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

Lecture_07 通用的职责分配软件原则 GRASP

主讲: 姜宁康 博士

5、GRASP原则五：高内聚 High Cohesion

- How to keep objects focused, understandable and manageable, and as a side effect support Low Coupling? 如何使对象功能专注、可理解、可管理，同时又支持低耦合?

5.1 GRASP rule 5：High Cohesion

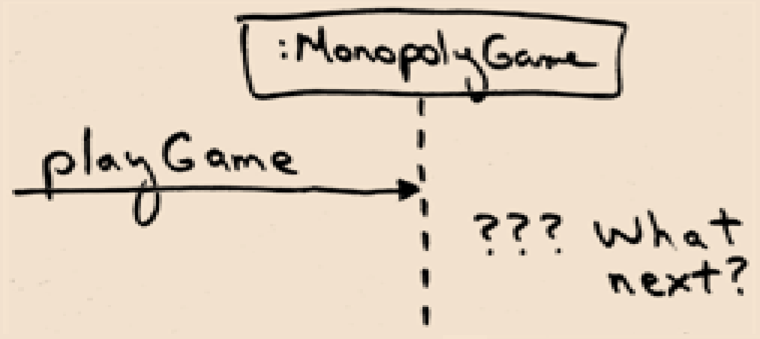
- Name：High Cohesion 高内聚
- Problem:
  - How to keep objects focused, understandable and manageable, and as a side effect support Low Coupling?
- Solution:
  - Assign responsibility so cohesion remains high 分配职责时保证高内聚
- Dosage(用法):
  - Used as an evaluation tool 用作评价工具
  - 更多的是一种理念，没有具体的可操作原则

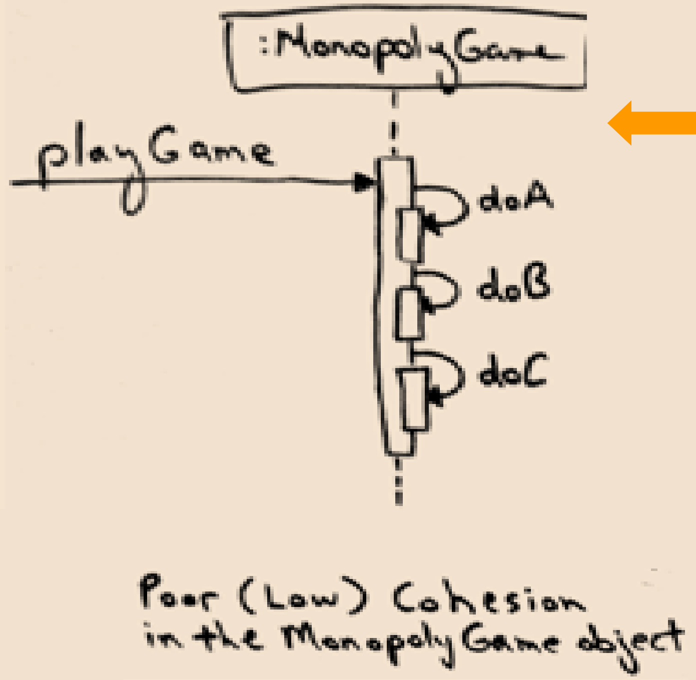
5.2 Cohesion Defined

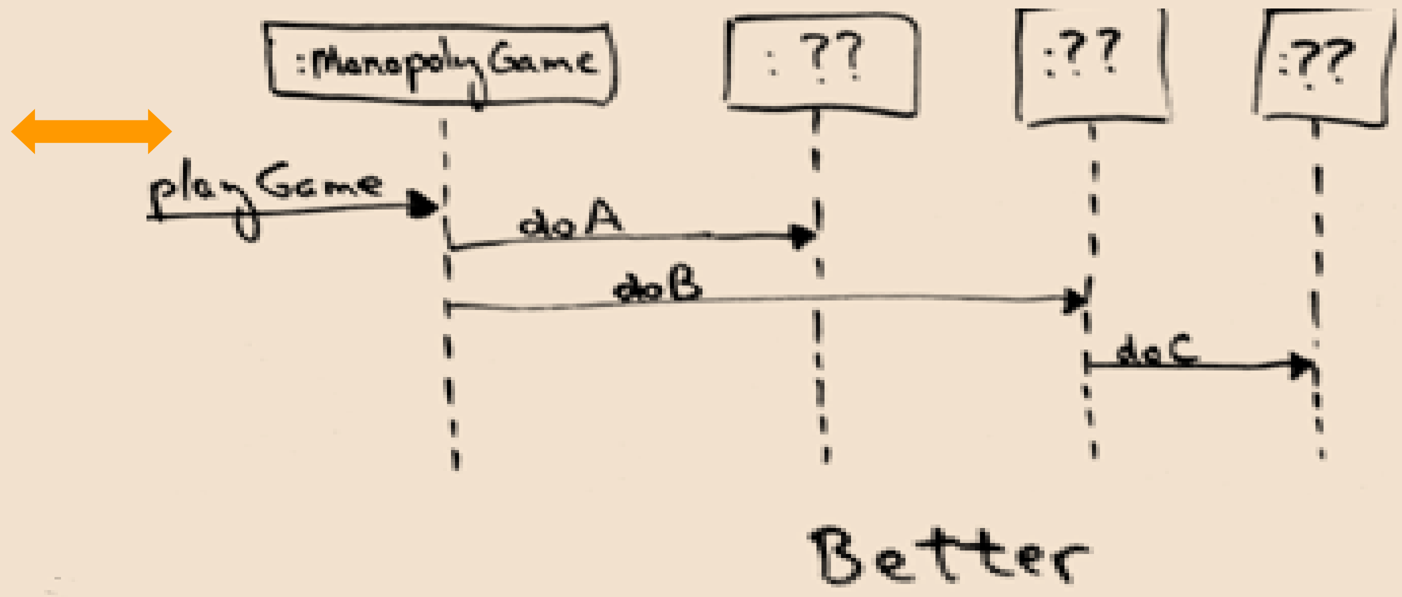
- 衡量概念之间相关度的两个指标
  - Cohesion，内聚：模块内元素之间联系紧密的程度，比如，一个类内部的操作之间
  - Coupling，耦合：两个模块之间联系的强度
- 内聚的“最佳实践”
  - 一个对象完成的功能不要太多 small number of responsibilities
  - 这些功能都是同一类别的 highly related responsibilities
  - 例如，教授：主要任务就是教学；研究员：主要任务是科研
- 评判练习，哪个更内聚
  - 一个类有2000行源代码100个方法
  - 另一个类有200行源代码10个方法
  - 谁能保证任务重的对象在完成功能时不会引用到类外部的资源（增加了耦合度）
- 比喻: “不是一家人，不进一家门”
  - “人” compared to “职责、操作”
  - “门” compared to “模块、类”

5.3 High Cohesion: Monopoly example

- Based on the Controller decision, we are now at the design point
  - two contrasting design approaches worth considering







5.4 Discuss: Cohesion

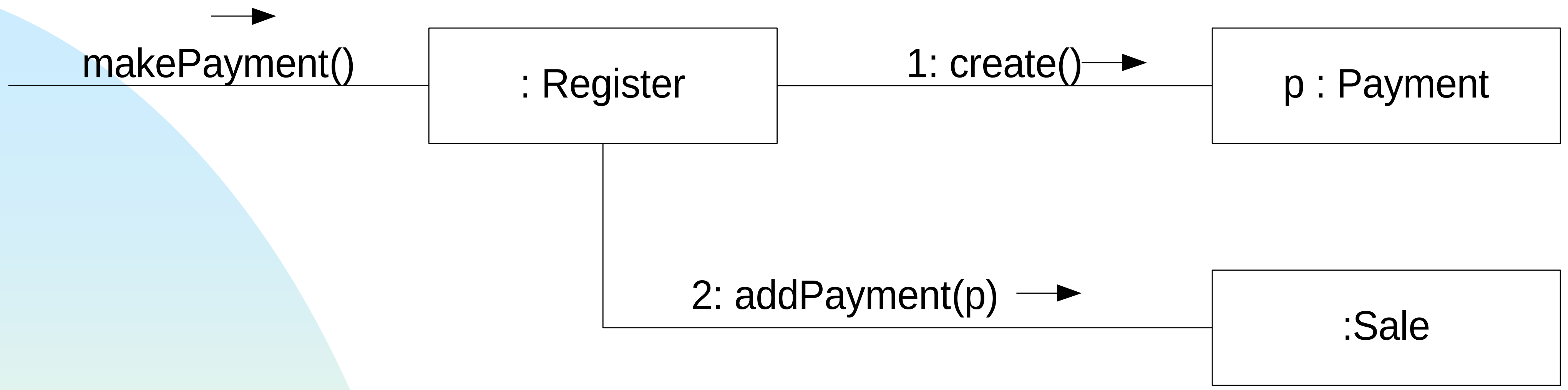
- 类低内聚的具有症状 A class with low cohesion
  - 做了许多相互无关的工作 does many unrelated things
  - 做了太多工作 does too much work
- 类低内聚的的原因 Low cohesion classes often represent
  - 大粒度的抽象 a very large grain of abstraction
  - 做了太多本应该委托给其他类去做的工作 have taken on responsibilities that should have been delegated to other objects
- 类低内聚的问题
  - 难以理解 Hard to understand
  - 难以重用 Hard to reuse
  - 难以维护 Hard to maintain
  - 没有稳定的时刻，总是在修改 (通常都会高耦合)

5.5 Discuss: NextGen POS Example

- The same example problem used in the Low Coupling pattern can be analyzed for High Cohesion
- Assume we have a need to create a (cash) Payment instance and associate it with the Sale
- What class should be responsible for this?

- Since Register records a Payment in the real-world domain, the Creator pattern suggests Register as a candidate for creating the Payment
- The Register instance could then send an addPayment message to the Sale, passing along the new Payment as a parameter

5.5 Discuss: NextGen POS Example



方案1：This assignment of responsibilities places the responsibility for making a payment in the Register. The Register is taking on part of the responsibility for fulfilling the makePayment system operation

In this isolated example, this is acceptable
but if we continue to make the Register class responsible for doing some or most of the work related to more and more system operations, it will become increasingly burdened with tasks and become incohesive 如果Register不停地这么做，就会变得臃肿！

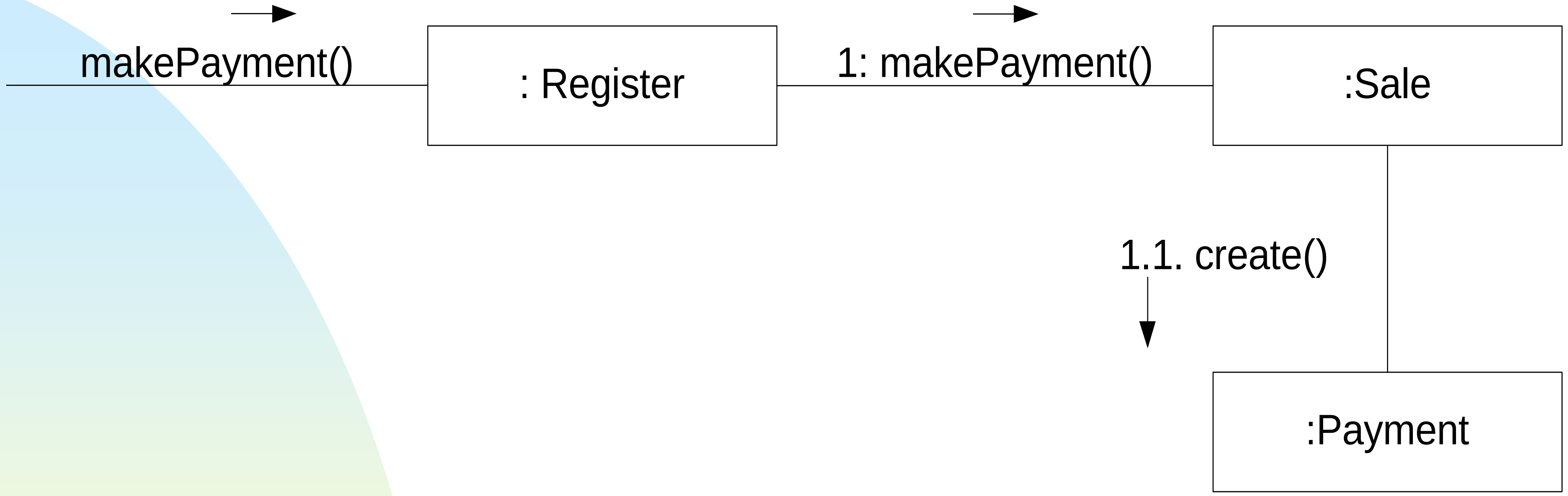
5.5 Discuss: NextGen POS Example

Imagine that there were fifty system operations, all received by Register. If it did the work related to each, it would become a "bloated" incohesive object

The point is not that this single Payment creation task in itself makes the Register incohesive, but as part of a larger picture of overall responsibility assignment, it may suggest a trend toward low cohesion

And most important in terms of developing skills as an object designer, regardless of the final design choice, the valuable thing is that at least a developer knows to consider the impact on cohesion

5.5 Discuss: NextGen POS Example

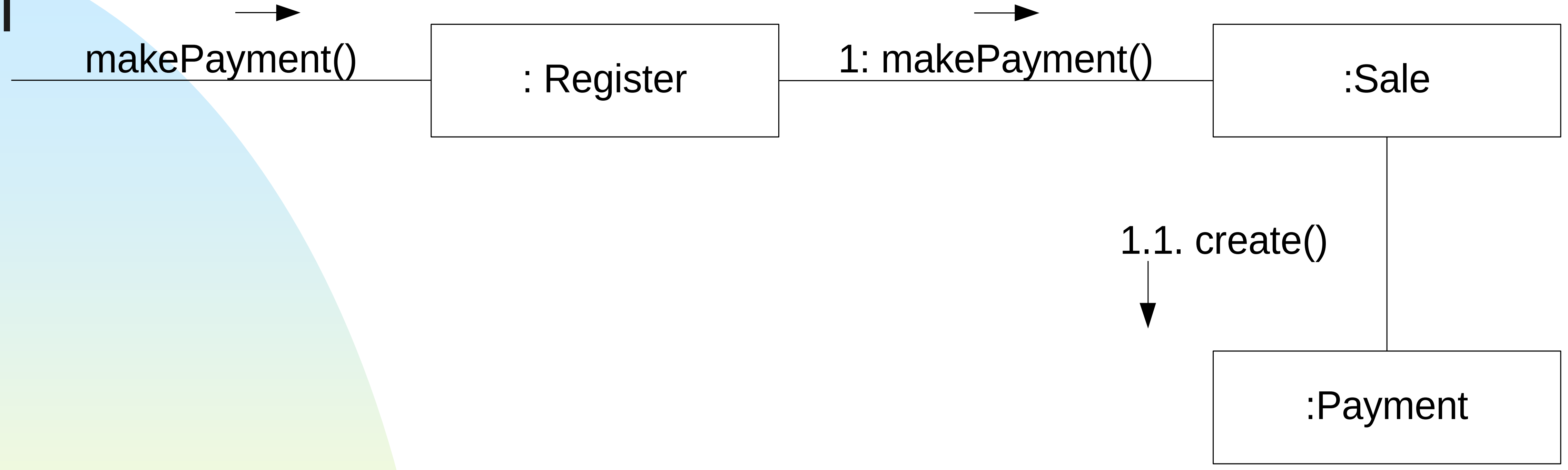


方案2：delegates the <u>payment creation responsibility</u> to the Sale, which supports higher cohesion in the Register 这是理想的方案

在实践中，高内聚必须和信息专家、低耦合等其他原则综合考虑（类似：大局观、系统性思考）

与低耦合一样，高内聚也是要记在心中、所有设计决策都需要加以考虑的原则

5.5 Discuss: NextGen POS Example



- **Delegating 委托 to Sale creates greater cohesion in Register**

小结

- 高内聚的类
  - 有较少数量的操作，操作的性质基本一致，不会做太多的事情
  - 如果同类别的工作太多，则会定义新的类分担任务，相互间合作
- 高内聚的类有许多有点
  - 易于维护
  - 易于理解
  - 易于重用
- 高内聚也是一种评估性原则，用于评估所有的设计决策是否合适

  It is an evaluative principle that a designer applies while evaluating all design decisions

- **本讲结束**