CSSE 374: Examination Results and Moving from Design to Code

Shawn Bohner

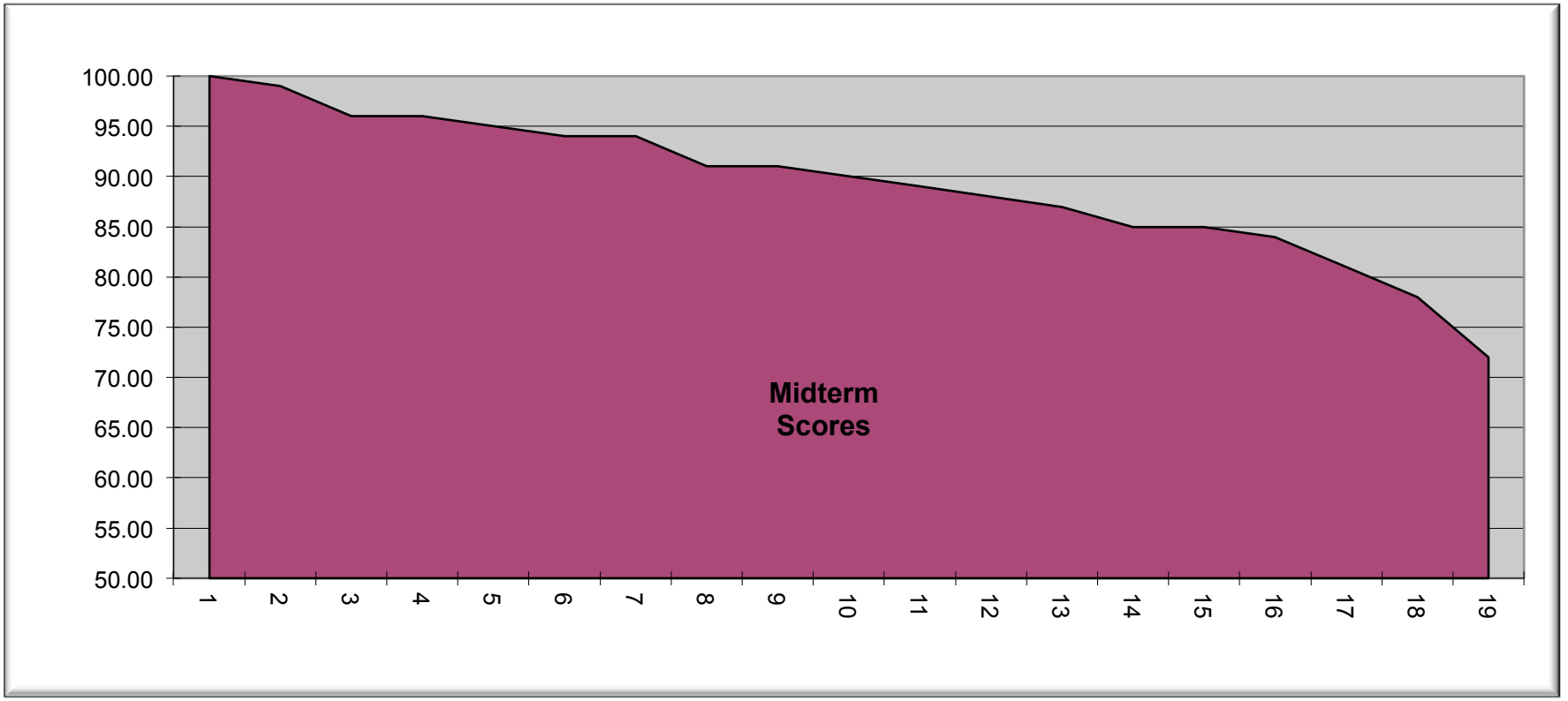
Office: Moench Room F212

Phone: (812) 877-8685

Email: bohner@rose-hulman.edu

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Examination #1 Results



Average Score 89.21%

Median Score 90.00%

Lowest Score 72.00%

Highest Score 100.00%

Exam 1 Stats (Comparative only – course grades will be determined later)

Cutoffs	Grade	# of Grade
90.0%	A	10
85.0%	B+	3
80.0%	B	4
75.0%	C+	1
70.0%	C	1
65.0%	D+	0
60.0%	D	0
0.0%	F	0

Cartoon of the Day

Used with permission. http://notinventedhe.re/on/2009-9-23

Before we get into Code

- Created Domain Model from requirements and use cases
- Used System Sequence Diagrams to identify system operations
- Clarified system operations with Operation Contracts
- Assigned "doing" responsibilities with Interaction Diagrams (Communication and Sequence Diagrams)
- Assigned "knowing" responsibilities with Design Class Diagrams

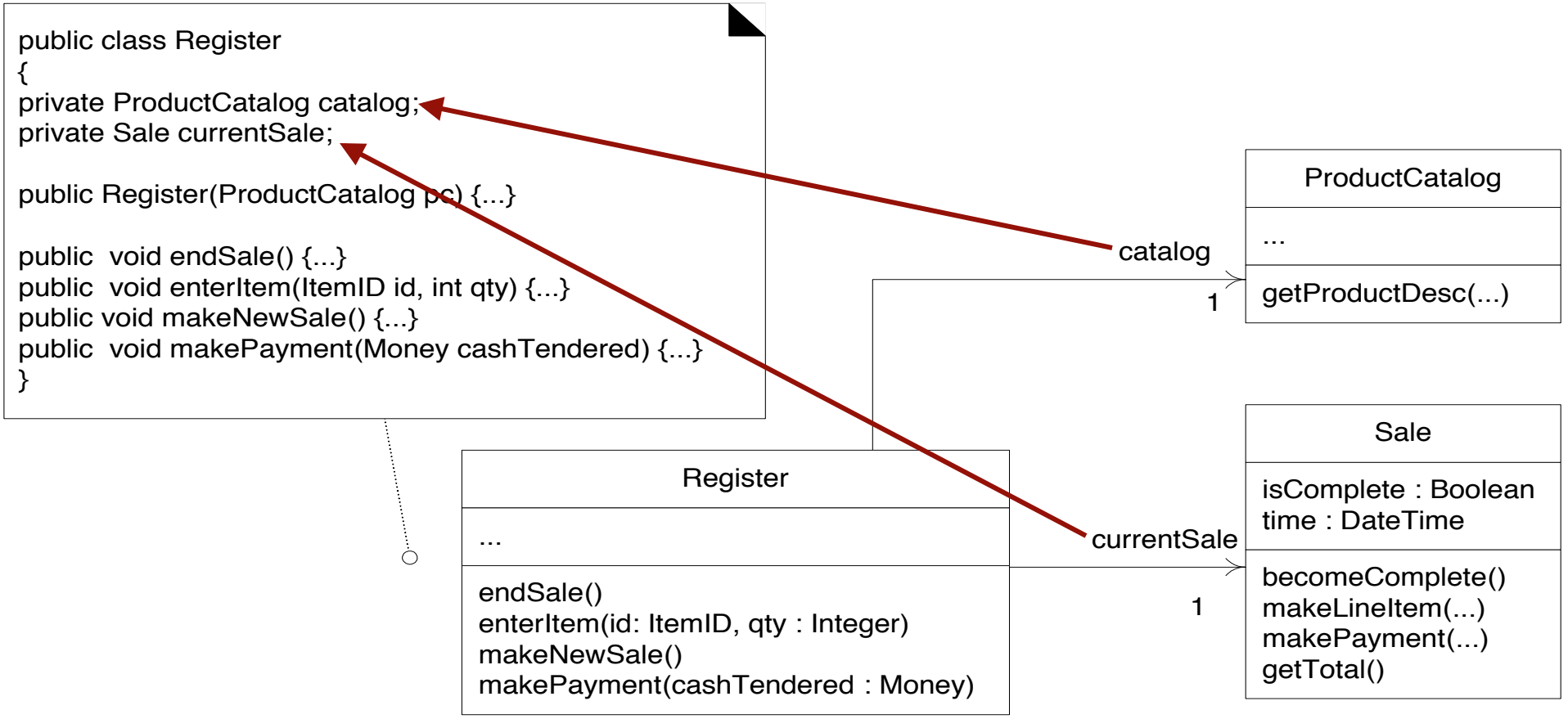
Depending on the system, many of these steps might just be sketches!

Moving from Design to Code

- Design provides starting point for Coding
  - DCDs contain class or interface names, superclasses, method signatures, and simple attributes
- Two primary tasks
  1. Define classes & interfaces
  2. Define methods
- Elaborate from associations to add reference attributes

Example: Defining Register Class

```
public class Register
{
private ProductCatalog catalog;
private Sale currentSale;

public Register(ProductCatalog pc) {...}

public  void endSale() {...}
public  void enterItem(ItemID id, int qty) {...}
public void makeNewSale() {...}
public  void makePayment(Money cashTendered) {...}
}
```

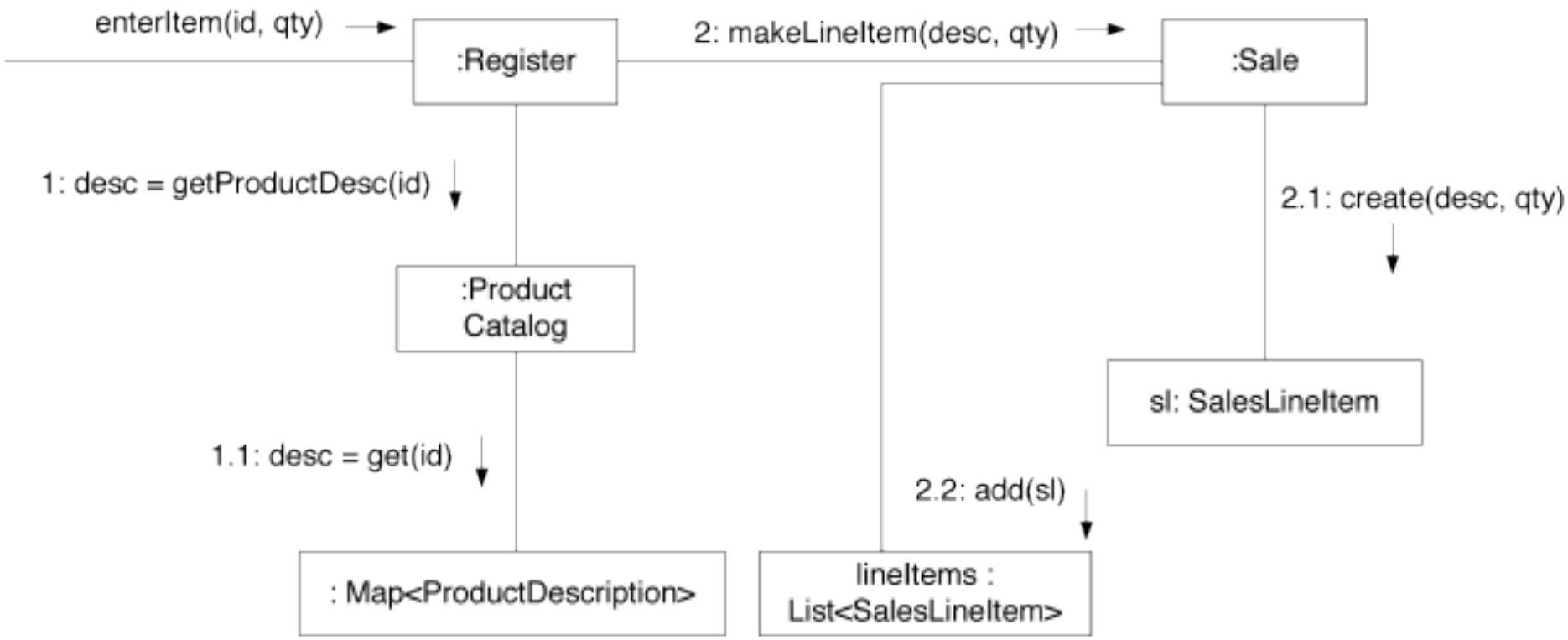


Create Class Definitions from DCDs

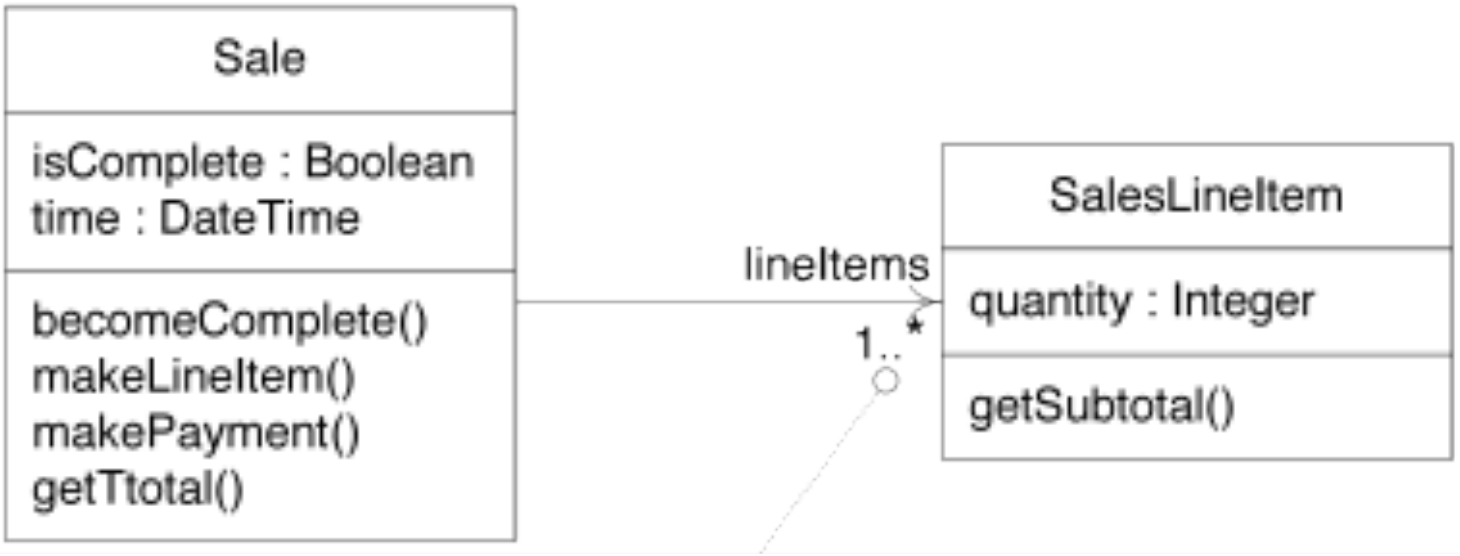
```
public class SalesLineItem
{
private int quantity;

private ProductDescription description;

public SalesLineItem(ProductDescription desc, int qty) { ... }

public Money getSubtotal() { ... }

}
```

Don't write the code on your diagram. Write it in your IDE.

Create Methods from Interaction Diagrams



Collection Classes in Code

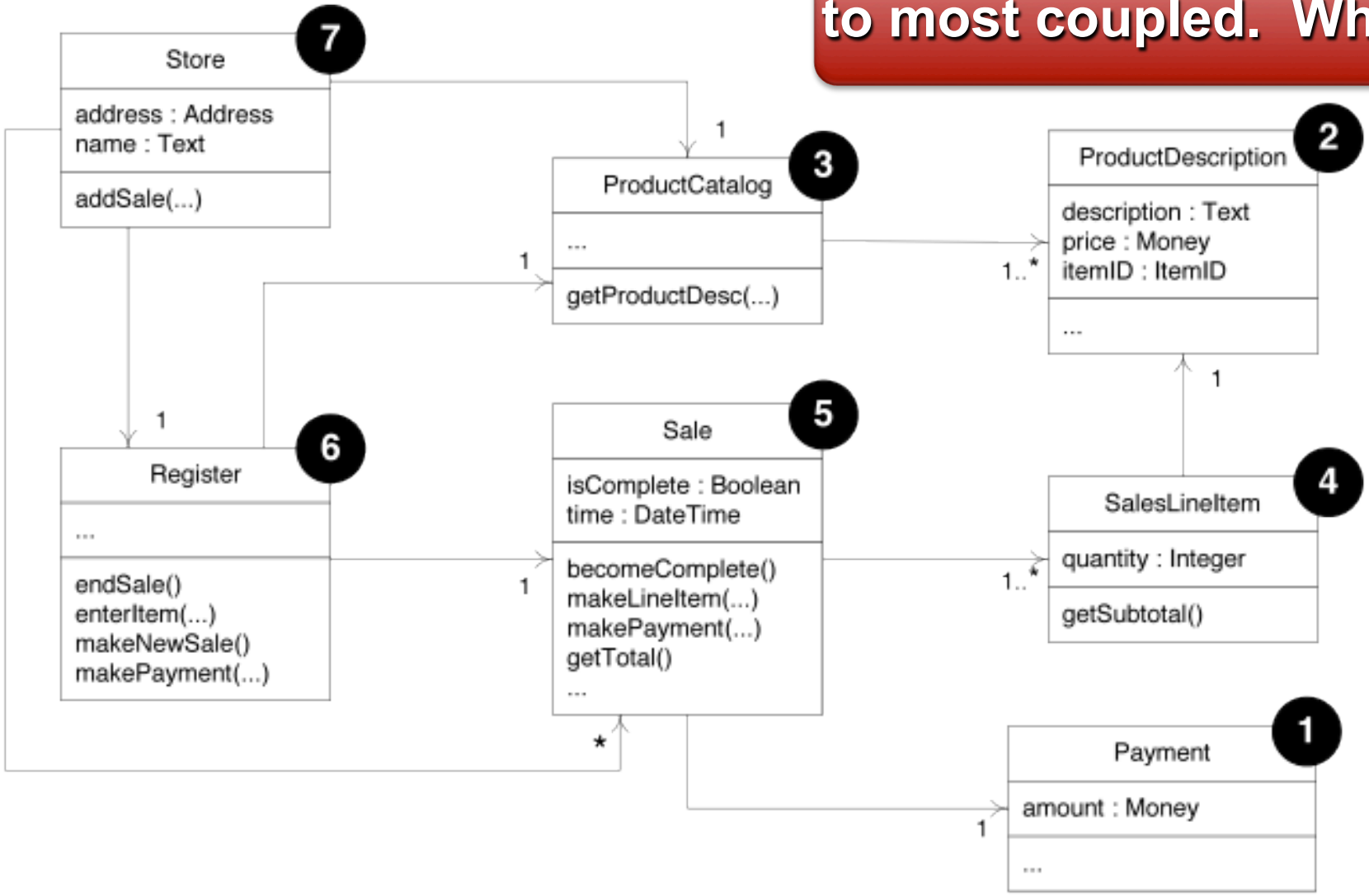



```
public class Sale {
            …
  private List<SalesLineItem> lineItems = new ArrayList<SalesLineItem>();
            …
}
```

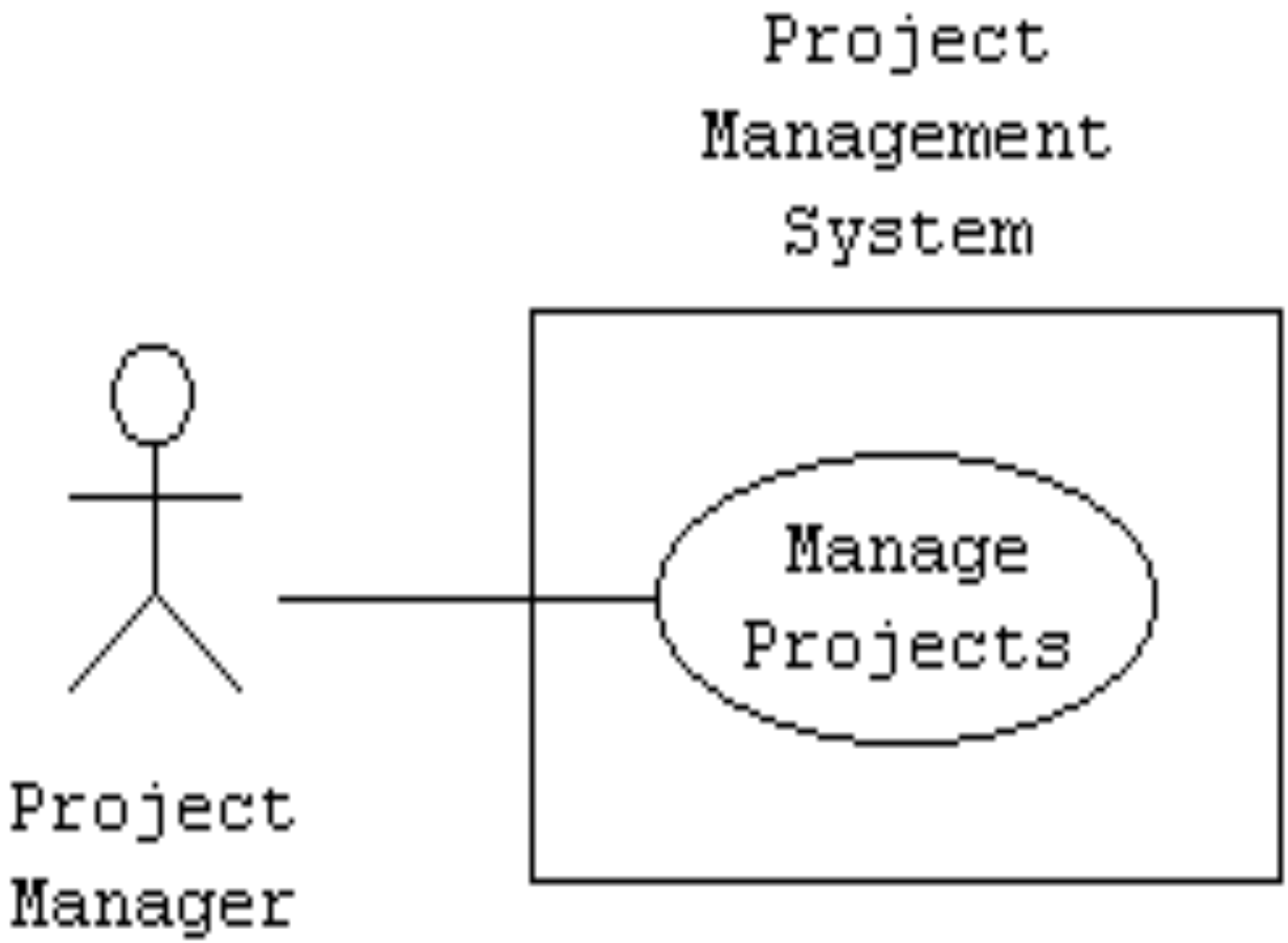
Guideline: If an object implements an interface, use the interface type for the variable.

What Order?

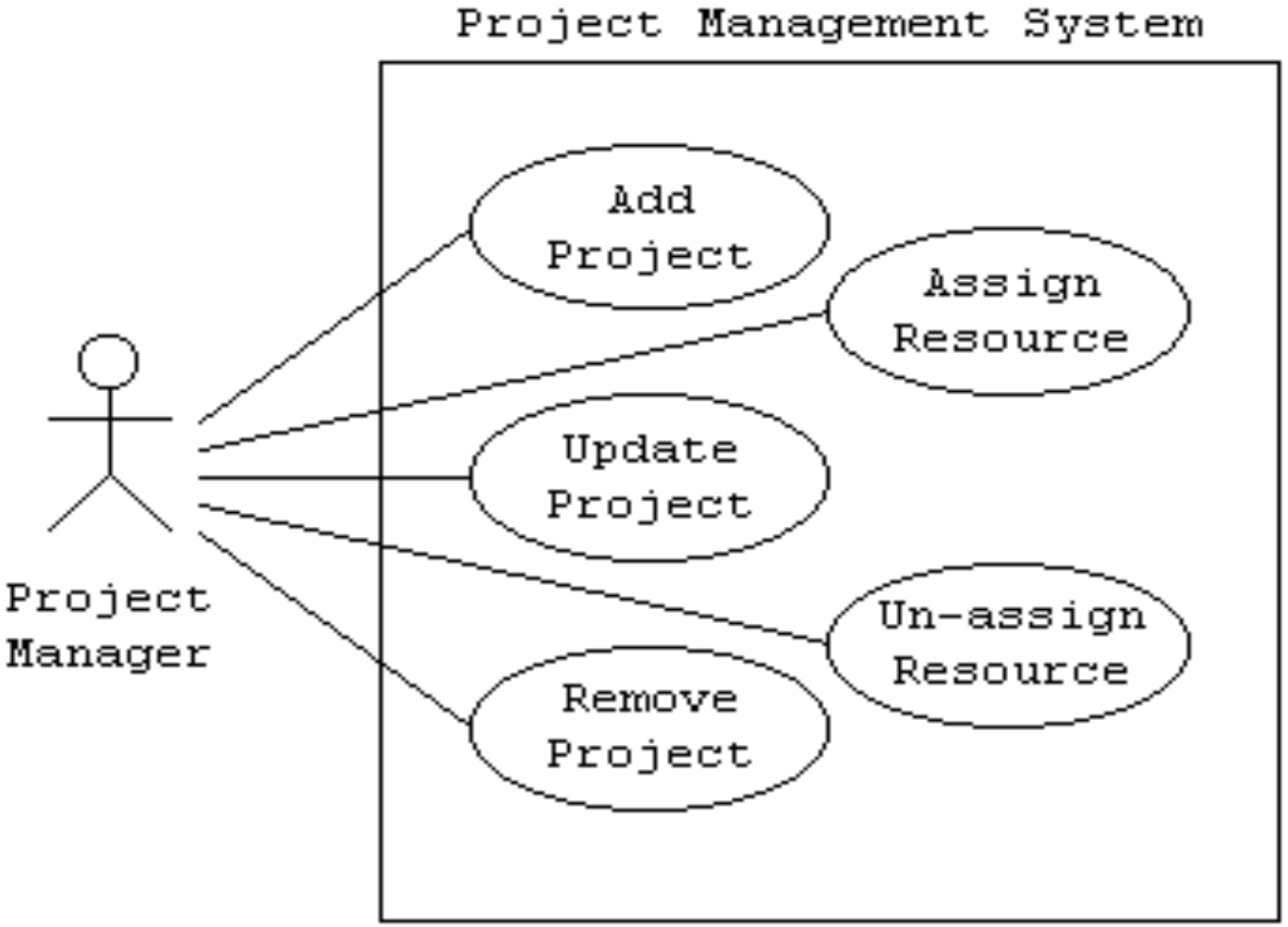
Typically, least coupled to most coupled. Why?



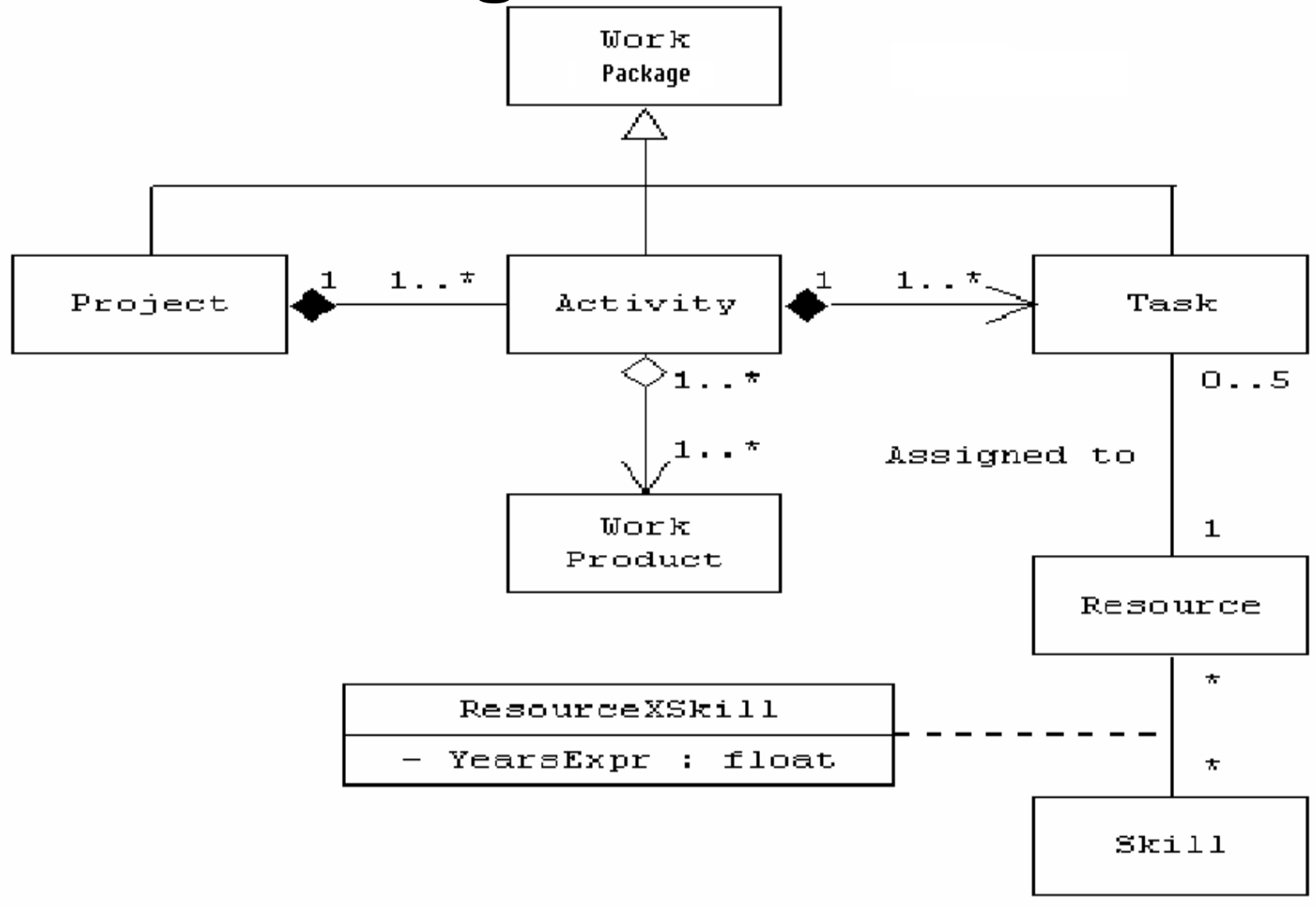
Walk Through Example: PM



PM: Use Case Diagram



PM: Class Diagram



PM: Class to Code

- class WorkPackage;
- class Project;
- class Activity;
- class Task;
- class WorkProduct;
- class Resource;
- class Skill;
- class ResourceXSkill;

PM: Class to Code

```
class WorkPackage
   { // Details omitted };

 class Project : public WorkPackage
 { private: CollectionByVal<Activity> theActivity; };

class Activity : public WorkPackage
 { private: Project *theProject;
                CollectionByVal<Task> theTask;
                 CollectionByRef<WorkProduct>
                                        theWorkProduct; };
```

PM: DCD Mapping

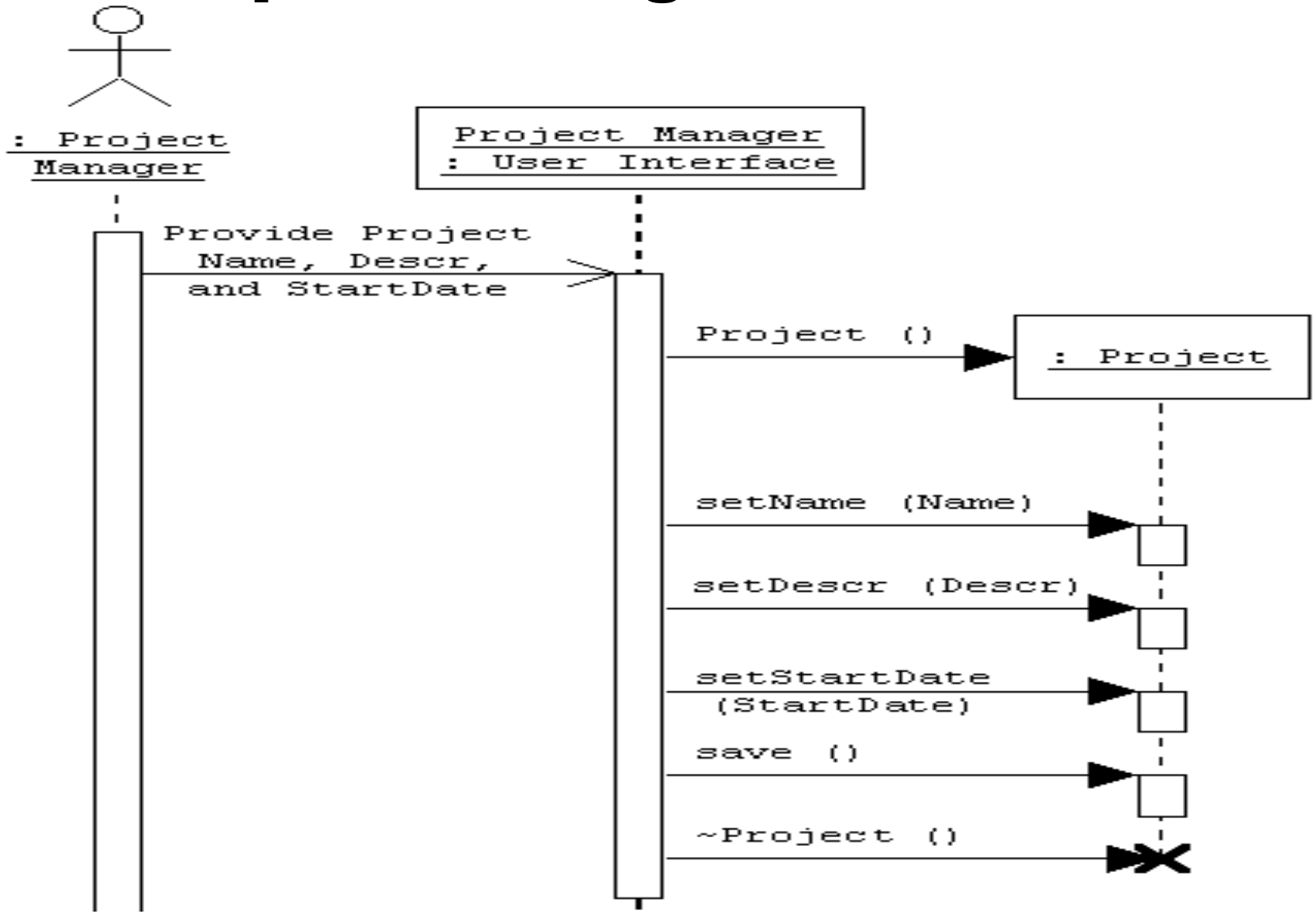
Project
Name : char * {private} - Descr : char * - StartDate : Date - NumberOfProjects : int = 0
+ <<constructor>> Project (Name : char *) : Project + <<constructor>> Project (void) : Project + <<destrcutor>> ~Project (void) + getName (void) : char * + setName (theName : char *) : void setDescr (Descr : char *) : void {public} getDescr (void): char * {public} + setStartDate (theStartDate : Date) : void getStartDate (void) : Date {public} # hasActivites (void) : bool + addActivity (theActivity : const Activity &) : void + getAllAcitivities (void) : CollectionByRef<Activity> + getNumberOfProjects (void) : int + save (void) : void + load (Name : char *) : void

PM: DCD Code

```
class Project
 { private:
     char *Name;
     char *Descr;
     Date StartDate;
     static int NumberOfProjects;
  public:
    Project (char *Name);
    Project (void); ~Project (void);
    char *getName (void);
    void setName (char *theName);
    void setDescr (char *Descr);
    char *getDescr (void);
    void setStartDate (Date
                          theStartDate);
```

```
Date getStartDate (void);
   void addActivity (const Activity
     &theActivity);
  CollectionByRef<Activity>
     getAllAcitivities (void);
   static int getNumberOfProjects (void);
   void save (void);
   void load (char *Name);
protected:
   bool hasActivities (void); };

 int Project::NumberOfProjects = 0;
```

PM: Sequence Diagram



Homework and Milestone Reminders

- Read Chapters 21, 23, 24 on TDD, Refactoring, More on Patterns, and Analysis Revisited
- Milestone 4 – Junior Project Design with GRASP
  - Due by 11:59pm on Friday, January 28th, 2011
- Coming Homework 5 – BBVS Design using more GRASP Principles
  - Due by 11:59pm Tuesday, January 25th, 2011