Introduction to UML

UML

The Unified Modeling Language, UML, is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems.

examples

- use cases
- class diagrams

Use Cases

casual $\rightarrow$ fully dressed

Casual Use Case

- Order from catalog
  - **Description**: Customer calls to order items from the catalog. The sales rep. identifies the item numbers, verifies that the items are in stock, and confirms the order with the customer, giving him the order number. The sales rep. then forwards the order to the Shipping dept.

Less Casual Use Case

- Order from catalog
- Actors: Customer, sales rep, shipping dept.
- Primary Actor: Customer
  - **Description**: Customer calls to order items from the catalog. The sales rep. identifies the item numbers, verifies that the items are in stock, and confirms the order with the customer, giving him the order number. The sales rep. then forwards the order to the Shipping dept.

Fully Dressed Use Case

- Scope: What is the system under discussion?
- Primary actor: Who has the goal?
- Level: How high or low level is that goal?
- Actors: Who or what participates in the use case?
- Stakeholders: Who or what has a vested interest in system behavior during the use case?
- Preconditions: What must be true before the use case runs?
- Guarantees: What must be true after the use case runs?
- **Main success scenario**: What happens in the use case when nothing goes wrong?
- Extension: What can happen differently?

Main scenario: flow-of-events format

1 Customer calls to order from catalog.
2 Sales representative identifies item numbers.
3 Sales representative verifies stock.
4 Sales representative confirms order.
5 Sales representative gives order number to Customer.
6 Sales representative passes order to Shipping.

Flow-of-events *iteration*

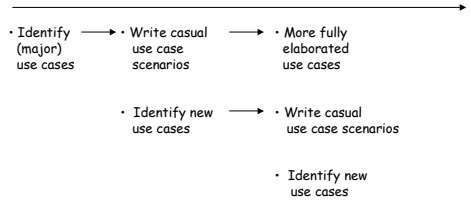
- ***For each*** item to be ordered:
  - Sales representative checks catalog number.
  - Sales representative verifies stock.
  - Sales representative records item.

Flow of Events *extensions*

Main success scenario
1 Customer calls to order from catalog.
2 Sales representative identifies item numbers.
3 Sales representative verifies stock.
4 Sales representative confirms order.
5 Sales representative gives order number to Customer.
6 Sales representative passes order to Shipping.

Extensions
2a Customer's catalog is outdated and item number is invalid. Sales representative tells client item is no longer available.

Use Case Evolution



Use Cases

All I am concerned about in this class is that your use cases effectively describe how your games function. You may use whatever format, information, or style that works best for you!

examples

- use cases
- class diagrams
- sequence diagram

Class Diagrams

A class diagram describes the types of objects in the system or a subsystem and their (static) relationship.

Class Diagram Evolution

Class Diagrams

Game

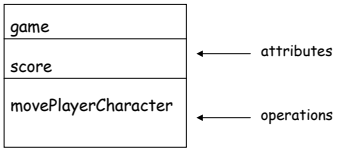
game is a domain concept

Class Diagrams

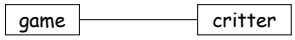
cGame

cGame is a POP class

Elaborated class diagrams

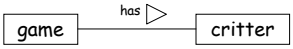


Association



game is associated with critter

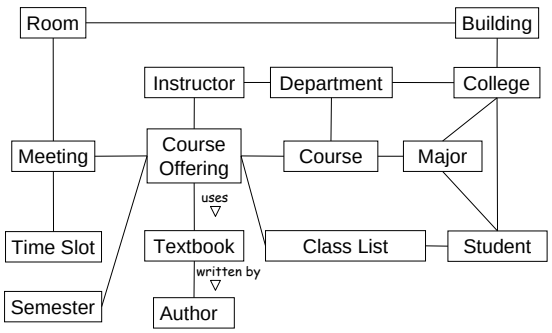
Association Names



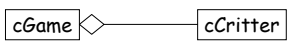
A is associated with B

- A has a B
- A has a method that returns a B
- vice versa
- etc.

Exercise: Identify Likely Association Names



Composition



cGame "has a" cCritter

A "has a" B

```
class cA
private:
        cB bObj;
```

instance member

composition

```
class cA
private:
        cB *pbObj;
```

reference member

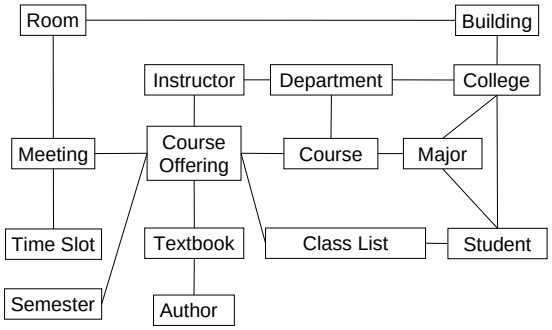
composition or aggregation

composition vs. aggregation

```
class cA
private:
        cB *pbObj;
```

- Composition: cA "owns" the cB object
  - cA initializes pbObj with new
  - cA destroys pbObj with delete
  - cascading delete
- Aggregation: cB is created/destroyed independently of cA

Exercise: Identify Likely Aggregations and Compositions

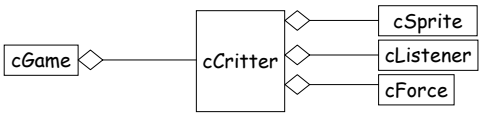


composition vs. aggregation

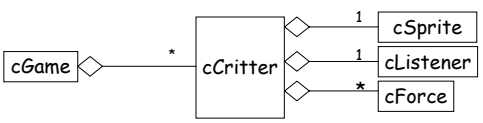
composition

aggregation (or composition or don't know)

Class Diagrams



Multiplicities



- cGame has zero or more cCritter
- cCritter has one cSprite
- cCritter has one cListener
- cCritter has zero or more cForce

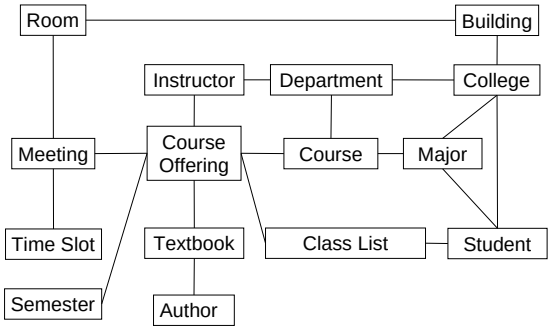
Multiplicities

- The default multiplicity is 1 or don't know.
- m..n means m through n (m and n fixed numbers).
- m..* means m or more.
- * means the same as 0..* (0 or more).
- a, b, c, ... means *one of* a, b, c ...
- 0,1 or 0..1 is a way of saying *optional*.

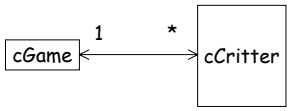
Association with Navigation

classB has a way to navigate to a classA object

Exercise: Identify Important Navigation Paths

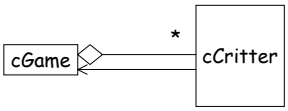


Class Diagrams

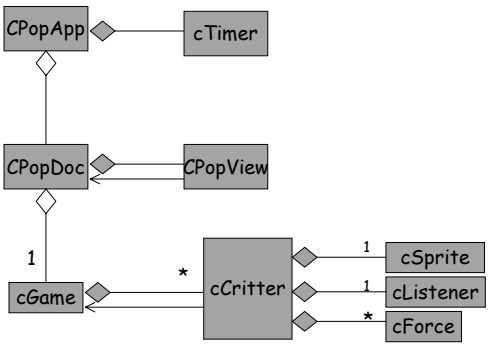


- cGame can access its cCritter objects
- cCritter can access its cGame

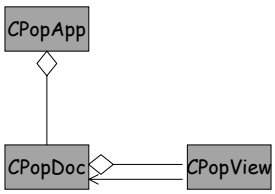
Class Diagrams



the bigger picture



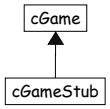
document-view architecture (design pattern)



recap

- class diagrams
  - association
  - composition
  - aggregation
  - multiplicity
  - navigation
  - inheritance

Inheritance

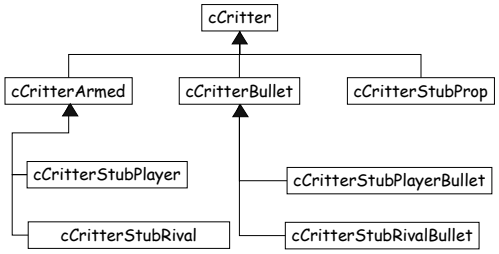


cGameStub is a cGame

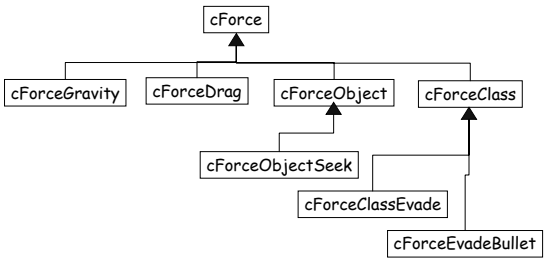
"is a"

class cGameStub : public cGame

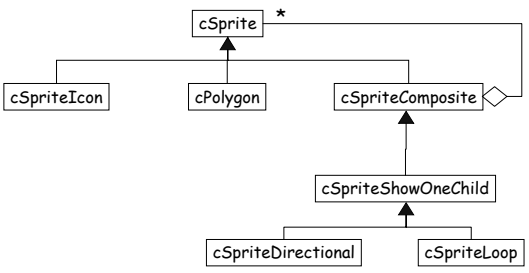
Class Diagrams: critters



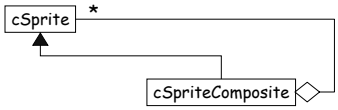
Class Diagrams: forces



Class Diagrams: sprites



design pattern



Class Diagrams

In project teams, construct a UML class diagram for your space invader game.