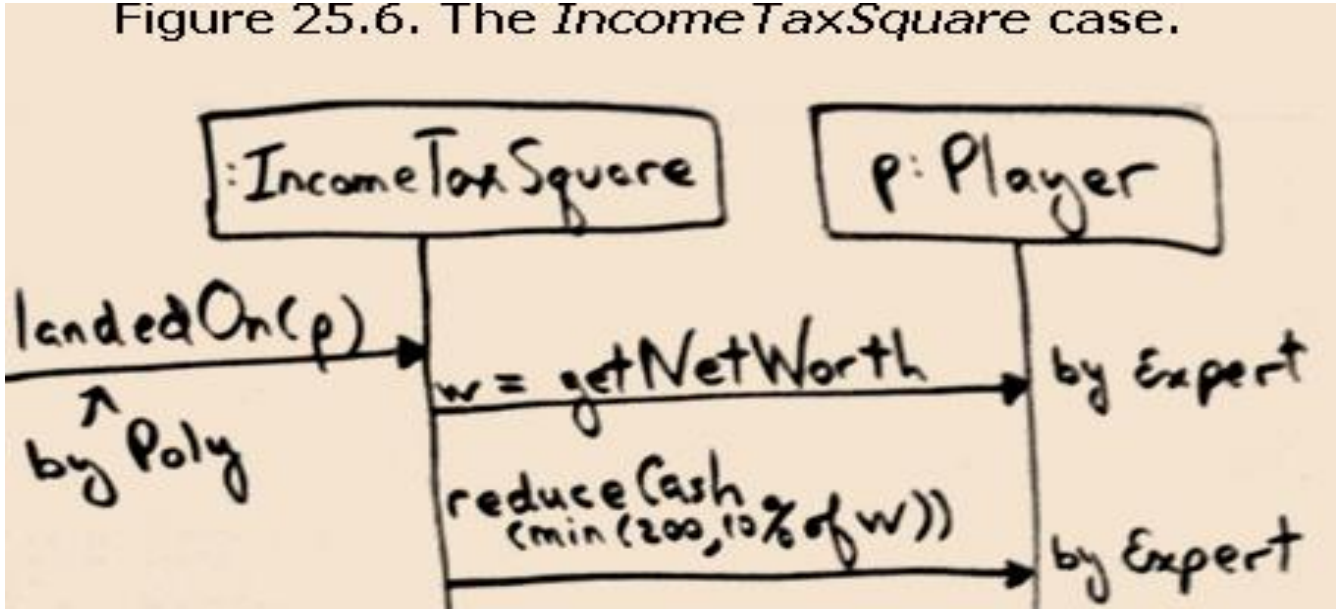


- 2) Regular square : do nothing

1.4 多态案例4

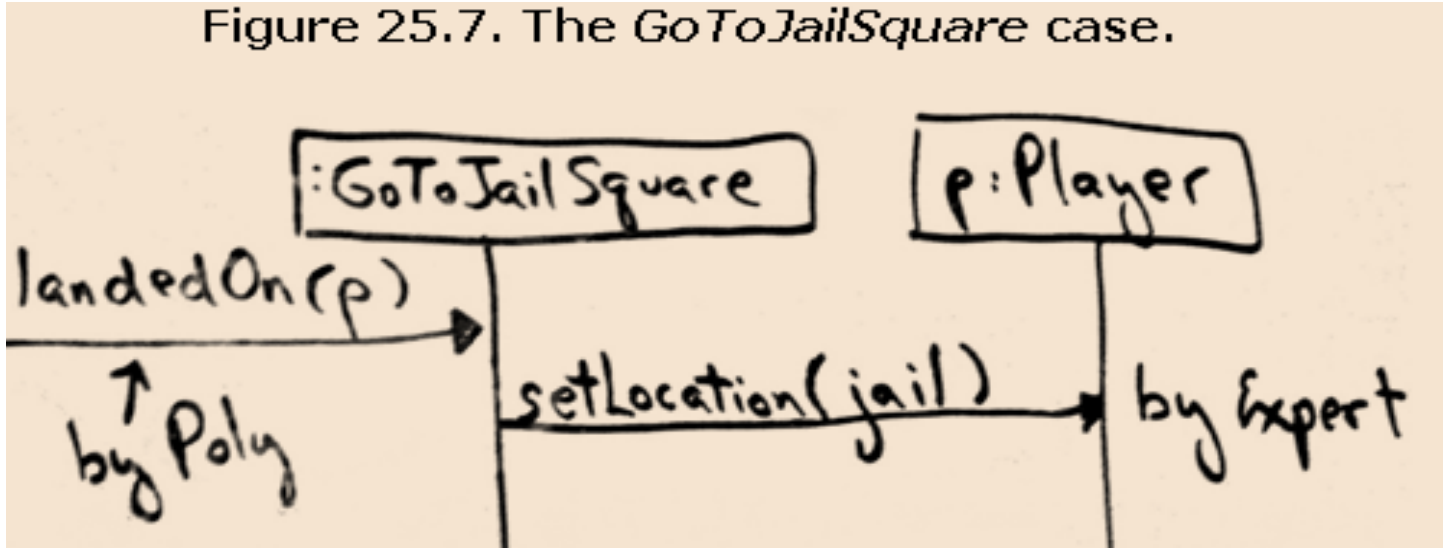
- 3) Income Taxsquare: decrease 10% of total money

Figure 25.6. The *IncomeTaxSquare* case.



- 4) GoToJail Square

Figure 25.7. The *GoToJailSquare* case.



- 本讲结束