SNS College of Engineering

Department of Information Technology

Mapping Designs to Code

Presented By

S.Rijutha

AP/IT

Mapping Designs to Code

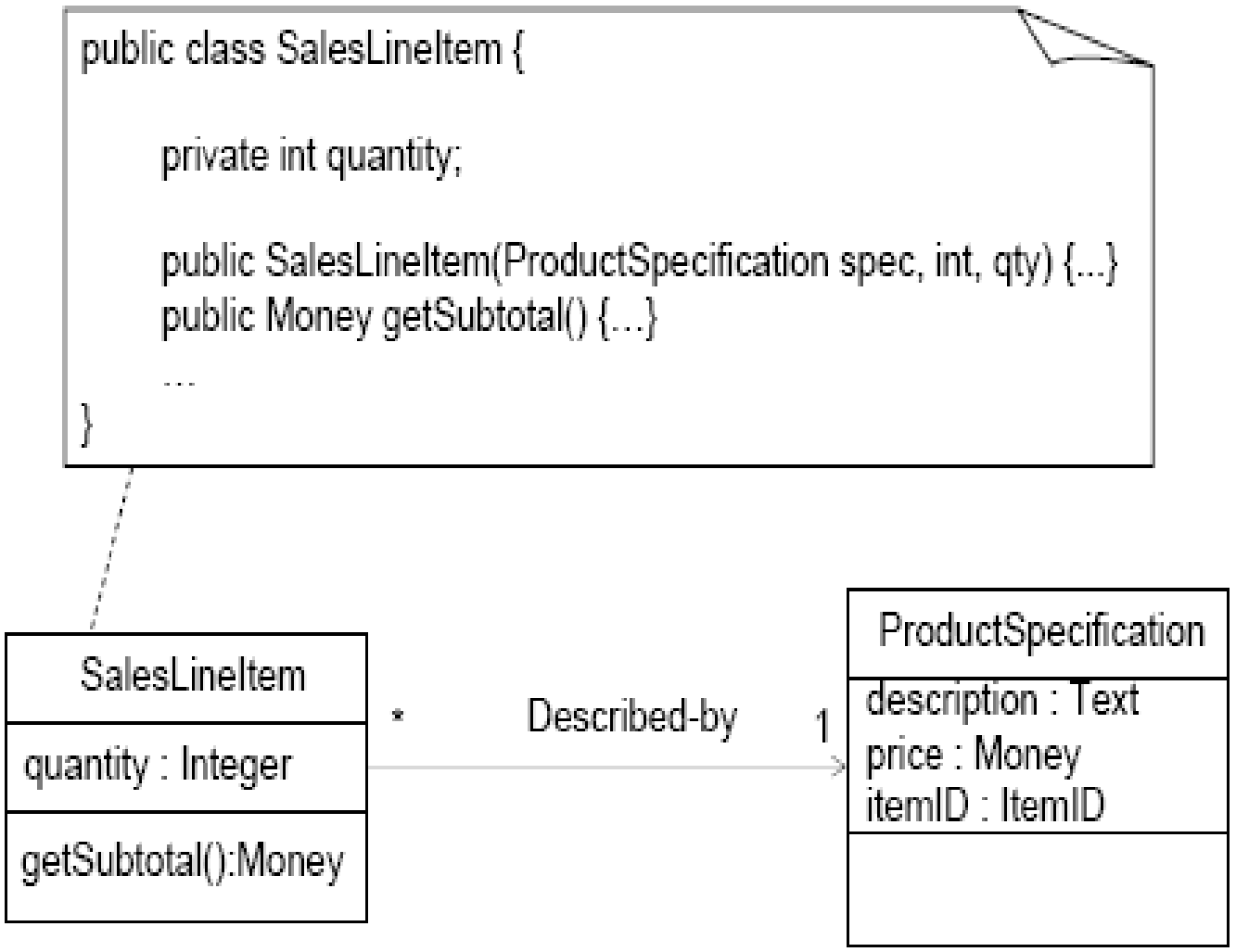
- Process: Write source code for
- Class and interface definitions
  - Method definitions

Design Class Diagrams

- DCDs contain class or interface names, classes, method and simple attributes.
- These are sufficient for basic class definitions.
- Elaborate from associations to add reference attributes.

Reference Attributes

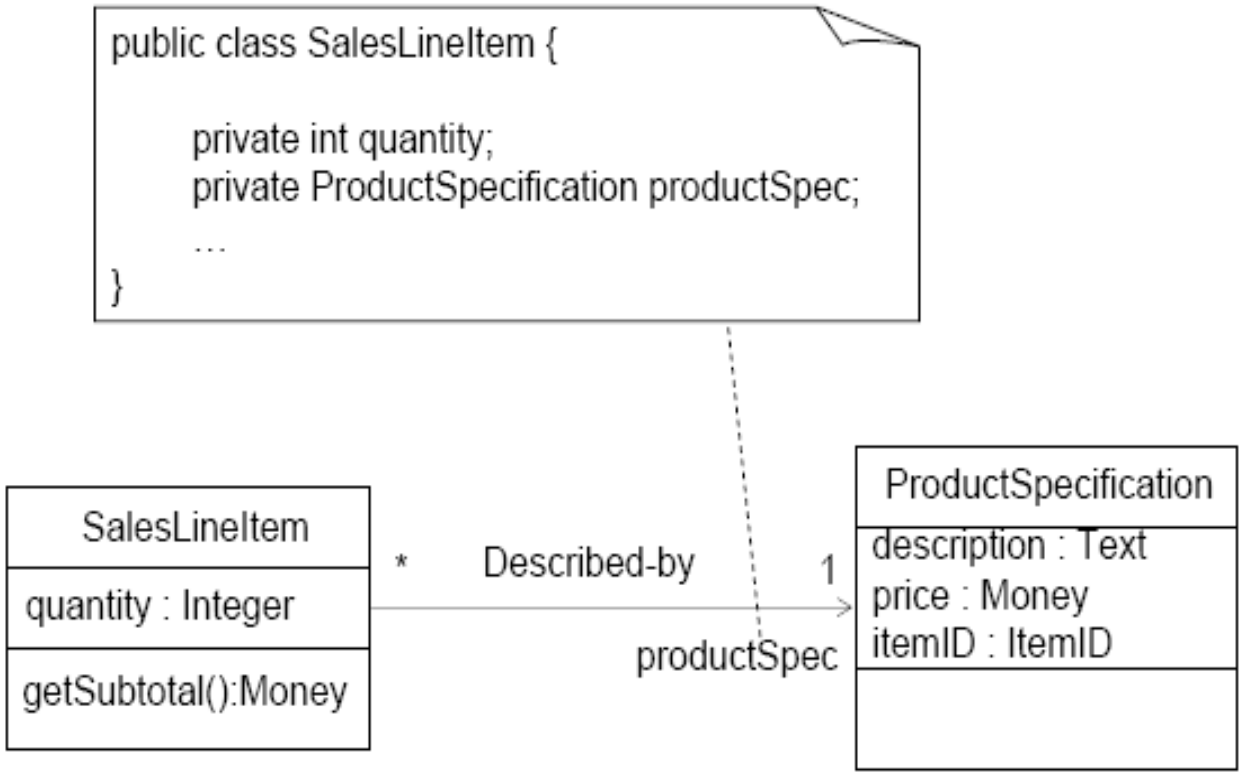
- **An attribute that refers to another complex objects.**
  - Reference Attributes are suggested by associations and navigability in a class diagram.
  - Example: A product specification reference on a Sales Line Item. So here we can use product spec as a complex reference attribute to sales line item class.


public class SalesLineItem {
private int quantity;
public SalesLineItem(ProductSpecification spec, int, qty) {...}
public Money getSubtotal() {...}
...
}
SalesLineItem
quantity : Integer
getSubtotal():Money
*
Described-by
1
ProductSpecification
description : Text
price : Money
itemID : ItemID

Role Names

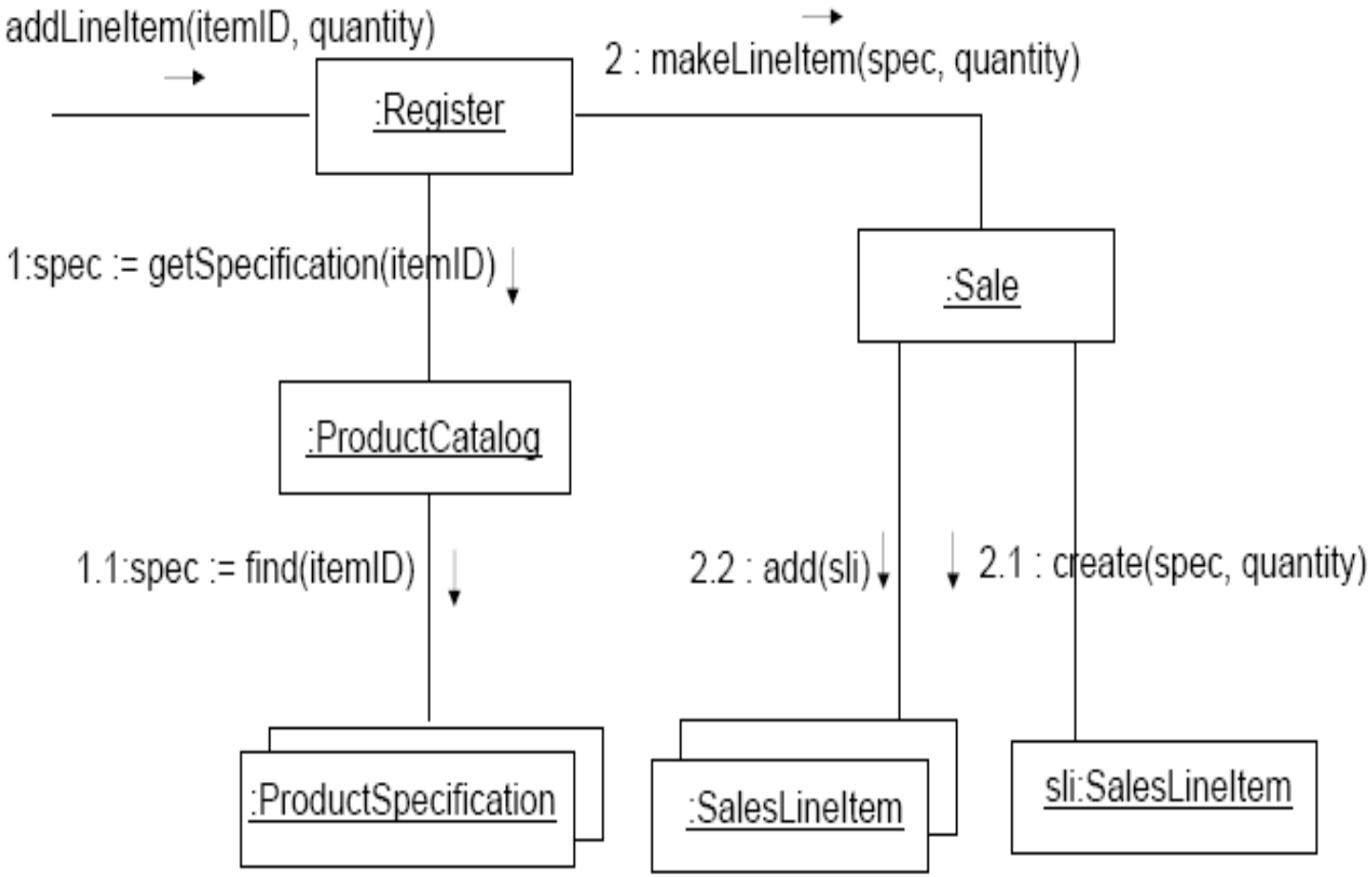
- Each end of an association is a role. Reference Attributes are often suggested by role names.

  (use role names as the names of reference attributes).


public class SalesLineItem {
private int quantity;
private ProductSpecification productSpec;
…
}
SalesLineItem
quantity : Integer
getSubtotal():Money
*
Described-by
1
productSpec
ProductSpecification
description : Text
price : Money
itemID : ItemID

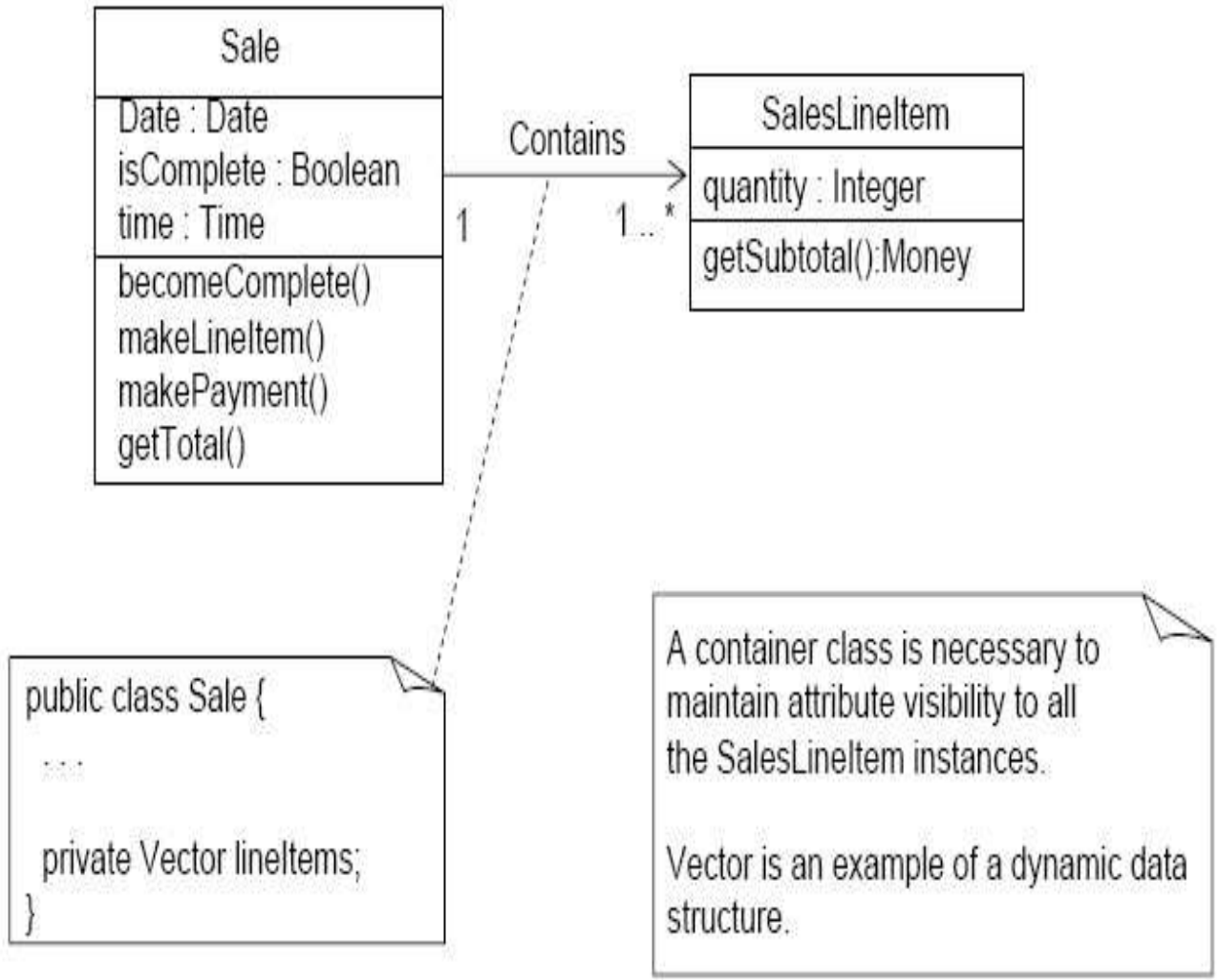
Creating Methods from Interaction Diagrams

- Interaction Diagrams are used to specify methods.
- They give most of the details for what the method does.

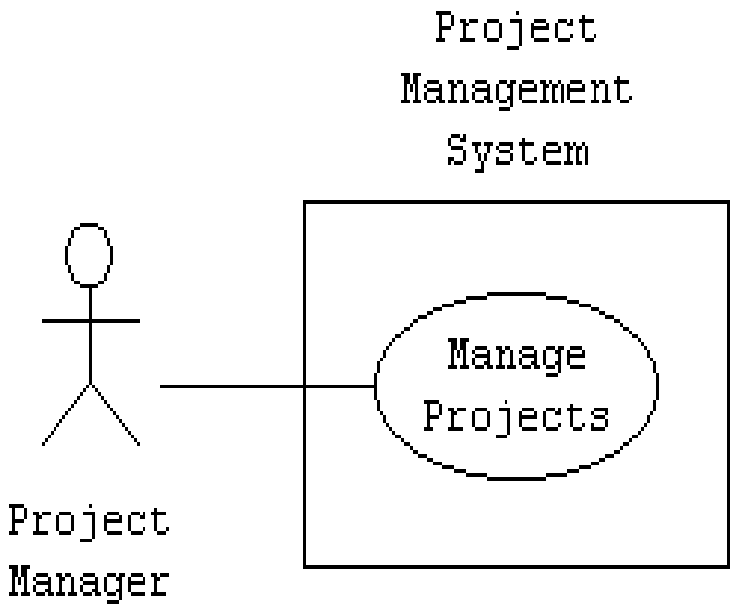

addLineItem(itemID, quantity)
:Register
2 : makeLineItem(spec, quantity)
1:spec := getSpecification(itemID)
:Sale
:ProductCatalog
1.1:spec := find(itemID)
2.2 : add(sli)
2.1 : create(spec, quantity)
:ProductSpecification
:SalesLineItem
sli:SalesLineItem

Containers and Collections

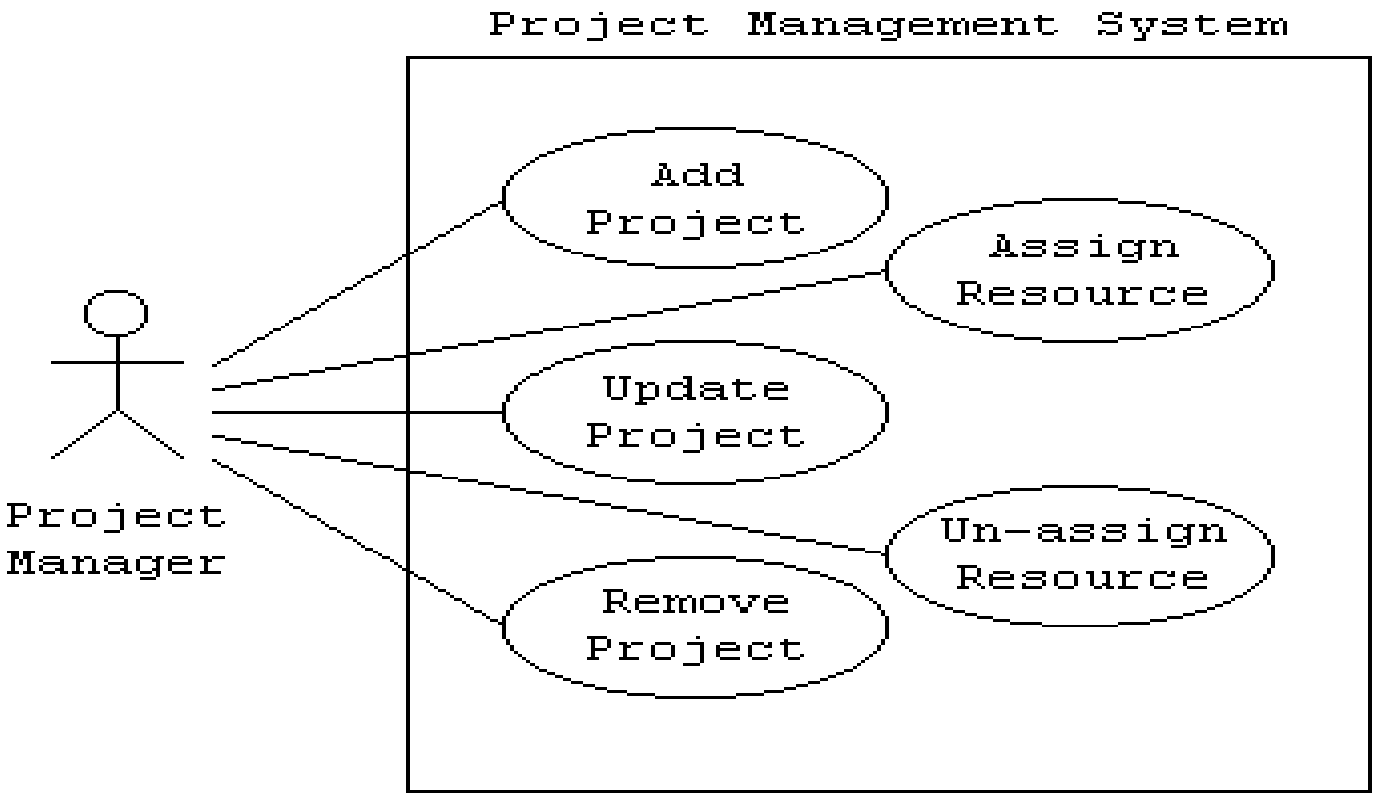
- Where an object must maintain visibility to a group of other objects, such as a group of Sales Line Items in a Sale, object-oriented languages often use an intermediate container or collection.
- These will be suggested by a multiplicity value greater than one on a class diagram.


Sale
Date : Date
isComplete : Boolean
time : Time
becomeComplete()
makeLineItem()
makePayment()
getTotal()
Contains
1
1 .. *
SalesLineItem
quantity : Integer
getSubtotal():Money
public class Sale {
. . .
private Vector lineItems;
}
A container class is necessary to maintain attribute visibility to all the SalesLineItem instances.
Vector is an example of a dynamic data structure.

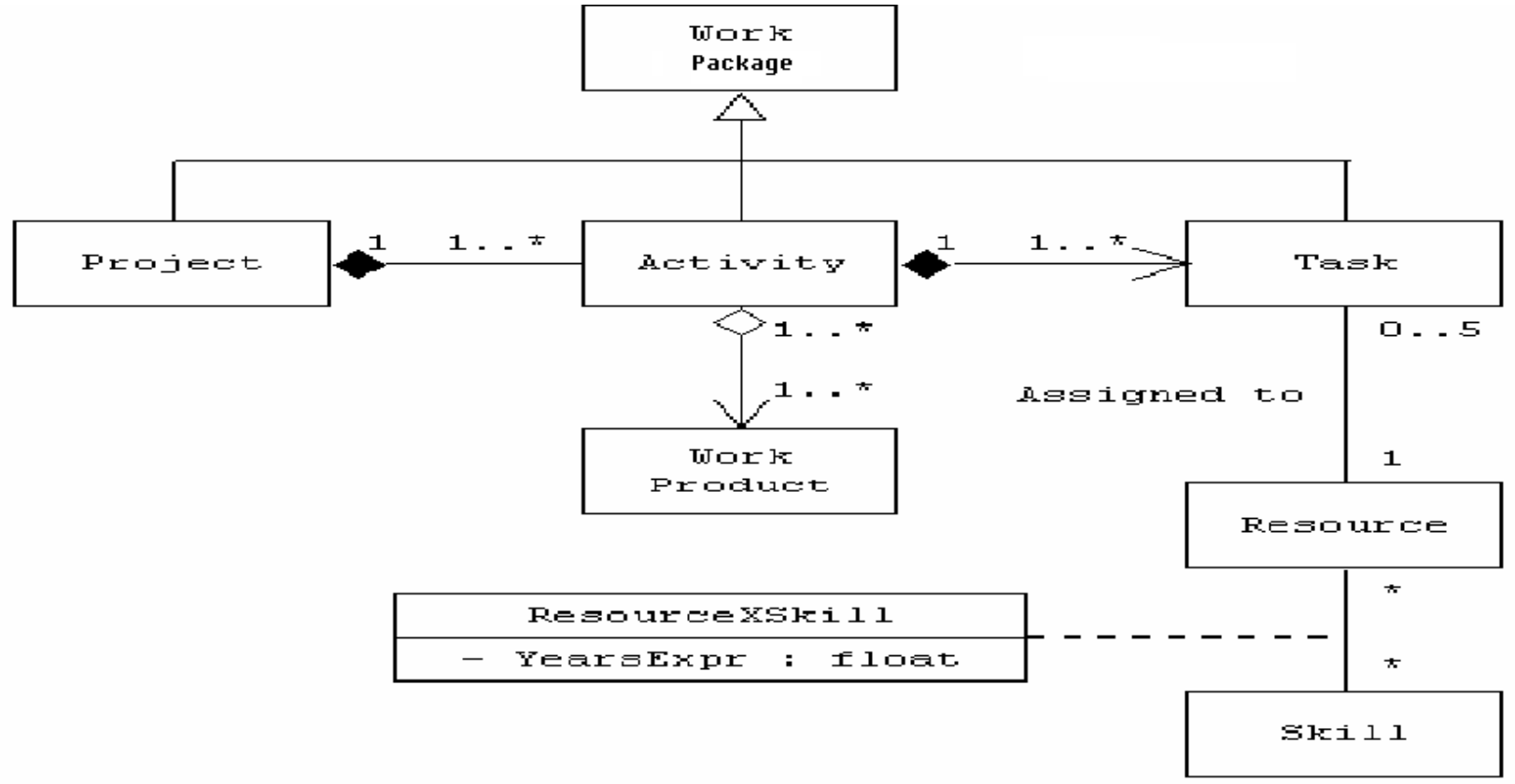
Working Example: PM



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

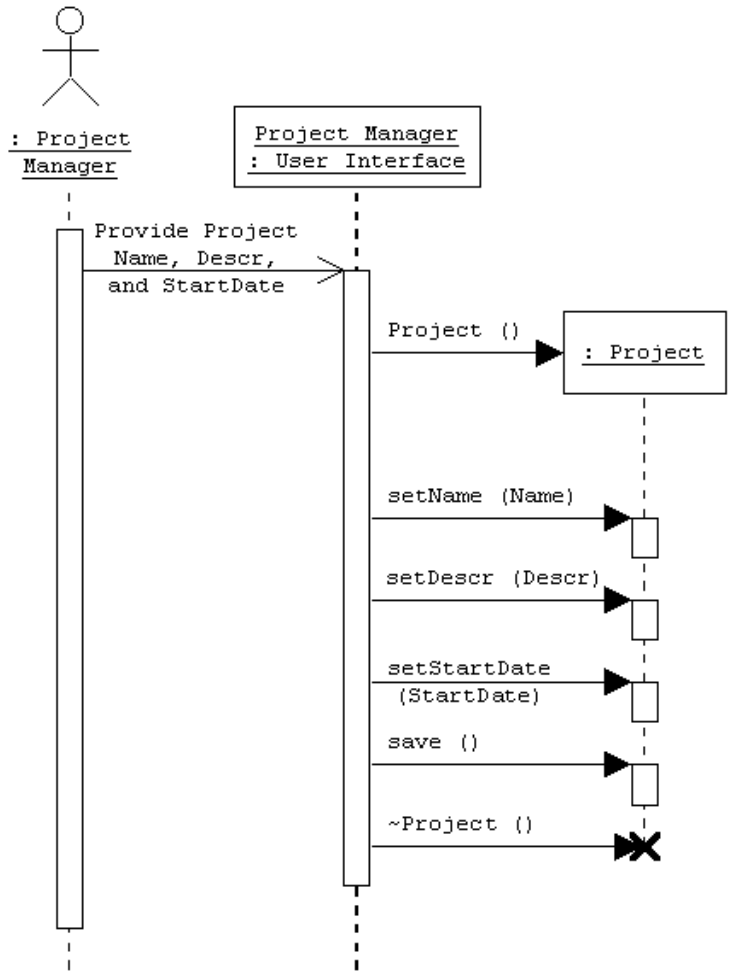
Project
Name : char * {private} – Descr : char * – StartDate : Date – NumberOfProjects : int = 0
+ <<constructor>> Project (Name : char *) : Project + <<constructor>> Project (void) : Project + <<destrcutor>> ~Project (void) + getName (void) : char * + setName (theName : char *) : void setDescr (Descr : 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);
   bool hasActivities (void); };
int Project::NumberOfProjects = 0;
```

PM: Sequence Diagram



PM: Sequence to Main

```
void main (void)

{ char *Name;              char *Descr;

Date StartDate; Project aProject;

  // provide project Name, descr, and  startdate

aProject.setName (Name);  aProject.setDescr (Descr);

    aProject.setStartDate (StartDate);  aProject.save (); }
```