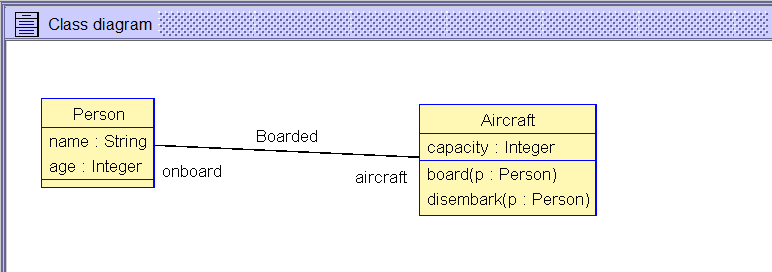
Q1



Board: set people onboard after must be = to set people onboard first union p : Person

b)

model AircraftSys

class Person

attributes

name : String

age : Integer

end

class Aircraft

attributes

capacity : Integer

operations

board(p : Person)

begin

insert (p, self) into Boarded

end

disembark(p : Person)

begin

delete (p, self) from Boarded

end

end

association Boarded between

Person[\*] role onboard

Aircraft[0..1]

end

constraints

context Aircraft

inv maxCap:

onboard->size() <= capacity

-- contract for board(p:Person)

context Aircraft::board( p : Person)

pre preBoard1:

not onboard->includes(p)

pre preBoard2:

onboard->size() < capacity

pre preBoard3:

p.age >= 18

post postBoard:

onboard = onboard@pre->union(Set{p})

--contract for disembark(p : Person)

context Aircraft::disembark( p : Person)

pre preDisembark1:

onboard->includes(p)

post postDisembark2:

onboard = onboard@pre - Set{p}

c)

-- Script generated by USE 5.0.1

!new Aircraft('cessna')

!cessna.capacity := 2

!new Person('p1')

!new Person('p2')

!new Person('p3')

!new Person('p4')

!p1.name := 'jim'

!p1.age := 28

!p2.name := 'tom'

!p2.age := 35

!p3.name := 'mary'

!p3.age := 17

!p4.name := 'jill'

!p4.age := 25

--will throw error since p3 under 18

--!cessna.board(p3)

--c

--will work

!cessna.board(p2)

!cessna.disembark(p2)

--will throw error since p3 not on board

--!cessna.disembark(p3)

-c

--expect error since p2 already on board

--!cessna.board(p2)

--c

!cessna.board(p1)

--expect error since cessna is at capacity

--!cessna.board(p4)

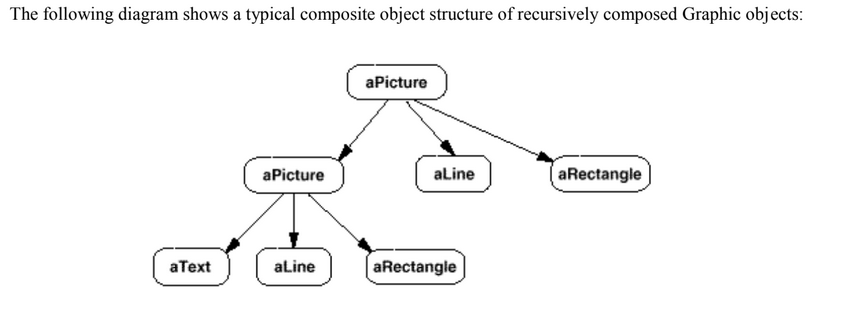
--c

--!cessna.disembark(p2)

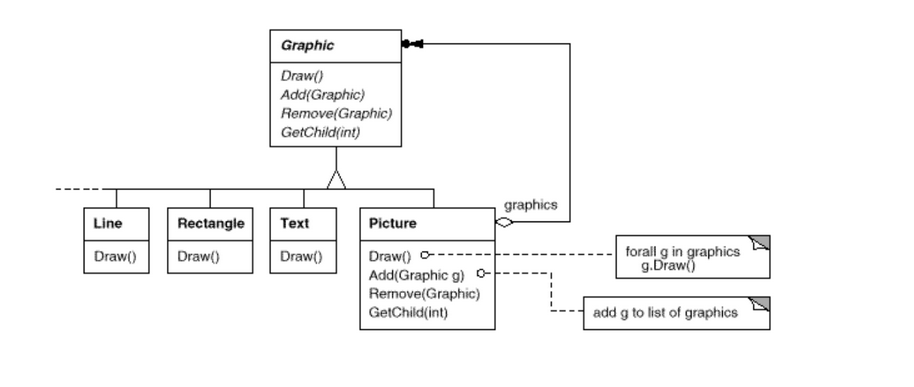
--!cessna.disembark(p1)

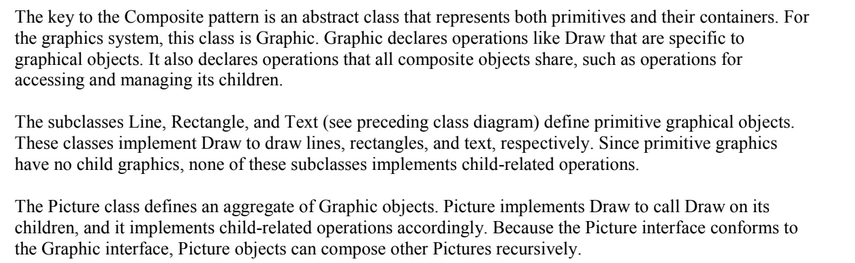
Q5

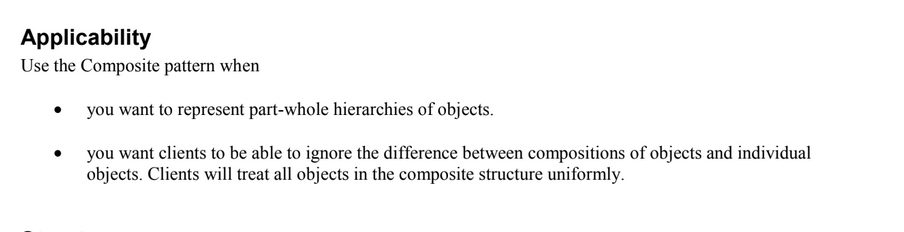
1. In [software engineering](https://en.wikipedia.org/wiki/Software_engineering), a **software design pattern** is a general, [reusable](https://en.wikipedia.org/wiki/Reusability) solution to a commonly occurring problem within a given context in [software design](https://en.wikipedia.org/wiki/Software_design). It is not a finished design that can be transformed directly into [source](https://en.wikipedia.org/wiki/Source_code) or [machine code](https://en.wikipedia.org/wiki/Machine_code). Rather, it is a description or template for how to solve a problem that can be used in many different situations. Design patterns are formalized [best practices](https://en.wikipedia.org/wiki/Best_practice) that the programmer can use to solve common problems when designing an application or system.
2. Object diagram:



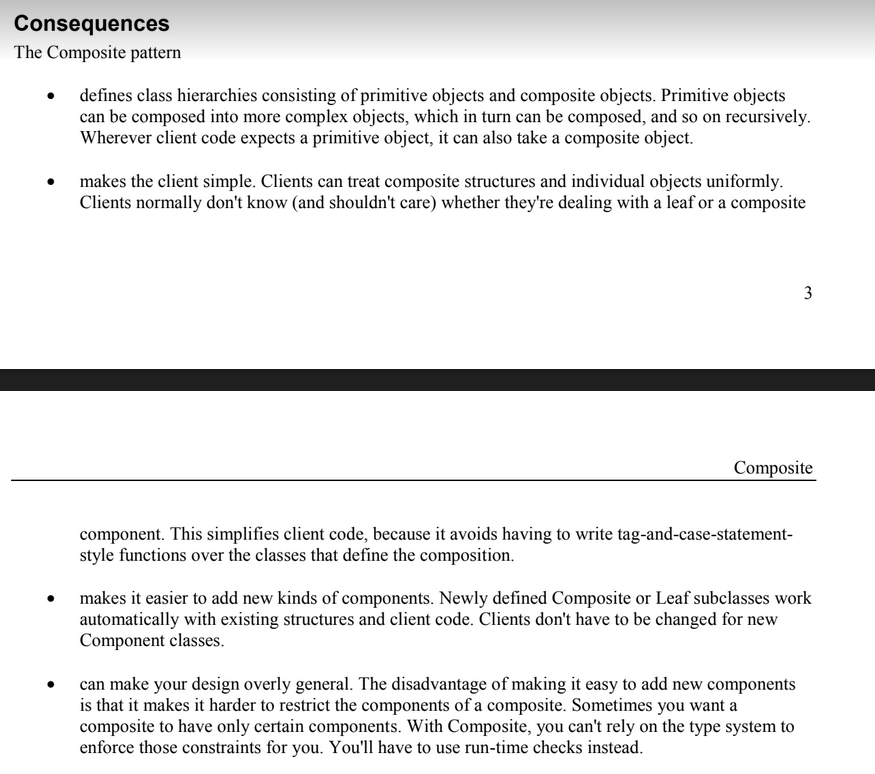
Class diagram:



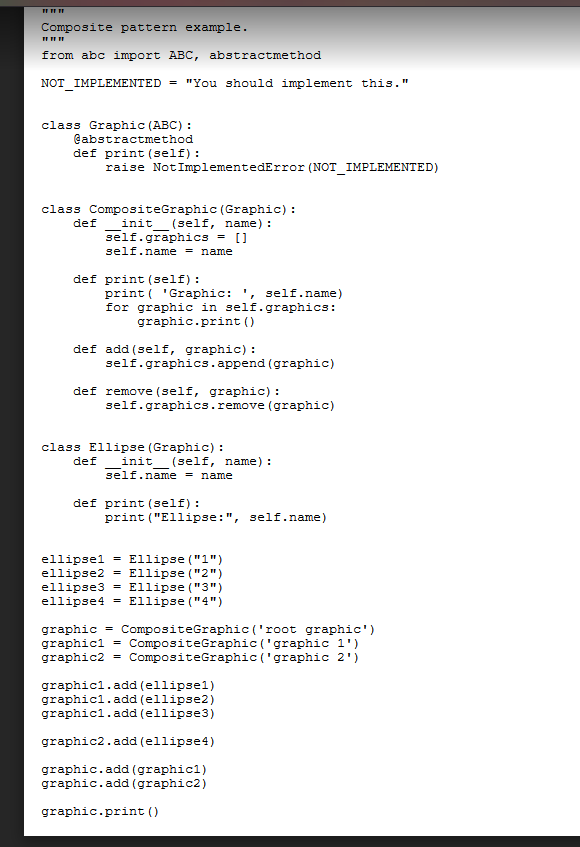




Advantages:



c)



(the important part of this is adding a bunch of objects to a list in another object – can shave down the code above)

Q3

Requirements engineering def:

The process of establishing what services are required and the constraints on the system’s operation and development

Problems waterfall model:

The main drawback of the waterfall model is the

difficulty of accommodating change after the

process is underway and so real projects rarely

follow it

• difficult to establish all requirements explicitly, no

room for uncertainty (customer uncertainty,

ambiguous requirements, evolving requirements)

• this makes it difficult to respond to changing

customer requirements

major mistake can be disastrous (software works but

does wrong thing, not really what customer needed

etc.)

• therefore, this model is only appropriate when the

requirements are well-understood

b)

inception

elaboration

construction

transition

inception:

establish business rationale

• decide project scope

• get commitment from sponsor to proceed further

• range from couple of hours chat and spreadsheet

plan to a serious feasibility study over a couple of months