Topic 5

Introduction to UML Diagrams

Objectives

- To introduce UML Diagrams
  - A diagrammatic way of showing the relationships among classes
  - This will help our understanding of the definitions of our collections and the usage of our collections in applications

1-2

UML Diagrams

- ***Unified Modeling Language (UML)*** is a standard notation for object-oriented design
  - Used to ***model*** object-oriented designs
  - Shows overall design of a solution
    - Shows class specifications
    - Shows how classes interact with each other
  - Diagrams use specific icons and notations
  - It is ***language independent***

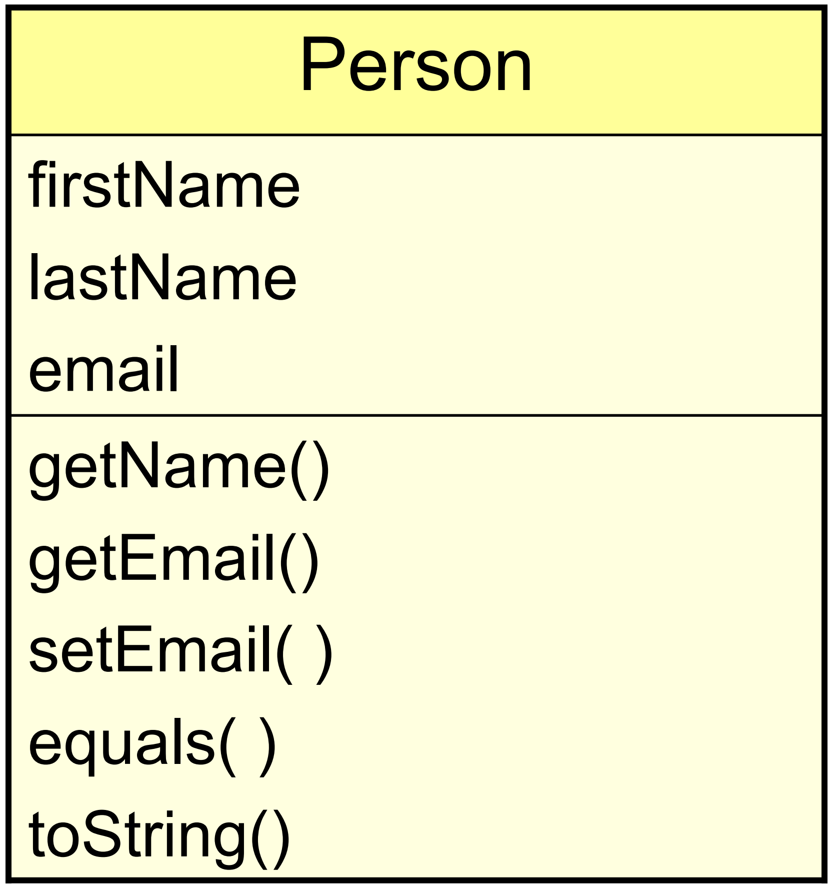
1-3

UML Class Diagram

- A class is represented in a UML diagram by a rectangle divided into **3** sections:
  - ***name*** of the class
  - ***attributes*** of the class (i.e. the data fields of the class, including variables and constants)
  - ***operations*** of the class (essentially equivalent to a Java method or a C++ function)

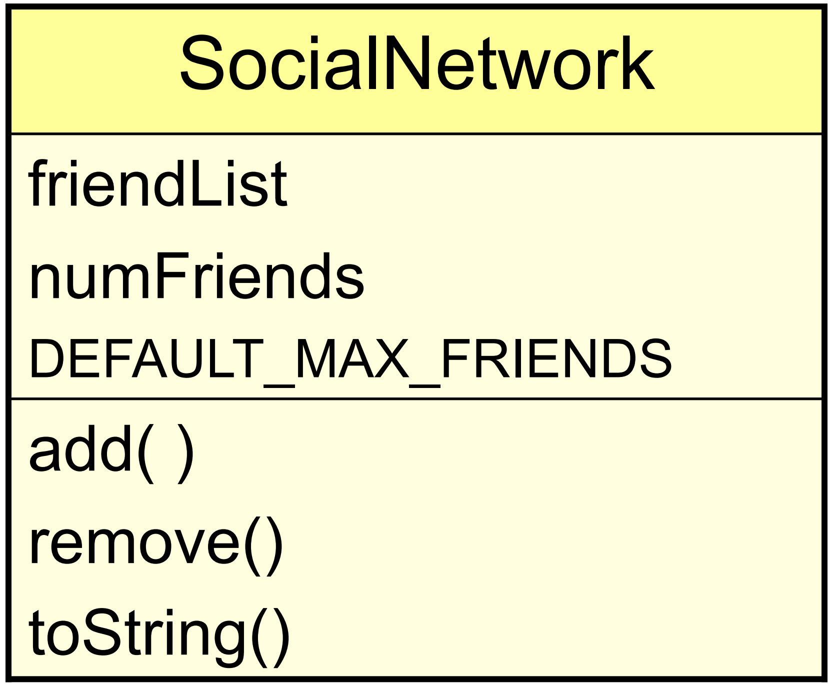
1-4

Example: UML Class Diagram



1-5

Example: UML Class Diagram



1-6

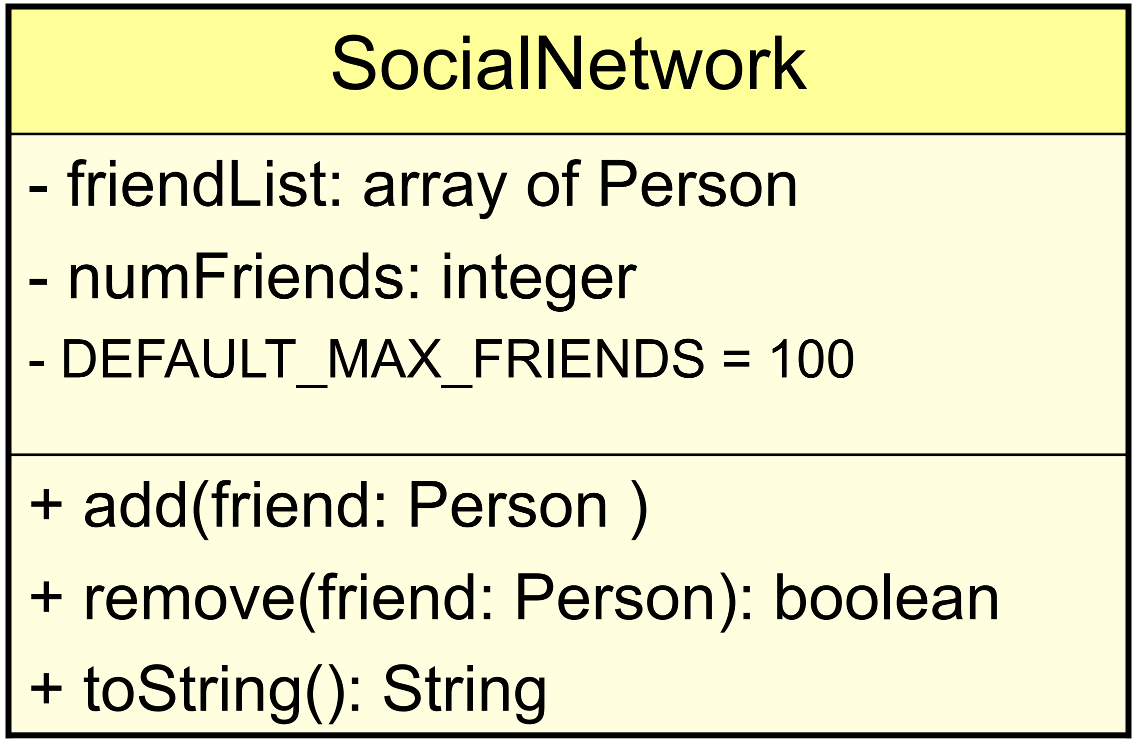
Features of UML Class Diagrams

- Attributes and operations may include:
  - ***visibility***: public (+) or private (-)
  - ***type*** of attribute or operation
  - ***parameter list*** for operations
- Including this information is of the form:

```
visibility    variable_name: type
visibility    variable_name: type = default_value
visibility    method_name(parameter_list): return_type
                                              {property}
```

1-7

Example: UML Class Diagram



SocialNetwork

- friendList: array of Person
- numFriends: integer
- DEFAULT_MAX_FRIENDS = 100

+ add(friend: Person)
+ remove(friend: Person): boolean
+ toString(): String

1-8

Features of UML Class Diagrams

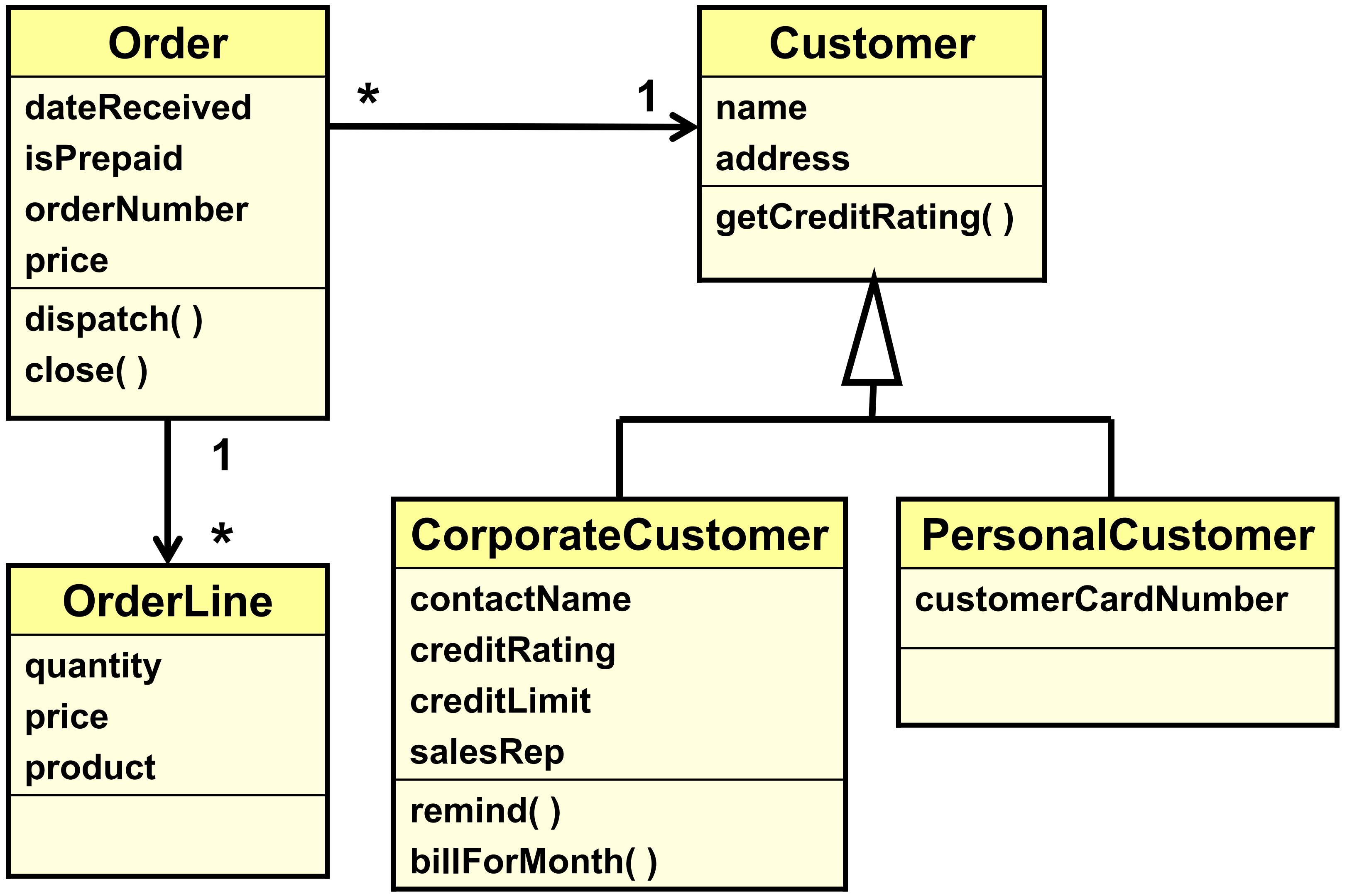
- Attributes and operations may be left incomplete, and completed as design is developed

Set of UML Class Diagrams

- A set of UML class diagrams shows:
  - The classes used in the system
  - The ***relationships*** among classes
  - The ***constraints*** on the connections among classes

1-10

Example: UML Diagram for Order Processing



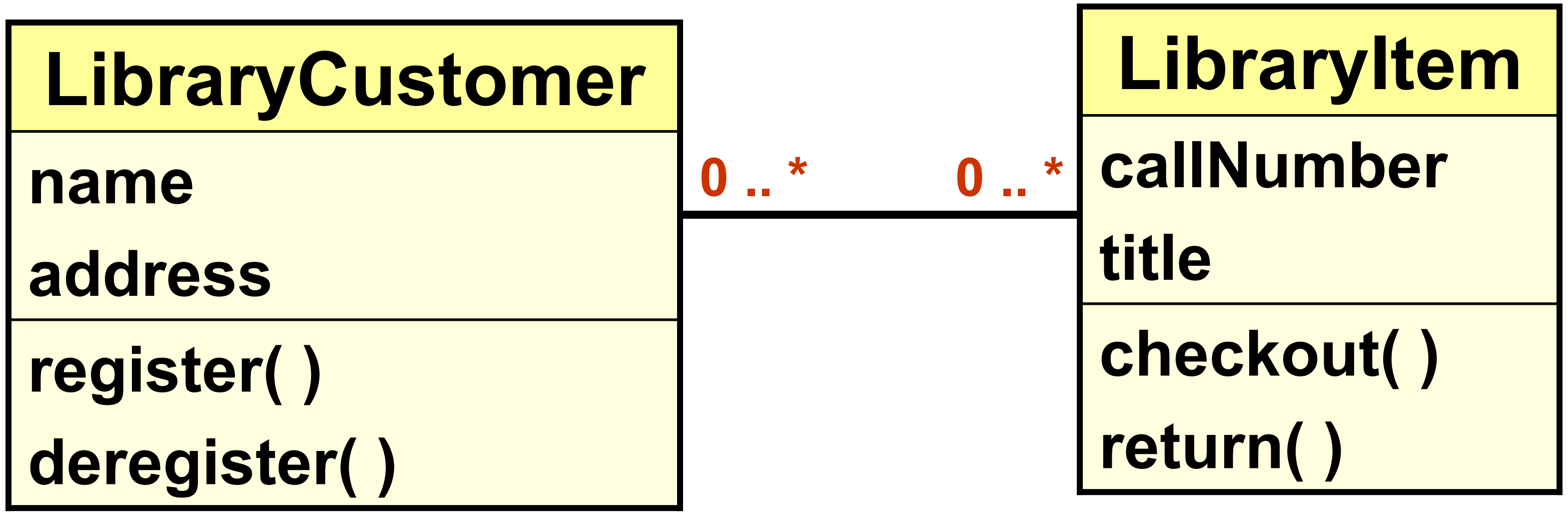
1-11

Features of Set of UML Diagrams

- ***Association between classes***:
  - Represents a relationship between objects of those classes
  - Indicated with a *solid line* between the classes
  - Can be annotated with ***cardinality***: indicates a numeric association between classes, such as:
    - **one-to-one**
    - **one-to-many (1..*)**
    - **many-to-many (*..*)**
    - **zero-to-many (0..*)**
    - **zero-to-5 (0..5)**
    - **etc.**

1-12

Example: Association Between Classes



1-13

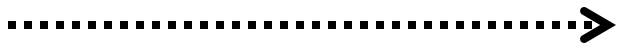
Association Between Classes

- What is the Order-Customer relationship in our Order Processing System?

- How would we annotate that a Library Customer can not check out more than 5 library items?

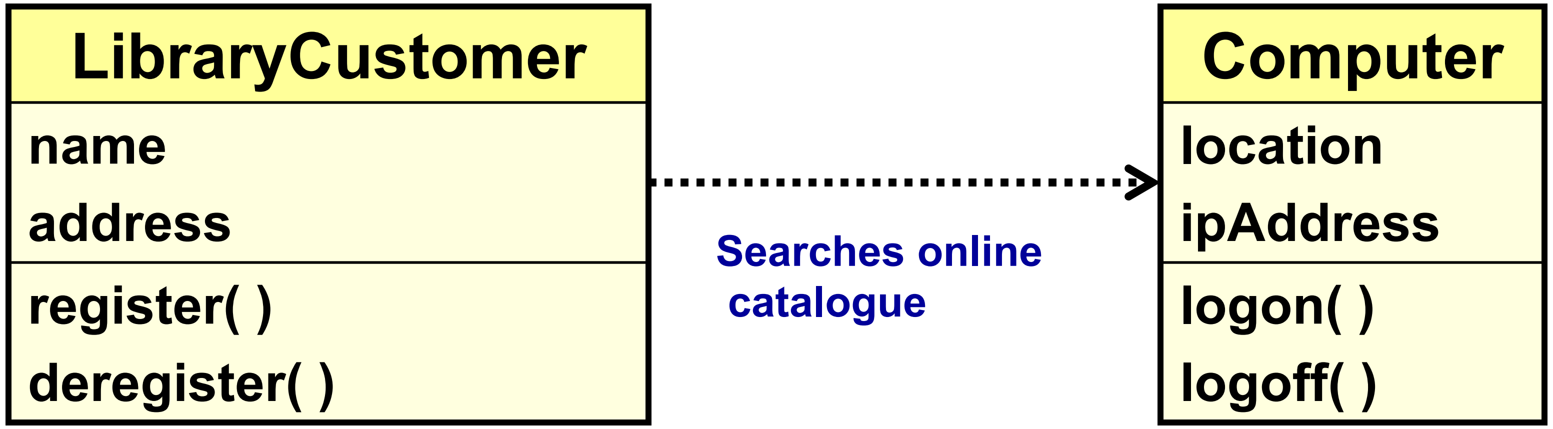
Features of Set of UML Diagrams

- ***Usage of another class***:
  - Broken line with an arrow indicates that one class makes use of the other



  - Line can be labeled with a message indicating the type of usage

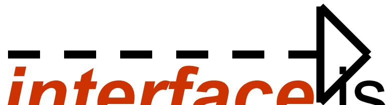
Example: One Class Indicating its Use of Another



1-16

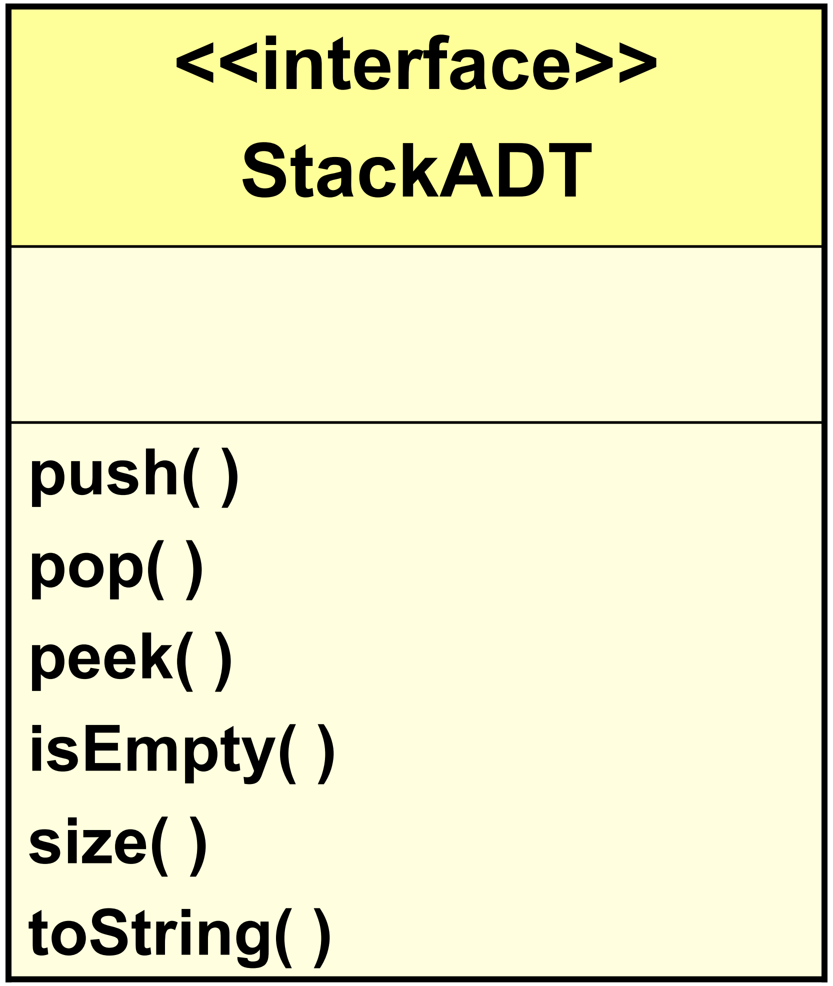
Features of Set of UML Diagrams

- ***Implementation of an interface***:
  - Indicated by a broken line with an open arrow

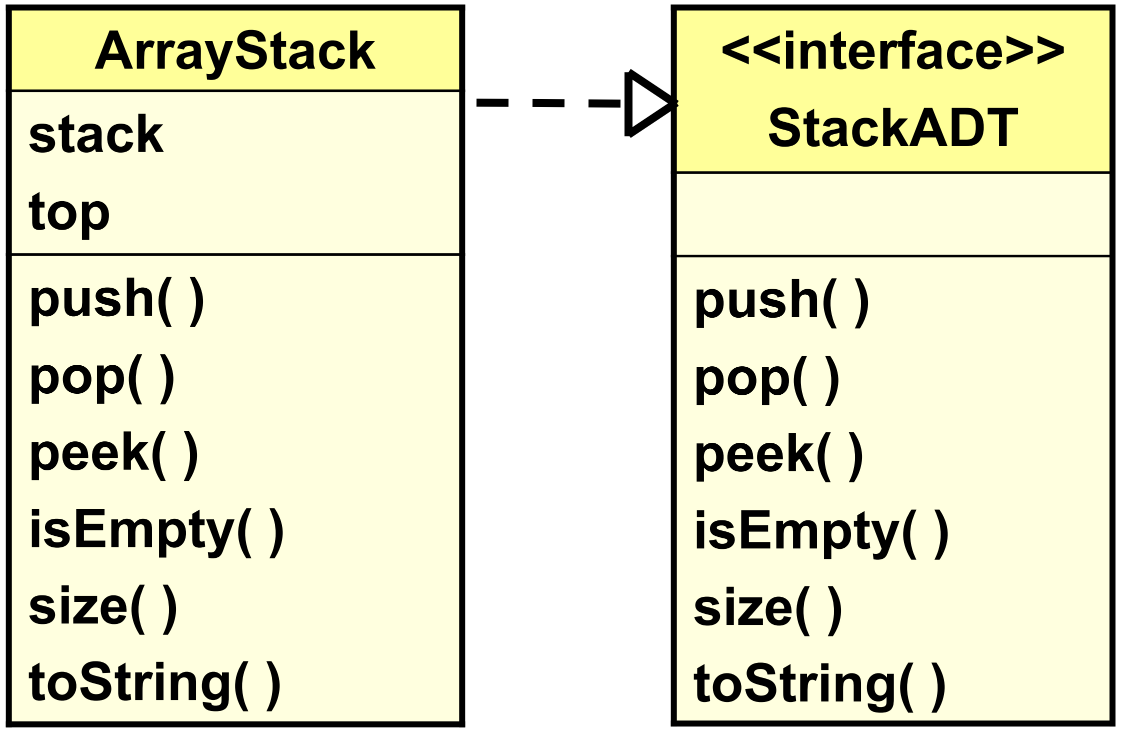


  - ***UML diagram for an interface*** is much like the UML diagram for a class
    - But there are no attributes (why not?)

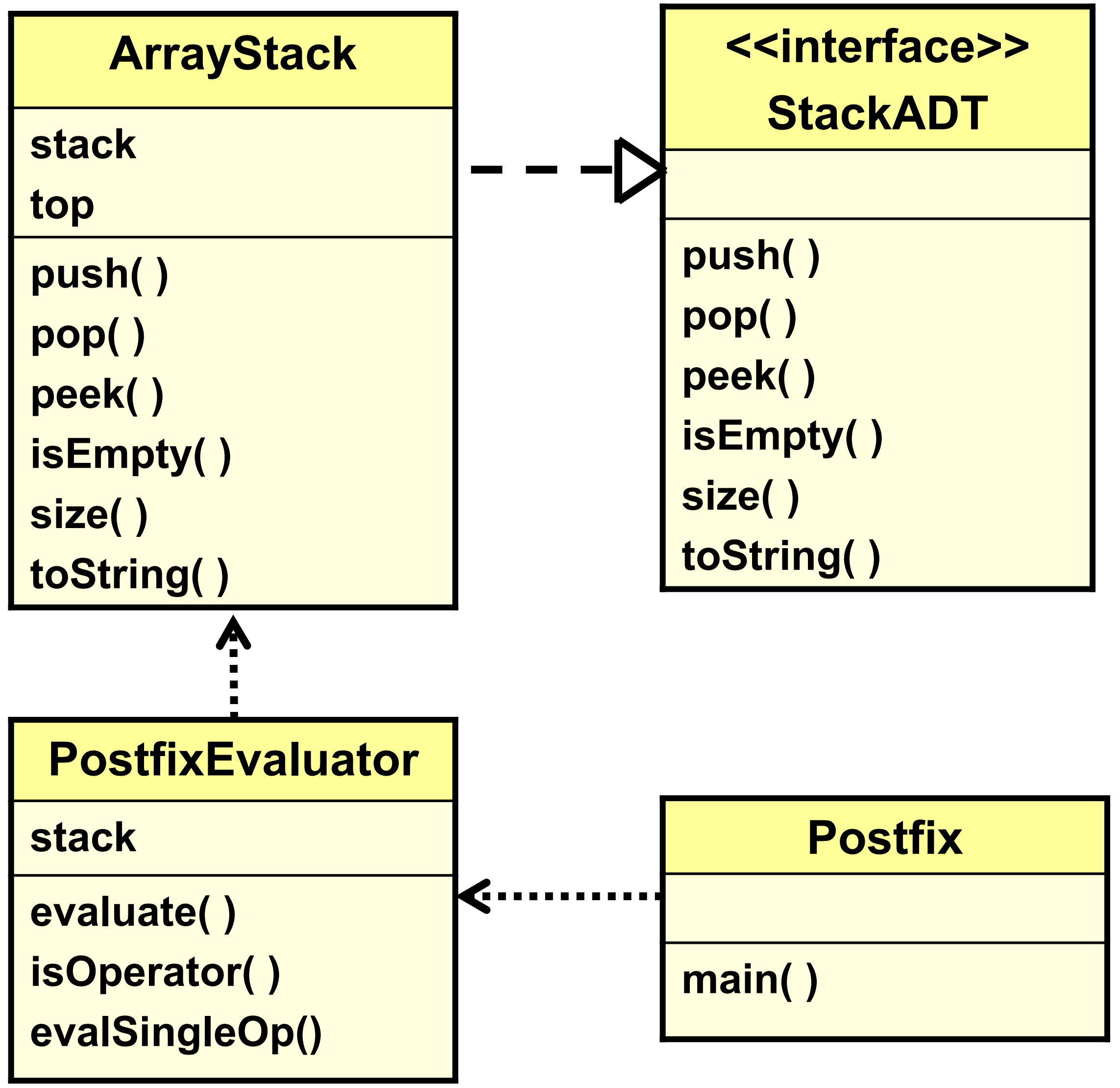
UML Diagram for StackADT Interface



UML Diagram for ArrayStack Implementation of StackADT



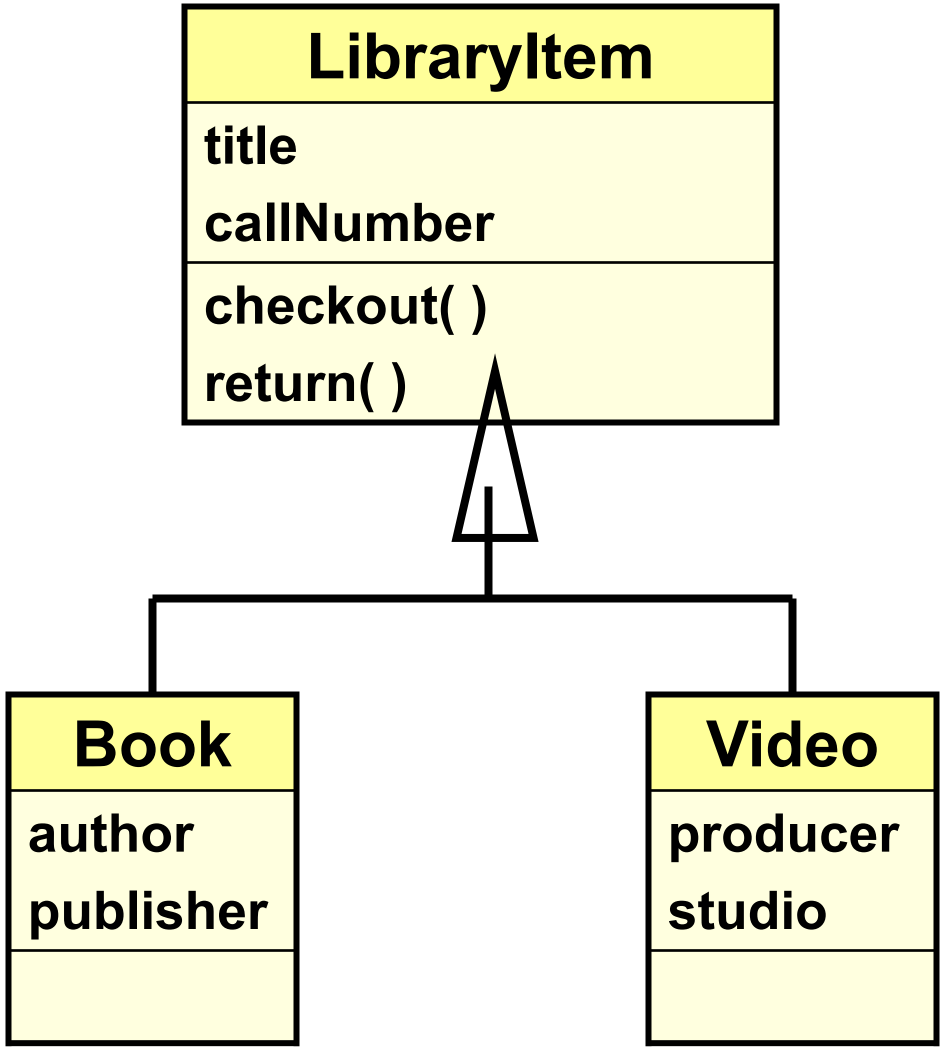
UML Diagram for Postfix Expression Program



Features of Set of UML Diagrams

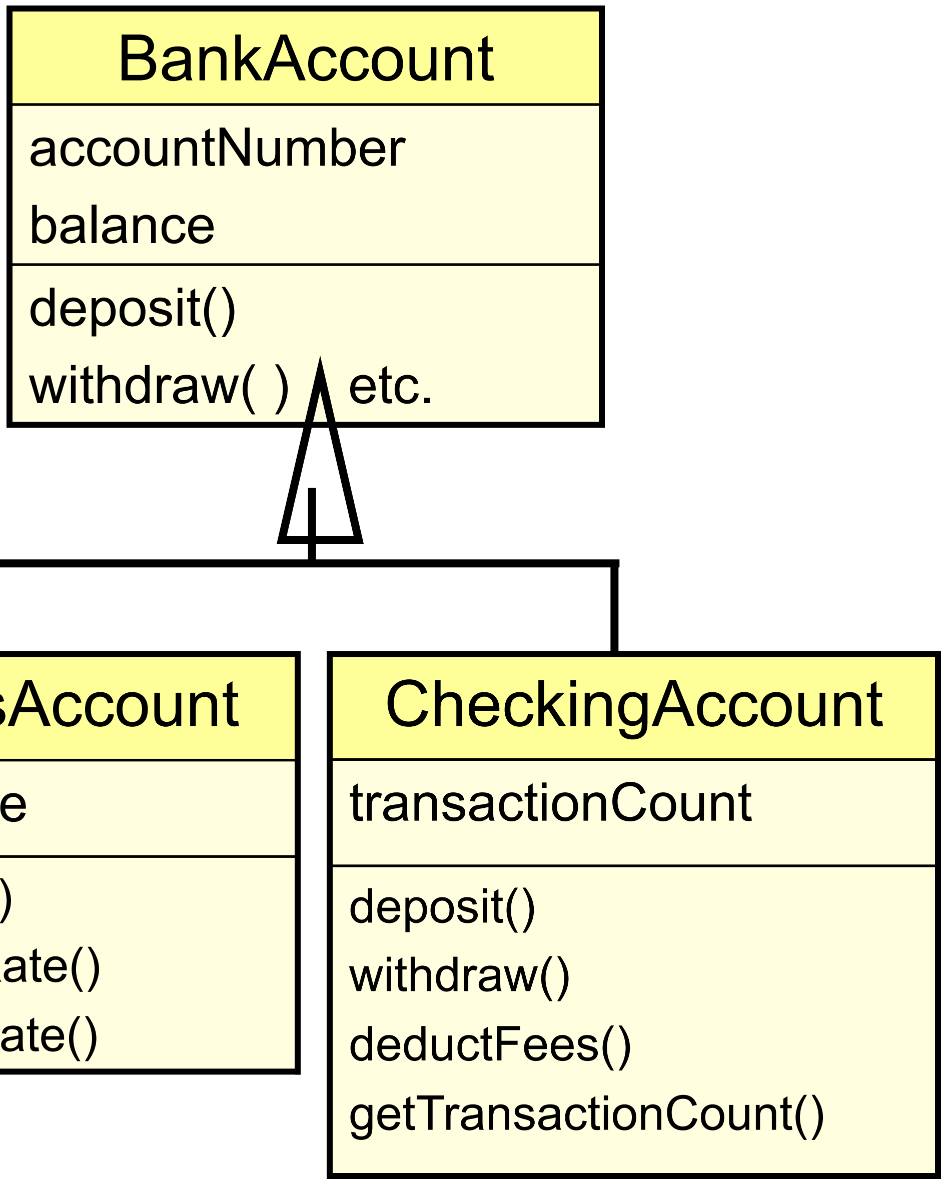
- ***Inheritance:***
  - An arrow on an association line indicates that one class is derived from the other

Example: Inheritance Relationships



1-22

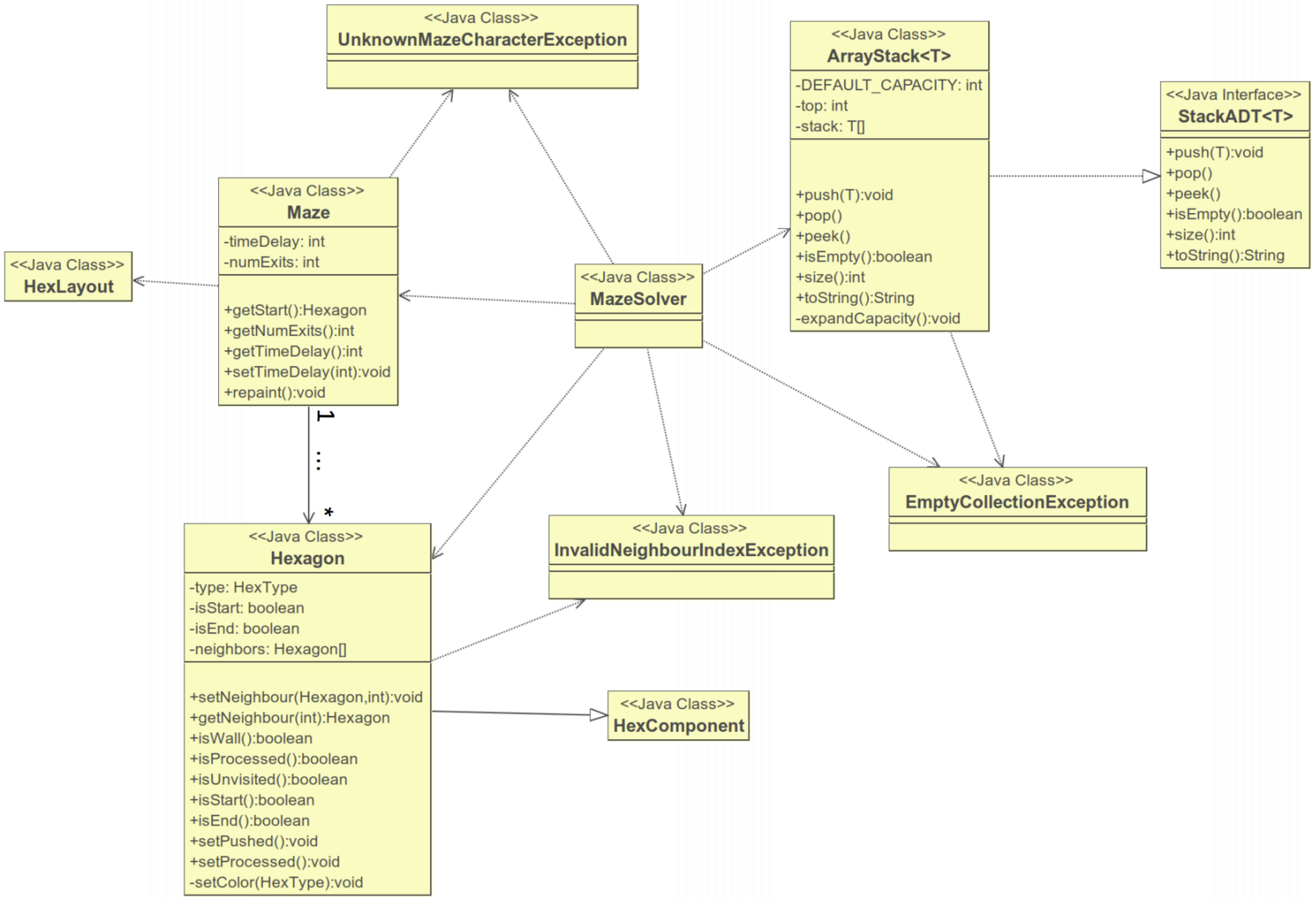
Example: Inheritance Relationships



1-23

Summary

- http://www.classdraw.com/help.htm



<<Java Class>>
UnknownMazeCharacterException
<<Java Class>>
ArrayStack<T>
-DEFAULT_CAPACITY: int
-top: int
-stack: T[]
+push(T):void
+pop()
+peek()
+isEmpty():boolean
+size():int
+toString():String
-expandCapacity():void
<<Java Interface>>
StackADT<T>
+push(T):void
+pop()
+peek()
+isEmpty():boolean
+size():int
+toString():String
<<Java Class>>
Maze
-timeDelay: int
-numExits: int
+getStart():Hexagon
+getNumExits():int
+getTimeDelay():int
+setTimeDelay(int):void
+repaint():void
<<Java Class>>
HexLayout
<<Java Class>>
MazeSolver
1
..
*
<<Java Class>>
EmptyCollectionException
<<Java Class>>
InvalidNeighbourIndexException
<<Java Class>>
Hexagon
-type: HexType
-isStart: boolean
-isEnd: boolean
-neighbors: Hexagon[]
+setNeighbour(Hexagon,int):void
+getNeighbour(int):Hexagon
+isWall():boolean
+isProcessed():boolean
+isUnvisited():boolean
+isStart():boolean
+isEnd():boolean
+setPushed():void
+setProcessed():void
-setColor(HexType):void
<<Java Class>>
HexComponent

Example – Social Network

- Users log into account using FaceSpace credentials.
- Users can view profile pages, which have a profile picture, basic friend information, posts, and links to other features.
- Users can post links, status updates, photos, and videos on their own profile page and on friends' profiles pages.
- Users can send friend requests.
- Users can view a list of their friends.
- Users can review friend requests and accept or reject.

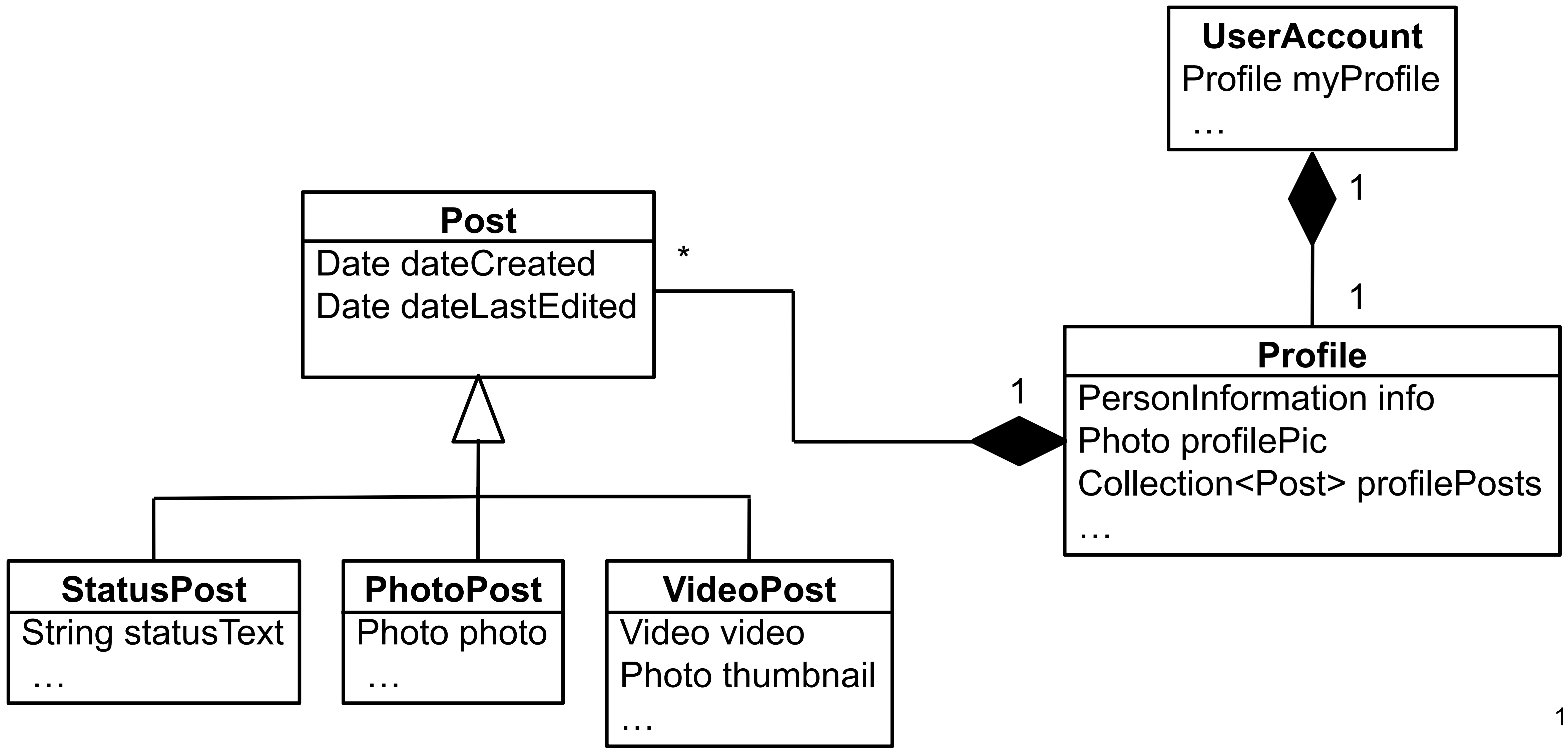
Example – Social Network

Candidate Classes – start with essential data representation.

UserAccount
Profile
PersonInformation
Post
PhotoPost
VideoPost
StatusPost
FriendList
FriendRequest

Any others?

Example – Social Network



Example – Social Network

Candidate Classes – consider program logic. What do we need?

“Users log into account…”

- We need a class to manage authentication.
- UserAuthenticator classes
- For now, sufficient to identify this as a single class
  - We will refine this later

“Users can view profile pages…”

- It is possible that all programming needed to view profiles can be done at the UI level.
- Question: Will the UI read information directly from the ‘model’ level, or do we need an intermediary class to help?

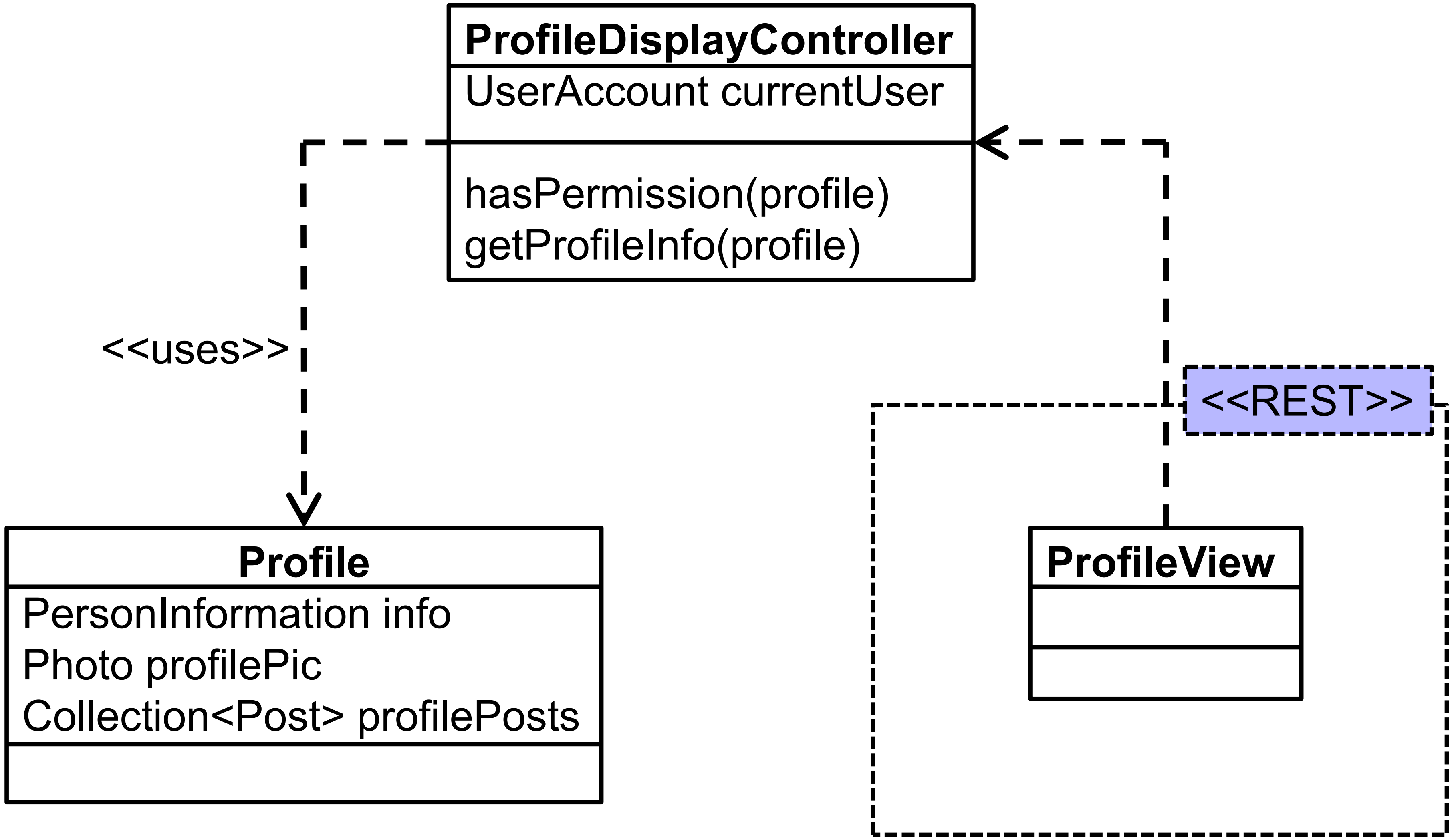
Example – Social Network

Recall that we want as little logic as possible at the domain/model level.

We will need additional programming logic to control which users have access to which profiles.

This suggests we will need a ‘controller’ class to sit between the UI and the model.

Example – Social Network



Example - Grails

```
package facespace

class UserAccount {

    String ownerName

    static hasOne = [userProfile : Profile]

    def getProfile(){
        return userProfile
    }

    static constraints = {
        userProfile nullable: true
    }
}
```

```
package facespace

class Profile {

    static belongsTo = [ownerAccount : UserAccount]
    static hasMany = [posts : Post]

    static constraints = {
        posts nullable: true
    }
}
```

Example - Grails

```
package facespace

class StatusPost extends Post{

    String statusText

    static constraints = {
        statusText(maxSize: 180)
    }
}
```

```
package facespace

class VideoPost extends Post{

    String videoURL
    String thumbnailURL

    static constraints = {
        videoURL nullable: true
        thumbnailURL nullable: true
    }
}
```

Example - Grails

```
package facespace

class BootStrap {

    def init = { servletContext ->
        def acc = new UserAccount(ownerName: 'Ethan').save()
        def p = new Profile(ownerAccount: acc).save()
        new StatusPost(statusText:'This is my first status update!', ownerProfile: p).save()
    }
    def destroy = {
    }
}
```

Example Grails

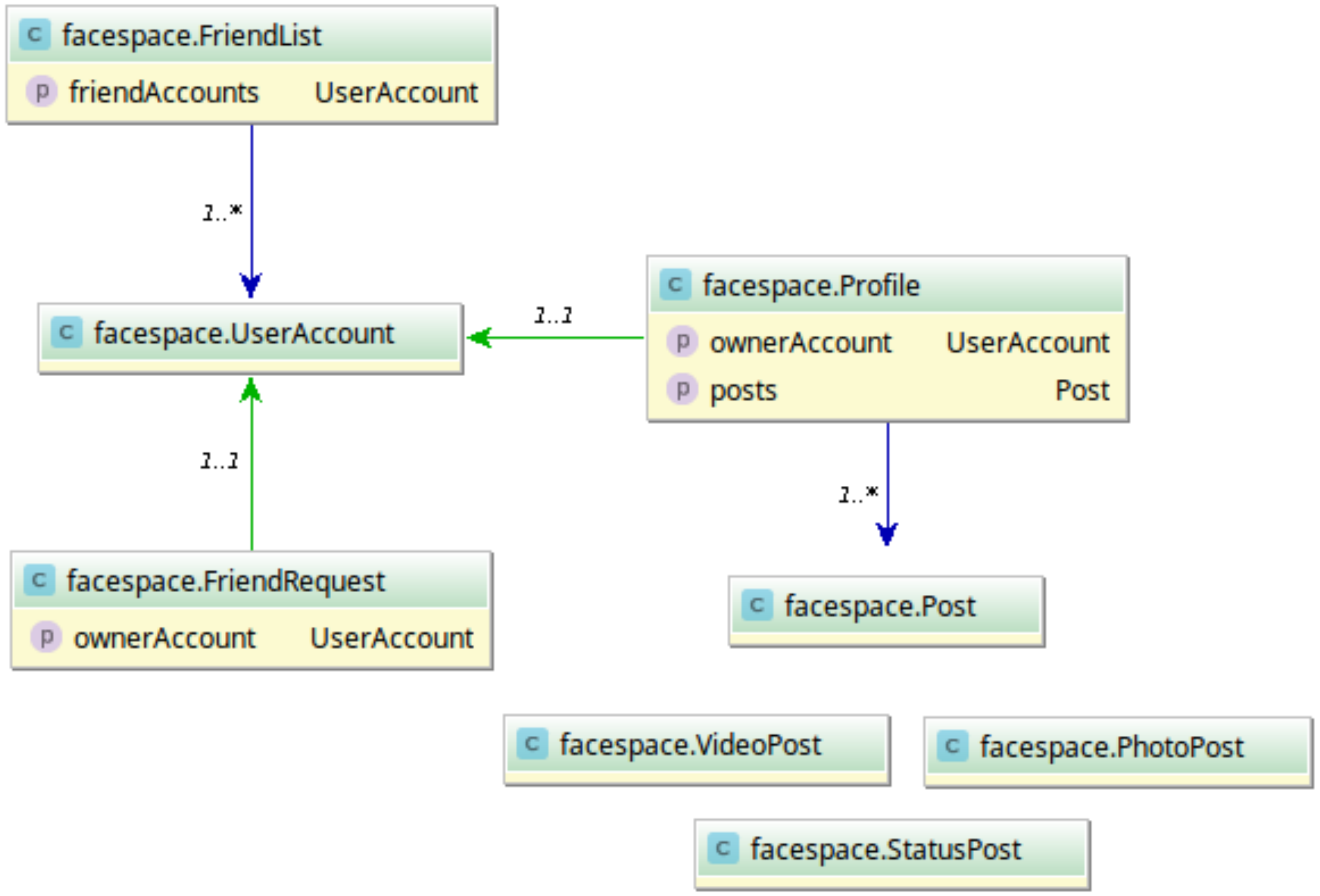
```
package facespace

class ProfileDisplayController {

    def index() {
        def acc = new UserAccount(ownerName: 'Ethan')
        def example = UserAccount.find(acc)

        def posts = UserAccount.find(example).getProfile().getPosts()
        for(Post p : posts){
            if(p instanceof StatusPost){
                String s = p.getStatusText();
                render s
            }
        }
    }
}
```

Example Grails


facespace.FriendList
friendAccounts UserAccount
1..*
facespace.UserAccount
1..1
facespace.Profile
ownerAccount UserAccount
posts Post
1..1
1..*
facespace.FriendRequest
ownerAccount UserAccount
facespace.Post
facespace.VideoPost
facespace.PhotoPost
facespace.StatusPost