– Use case diagrams
– Class diagrams
– Object diagrams
– Sequence diagrams
– Collaboration diagrams
– Statechart diagrams
– Activity diagrams

Objectives

Understand the concept of Classes, Attributes and Operations

Relationships

How to Use Class Diagrams

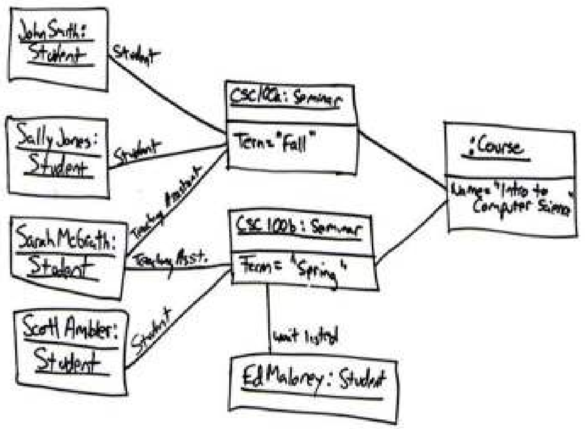
Suggested reading:

Practical UML: A hands on introduction for developers

http://edn.embarcadero.com/article/31863

Classes

- A class describes a group of objects with
  - similar properties (attributes),
  - common behaviour (operations),
  - common relationships to other objects,
  - and common meaning (“semantics”).



Objects and classes are different; a class is a type, an object is an instance

- state and identity is associated with objects

Finding classes

- **Finding classes in use case, or in text descriptions:**

  •Consider **grammatical parses during early stage**

  •Look for **nouns and noun parses** in the description of a use case or a problem statement;

  •These are only included in the model if they explain the nature or structure of information in the application.

Example

The SafeHome security function enables the homeowner to configure the security system when it is installed, monitors all sensors connected to the security system, and interacts with the homeowner through the Internet , a PC or a control panel . During installation , the SafeHome PC is used to program and configure the system . Each sensor is assigned a number and type , a master password is programmed for arming and disarming the system, and telephone number(s) are input for dialing when a sensor event occurs. When a sensor event is recognized , the software invokes an audible alarm attached to the system. After a delay time that is specified by the homeowner during system confi guration activities, the software dials a telephone number of a monitoring service, provides information about the location , reporting the nature of the event that has been detected. The telephone number will be redialed every 20 seconds until telephone connection is obtained.

The homeowner receives security information via a control panel, the PC, or a browser, collectively called an interface . The interface displays prompting messages and system status information on the control panel, the PC, or the browser window. Homeowner interaction takes the following form . . .

Example

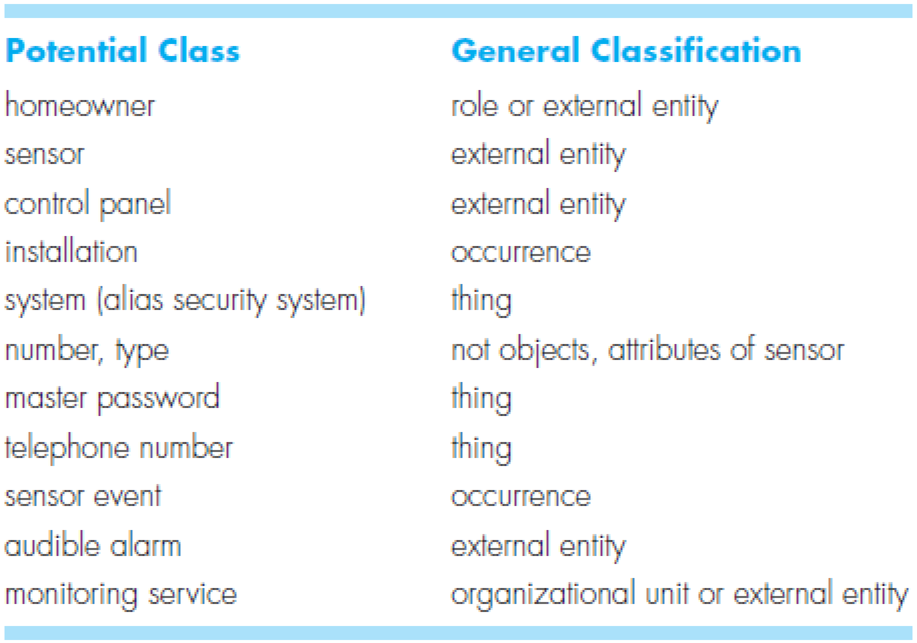
The **SafeHome security function** *enables* the **homeowner** to *configure* the **security system** when it is *installed, monitors* all **sensors** *connected* to the security system, and *interacts* with the homeowner through the **Internet**, a **PC** or a **control panel**.

During **installation**, the SafeHome PC is used to *program* and *configure* the **system**. Each sensor is assigned a **number** and **type**, a **master password** is programmed for *arming* and *disarming* the system, and **telephone number(s)** are *input* for *dialing* when a **sensor event** occurs. When a sensor event is *recognized* , the software *invokes* an **audible alarm** attached to the system. After a **delay time** that is *specified* by the homeowner during system confi guration activities, the software dials a telephone number of a **monitoring service**, *provides* **information** about the **location**, *reporting* the nature of the event that has been detected. The telephone number will be *redialed* every 20 seconds until **telephone connection** is *obtained.*

The homeowner *receives* **security information** via a control panel, the PC, or a browser, collectively called an **interface**. The interface *displays* **prompting messages** and **system status information** on the control panel, the PC, or the browser window. Homeowner interaction takes the following form . . .

Example

- Extracting the nouns, we can propose a number of potential classes:



Potential Class	General Classification
homeowner	role or external entity
sensor	external entity
control panel	external entity
installation	occurrence
system (alias security system)	thing
number, type	not objects, attributes of sensor
master password	thing
telephone number	thing
sensor event	occurrence
audible alarm	external entity
monitoring service	organizational unit or external entity

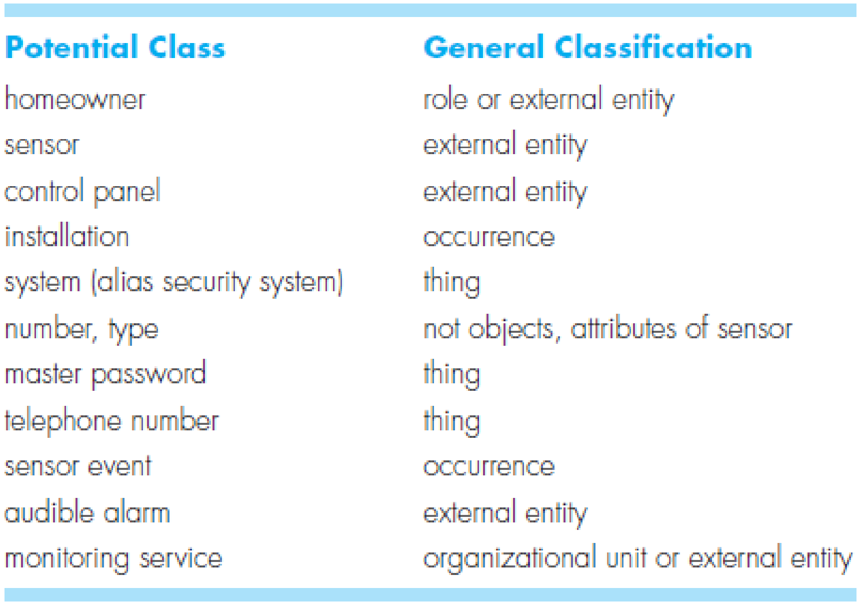
Further

Coad and Yourdon [Coa91] suggest six selection characteristics that **should be used as you consider each potential class for inclusion in the analysis model**:

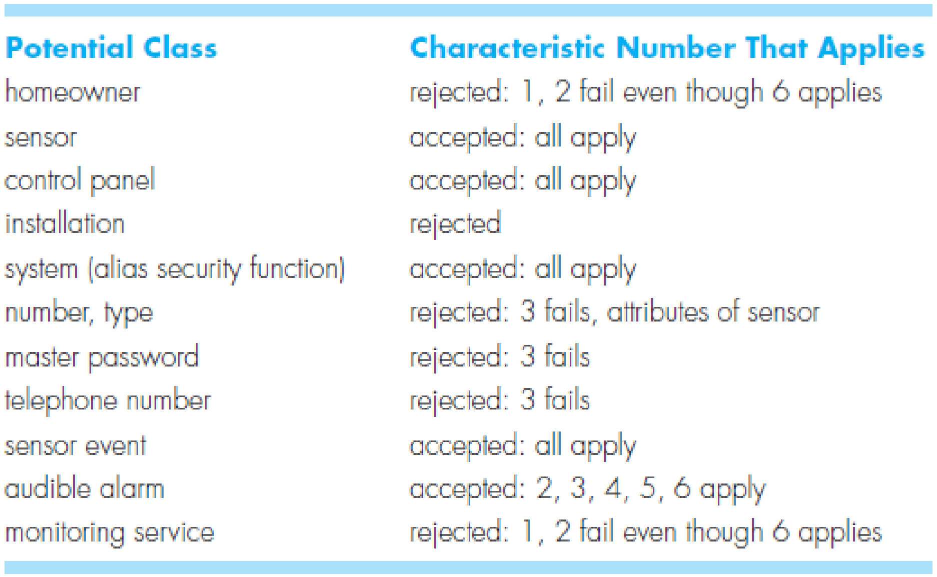
Retained information	Needed services	Multiple attributes	Common attributes	Common operations	Essential requirements
• The potential class will be useful during analysis only if information about it must be **remembered so that the system can function.**	• The potential class **must have a set of identifiable operations** that can change the value of its attributes in some way.	• During requirement analysis, the focus should be on “major” information; a class with a single attribute may, in fact, be useful during design, but is probably better represented as an attribute of another class during the analysis activity.	• A set of attributes can be defined for the potential class and these attributes apply to all instances of the class.	• A set of operations can be defined for the potential class and these operations apply to all instances of the class.	• External entities that appear in the problem space and produce or consume information essential to the operation of any solution for the system will almost always be defined as classes in the requirements model.

Further

Retained information	Needed services	Multiple attributes	Common attributes	Common operations	Essential requirements

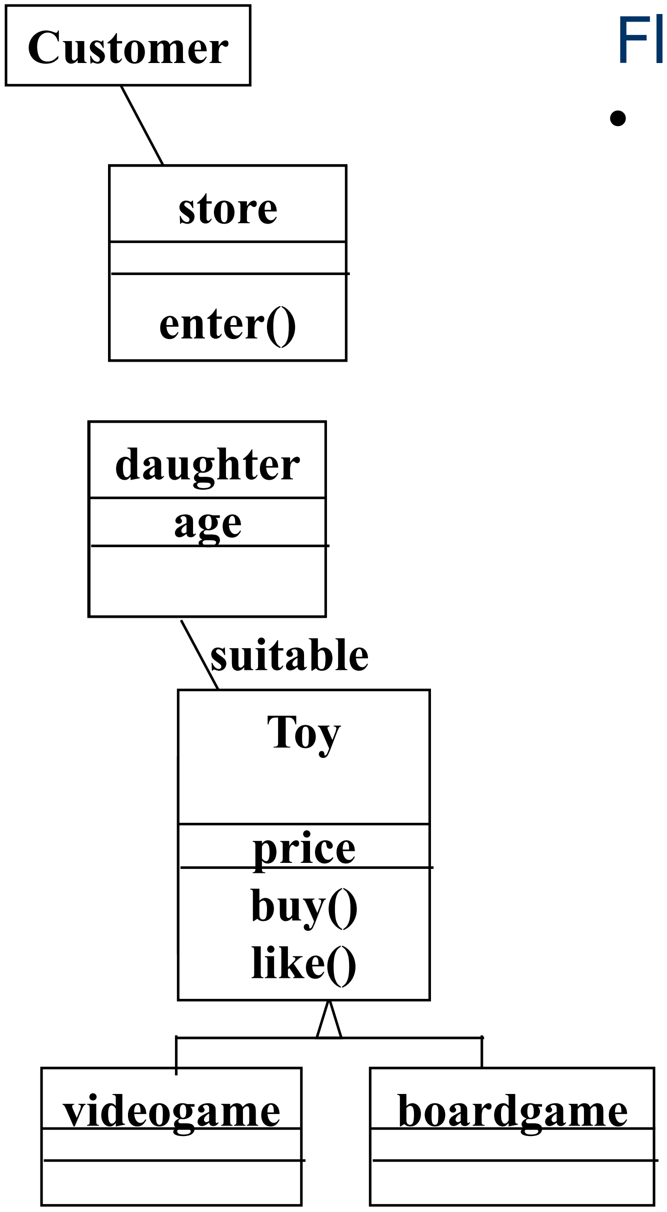


Potential Class	General Classification
homeowner	role or external entity
sensor	external entity
control panel	external entity
installation	occurrence
system (alias security system)	thing
number, type	not objects, attributes of sensor
master password	thing
telephone number	thing
sensor event	occurrence
audible alarm	external entity
monitoring service	organizational unit or external entity



Potential Class	Characteristic Number That Applies
homeowner	rejected: 1, 2 fail even though 6 applies
sensor	accepted: all apply
control panel	accepted: all apply
installation	rejected
system (alias security function)	accepted: all apply
number, type	rejected: 3 fails, attributes of sensor
master password	rejected: 3 fails
telephone number	rejected: 3 fails
sensor event	accepted: all apply
audible alarm	accepted: 2, 3, 4, 5, 6 apply
monitoring service	rejected: 1, 2 fail even though 6 applies

Generating a Class Diagram from Flow of Events



Flow of events:

- The customer enters the store to buy a toy. It has to be a toy that his daughter likes and it must cost less than 50 Euro. He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

  An assistant helps him. The suitability of the game depends on the age of the child. His daughter is only 3 years old. The assistant recommends another type of toy, namely a boardgame. The customer buy the game and leaves the store

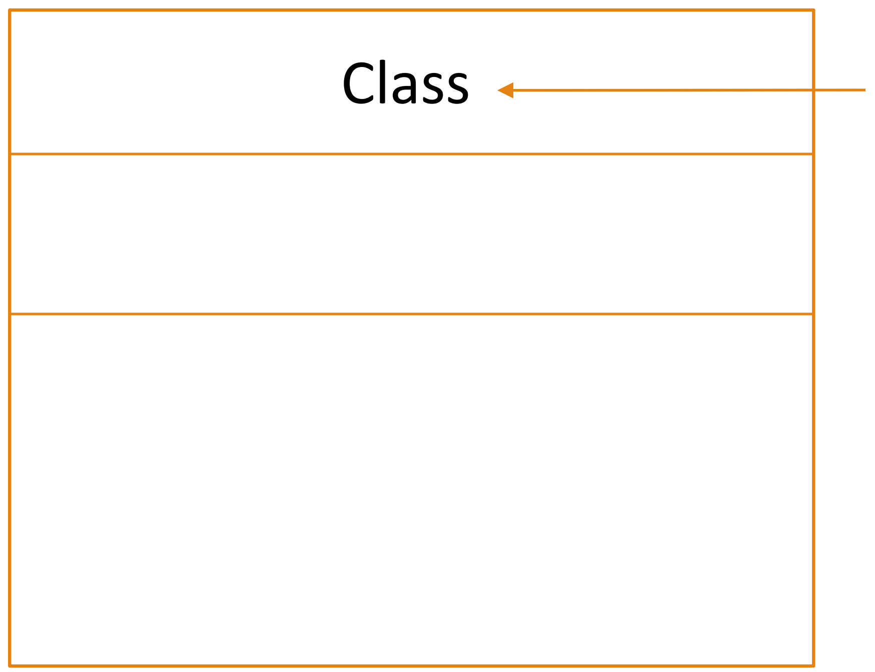
Finding classes

- **Don’t create classes for concepts which:**
  - Are beyond the scope of the system;
  - Refer to the system as a whole;
  - Duplicate other classes;
  - Are too vague or too specific (few instances);

Finding classes

- **Finding classes in other sources:**
  - Reviewing background information;
  - Users and other stakeholders;
  - Analysis patterns;
  - CRC (Class Responsibility Collaboration) cards.

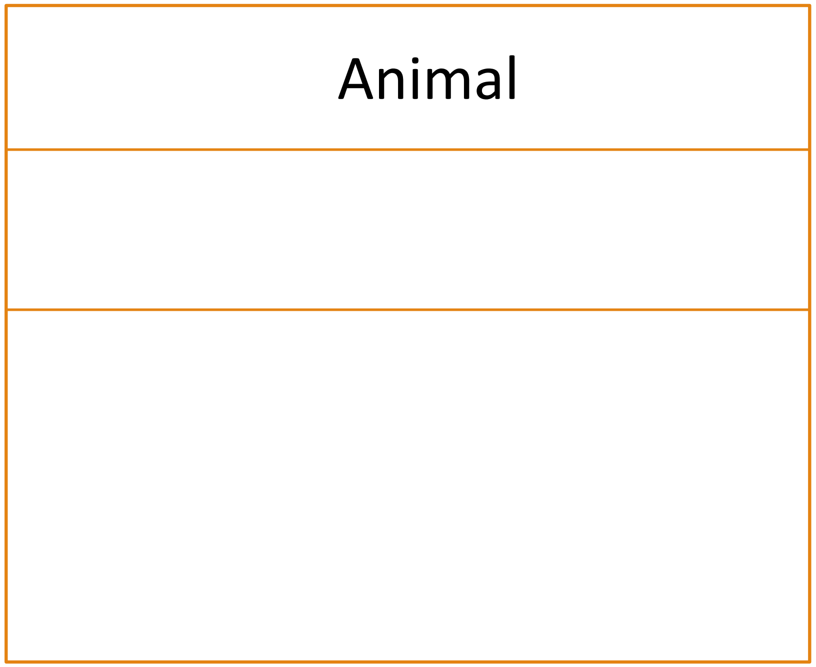
Class



- Mandatory
- Must have unique name

Example: Zoo system

- Tiger
- Bear
- Panda
- Snake
- Lion
- ……

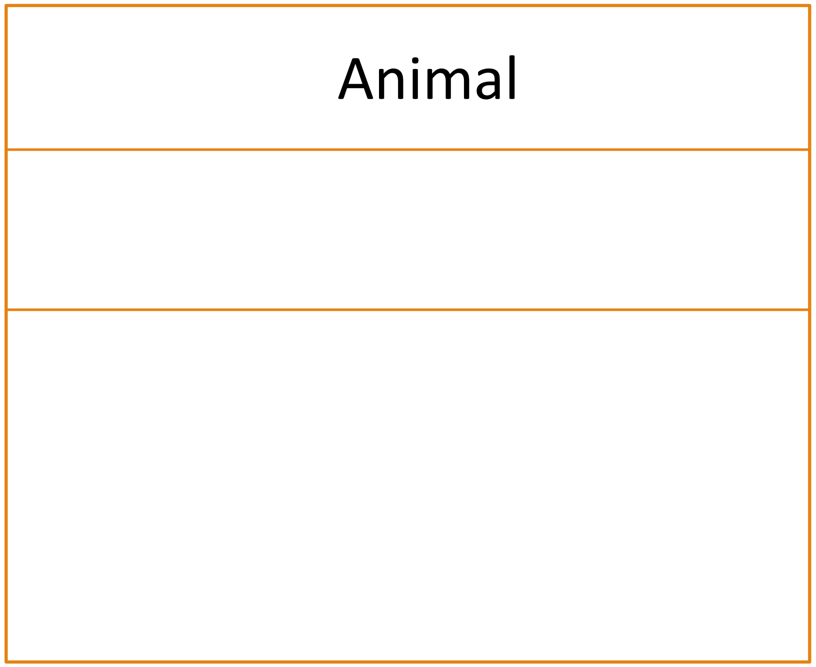


- Must have unique name

Example: Zoo system

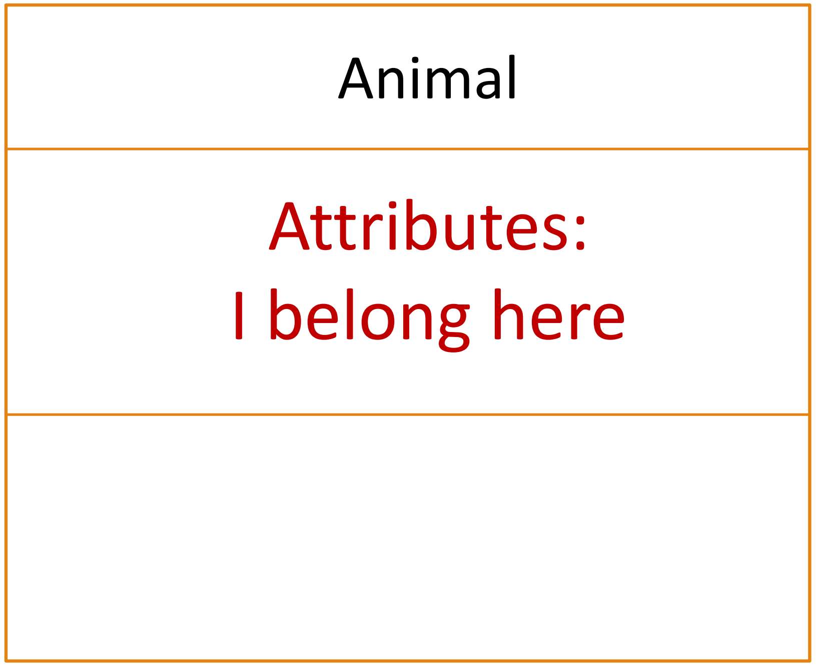
- Tiger
- Bear
- Panda
- Snake
- Lion
- ……

Instances of animal class



You do that through…..

Attributes



Describes a class that has been selected for inclusion in the analysis of model

Also known as fields, variables or properties

So where do they go?

represent useful information about instances of a class.

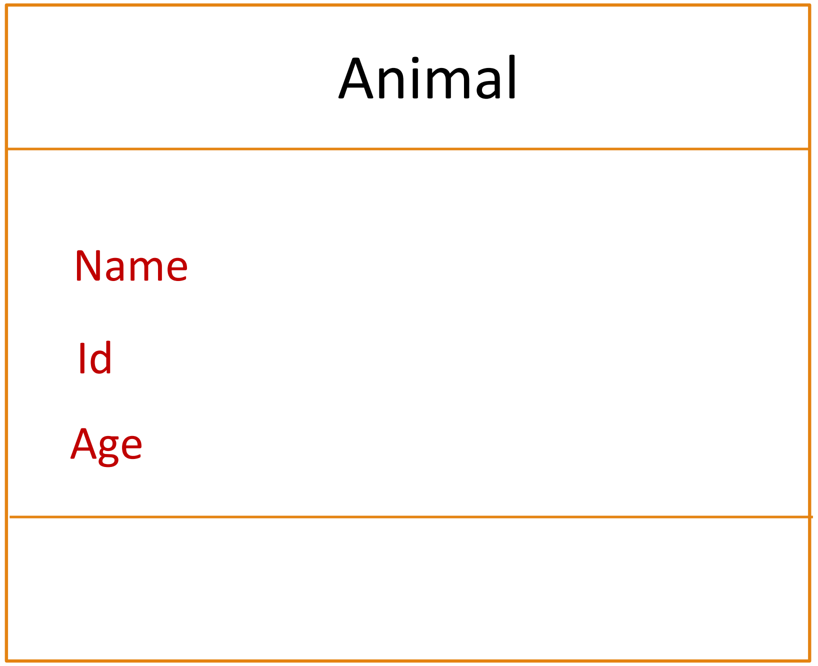
Attributes

- Tiger

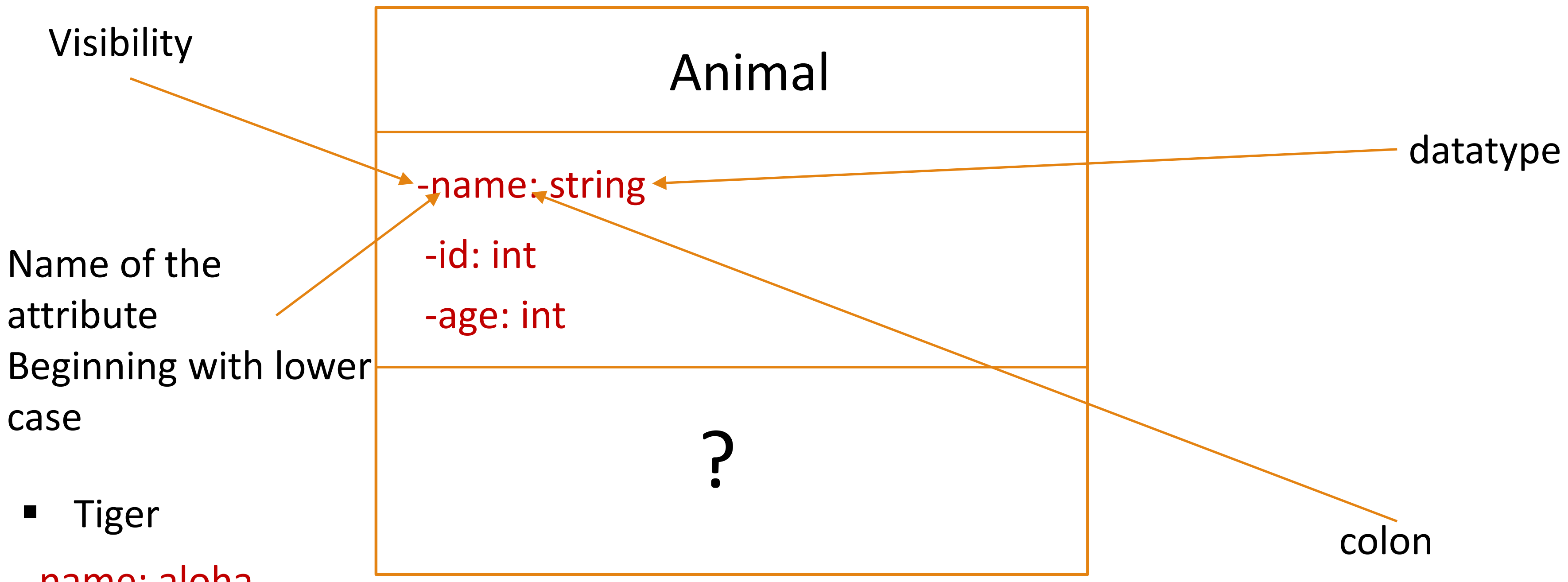
Name: aloha

Id: 155

Age: 10



Attributes



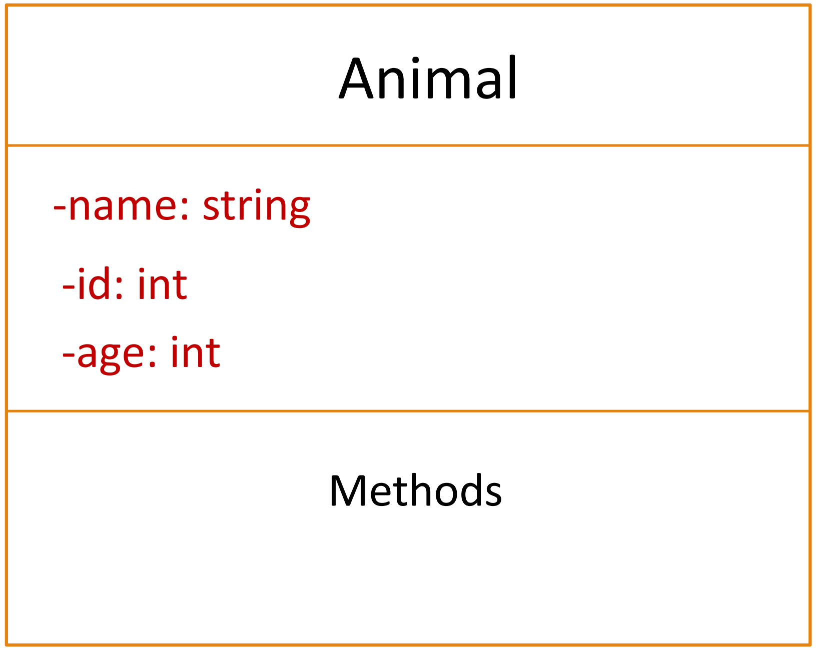
- Tiger

-name: aloha

-id: 155

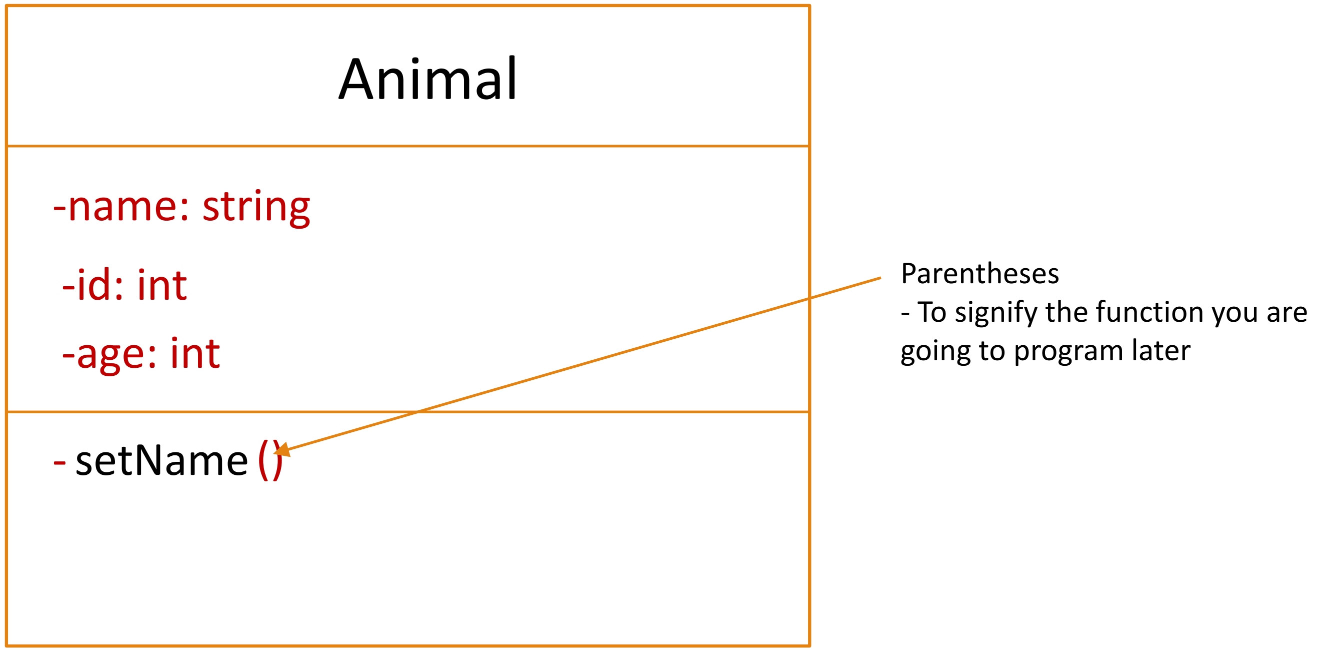
-age: 10

Methods



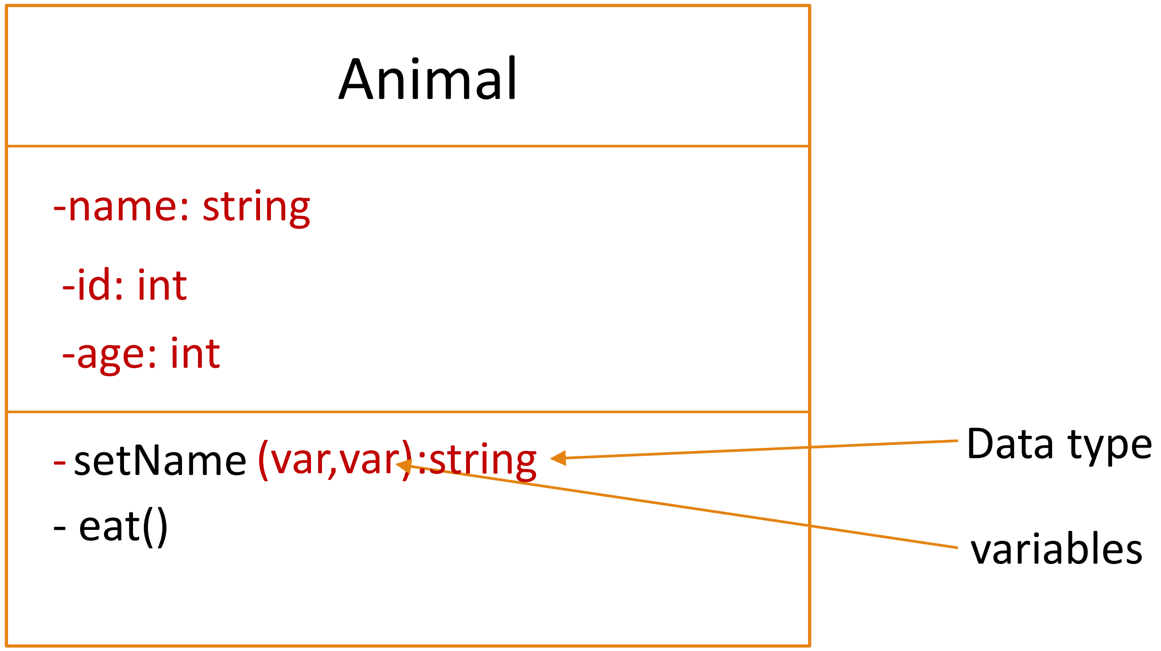
- also known as operations or functions
- Operations define the behavior of an object.
- Allow you to specify any behavioral feature of a class
- Must have the knowledge of the nature of the class attributes

Methods

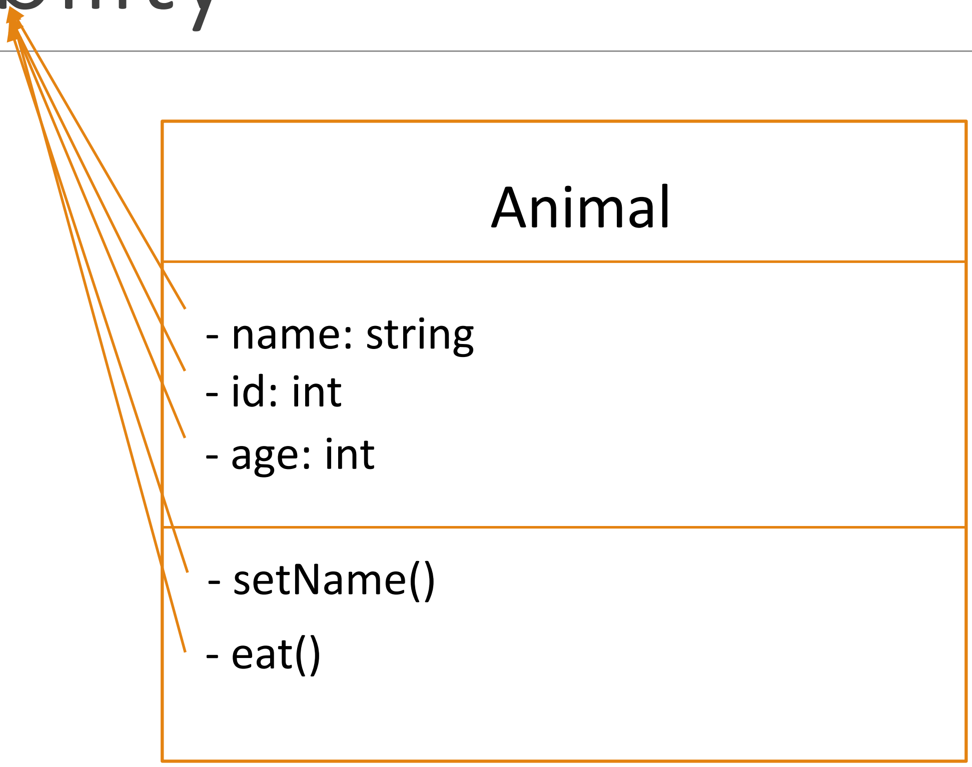


Some of these verbs will be legitimate operations and can be easily connected to a specific class.

Methods


Animal
-name: string
-id: int
-age: int
- setName (var,var):string
- eat()
Data type
variables

Visibility



Visibility of an attribute or a method sets the accessibility for that attribute or method

- private cant be accessed by any other class or sub class

+ public can be accessed by any other class or sub class

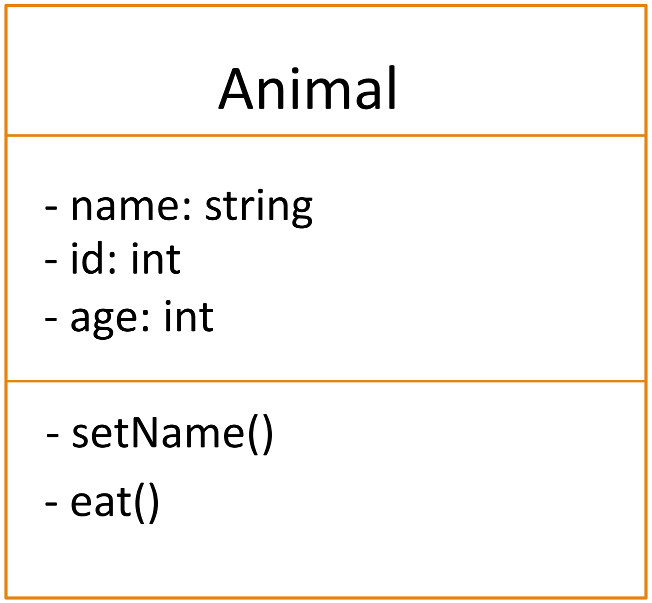
protected

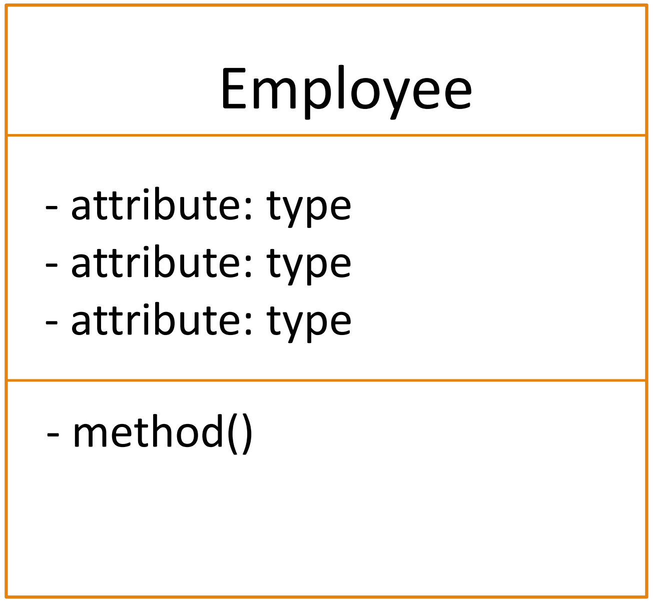
Means attribute or method is protected
Can be access by same class or sub class

~ Package/default

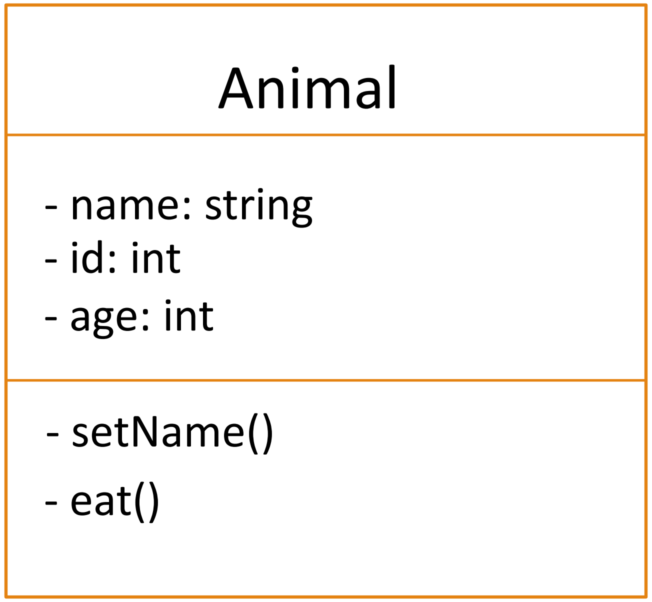
Sets the visibility to package or default which means it can be used by any other class as long as its in same package

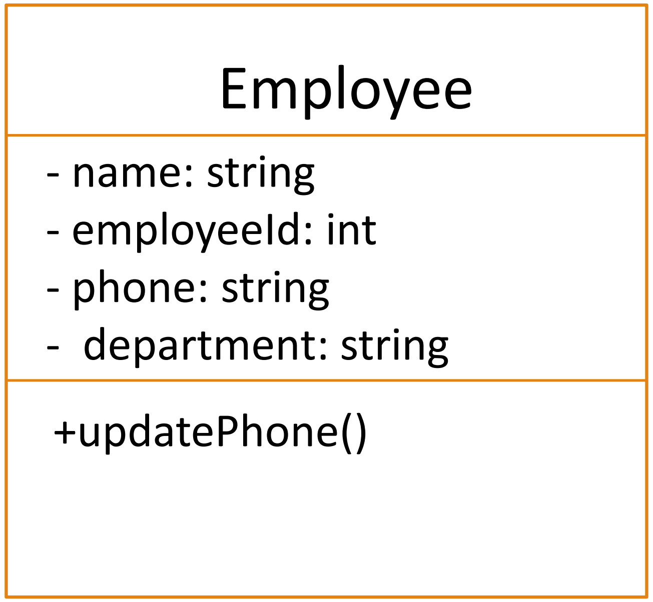
Example


Animal
- name: string
- id: int
- age: int
- setName()
- eat()


Employee
- attribute: type
- attribute: type
- attribute: type
- method()

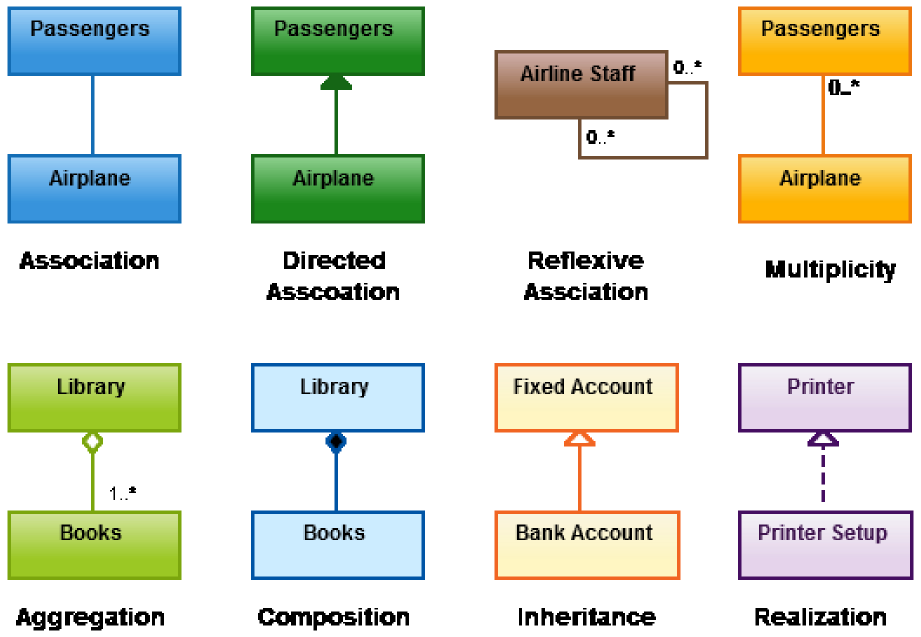
Example


Animal
- name: string
- id: int
- age: int
- setName()
- eat()


Employee
- name: string
- employeeId: int
- phone: string
- department: string
+updatePhone()

Relationships

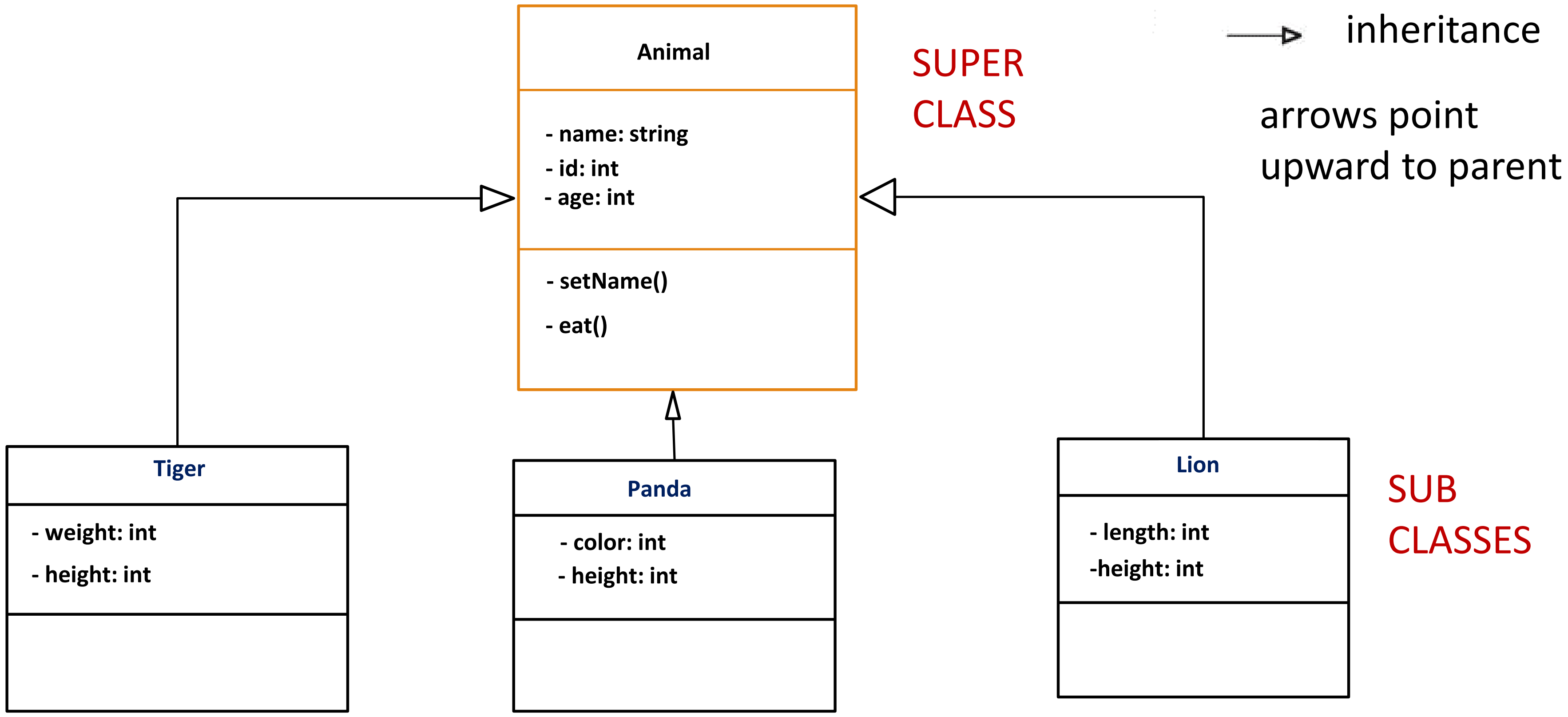
- Classes and objects do not exist in isolation from one another
- A relationship represents a connection among things.
- In UML, there are different types of relationships:
  - Association
    - Aggregation
    - Composition
  - …more…



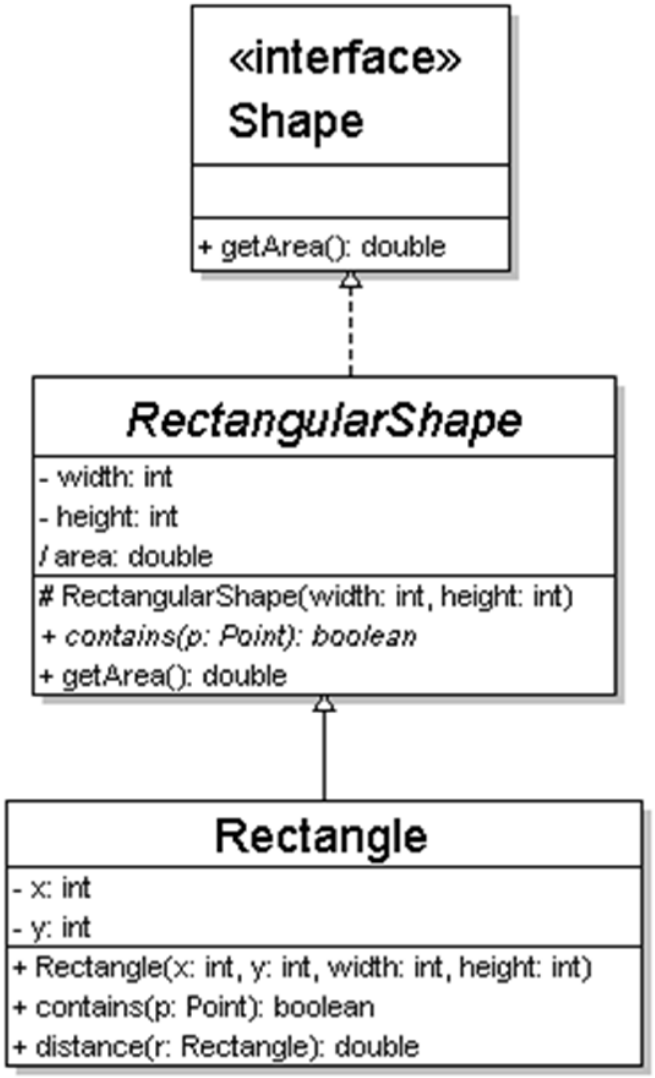
http://creately.com/blog/diagrams/class-diagram-relationships/

Relationships: example

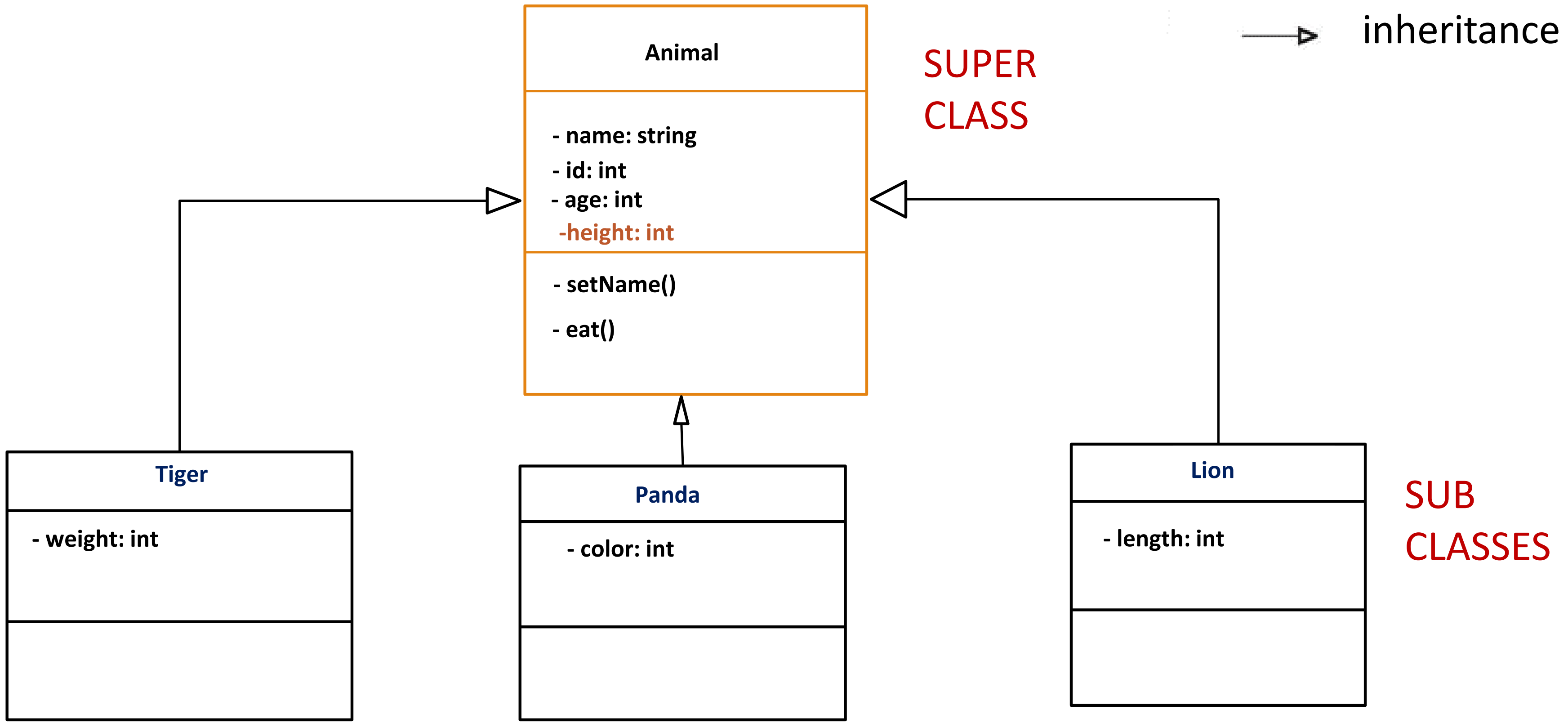
Generalization (inheritance) relationships



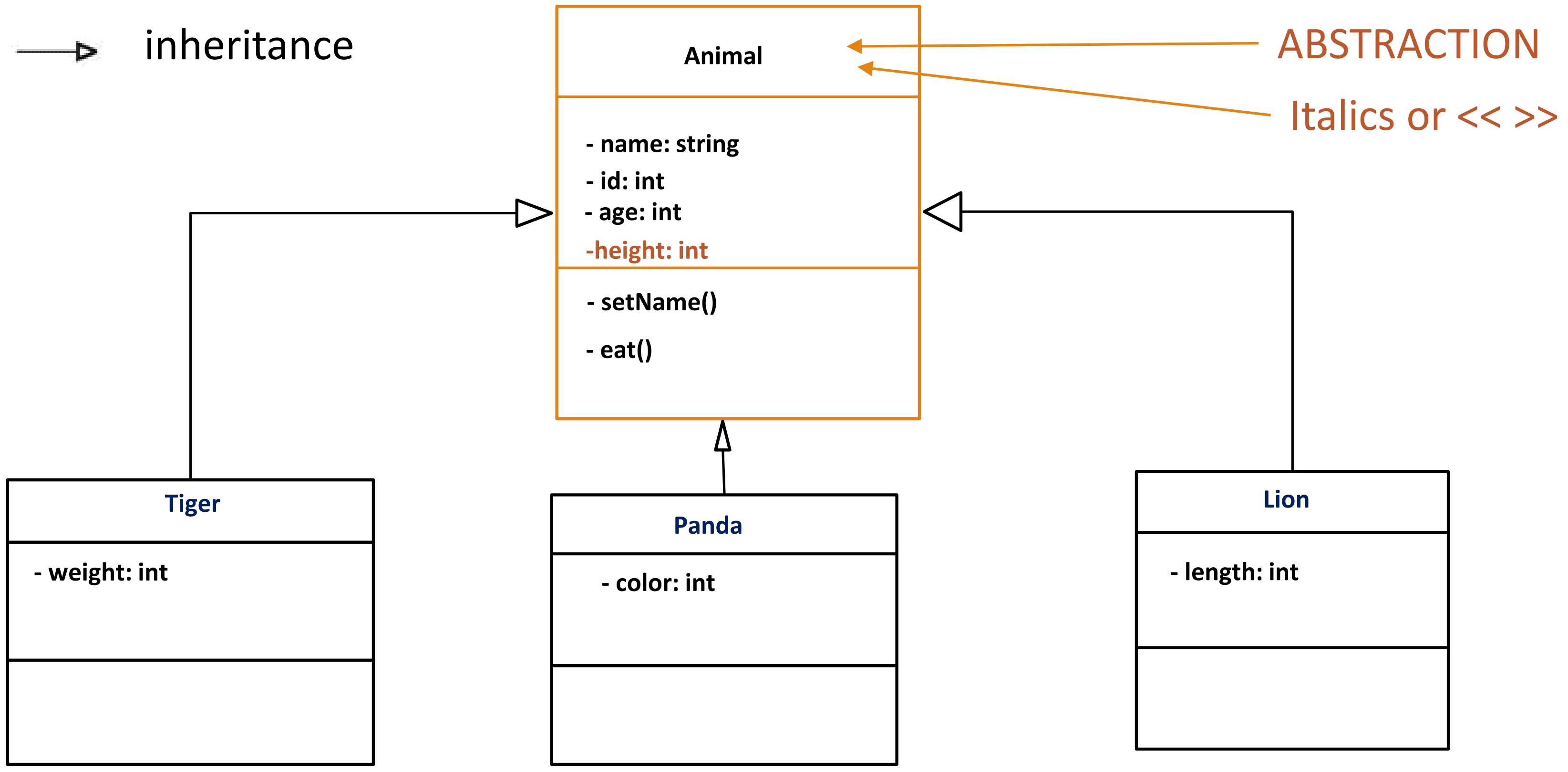
- hierarchies drawn top-down
- arrows point upward to parent
- line/arrow styles indicate whether parent is a(n):
  - <u>class</u>:
    solid line, black arrow
  - <u>abstract class</u>:
    solid line, white arrow
  - <u>interface</u>:
    dashed line, white arrow
- often omit trivial / obvious generalization relationships, such as drawing the Object class as a parent


«interface»
Shape
+ getArea(): double
RectangularShape
- width: int
- height: int
/ area: double
# RectangularShape(width: int, height: int)
+ contains(p: Point): boolean
+ getArea(): double
Rectangle
- x: int
- y: int
+ Rectangle(x: int, y: int, width: int, height: int)
+ contains(p: Point): boolean
+ distance(r: Rectangle): double

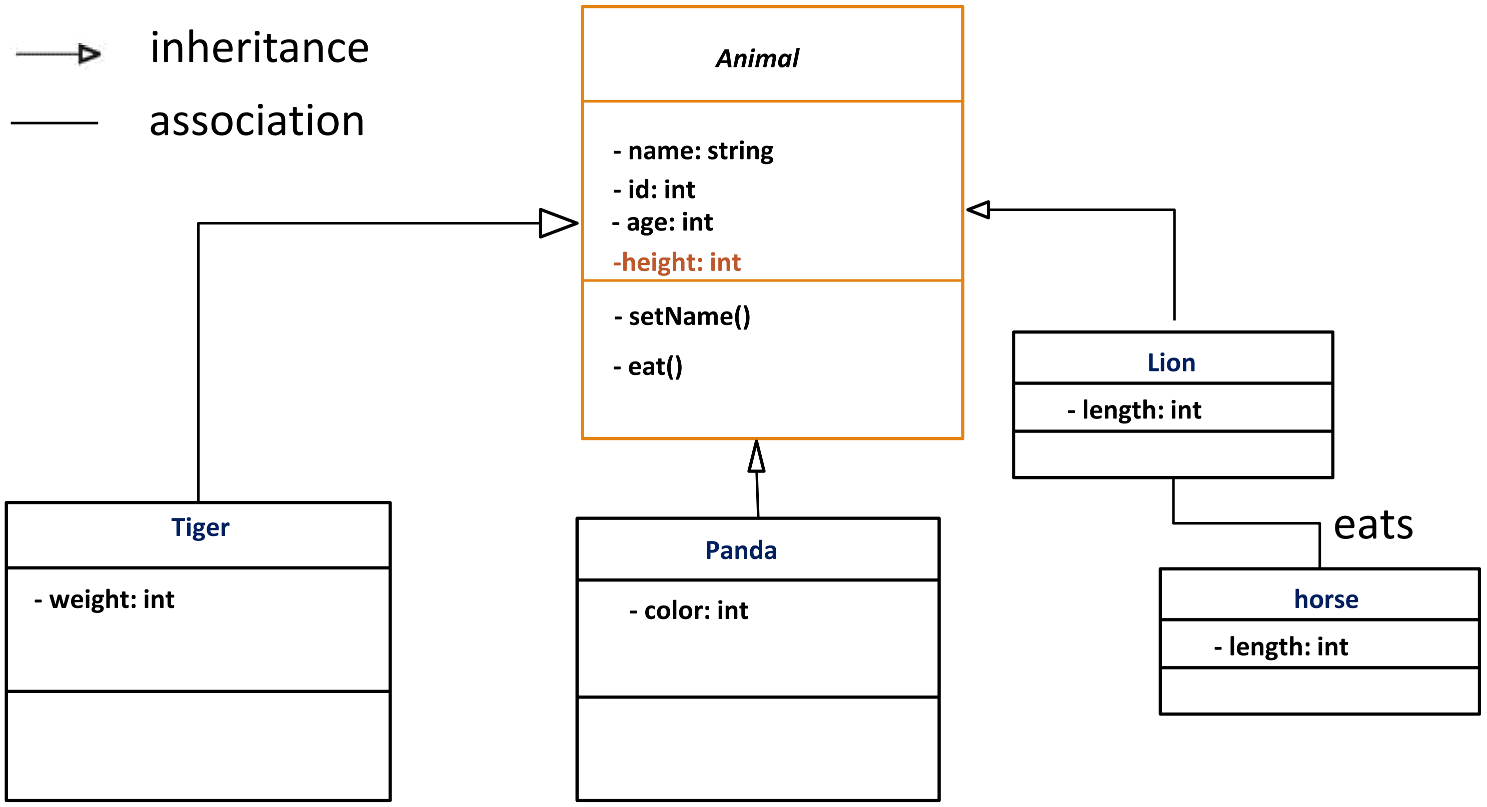
Relationships: example



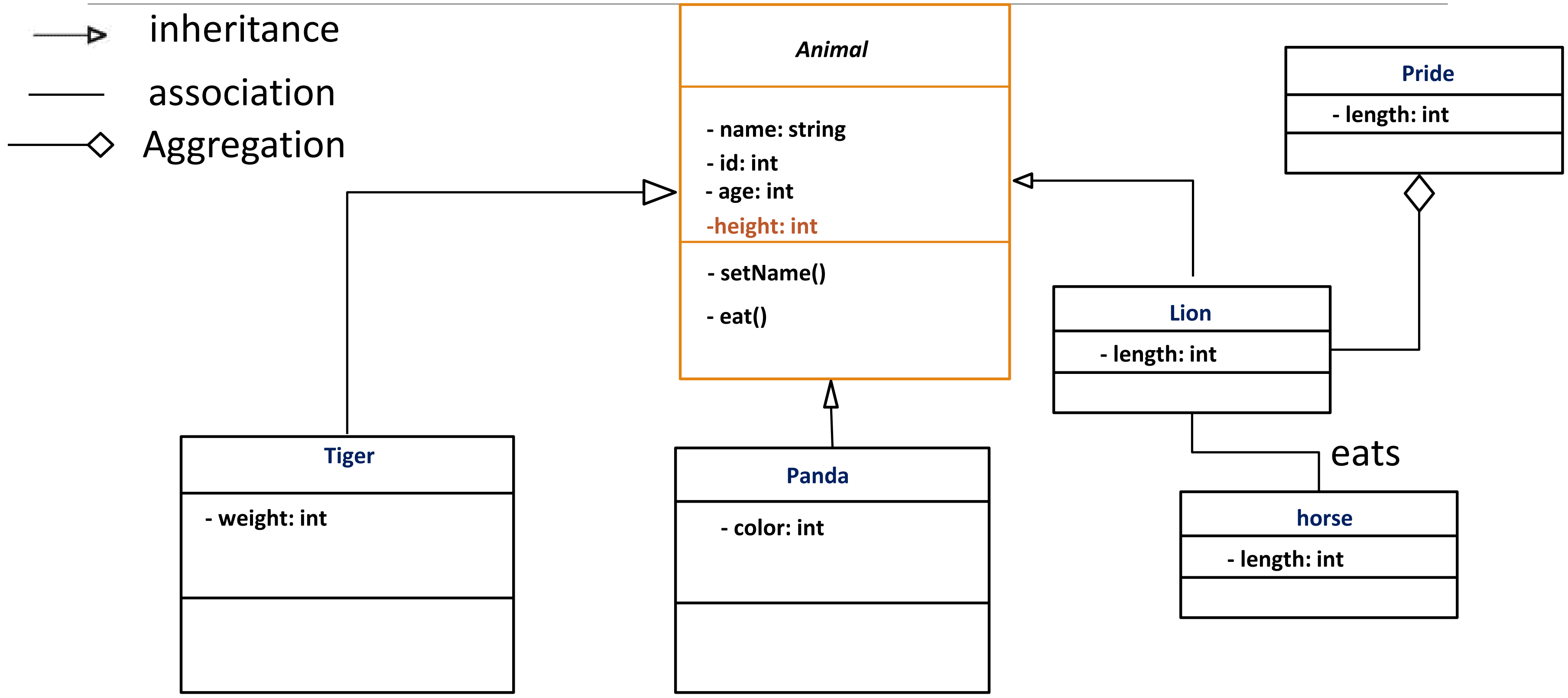
Relationships



Relationships

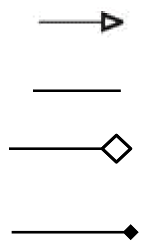


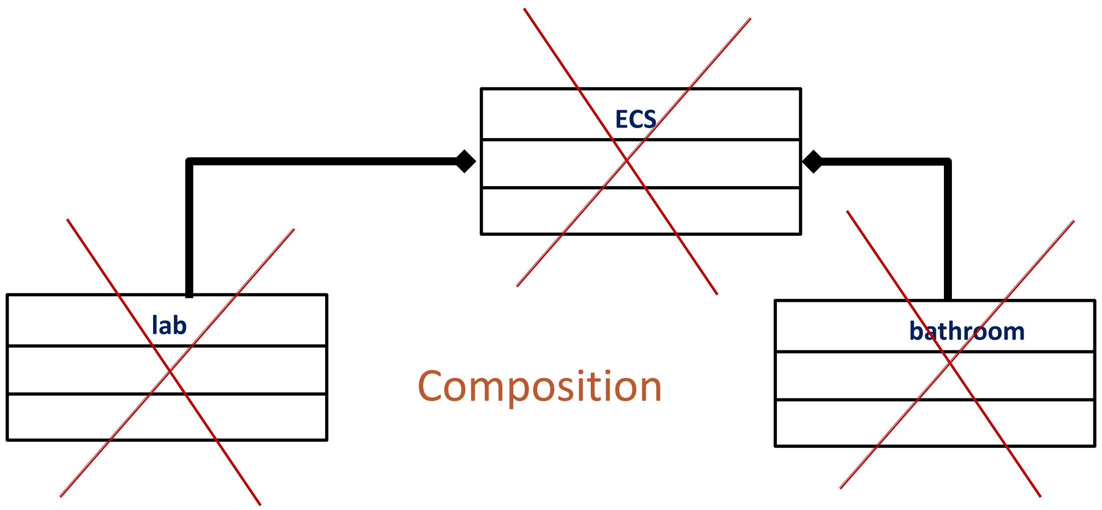
Relationships



Special type of association that specifies a while and its parts

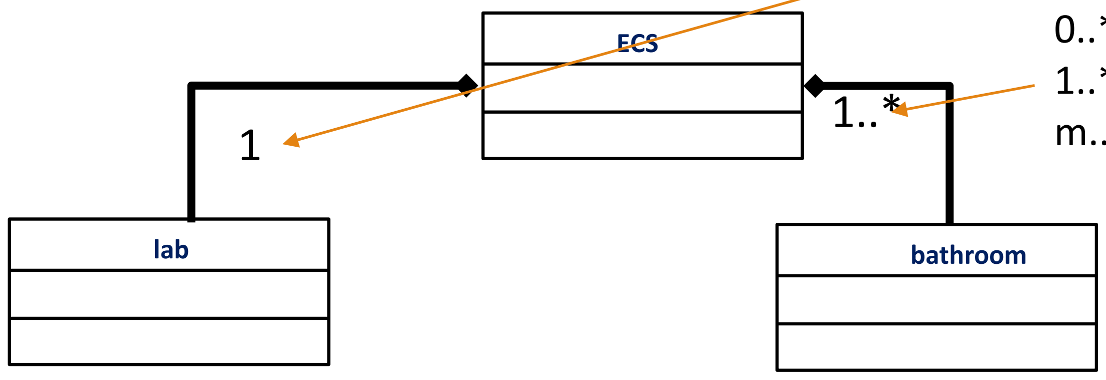
Relationships





When a child object wouldn't be able to exist without its parent object.

Multiplicity

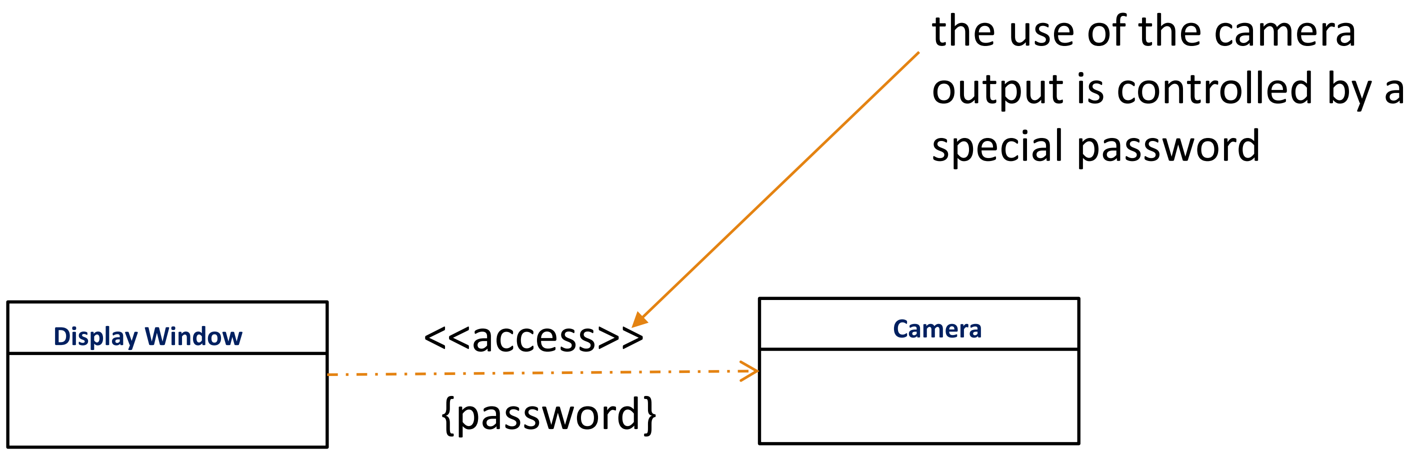


0..1 zero to one (optional)
N specific number
0..* Zero to many
1..* one to many
m..n specific number range

Allows you to set numerical constraints on your relationships.

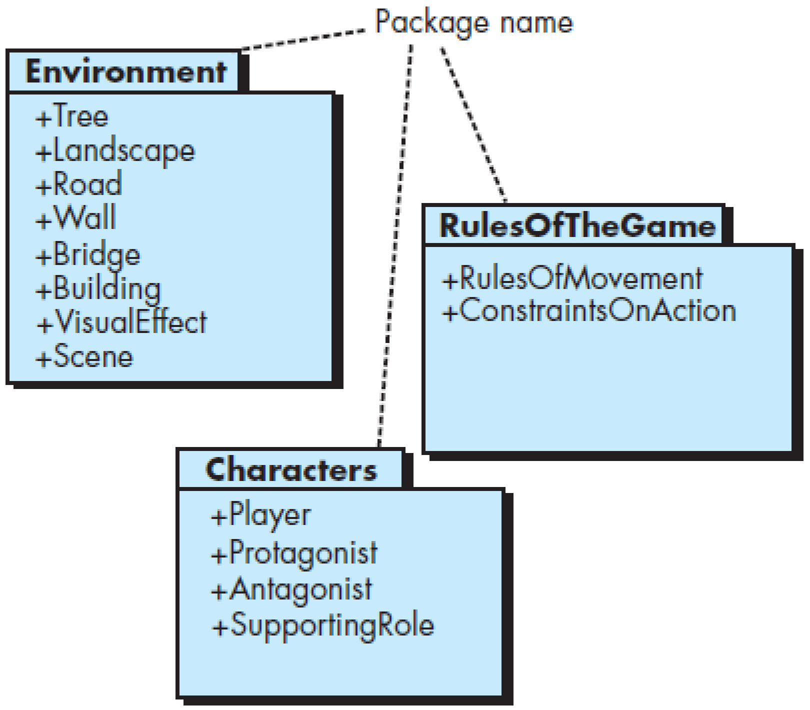
Dependencies

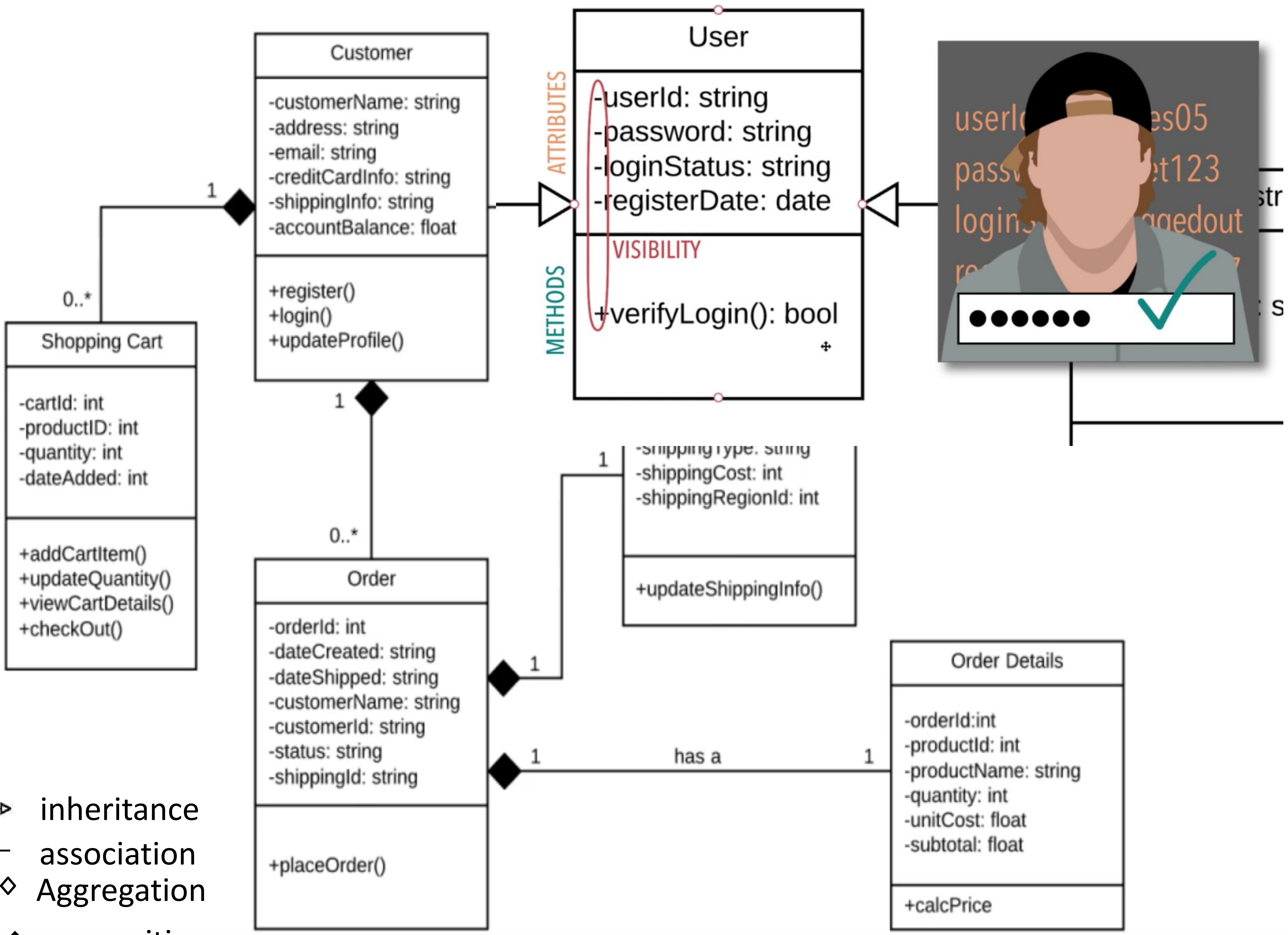
relationship between two objects is not a simple association


the use of the camera output is controlled by a special password
Display Window
<<access>>
{password}
Camera

Analysis Packages

categorized in a manner that packages them as a grouping



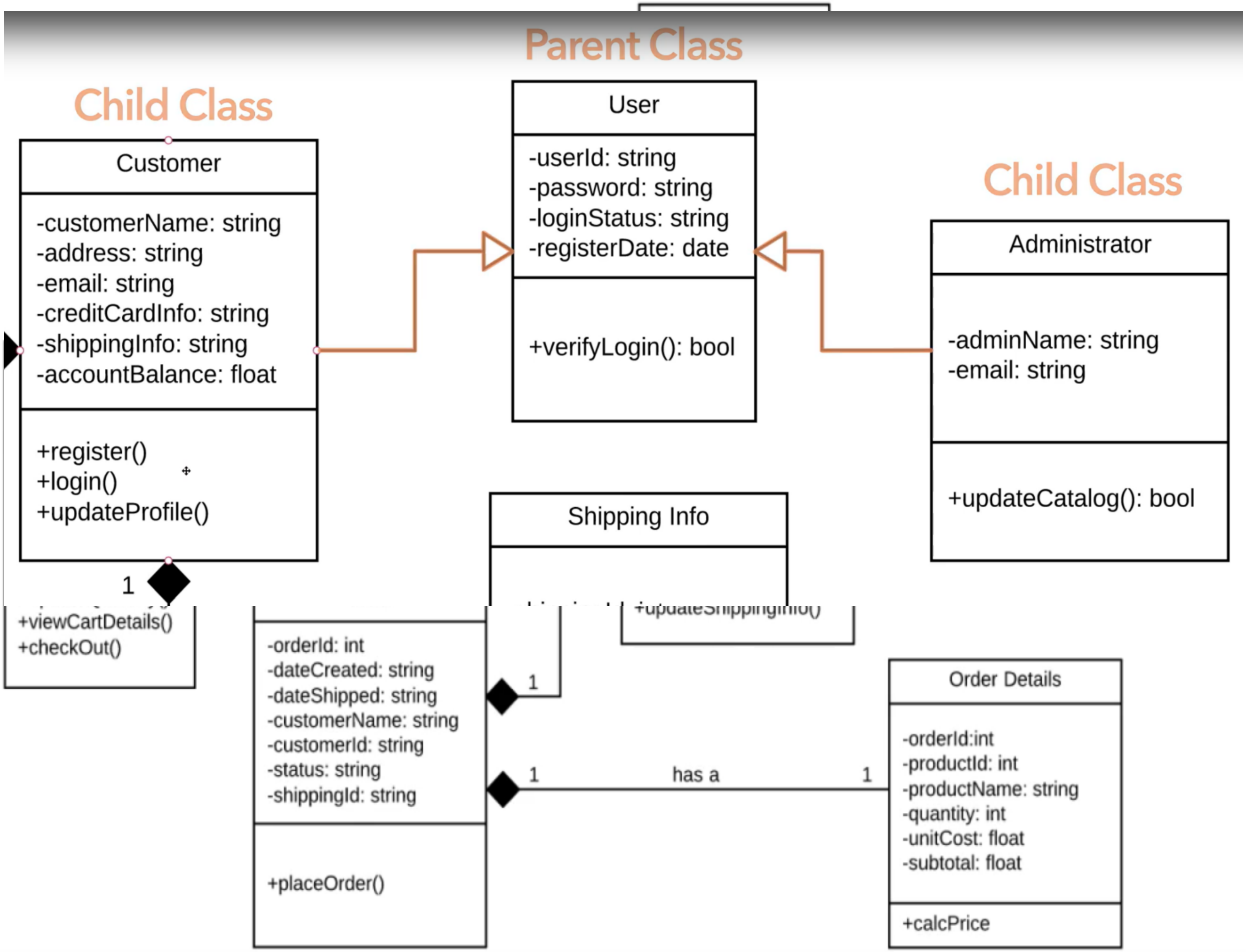


⟶▷ inheritance

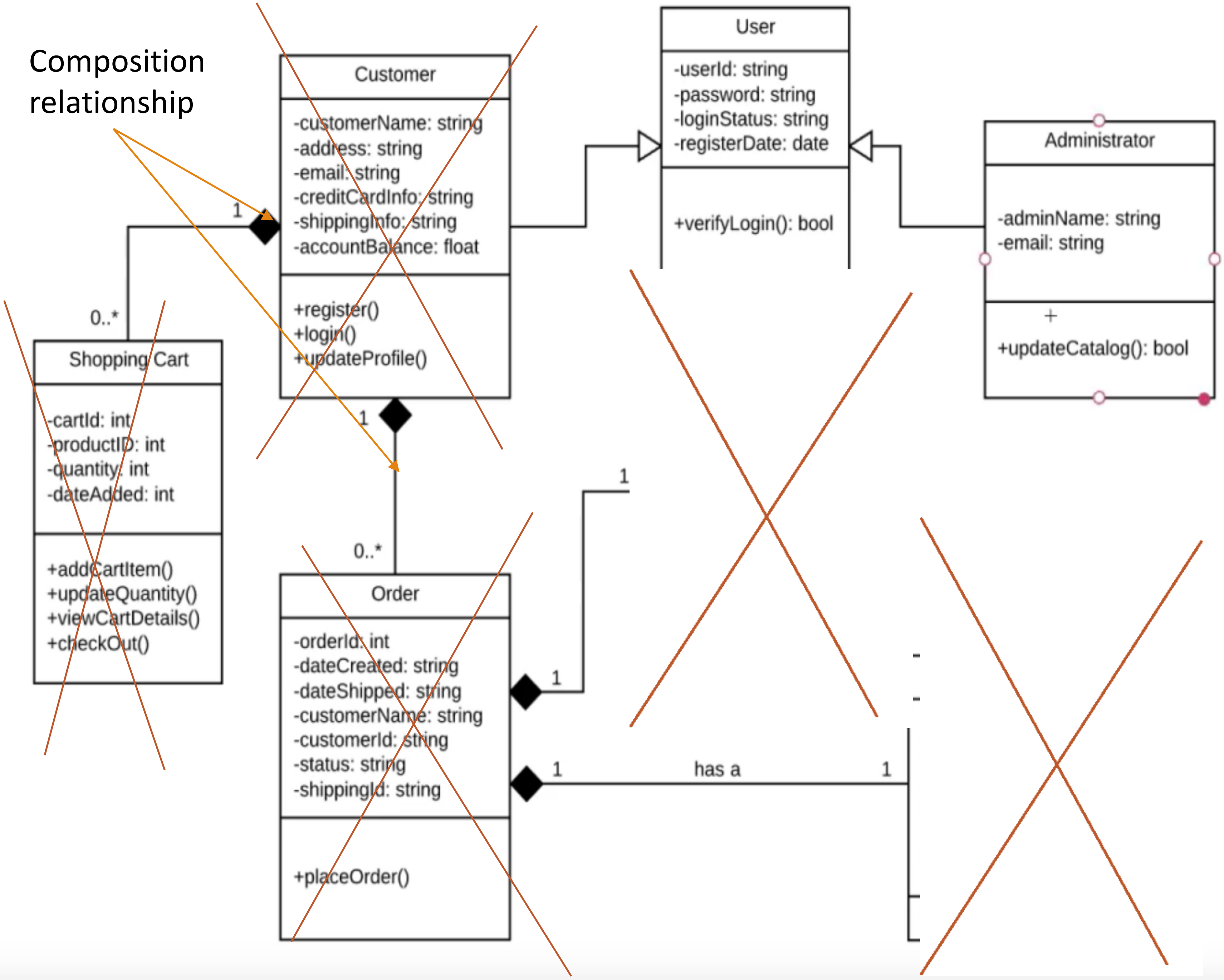
—— association

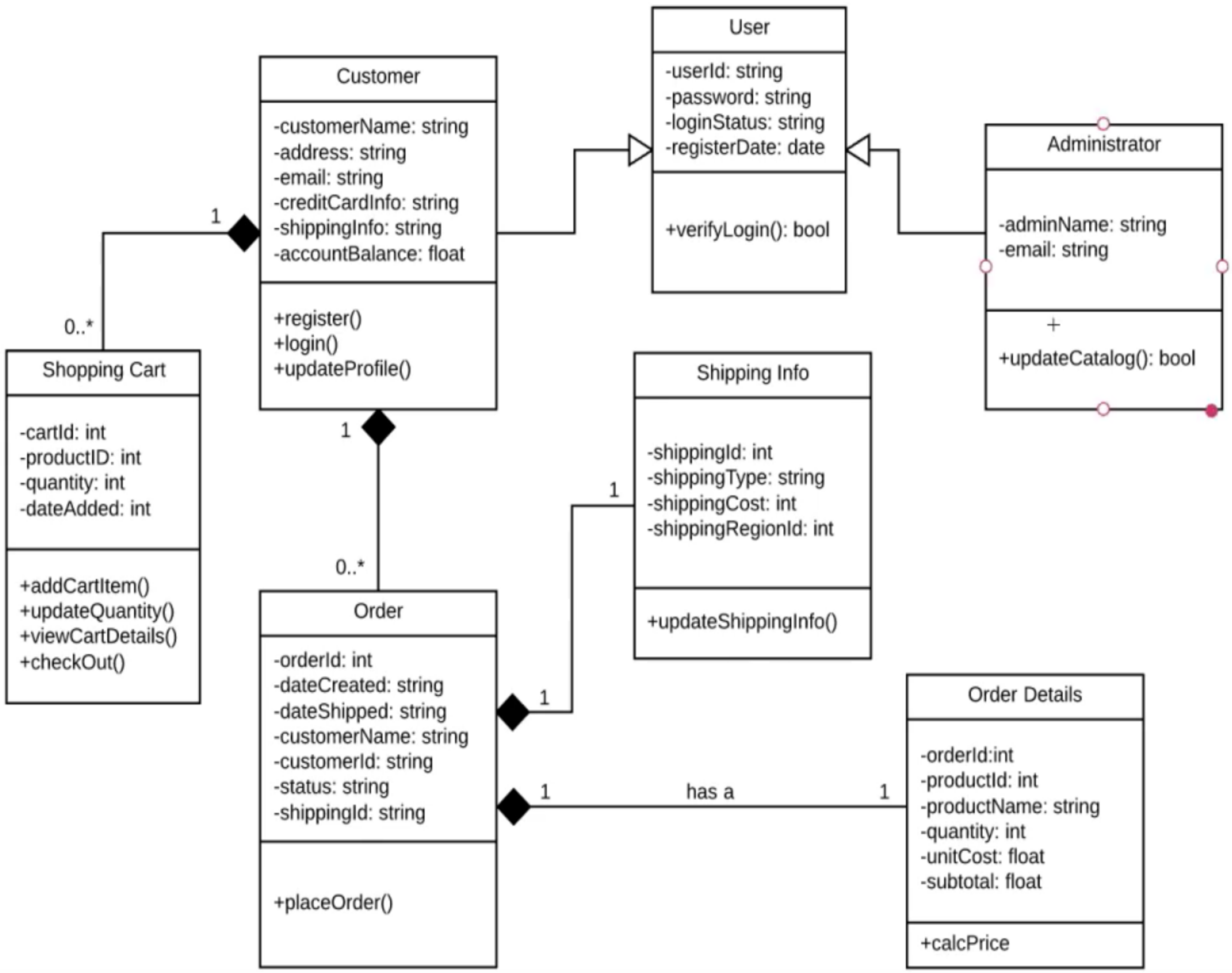
——◇ Aggregation

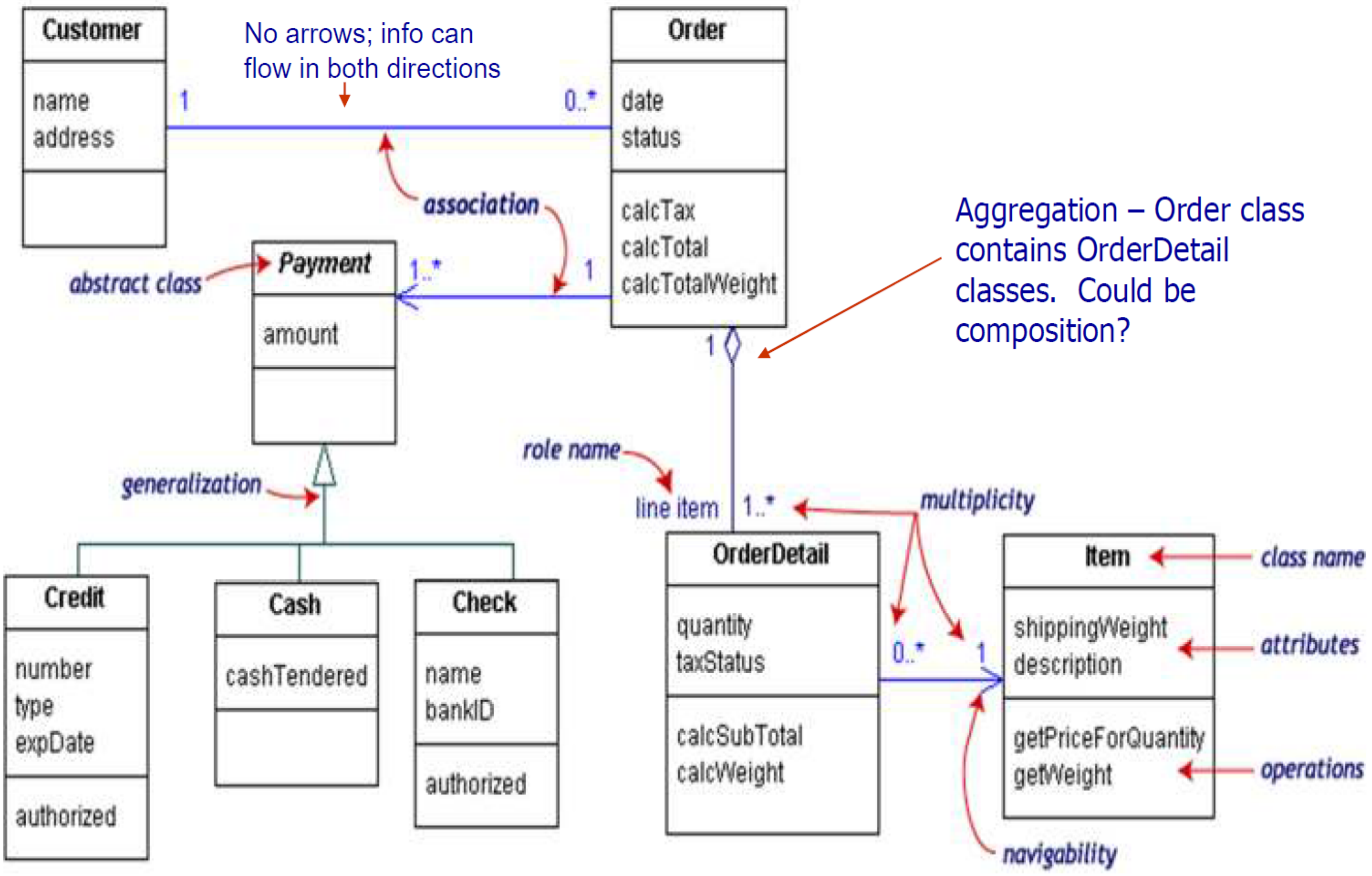
——◆ composition


Parent Class
User
-userId: string
-password: string
-loginStatus: string
-registerDate: date
+verifyLogin(): bool
Child Class
Customer
-customerName: string
-address: string
-email: string
-creditCardInfo: string
-shippingInfo: string
-accountBalance: float
+register()
+login()
+updateProfile()
Child Class
Administrator
-adminName: string
-email: string
+updateCatalog(): bool
Shipping Info
1
+viewCartDetails()
+checkOut()
-orderId: int
-dateCreated: string
-dateShipped: string
-customerName: string
-customerId: string
-status: string
-shippingId: string
+placeOrder()
1
1
has a
1
Order Details
-orderId:int
-productId: int
-productName: string
-quantity: int
-unitCost: float
-subtotal: float
+calcPrice

Composition relationship




User
-userId: string
-password: string
-loginStatus: string
-registerDate: date
+verifyLogin(): bool
Customer
-customerName: string
-address: string
-email: string
-creditCardInfo: string
-shippingInfo: string
-accountBalance: float
+register()
+login()
+updateProfile()
Administrator
-adminName: string
-email: string
+
+updateCatalog(): bool
Shopping Cart
-cartId: int
-productID: int
-quantity: int
-dateAdded: int
+addCartItem()
+updateQuantity()
+viewCartDetails()
+checkOut()
Shipping Info
-shippingId: int
-shippingType: string
-shippingCost: int
-shippingRegionId: int
+updateShippingInfo()
Order
-orderId: int
-dateCreated: string
-dateShipped: string
-customerName: string
-customerId: string
-status: string
-shippingId: string
+placeOrder()
Order Details
-orderId:int
-productId: int
-productName: string
-quantity: int
-unitCost: float
-subtotal: float
+calcPrice
1
0..*
1
0..*
1
1
1
has a
1



Aggregation – Order class contains OrderDetail classes. Could be composition?

Class diagram pros/cons

Class diagrams are great for:
– discovering related data and attributes
– getting a quick picture of the important entities in a system
– seeing whether you have too few/many classes
– seeing whether the relationships between objects are too complex, too many in number, simple enough, etc.
– spotting dependencies between one class/object and another

• Not so great for:
– discovering algorithmic (not data-driven) behavior
– finding the flow of steps for objects to solve a given problem
– understanding the app's overall control flow (event-driven? web-based? sequential? etc.)

Assignment

Design a class diagram based on your project.

Deadline 10/08/2017

Mid-term exam

10/25/2017

Open Book exam

Exercise

- Consider this Texas Hold 'em poker game system:

- 2 to 8 human or computer players
- Each player has a name and stack of chips
- Computer players have a difficulty setting: easy, medium, hard

- Summary of each hand:

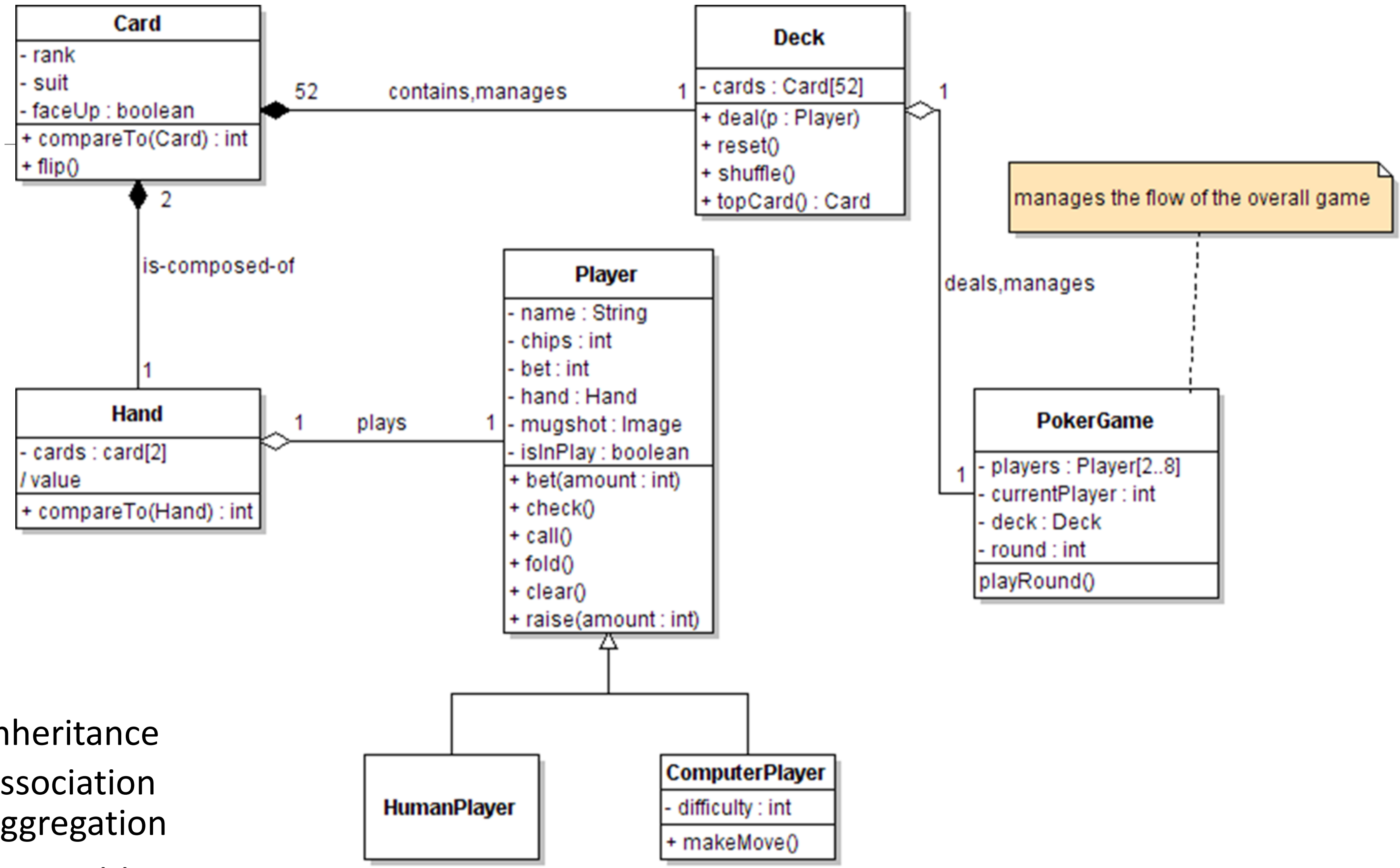
- Dealer collects ante from appropriate players, shuffles the deck, and deals each player a hand of 2 cards from the deck.
- A betting round occurs, followed by dealing 3 shared cards from the deck.

- As shared cards are dealt, more betting rounds occur, where each player can fold, check, or raise.

- At the end of a round, if more than one player is remaining, players' hands are compared, and the best hand wins the pot of all chips bet.

- What classes are in this system? What are their responsibilities? Which classes collaborate?

- Draw a class diagram for this system. Include relationships between classes



—▷ inheritance

— association

—◇ Aggregation

—◆ composition