Software technology

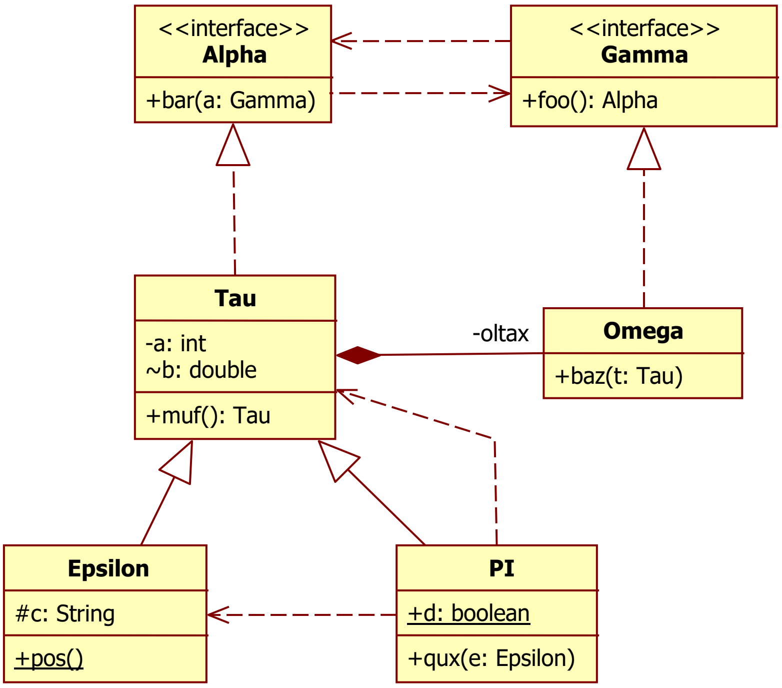
Exercise Book
UML
Solutions

Elaborated by
Balázs Goldschmidt, Zoltán László, Balázs Simon

Table of contents

1 Class diagram

Based on the diagram evaluate each statement using the key given below !



A – only the first part of the statement is true
B – only the second part of the statement is true
C – both parts are true, the conclusion is false
D – both parts are true, the conclusion is true
E – both parts are false

[B] method **qux(e: Epsilon)** of **PI** can receive an object of **Tau** as parameter, because **PI** knows **Tau**.

[B] method **foo()** of **Omega** cannot create an object of **Tau**, because the interface **Gamma** (it declares the operation **foo()**) doesn't know **Tau**.

[A] method **pos()** of **Epsilon** cannot modify the attribute **c**, because the visibility of the attribute **c** is package.

[C] method **muf()** of **Tau** can call the method **baz(t:Tau)** of **Omega**, because **Omega** knows **Tau**.

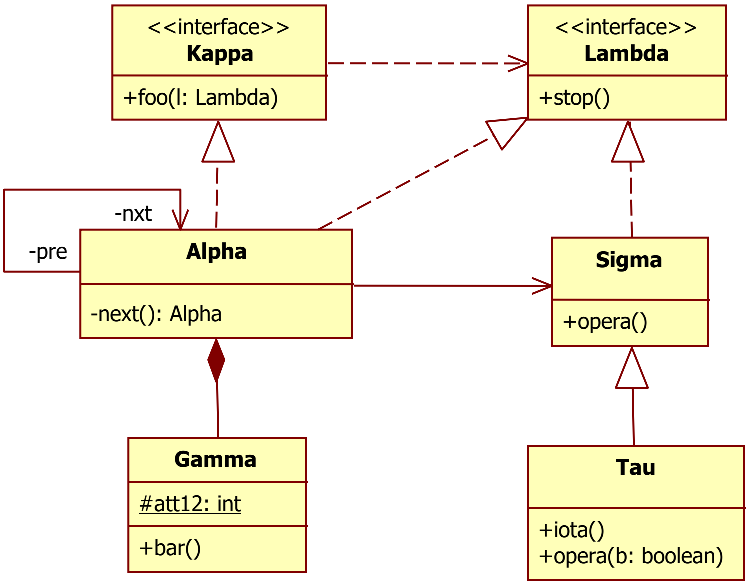
[D] method **baz(t:Tau)** of **Omega** can modify the attribute **b** of **Tau**, because the attribute **b** is not private.

[B] method **qux(e:Epsilon)** of **PI** cannot modify the attribute **d**, because the attribute **d** is static.

[C] method **bar(a: Gamma)** of **Epsilon** cannot call the method **qux(e: Epsilon)** of **PI** received as parameter, because **Epsilon** doesn't know **PI**.

[C] method **pos()** of **Epsilon** cannot modify the attribute **a**, because the attribute is not static.

Based on the diagram evaluate each statement using the key given below !



A – only the first part of the statement is true
B – only the second part of the statement is true
C – both parts are true, the implication is false
D – both parts are true, the implication is true
E – both parts are false

[C] **Alpha** can call the method **opera()** of a **Sigma**, because both implements **Lambda**.

[B] The method **next( )** of **a1** instance of the **Alpha** returns a reference only to itself, because the **next()** is private.

[C] Method **iota()** can be called from the method **stop()** of **Tau**, because the **iota()** is public.

[B] The **att12** of the **Gamma** can be accessed from the **next()** of an **Alpha**, bacause **Gamma** is the component of the **Alpha**.

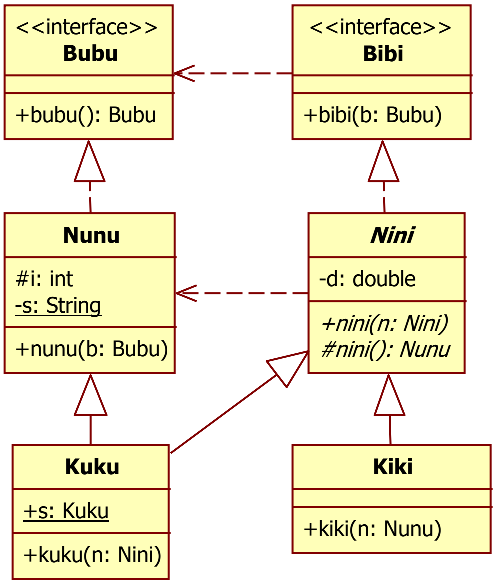
[E] Method **next()** of **Alpha** can be called from the **bar()** of **Gamma**, because the component class can access all method of the whole class.

[B] **Tau** can be substituted by **Sigma**, because both has **opera()** method.

[D] Method **foo(l:Lambda)** of **Alfa** can receive a **Tau** parameter, because **Tau** implements **Lambda**.

[B] Method **opera(b: boolean)** overwrites the **opera()** method of **Sigma**, because **Tau** is the specialization of **Sigma**.

Based on the diagram evaluate each statement using the key given below !



A – only the first part of the statement is true
B – only the second part of the statement is true
C – both parts are true, the implication is false
D – both parts are true, the implication is true
E – both parts are false

[B] **bibi** method of **Nini** cannot receive a **Kuku** object as a parameter, because **Bibi** doesn't know **Kuku**.

[C] There is such **s** attribute, which cannot be modified by **kiki** method of **Kiki**, because not all **s** attribute is public.

[E] **kiki** method of **Kiki** cannot receive a **Kuku** object as a parameter, because if it would receive, it violates the Liskov-principle.

[B] The interfaces of **Kuku** and **Kiki** are identical, because both implements the **Bibi** interface.

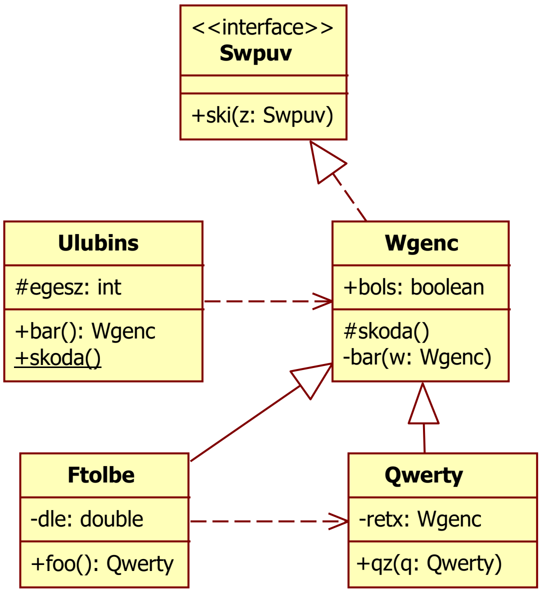
[E] **Kuku** cannot be instantiated, because it doesn't implement the abstract **nini** methods.

[C] **nunu** method of **Nunu** cannot instantiate a **Kiki** object, because **Kiki** is depending on **Nunu**.

[E] **bubu** method of **Kuku** cannot modify the **i** attribute, because **i** is static.

[C] **Kuku** implements both **Bubu** and **Bibi** interfaces, because the multiple inheritance between interfaces is allowed in Java.

Based on the diagram evaluate each statement using the key given below!



A – only the first part of the statement is true
B – only the second part of the statement is true
C – both parts are true, the implication is false
D – both parts are true, the implication is true
E – both parts are false

[B] **qz(q:Qwerty)** method of **Qwerty** can receive an **Ftolbe** type object as parameter, because **Ftolbe** is the descendant of the **Qwerty's** ancestor.

[D] There is such **skoda()** method, which cannot modify the **bols** attribute, because **bols** is not class-scope.

[D] **skoda()** method of **Wgenc** cannot call every method named **bar**, because **Wgenc** doesn't know about **Ulubins**.

[E] The return value of the **skoda()** method of **Ulubins** is **Ulubins**, because the predefined return value of every static method is the instance of the method's own class.

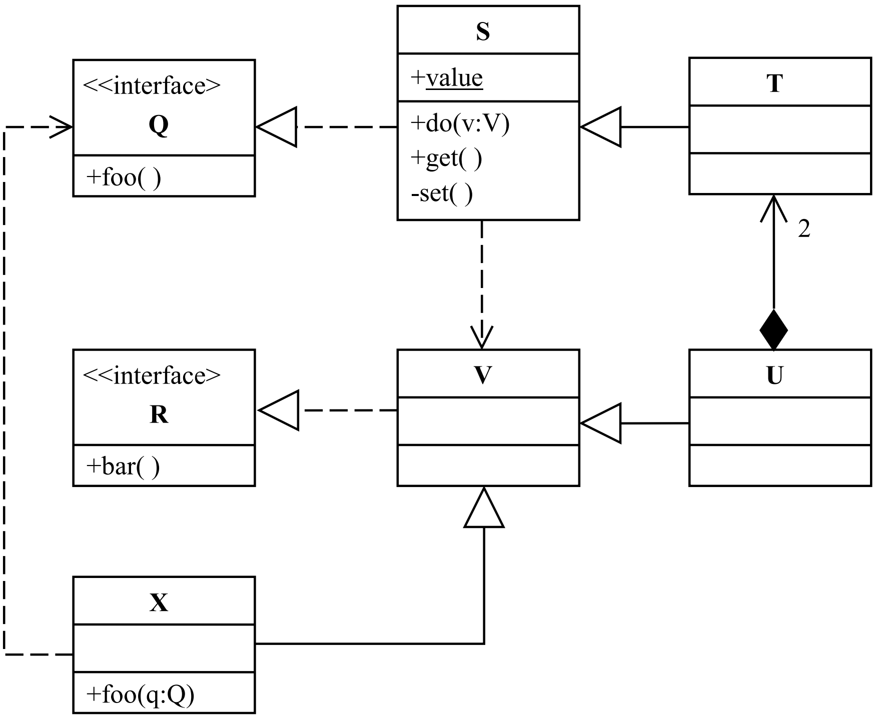
[A] **bar(w:Wgenc)** method of **Wgenc** can call the **skoda()** method of a **Folbe** type parameter object, because the visibility of **skoda()** is „package”.

[A] **ski(z:Swpuv)** method of **Ftolbe** can call only the **ski(z:Swpuv)** method of the parameter object, because the parameter object must not have other methods.

[C] **foo()** method of **Ftolbe** can instantiate a **Ftolbe** object, because **foo()** is not static.

[C] **bar()** method of **Ulubins** cannot instantiate an object of **Qwerty**, because **Qwerty** doesn't depend on **Ulubins**.

Based on the diagram evaluate each statement using the key given below!



A – only the first part of the statement is true
B – only the second part of the statement is true
C – both parts are true, the implication is false
D – both parts are true, the implication is true
E – both parts are false

[C] **foo(q:Q)** method of **X** can receive a type **T** parameter, because **T** also has a method named **foo**.

[B] **set()** of **S** cannot modify the **value** attribute, because their visibilities are different.

[B] When **V** is deleted two instances of **T** must also be deleted, because **U** has two **T** components and **U** is the descendant of **V**.

[B] **S** can call the **foo()** of **T**, because **T** is the descendant of **S**.

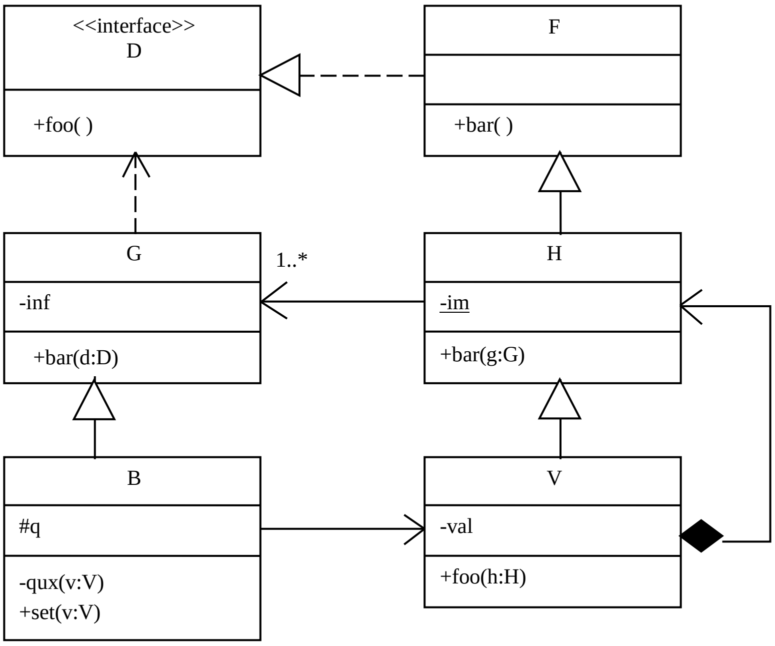
[A] **X** can call the **foo()** method of an object which implements the **Q** interface, because **X** implements **Q**.

[B] **T** can create an object of **U**, because **S** can create **V** and **T** is the descendant of **S** and **U** is the descendant of **V.**

[B] **foo(q:Q)** method of **X** can call the **get()** of an **S** type object received as parameter, because **S** implements **Q**.

[B] **bar()** of **X** can call the **foo()** method of an object implements **Q**, because **foo(q:Q)** of **X** can do the same.

Based on the diagram evaluate each statement using the key given below!



A – only the first part of the statement is true **B** – only the second part of the statement is true
C – both parts are true, the implication is false **D** – both parts are true, the implication is true
E – both parts are false

[B] method **set(v:V)** of **B** cannot modify the attribute **q**, because their visibilities are different.

[E] an instance of **B** cannot call the method **foo()** of **V**, because **V** has only a different (**foo(h:H))** method.

[B] method **bar(d:D)** of **G** can call the method **bar()** of the received parameter object of **F**, because **F** implements interface **D**.

[B] **V** can contain **G**, because **V** can contain **H** and each **H** is associated with at least one **G** instance.

[C] method **bar(d:D)** of **G** can receive a parameter type **H**, because **H** has a method **bar()**.

[B] method **bar(g:G)** of **H** can receive a parameter type **V**, because **V** implements interface **D.**

[B] method **qux(v:V)** of **B** can modify the attribute **val** of the received object, because both the method, and the attribute are private.

[C] method **bar(g:G)** of **H** can modify the attribute **im**, because the attribute is class-scope.

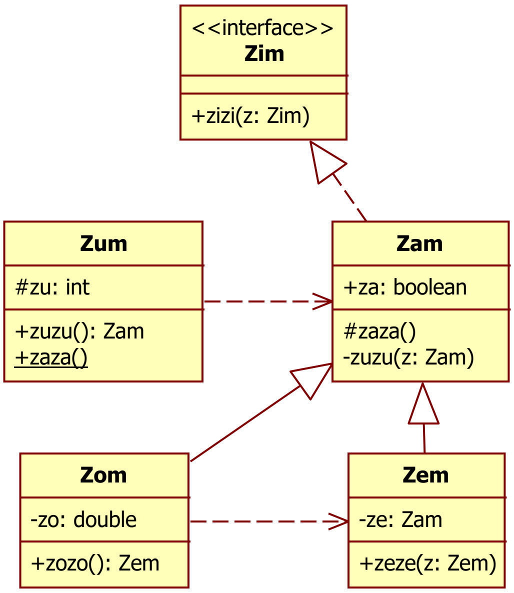
[E] method **bar(d:D)** of **G** can receive a parameter type **B**, because **G** is a descendants of **B**

[E] method **bar(g:G)** of **H** cannot modify the attribute **im**, because the attribute is constant

[E] method **bar(d:D)** of **G** can call the method **bar()** of the received parameter object of **F**, because the two mentioned **bar** methods have the same signature.

[E] **B** has a method **foo()**, because **B** implements interface **D**

Based on the diagram evaluate each statement using the key given below !



A – only the first part of the statement is true
B – only the second part of the statement is true
C – both parts are true, the implication is false
D – both parts are true, the implication is true
E – both parts are false

[A] **zuzu** method of **Zam** can call the **zaza** method of **Zom** received as parameter, because **zaza** is abstract.

[A] **zozo** method of **Zom** can intantiate a **Zom** object, because **Zom** is a descendant of **Zem**.

[A] **zaza** method of **Zum** cannot modify the **zu** attribute, because **zu** is privat.

[C] **zeze** method of **Zem** cannot receive a **Zom** object as parameter, because **Zom** depends on **Zem**.

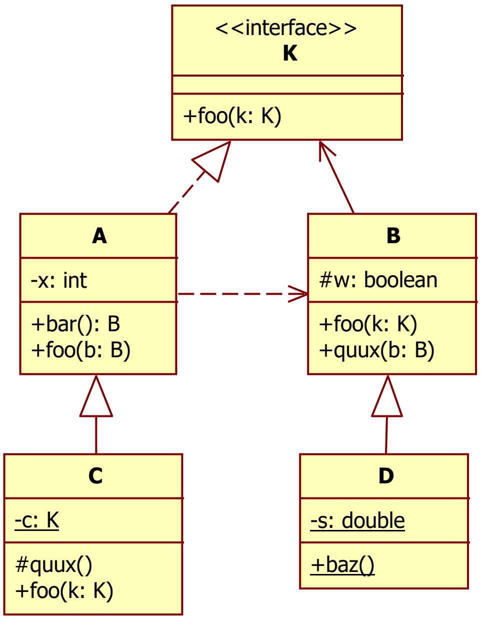
[C] **zuzu** method of **Zum** cannot instantiate a **Zem** object, because **Zem** doesn't depend on **Zum**.

[D] There is such **zaza** method, which cannot modify the **za** attribute, because **za** isn't static.

[D] **zaza** method of **Zam** cannot call each **zuzu** method, because **Zam** doesn't know about **Zum**.

[E] **zizi** method of **Zom** cannot call the **zizi** method of **Zem**, because none of them have **zizi** method.

Based on the diagram evaluate each statement using the key given below !



A – only the first part of the statement is true
B – only the second part of the statement is true
C – both parts are true, the implication is false
D – both parts are true, the implication is true
E – both parts are false

[B] The **quux** method of **C** cannot modify the **c** attribute, because **quux** is not private.

[C] The interface of **C** and the interface of **D** are different, because **D** doesn’t implement **K**.

[B] The **foo** method of **D** cannot call the **foo(k:K)** method of a **C** object, received as a parameter, because **D** doesn’t depend on **C**.

[A] None of the methods of **C** can modify the **w** attribute of a **B** parameter, because **C** doesn’t depend on **B**.

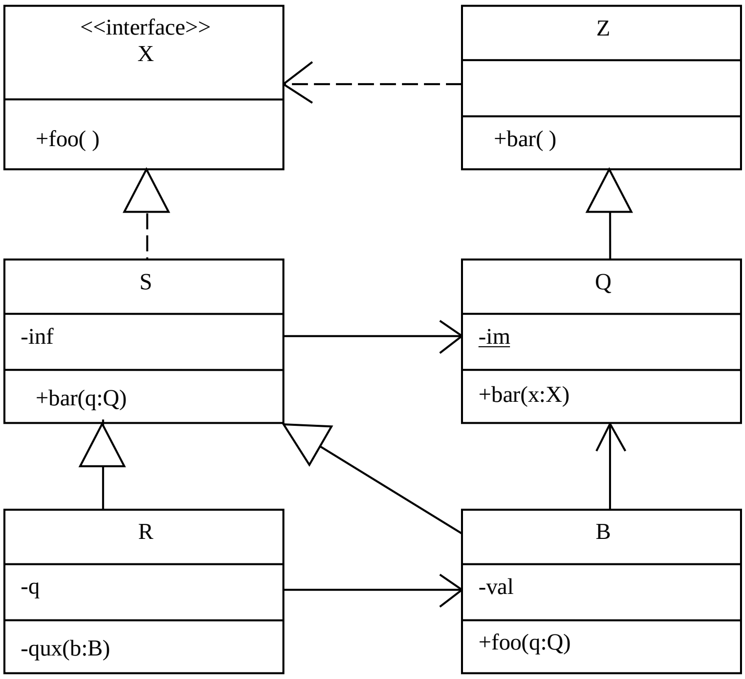
[A] The **baz** method of **D** cannot modify the **w** attribute of **B**, because **w** is static**.**

[D] There is such **foo** method, which cannot receive an object instance of **B**, as a parameter, because **B** doesn’t implement **K**.

[C] The **bar** method of **A** cannot create an object of **D**, because **D** doesn’t depend on **A**.

[D] Each method declared in **D** can modify the **s** attribute, because **s** is class-scope.

Based on the diagram evaluate each statement using the key given below !



A – only the first part of the statement is true
B – only the second part of the statement is true
C – both parts are true, the implication is false
D – both parts are true, the implication is true
E – both parts are false

[C] **S** can be substituted by **B**, because **B** implements the interface **X**.

[A] Method **bar(x:X)** of **Q** can receive an object of **R**, because the interfaces of **Q** and **S** are identical.

[E] The interface of **B** contains the method **bar(x:X)**, because the method is class-scope.

[A] Method **bar()** of **Q** cannot modify the attribute **im**, because the attribute is constant.

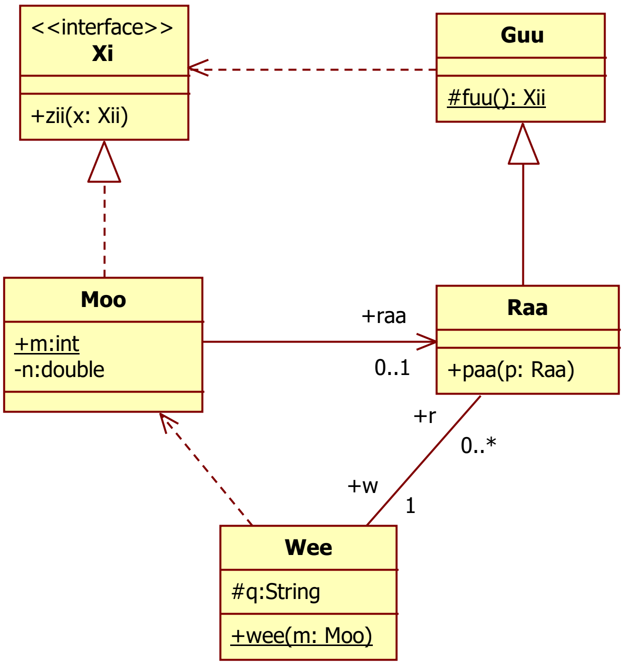
[E] **Q** can call the method **bar(q:Q)** of **S**, because both classes implement the interface **X**.

[B] Method **foo(q:Q)** of **B** cannot access the attribute **val**, because the attribute is private.

[E] **Q** can be substituted by **S**, because **S** is a descendant of **Q.**

[A] **Q** doesn't implement the method **foo()**, consequently **Q** doesn't depend on the interface **X**.

Based on the diagram evaluate each statement using the key given below !



A – only the first part of the statement is true
B – only the second part of the statement is true
C – both parts are true, the implication is false
D – both parts are true, the implication is true
E – both parts are false

[E] method **paa(p:Raa)** of **Raa** can receive an object of **Guu** as parameter, because **Raa** as function parameter can be substituted by **Guu**.

[C] method **paa(p:Raa)** of **Raa** cannot modify **m** attribute of **Moo**, because **m** attribute of **Moo** is static.

[B] method **zii(x:Xii)** of **Moo** cannot multiply the value of attributes **m** and **n**, because attribute **n** is private.

[D] The interfaces of **Moo** and **Xii** are identical, because **Moo** doesn't define additional method.

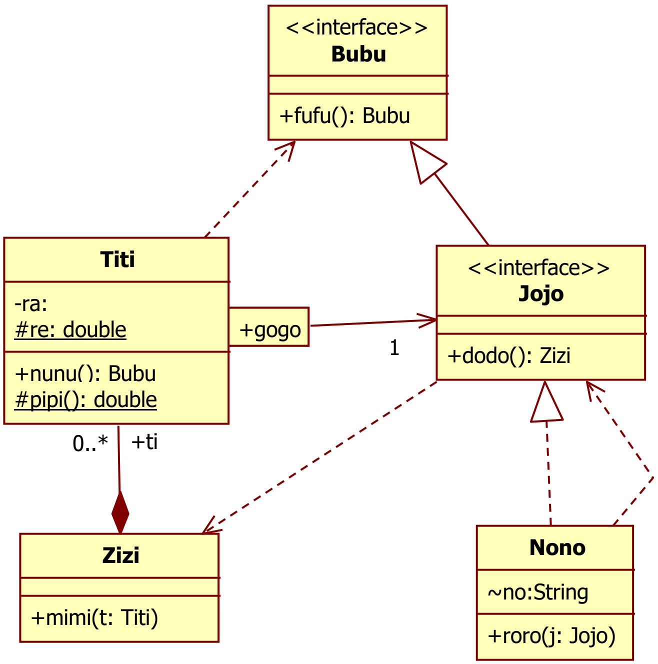
[B] method **paa(p:Raa)** of **Raa** cannot call the method **fuu():Xii**, because **Guu** doesn't implement **Xii.**

[A] method **wee(m:Moo)** of **Wee** cannot modify the value of the attribute **q**, because the visibility of **q** is *package*.

[B] **Raa** doesn't depend on **Xii**, because **Guu** doesn't implement interface **Xii**.

[C] method **wee(m:Moo)** of **Wee** can call the method **zii(x:Xii)** of parameter **m**, because it doesn't violate the Demeter's law.

Based on the diagram evaluate each statement using the key given below !



A – only the first part of the statement is true
B – only the second part of the statement is true
C – both parts are true, the implication is false
D – both parts are true, the implication is true
E – both parts are false

[B] The **dodo()** method of **Nono** cannot modify the value of attribute **no**, because the **String** type of Java is immutable.

[C] The **mimi(t:Titi)** method of **Zizi** mustn't call the **dodo()** method of the **Jojo** type returned by the **nunu()** method of the **t** parameter, because it violates the Demeter's law.

[E] The **dodo()** method of **Nono** cannot create an instance of **Zizi**, because **Nono** doesn't depend on **Zizi**.

[B] **Zizi** can call the method **roro(j:Jojo)** on the return value of the **ti.nunu(),** because **Nono** implements the interface **Bubu.**

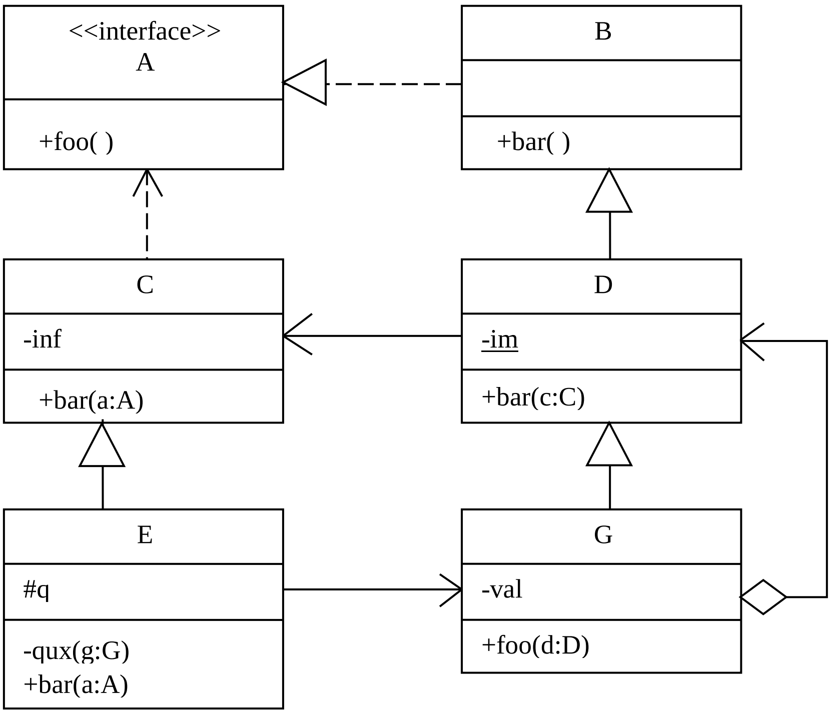
[B] The **Titi** object accesses only one object instance of interface **Jojo**, because the multiplicity is 1 on the **Jojo**-side of the association between them.

[E] The **roro(j:Jojo)** method of **Nono** can receive an object of interface **Bubu** as **j** parameter, because **Jojo** implements **Bubu**.

[A] The **pipi()** method of **Titi** cannot multiply the attributes **ra** and **re**, because private attribute can be accessed only by private method.

[B] **Zizi** depends on **Bubu**, because **Titi** depends on **Bubu**.

Based on the diagram evaluate each statement using the key given below !



A – only the first part of the statement is true
B – only the second part of the statement is true
C – both parts are true, the implication is false
D – both parts are true, the implication is true
E – both parts are false

[D] **B** can be substituted by **G**, because **G** is a descendant of **B**.

[A] Method **bar(c:C)** of **D** can modify attribute **im**, because **im** is not class-scope.

[E] Method **qux(g:G)** of **E** cannot call the method **foo( )** of the object received as a parameter, because **G** has no **foo( )** method.

[B] Method **bar(c:C)** of **G** cannot receive a parameter type **E**, because **G** doesn't depend on **E**.

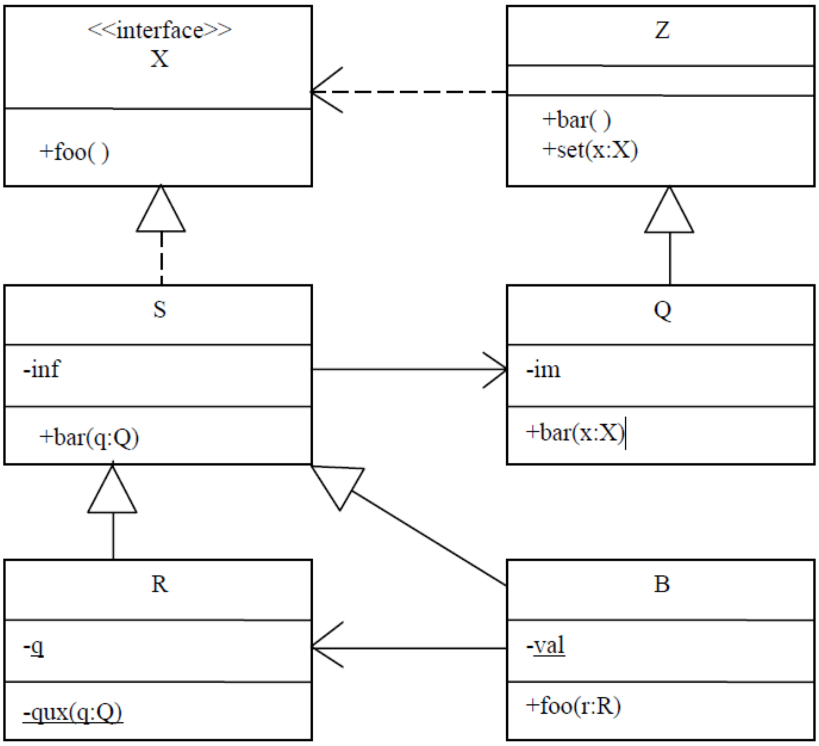
[E] **E** can be substituted by **G**, because both classes implement the interface **A**.

[B] Method **bar(a:A)** of **C** can call the method **bar( )** of **B** (**B** is received as a parameter), because **B** implements the method **bar( )**.

[A] Method **qux(g:G)** of **E** can modify attribute **q**, because **q** is private.

[B] Method **bar(a:A)** of **C** can receive a parameter type **E**, because **E** depends on **A.**

Based on the diagram evaluate each statement using the key given below !



A – only the first part of the statement is true **B** – only the second part of the statement is true
C – both parts are true, the implication is false **D** – both parts are true, the implication is true
E – both parts are false

[E] **Q** can be substituted by **S,** because **S** is a descendant of **Q**.

[A] method **qux(q:Q)** of **R** cannot receive a parameter type **Z**, because the method is abstract.

[E] method **foo(r:R)** of **B** cannot call the method **foo()** of the received parameter, because **R** has no such method.

[E] method **bar(q:Q)** of **S** cannot modify attribute **inf** of **S**, because the attribute is constant.

[E] method **bar(q:Q)** of **S** cannot call the method **bar()** of the received parameter, because **Q** has no such method.

[B] method **set(x:X)** of **Z** cannot receive a parameter type **B**, because **B** implements interface **X**.

[B] **B** can modify the attribute **im** of **Q**, because **S** is depending on **Q**.

[B] **R** and **B** are interchangeable, because they have common ancestor.

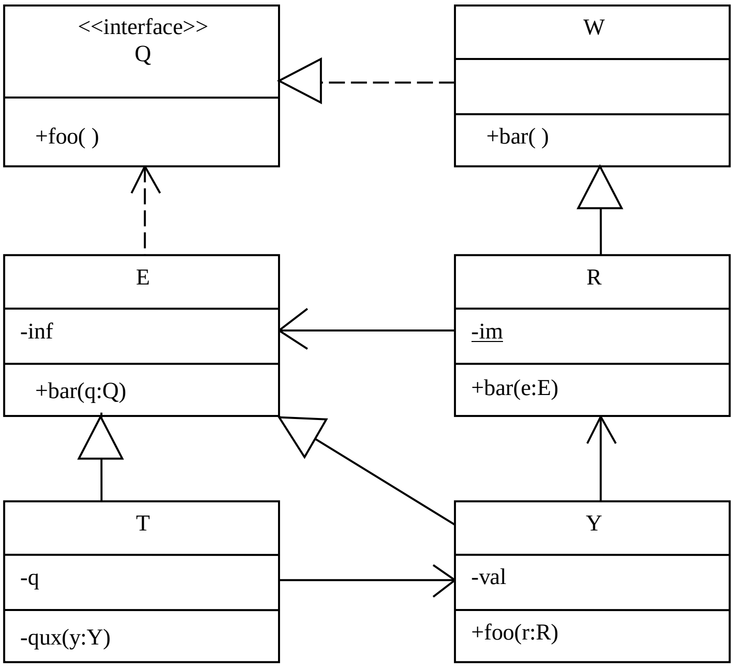
[B] method **qux(q:Q)** of **R** can receive a parameter type **Z**, because the method is class-scoped.

[D] method **foo(r:R)** of **B** can call the method **foo()** of the received parameter, because **R** has such method.

[D] method **set(x:X)** of **Z** can receive a parameter type **B**, because **B** implements interface **X**.

[D] method **bar(q:Q)** of **S** can call the method **bar()** of the received parameter, because **Q** has such method.

Based on the diagram evaluate each statement using the key given below !



A – only the first part of the statement is true
B – only the second part of the statement is true
C – both parts are true, the implication is false
D – both parts are true, the implication is true
E – both parts are false

[B] **R** doesn’t implement the interface **Q**, because **R** has methods, which are not declared in **Q**.

[C] method **bar(q:Q)** of **Y** can receive a parameter type **R**, because **Y** is depending on **R**.

[B] method **bar(e:E)** of **R** cannot receive a parameter type **Y**, because **Y** can call **R** along the association **Y**-**R**.

[E] method **bar(q:Q)** of **E** can receive a parameter type **E**, because **E** implements the interface **Q**.

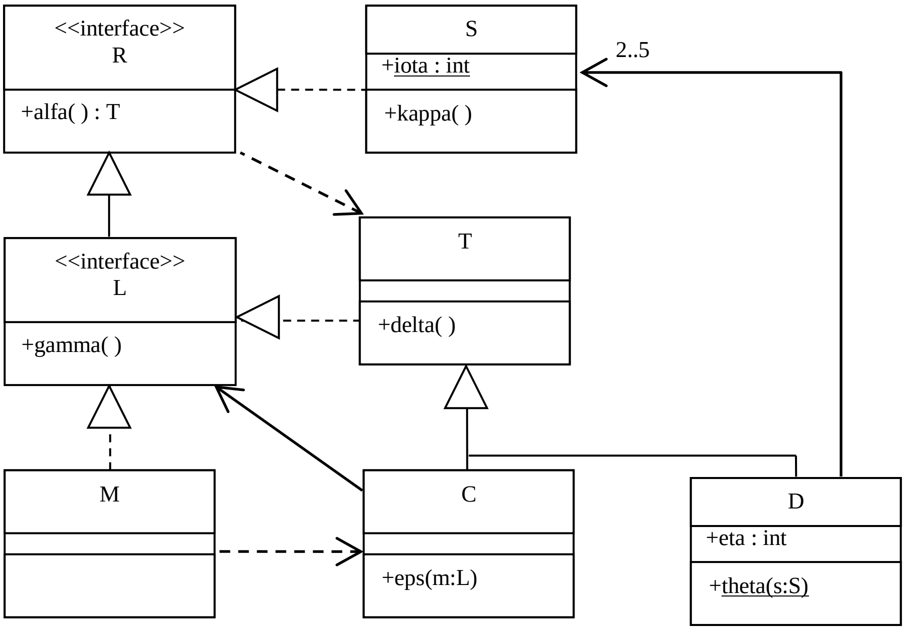
[C] method **foo(r:R)** of **Y** cannot modify the attribute **im** of the received parameter, because the attribute is static.

[B] method **qux(y:Y)** of **T** can modify the attribute **val** of the received parameter, because the method is private.

[E] **E** can be substituted by **R**, because their interfaces are identical.

[E] method **bar(q:Q)** of **E** cannot call the method **foo()** of the received parameter **W**, because **W** has no method **foo().**

Based on the diagram evaluate each statement using the key given below !



A – only the first part of the statement is true
B – only the second part of the statement is true
C – both parts are true, the implication is false
D – both parts are true, the implication is true
E – both parts are false

[A] method **alfa():T** of **M** can return an object of class **C**, because **C** depends on **M**.

[B] method **theta(s:S)** of **D** can receive a parameter type **C**, because both **S** and **C** implement interface **R**.

[B] **M** can be substituted by **C**-vel, because both classes implement interface **R**.

[B] method **theta(s:S)** of **D** can be called at most 5 times, because **D** is connected to at most 5 instances of **S**

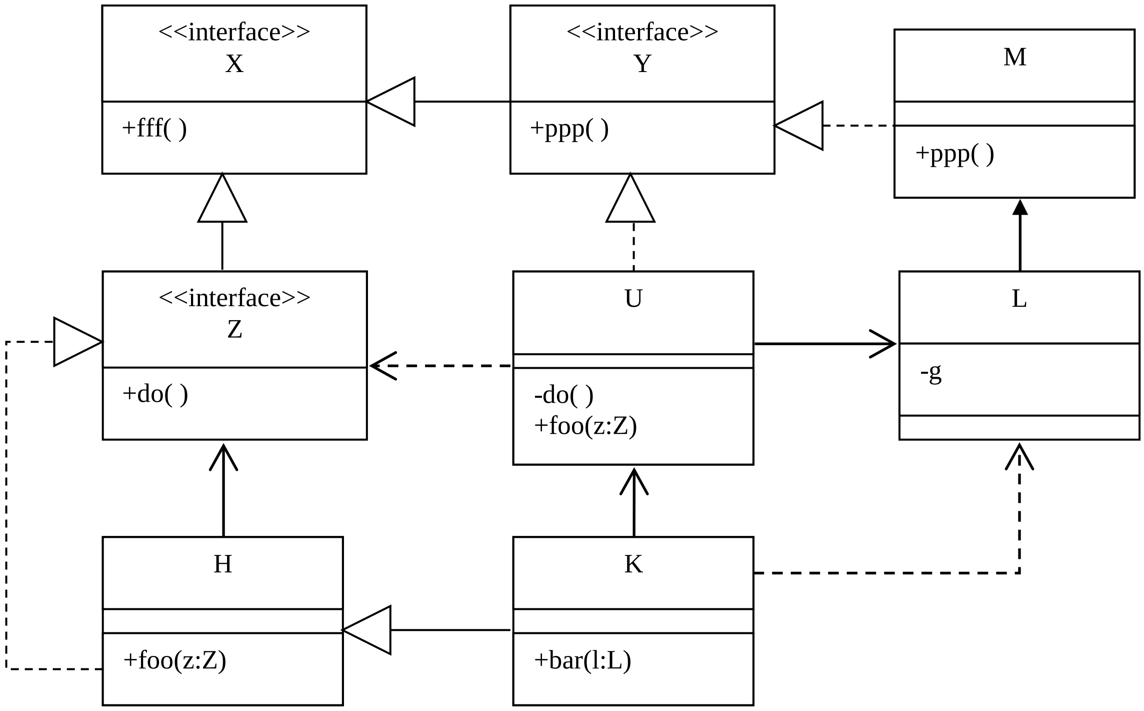
[C] method **theta(s:S)** of **D** can modify the attribute **iota** of the received parameter, because **theta(s:S)** is static.

[E] class T has no method **alfa():T**, because **T** doesn't implement interface **R.**

[E] method **eps(m:L)** of **C** cannot call the method **gamma()** of the received parameter, because the visibility of the latter method is protected.

[E] **S** doesn't implement the method **alfa():T**, because **S** doesn't depend on **T.**

Based on the diagram evaluate each statement using the key given below !



A – only the first part of the statement is true
B – only the second part of the statement is true
C – both parts are true, the implication is false
D – both parts are true, the implication is true
E – both parts are false

[B] method **foo(z:Z)** of **H** can receive a parameter type **U**, because **U** implements the interface **Y**.

[A] **U** cannot call the method **bar(l:L)** of **K**, because **K** doesn't depend on **L**.

[A] **K** implements the interface **Z**, accordingly **K** can call the method **do( )** of **U**.

[E] **K** can be substituted by **U**, because **K** is a descendant of **U**.

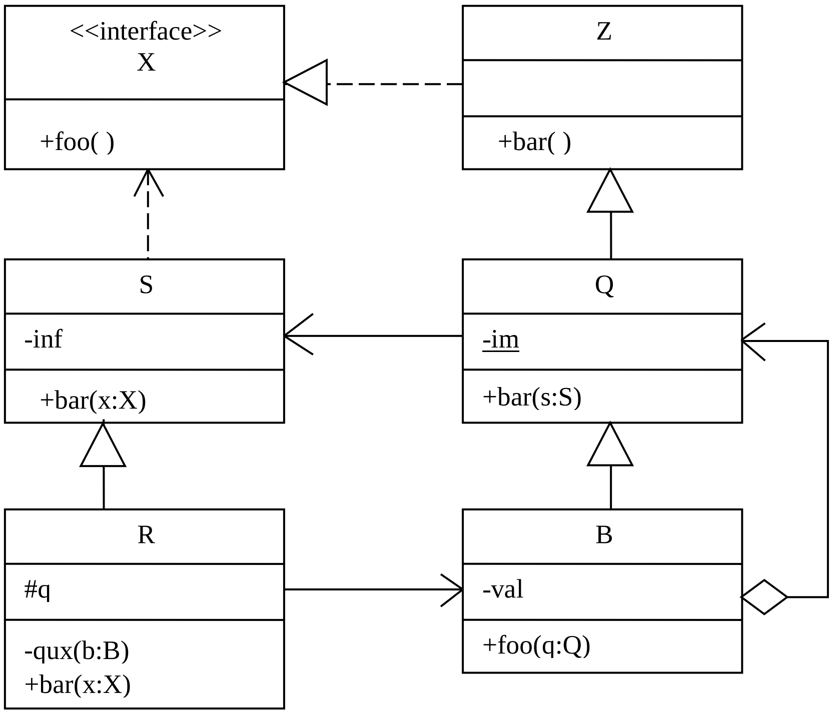
[B] **U** can be substituted by **H**, because both classes implement the interface **X**.

[A] **L** has no public attribute, accordingly **L** cannot call the public method **ppp( )** of **M**.

[E] **K** cannot create an instance of **L**, because the attribute **g** of **L** is protected.

[C] method **foo(z:Z)** of **U** cannot call the method **foo(z:Z)** of **H** (**H** is received as a parameter), because **U** doesn't implement the interface **Z**.

Based on the diagram evaluate each statement using the key given below !



A – only the first part of the statement is true
B – only the second part of the statement is true
C – both parts are true, the implication is false
D – both parts are true, the implication is true
E – both parts are false

[E] **R** can be substituted by **B**, because both classes implement the interface **X**.

[D] **Z** can be substituted by **B**, because **B** is a descendant of **Z**.

[B] Method **bar(s:S)** of **Q** cannot modify attribute **im**, because **im** is static.

[E] Method **qux(b:B)** of **R** cannot call the method **foo( )** of the object received as a parameter, because **B** doesn’t implement the interface **X**.

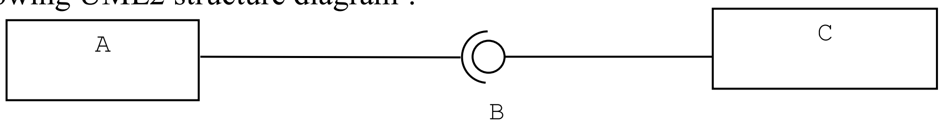
[B] Method **bar(x:X)** of **S** can call the method **bar( )** of **Z** (**Z** is received as a parameter), because **Z** implements the interface **X**.

[E] Method **qux(b:B)** of **R** cannot modify attribute **q**, because **q** is private.

[B] Method **bar(s:S)** of **B** cannot receive a parameter type **R**, because **B** doesn’t depend on **R**.

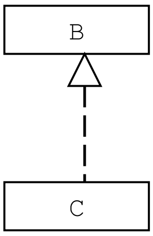
[B] Method **bar(x:X)** of **S** can receive a parameter type **R**, because **R** depends on **X.**

Let’s have the following UML2 structure diagram !

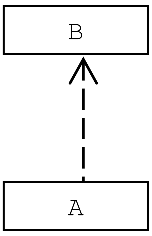


Supposing that there are no other relationships between the elements of the diagram given above, tick the conform diagram(s) !

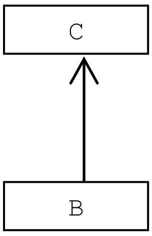
☑



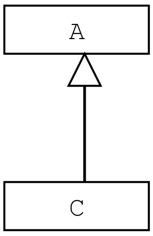
☑



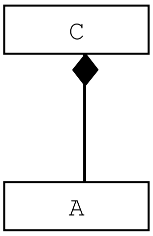
□



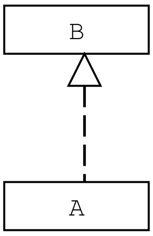
□



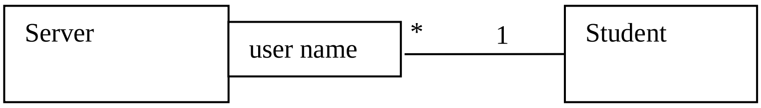
□



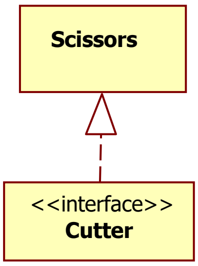
□



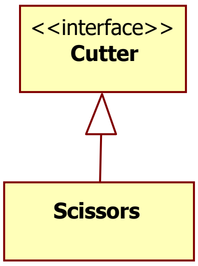
A server serves many students, and students have access right to more servers. Draw a UML class diagram, in which the „user name” is a qualifier ! Don’t forget the multiplicity !



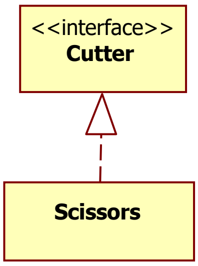
Mark the syntactically ***and*** semantically correct figures!



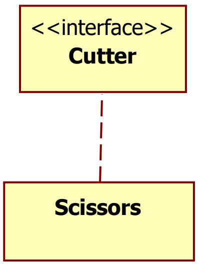
□



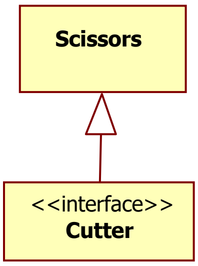
□



☑

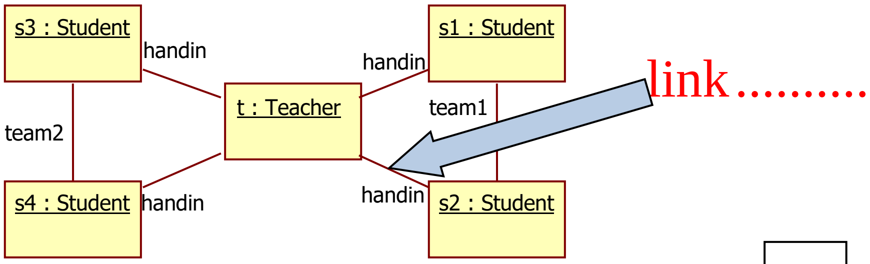


□



□

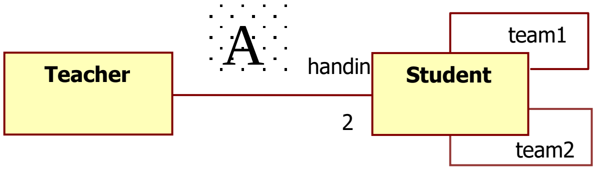
Define the pointed UML2 meta-model element!

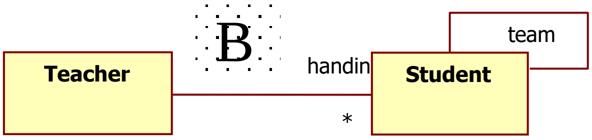


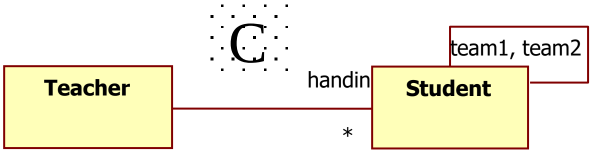
link.........

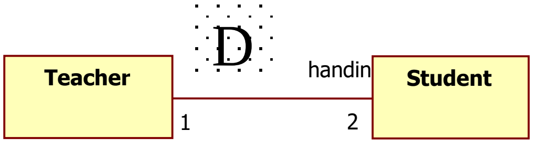
Which is the class diagram which best corresponds to the object diagram?

B



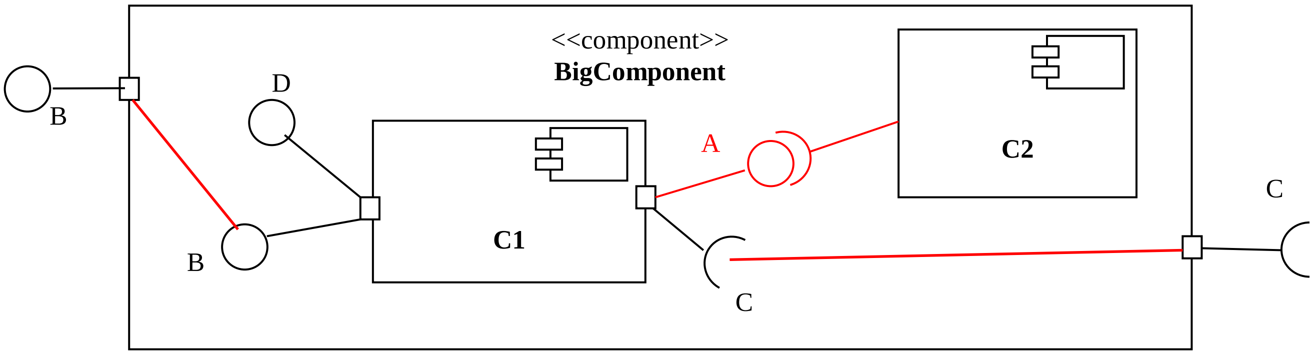




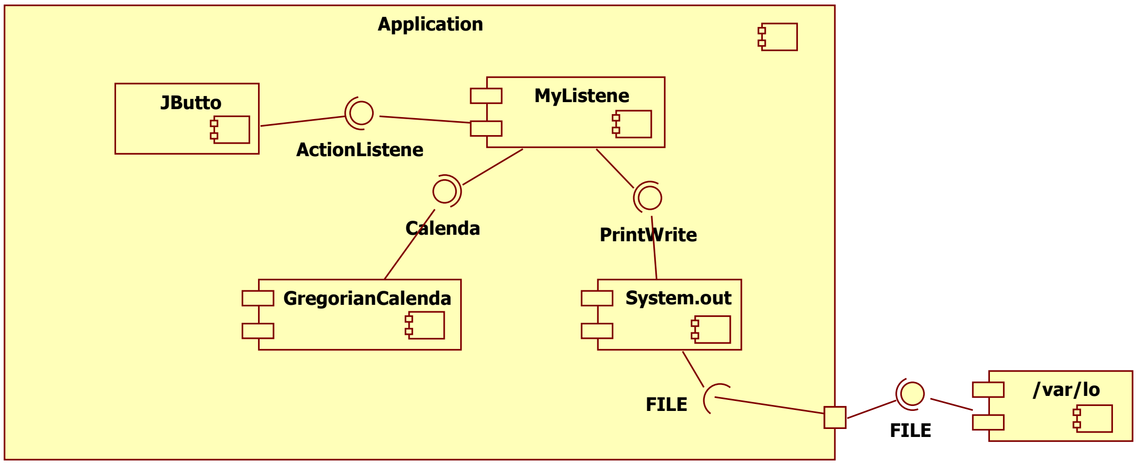


2 Component diagram

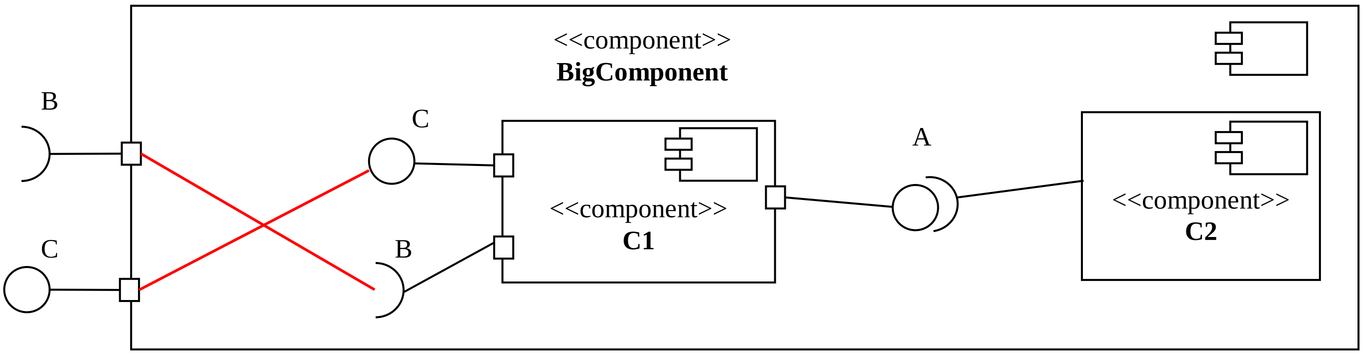
Insert the missing elements into the UML2 diagram, taking into consideration that the component C2 uses the interface A, which is implemented by the component C1!



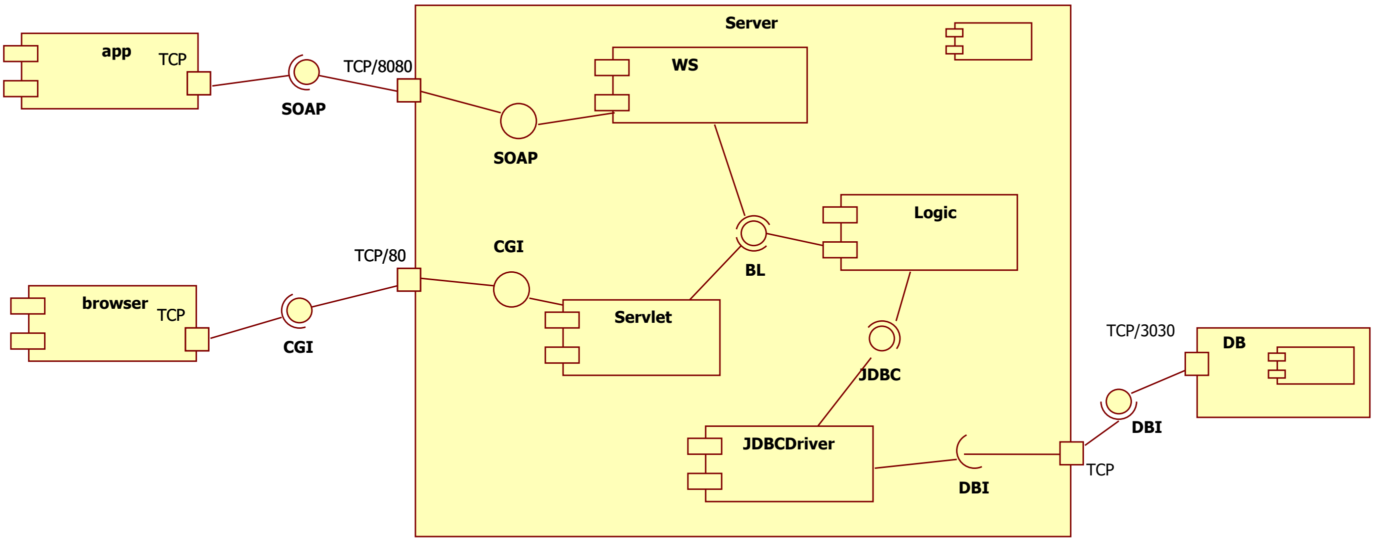
Draw a UML2 component diagram based on the specification given below!
Pressing a button in a Swing Application, an instance of the MyListener class writes the actual date getting from a GregorianCalendar object (implements the Calendar interface) to the System.out which implements the PrintWriter interface. The System.out is implemented in C, and it is directed into a logfile (/var/log) through a FILE* type variable.



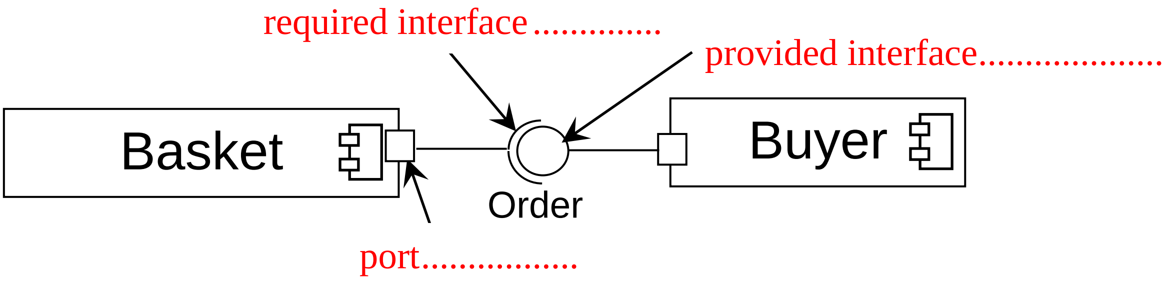
Insert the missing relationships into the UML2 diagram!



Draw a UML2 component diagram based on the specification given below!
The Server has a TCP/80 port through which browsers can be connected. Browsers use the CGI interface on this port. Mobile applications can connect to the Server, they use the SOAP interface on the TCP/8080 port. The Server uses a data base system through the DBI interface. The data base system publishes this interface through its TCP/3030 port. The Server contains four components. The SOAP interface implemented by the WebService component, the CGI interface is implemented by the Servlet component. Both component use the Logic component which provides the BusinessLogic API. The Logic component accesses the data base system through the JDBCDriver component, which implements the JDBC interface.

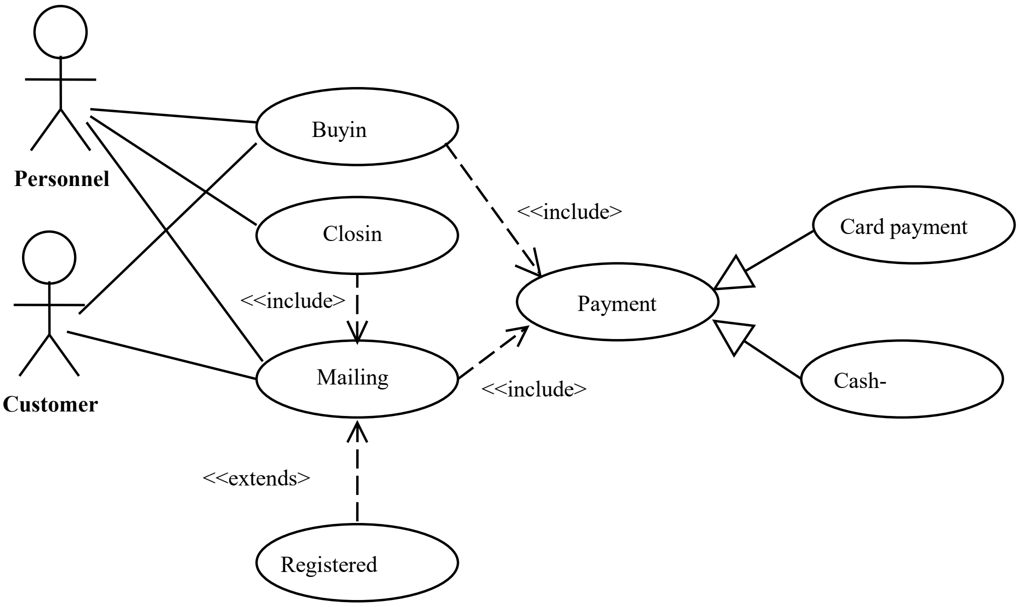


Give the UML2 name of the pointed items !

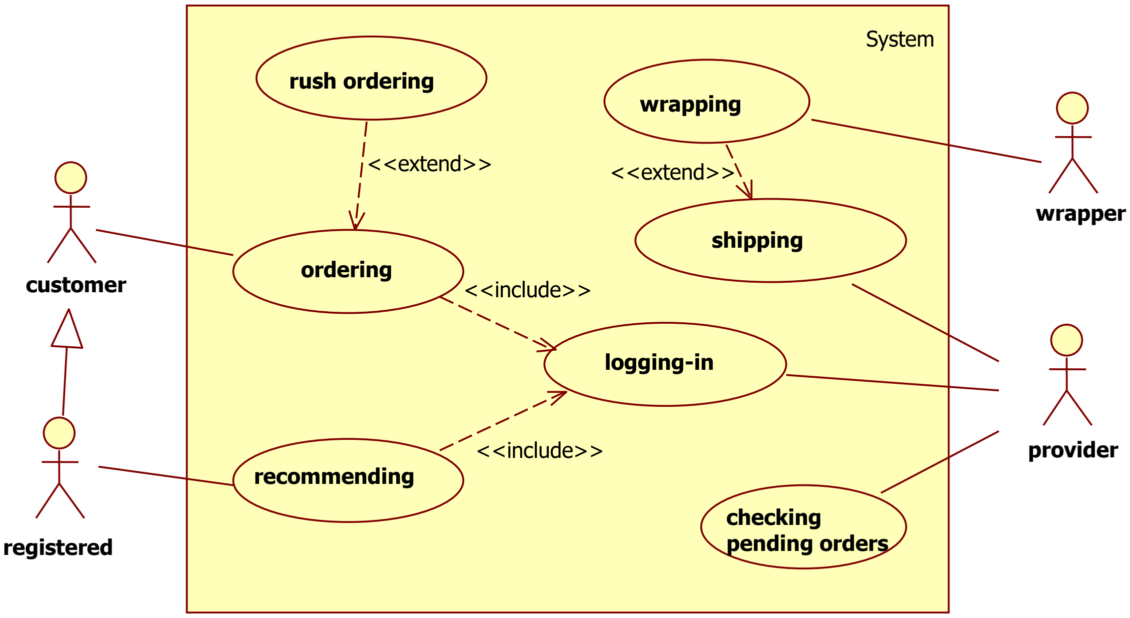

required interface.............
provided interface...................
Basket
Buyer
Order
port................

3 Use-case diagram

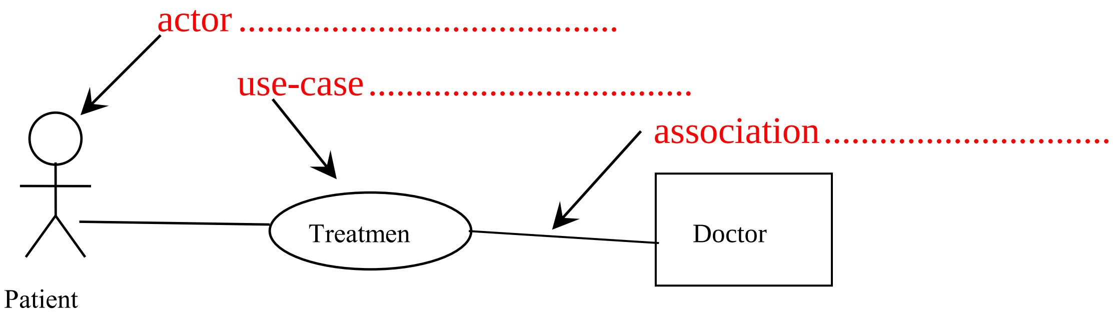
Draw a UML2 use case diagram based on the story given below!
In a shop the customer - in cooperation with the personnel - can buy some products (cigarettes, newspapers, sandwiches, etc.) and posting letters. Payment can be done by bankcard or cash. In dedicated shops registered letters can be posted also. When shop closing the personnel creates a daily balance of traffic, and posts it to the tax office.



Draw a UML2 use-case diagram based on the story given below!
The Emezen webshop recognizes two kinds of user. The simple customer can submit orders, the registered user moreover can recommend new products to the selection list. The ordering can be extended in the case of rush ordering. Logging in is the obligatory part of the user's activities. The provider must also login into the system. The provider can check the pending orders and supervises the shipping. Sometimes special wrapping (eg. Xmas, birthday) is required as a part of shipping. The wrapping is done by a wrapper fellow.

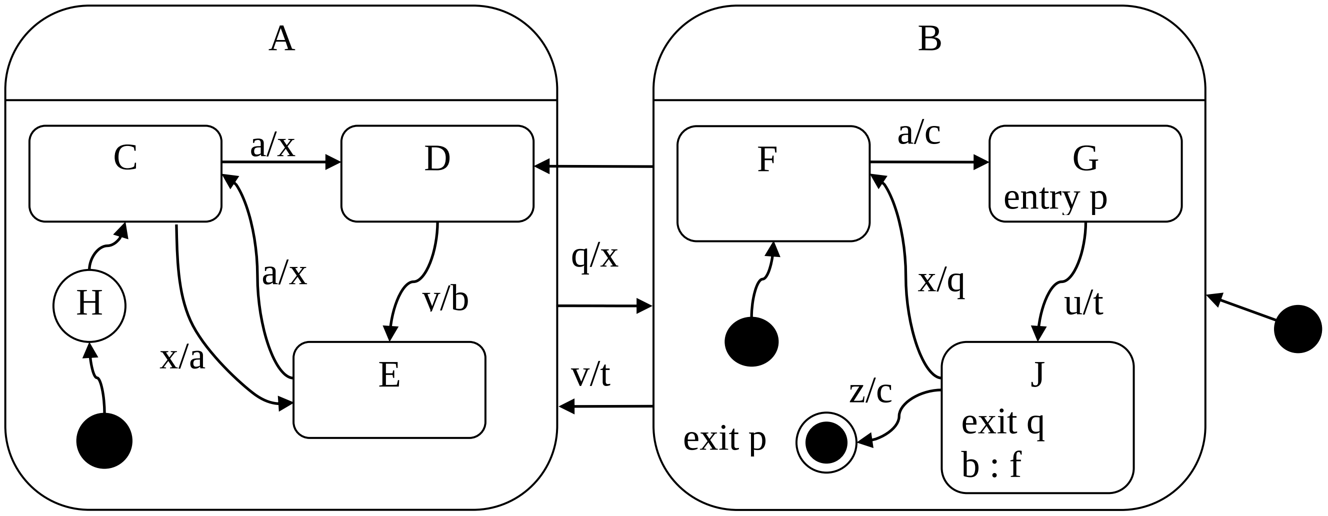


Define the UML2 name of the pointed items !


actor ....
use-case ....
association ....
Treatmen
Doctor
Patient

4 State diagram

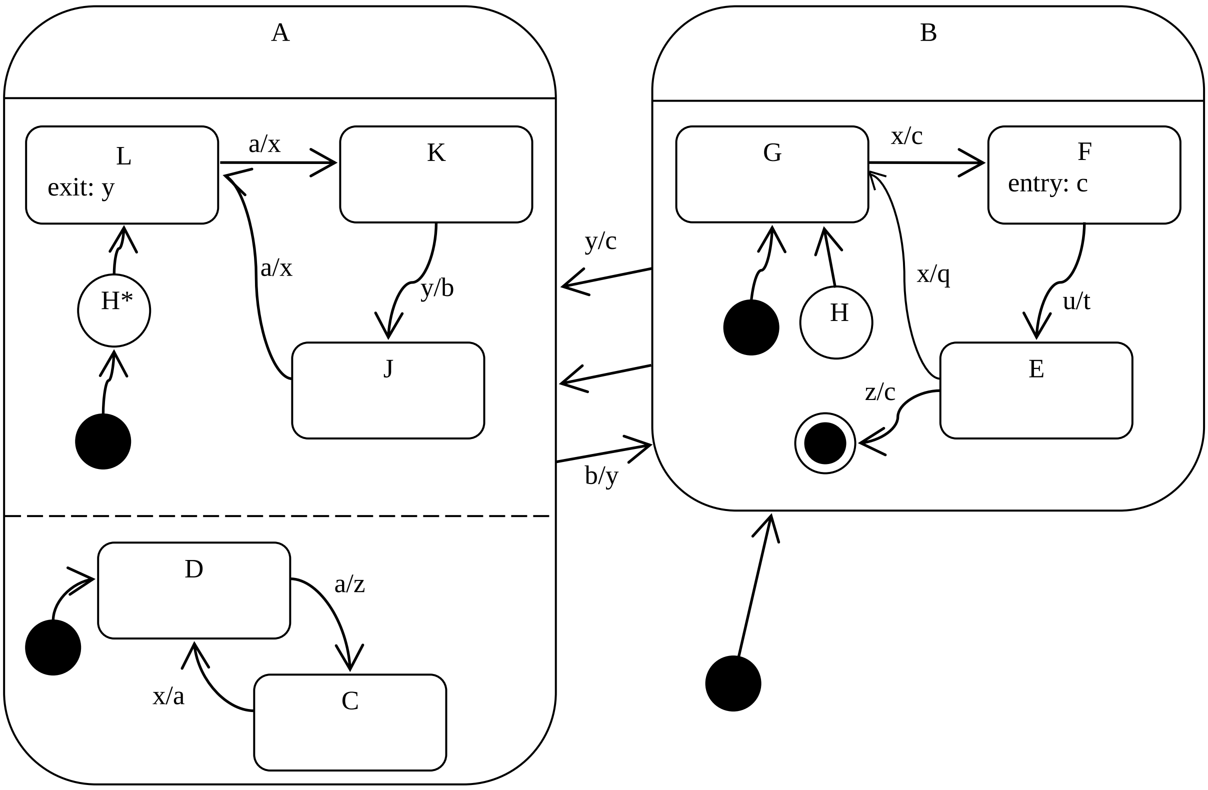
Classify the statements from the UML2 state-chart given below!



After start, receiving the event-sequence **a, u, b, z, a, v**

True	False	Statement
□	□	we’ve touched state E exactly two times
□	□	there was such event, which made no action
□	□	we didn’t execute action „b”
□	□	we’ve touched 5 different states
□	□	we are in C
□	□	action “a” was executed exactly once

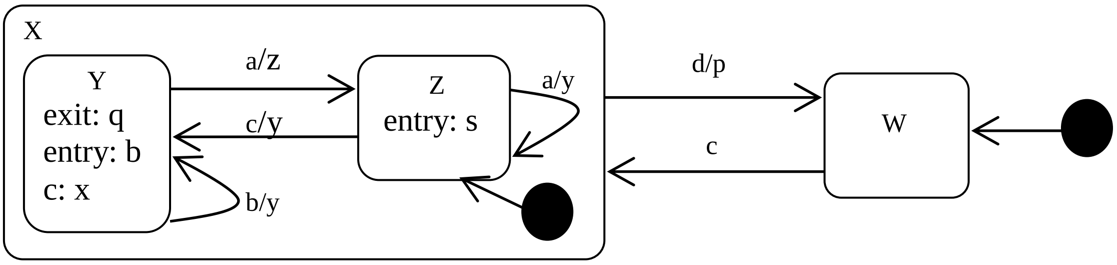
Classify the statements from the UML2 state-chart given below!



After start, receiving the event-sequence **x, y, a, b, x, u, z**

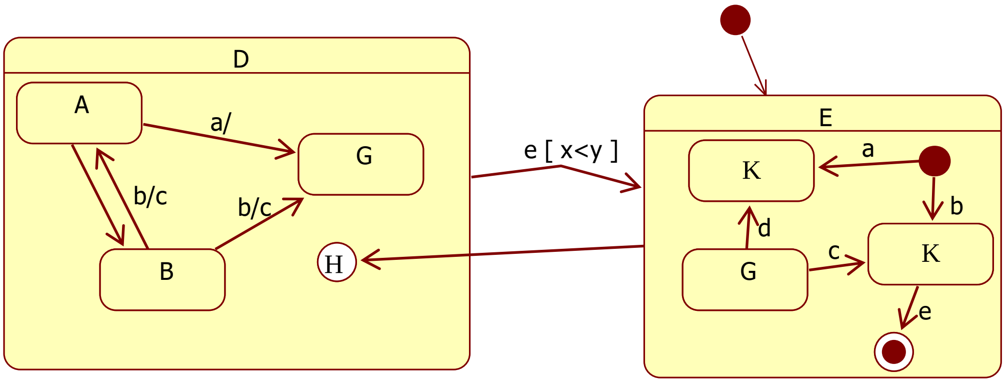
True	False	Statement
□	[✓ □]	we are in state E.
□	[✓ □]	we've touched state J and C at the same time.
[✓ □]	□	we've touched state K exactly two times
□	[✓ □]	every state was touched.
□	[✓ □]	action „c" is executed less then five times.
[✓ □]	□	there is such transition, when the action „x", „y"and „z" are all executed.
[✓ □]	□	we've touched state K and D at the same time.
□	[✓ □]	we are in state L/D.

Let's have the following UML2 state-chart! Draw an equivalent state tranzition table! Mark the unspecified blocks by "---"!



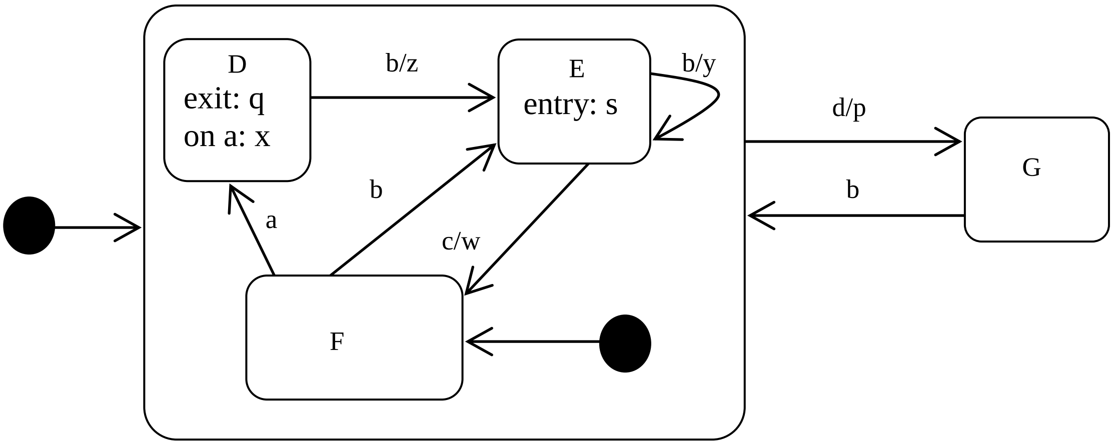
	d	c	b	a
W	---	Z/s	---	---
Z	W/p	Y/y, b	---	Z/y, s
Y	W/q, p	Y/x	Y/q, y, b	Z/q, z, s

List the syntactical and semantical errors in the diagram given below!



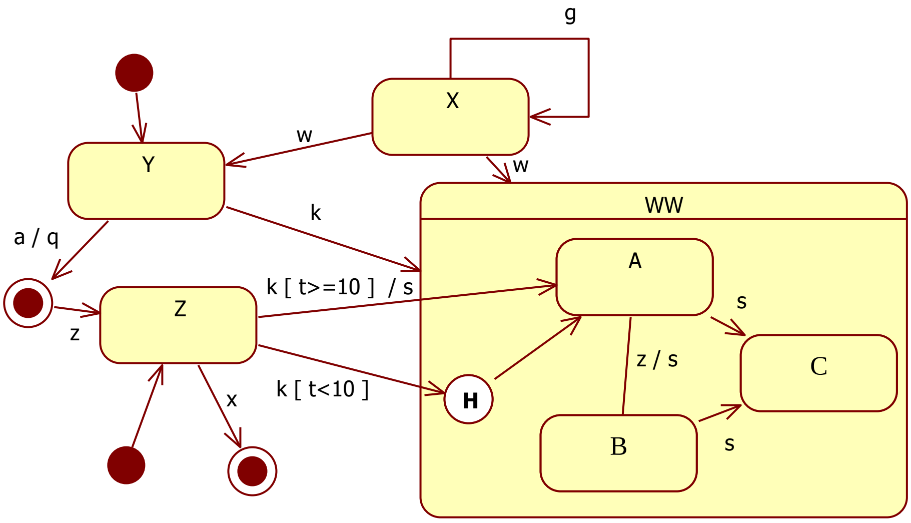
start of E is directed into two states..
2 K states are in E ....
history of the D has no predefined state ....
G of E has no input transition..
no event on the A->B transition ....
events are assigned to the transitions from the start of E...................

Let's have the following UML2 state-chart ! Draw an equivalent state transition table !



	a	b	c	d
D	D/x	E/q, z, s		G/q, p
E		E/y, s	F/w	G/p
F	D	E/s		G/p
G		F		

List the syntactical and semantical errors in the diagram given below!



Leaving from endstates

Entering into initial states

(initial and end states are changed)

More than one initial states

No initial state in state WW

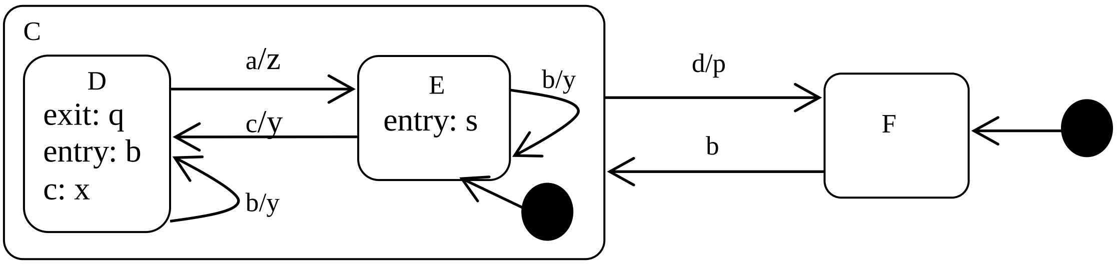
End state are not accessible form WW

Two transitions are connected to state Y in case of event „k"

Two predefined states are connected to History indicator in WW

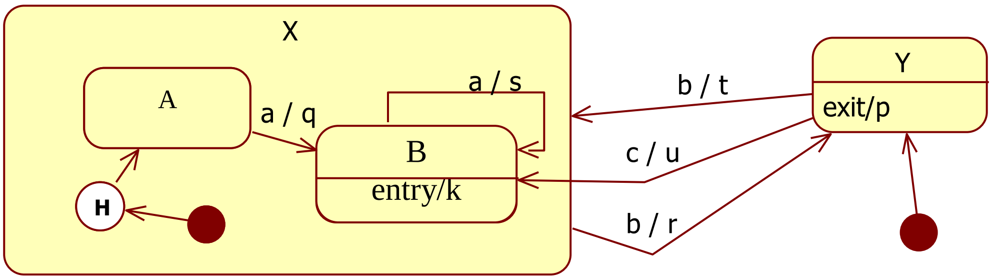
X has no input transition.(untuchable state)

Let's have the following UML2 state-chart ! Draw an equivalent state transition table !



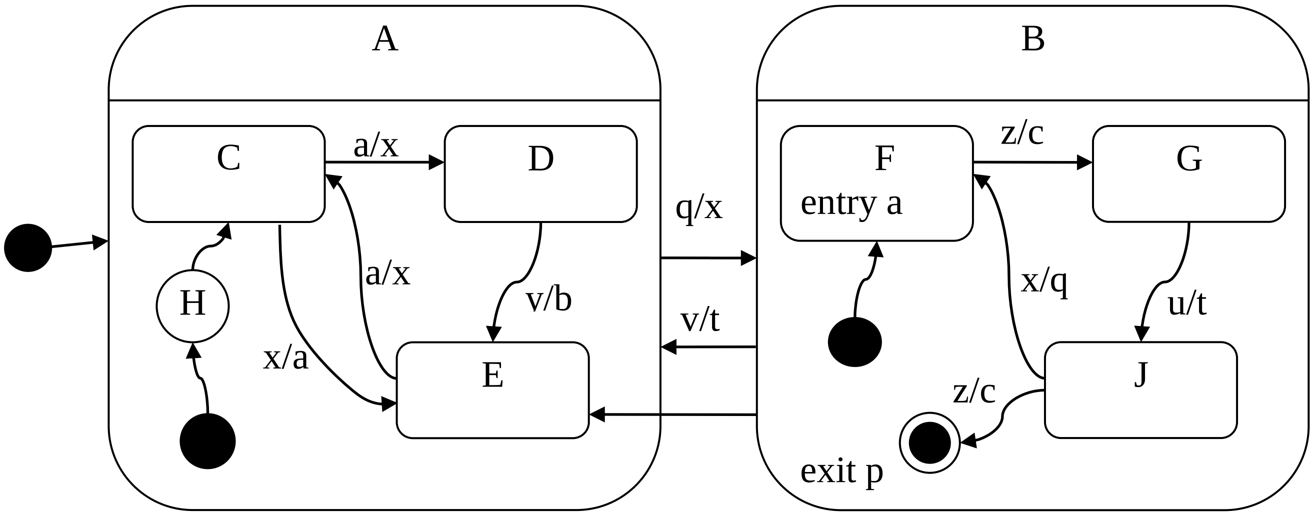
	a	b	c	d
D	E/q, z	D/q, y, b	D/x	F/q, p
E		E/y, s	D/y, b	F/p
F		E/s		

Let’s have the following UML2 state-chart! Draw an equivalent state transition table! Mark the unspecified blocks by “---“! The first row shall be the initial state!



	a	b	c
YA	---	A/p,t	B/p,u,k
YB	---	B/p,t,k	B/p,u,k
A	B/q,k	YA/r	---
B	B/s,k	YB/r	---

Classify the statements from the UML2 state-chart given below!

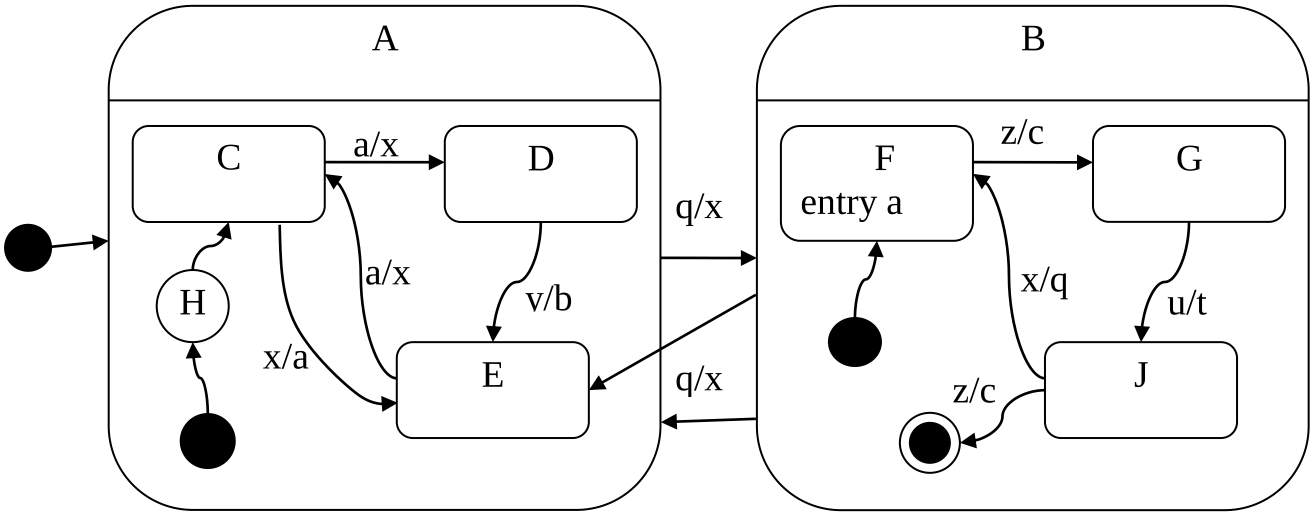


True	False	Statement
□	[✓ □]	receiving “p” in J, the next state is E
[✓ □]	□	during the transition from A to B, action “a” is always executed
[✓ □]	□	starting from D we return to D in two steps
□	[✓ □]	receiving “v” in F, the next state is H

After start, receiving the event-sequence **x, q, z, u, z**

True	False	Statement
[✓ □]	□	action “a” is executed two times
[✓ □]	□	action “p” is executed only once
[✓ □]	□	we’ve touched state E exactly two times
□	[✓ □]	we are in C

Classify the statements from the UML2 state-chart given below!

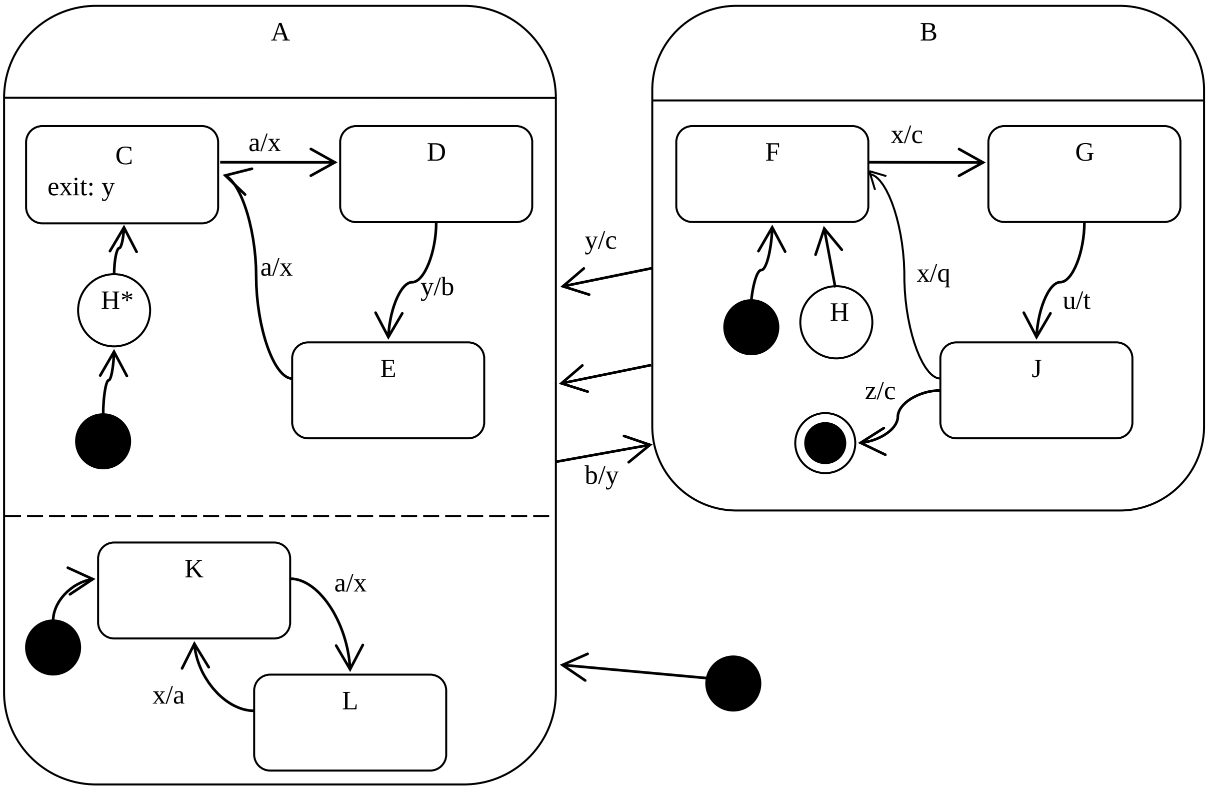


True	False	Statement
□	□	starting from D we return to D in two steps
□	□	from J we can move to E in one step
□	□	receiving “q” in F, the next state is H
□	□	during a transition from B to A, action “a” can be occurred

After start, receiving the event-sequence **x, q, z, q**

True	False	Statement
□	□	action “x” are executed two times
□	□	we are in E
□	□	action “a” are executed two times

Classify the statements from the UML2 state-chart given below!



True	False	Statement
□	☑	from state H we enter to F in any case
☑	□	leaving state B, action “c” is executed exactly once
☑	□	from state G we can enter to L in two steps
□	☑	we can be in state J and state K in the same time

After start, receiving the event-sequence **a, b, x, y, a, b**

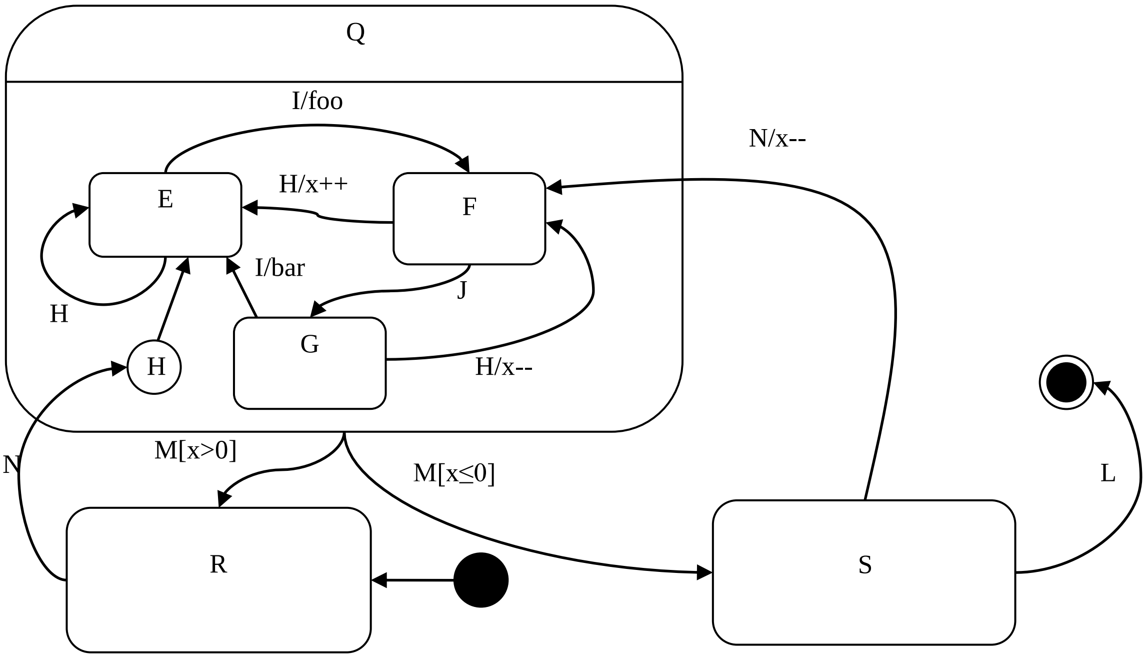
True	False	Statement
□	☑	action “x” is executed exactly two times
□	☑	we are in G
☑	□	action “c” is executed exactly two times
☑	□	we’ve touched state L exactly two times

Draw a UML2 state chart !

Let's have an object with three main states (**Q**, **R**, **S**). Within the main state **Q**, the internal control is defined by the state transition table given below.:

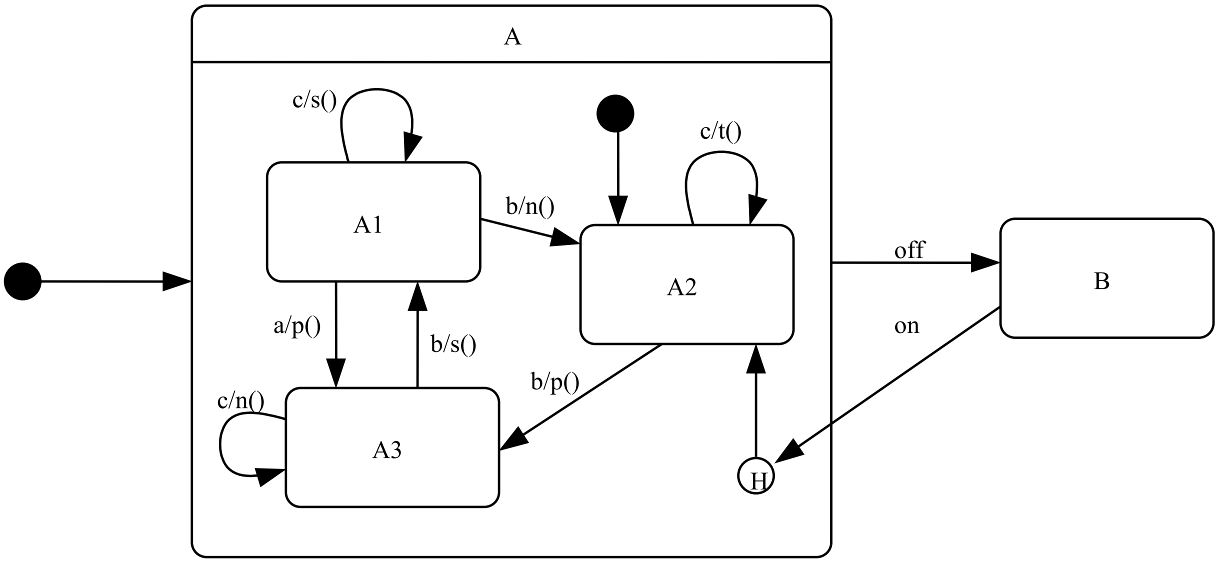
	H	**I**	**J**
E	E	F/foo	-
F	E/x++	-	G
G	F/x--	E/bar	-

Receiving an event **M** in state **Q**, if **x** is positive the next state is **R**, if **x** is non-positive the next is **S**. It returns from **R** to **Q** when receives an event **N**, and its internal control restores its state was in effect when the object has leaved state **Q**. The predefined substate is **E**. Event **N** also transfers it from **S** to **F**, but in this case **x** is decremented. The beginning state is **R**. Event **L** in state **S** stops the state machine.



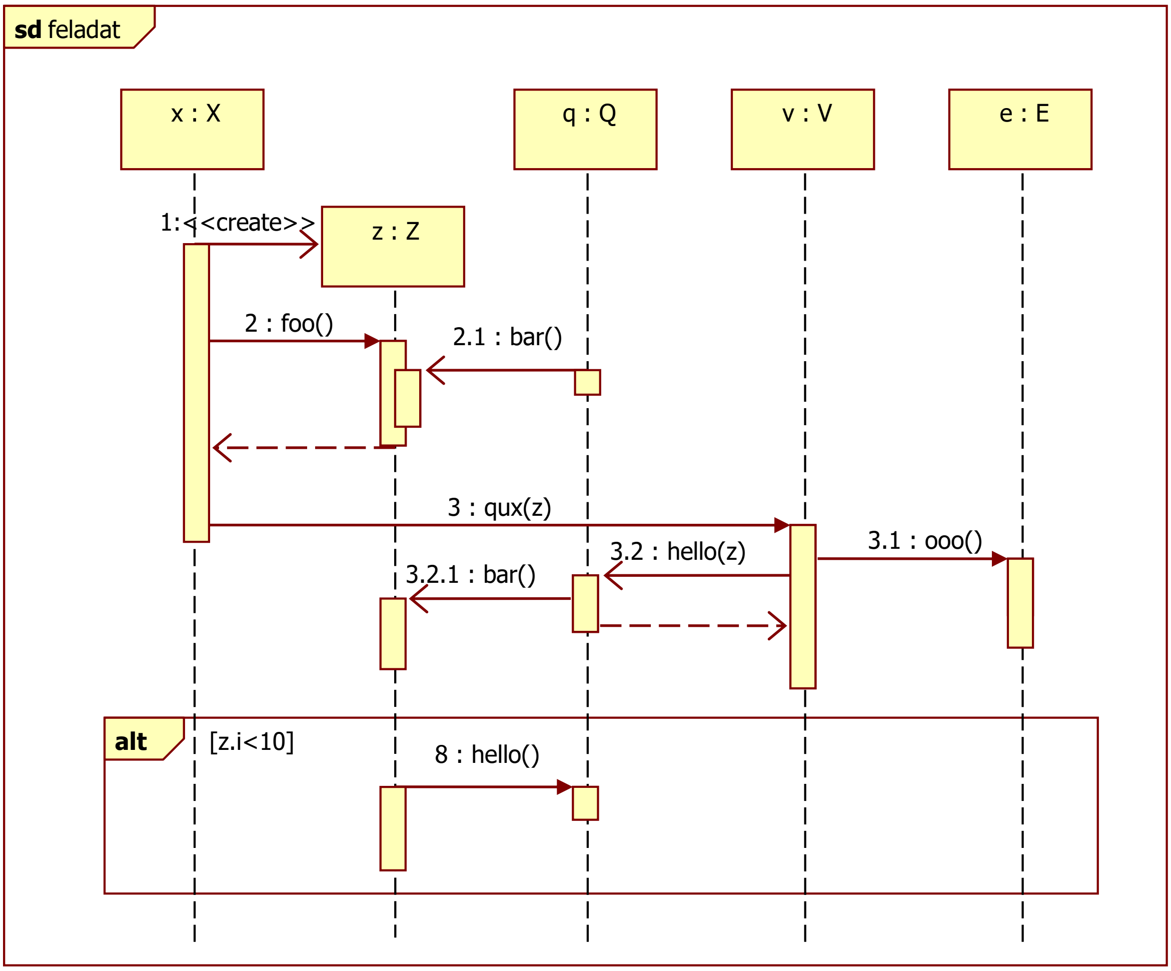
Let's have an object with two main states (A, B). Within the main state A, the internal control is defined by the state transition table given below. Receiving an event off in state A, the object changes its main state to B. It returns to the main state A, when receives an event on, and its internal control restores its state was in effect when the object has leaved state A. The beginning state is A2 within A. Draw a UML state chart !

	a	b	c
A1	A3/p()	A2/n()	A1/s()
A2	-	A3/p()	A2/t()
A3	-	A1/s()	A3/n()



5 Sequence diagram

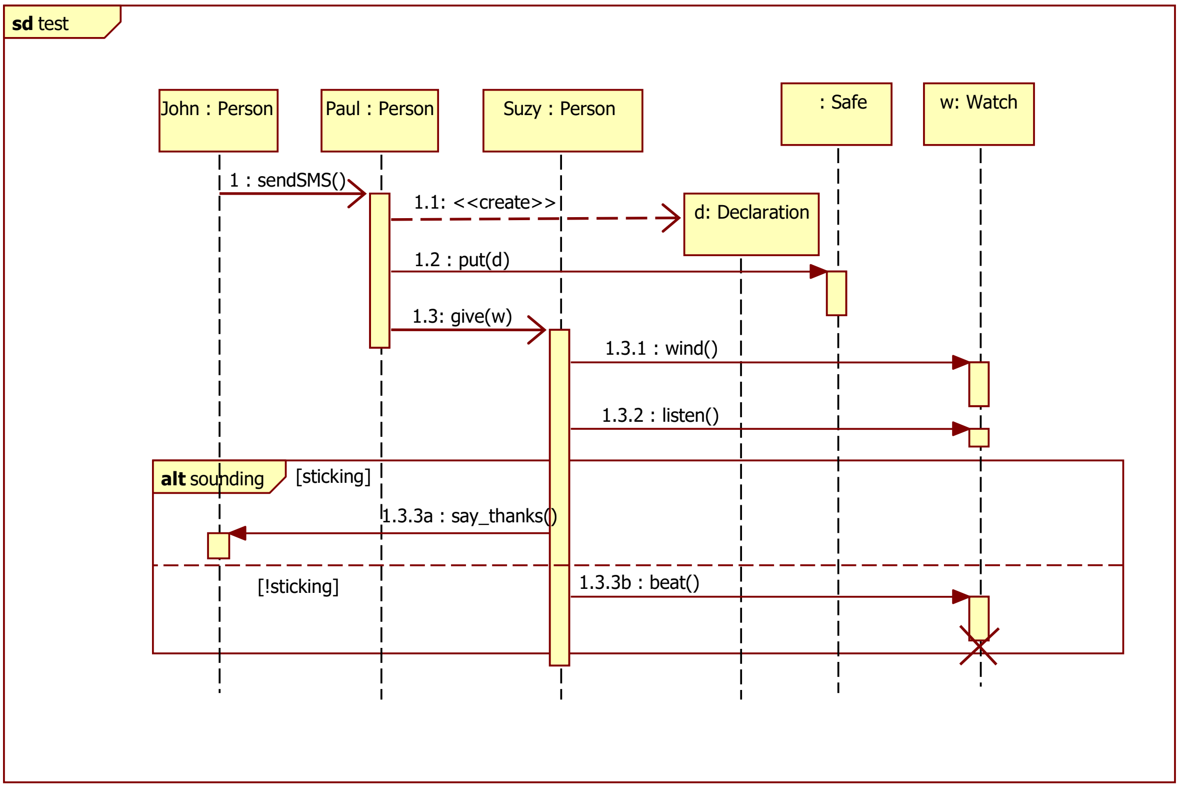
List the syntactical and semantical errors in the diagram given below!



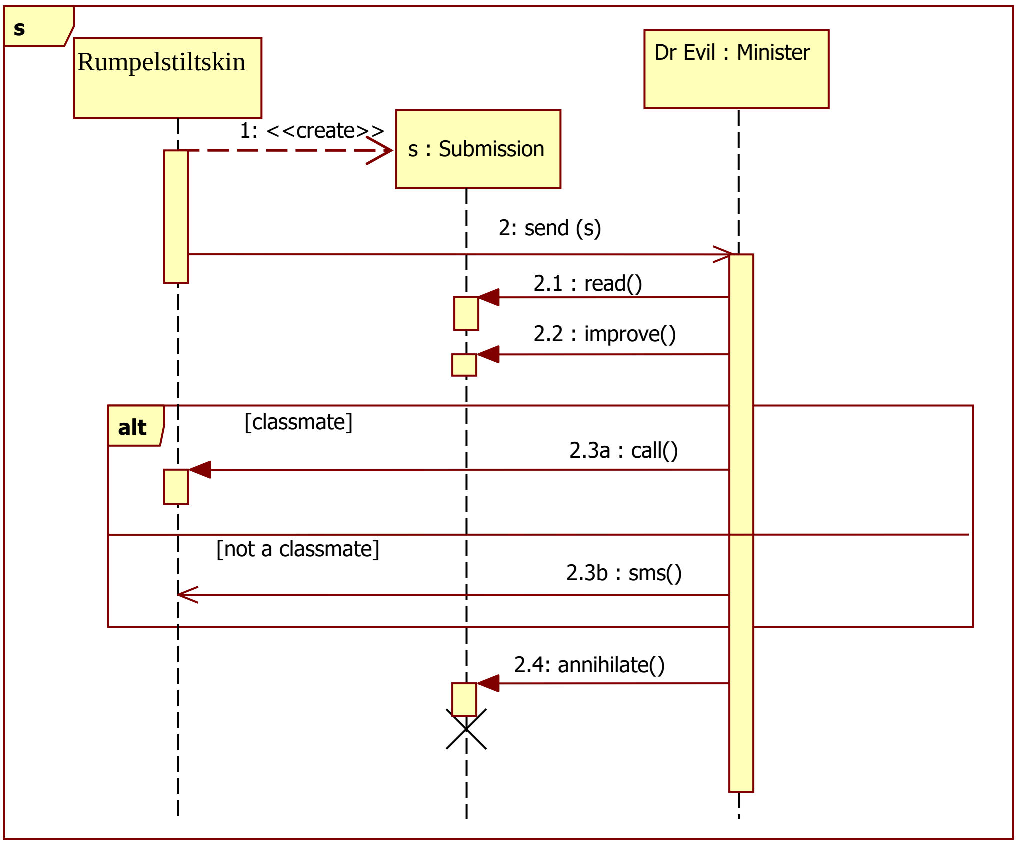
- 1:<<create>> is solid (instead of dashed)
- 2.1:bar() has no antecedent (Q has no reference to Z)
- X must be waiting for the return of the synchronous 3:qux(z) calling.
- before returning from the 3.1: ooo() calling, the 3.2:hello(z) message is sent.
- return from the asynchronous 3.2:hello(z) calling.
- alt box has no alternative
- 8:hello() has no antecedent

9. Draw a UML2 **sequence diagram** based on the story given below! Apply hierarchical numbering!

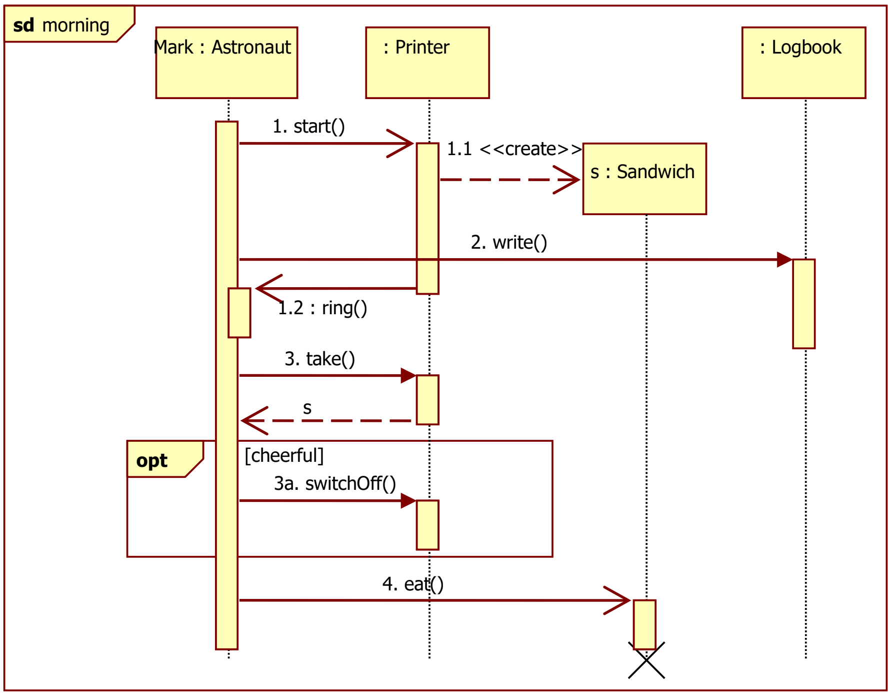
John sends an SMS to Paul in which John asks Paul to give away his old wind-up watch to Suzy. Paul creates a declaration and puts it into his safe, and then he gives his watch to Suzy. Suzy winds up the watch and she is listening its sound. If it's sticking, Suzy says thanks to Paul. If it is not sticking Suzy beats the watch.



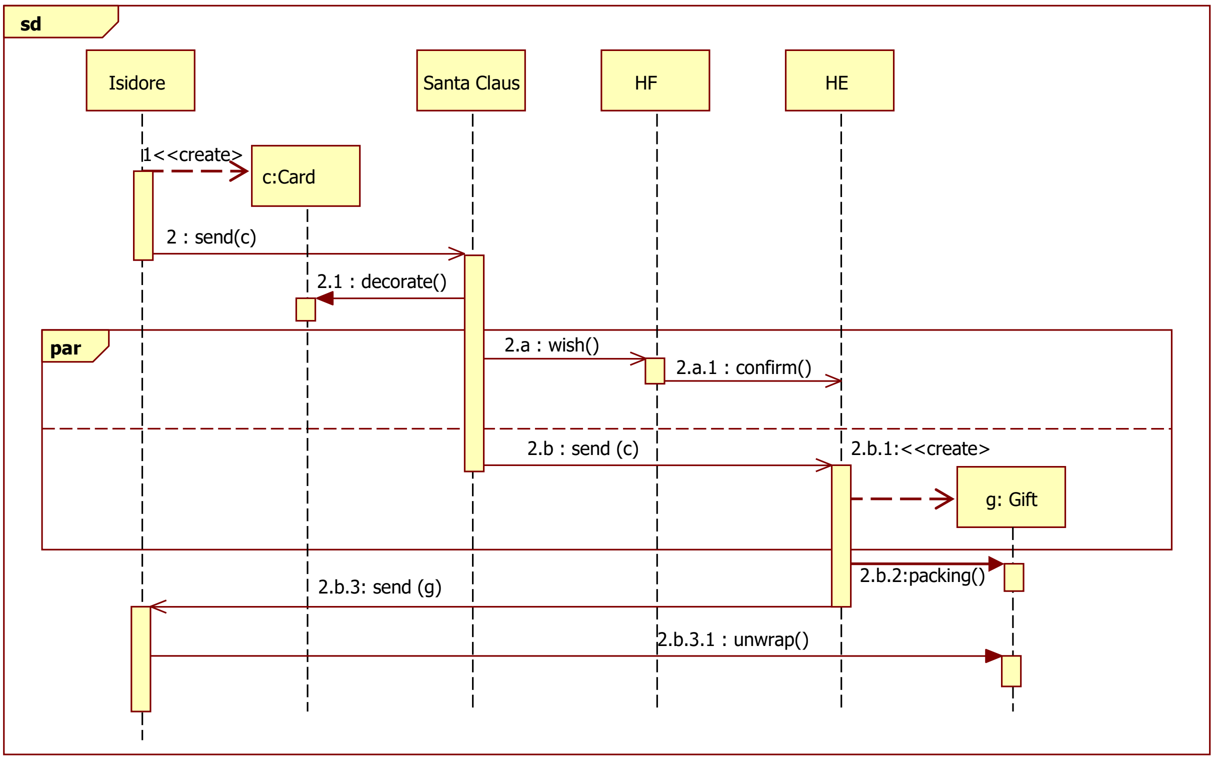
Draw a UML2 **sequence diagram** based on the story given below!
In a remote corner of Nernia, there is a young man, Rumpelstiltskin, who wants to open a cafe shop. He makes a submission, and sends it to Dr Evil, the minister. The minister reads the submission and then improves the spelling. If Rumpelstiltskin was a classmate of the minister, Dr Evil calls him. If not, Dr Evil sends an SMS him. At the end, the minister annihilates the submission.



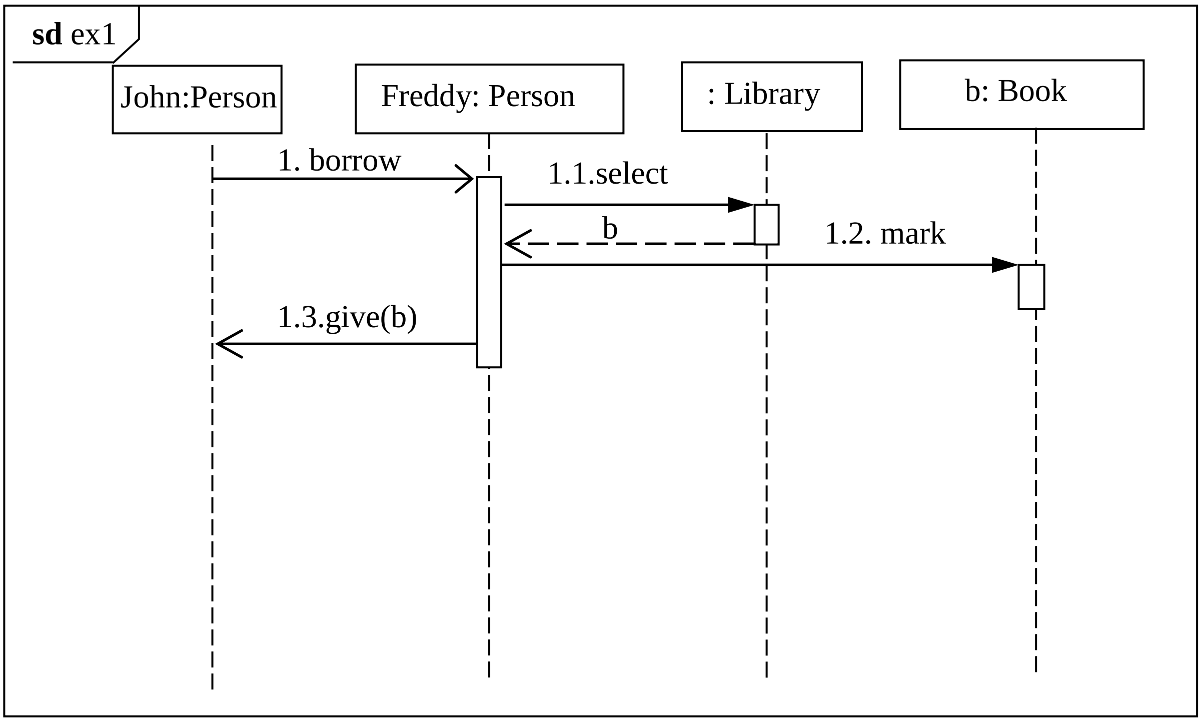
Draw a UML2 sequence diagram based on the following story! Use hierarchical numbering!
On the spaceship WoggaWogga3, astronaut Marc Schuldenberg starts his 3D-printer to print a ham&cheese sandwich. During printing, Marc writes remarks into the logbook. During writing, the printer rings the bell when the sandwich is ready. When Marc finishes the logging, takes out the sandwich from the printer. If Marc is cheerful, then he switches off the printer. Finally, he eats the sandwich.



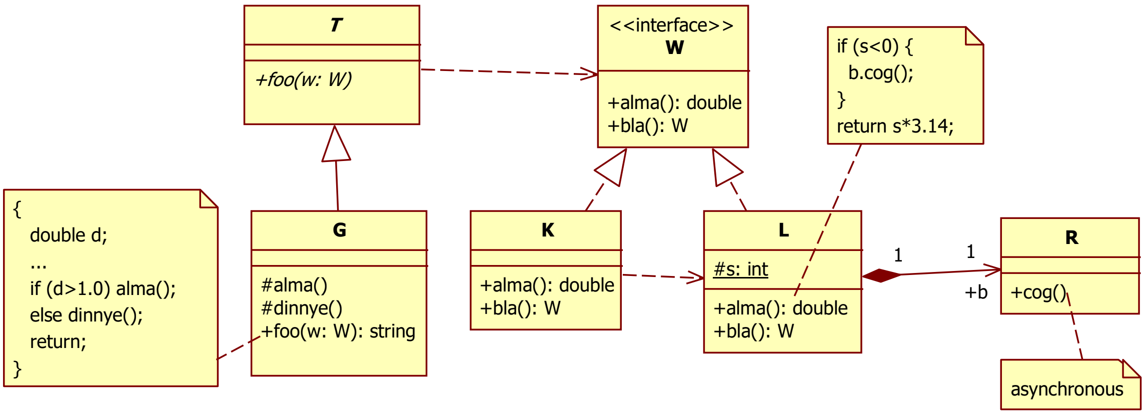
Draw a UML2 **sequence diagram** based on the story given below!
Isidore creates a request card specifying his christmas wish, and sends it to Santa Claus. Santa Claus decorates it and later forwards the wish to the Heavenly Financial Office (HFO) and the card to the Heavenly Export Authority (HEA) in unknown sequence. Receiving the card the HEA conjures the gift. Meantime the HFO makes decision and sends the confirmation to the HEA. When HEA receives the confirmation, packs the the gift and sends the pack to Isidore. Isidore unwraps the present.

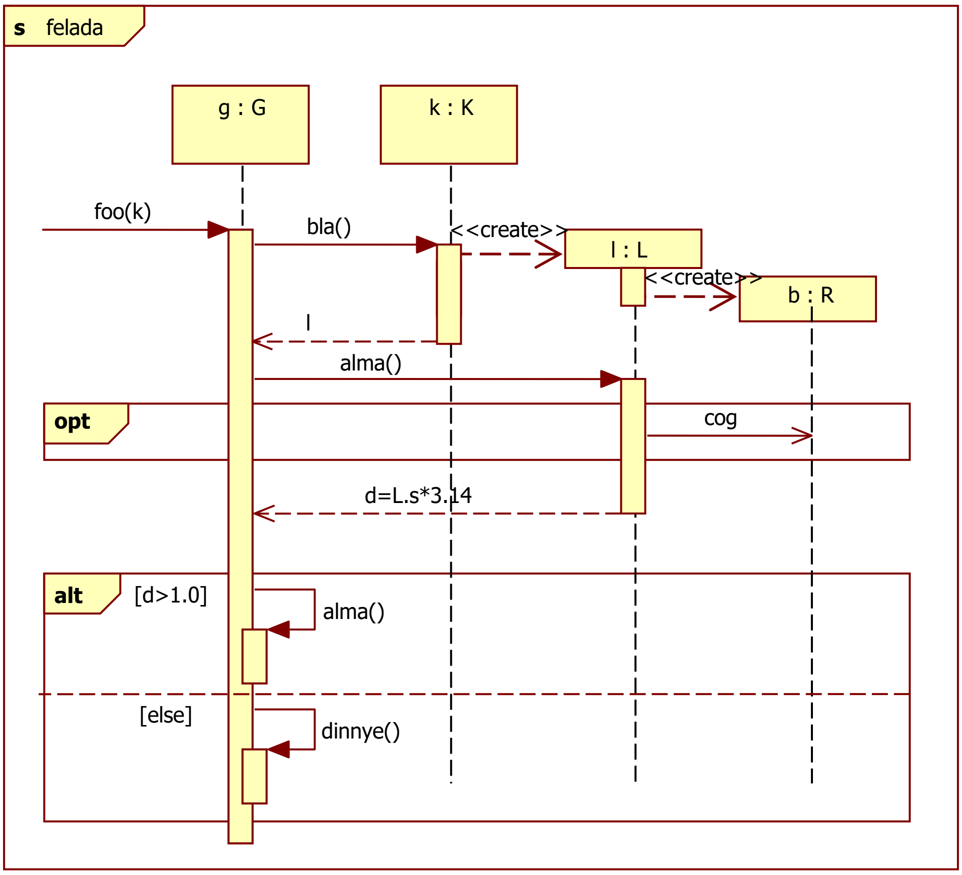


Draw a UML2 sequence diagram !. Apply hierarchical numbering on messages !
John borrows the best Software Engineering book from Freddy. At home Freddy selects the book from his library, and marks the book: writing his name onto the first page. Next day he gives the book to John.



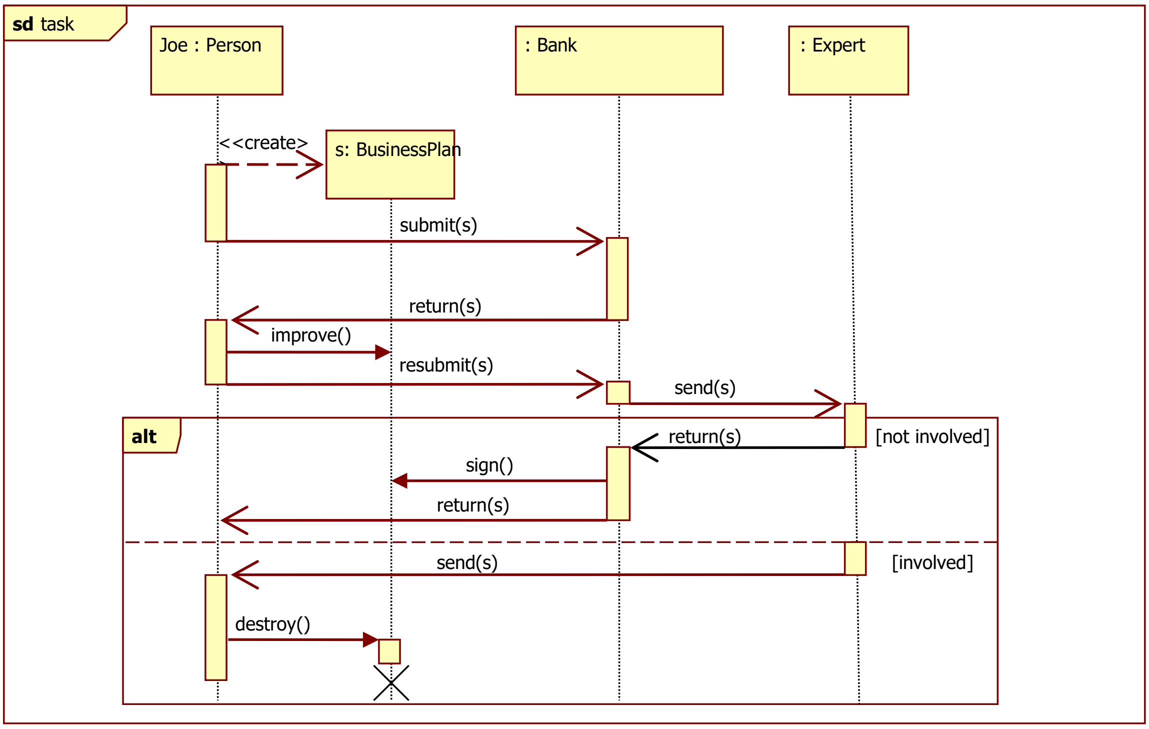
Based on the class diagram given below, draw a UML2 sequence diagram, in which each method occures exactly once (more methods with identical signature are not allowed)! No numbering! Use the Java code-excerpts in the notes! "... " marks non-specified part. Use each return value! The first method-calling will be the *foo(w:W)* with proper parameter!



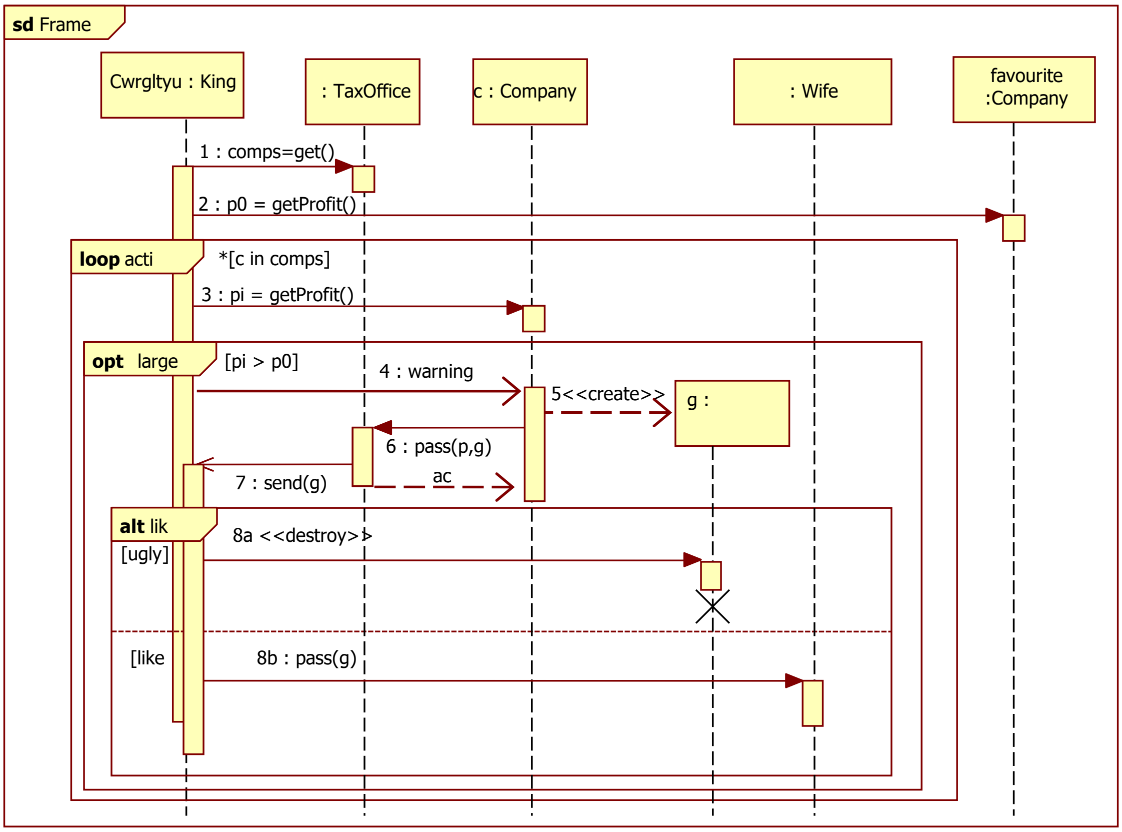


Draw a UML2 sequence diagram based on the story given below!

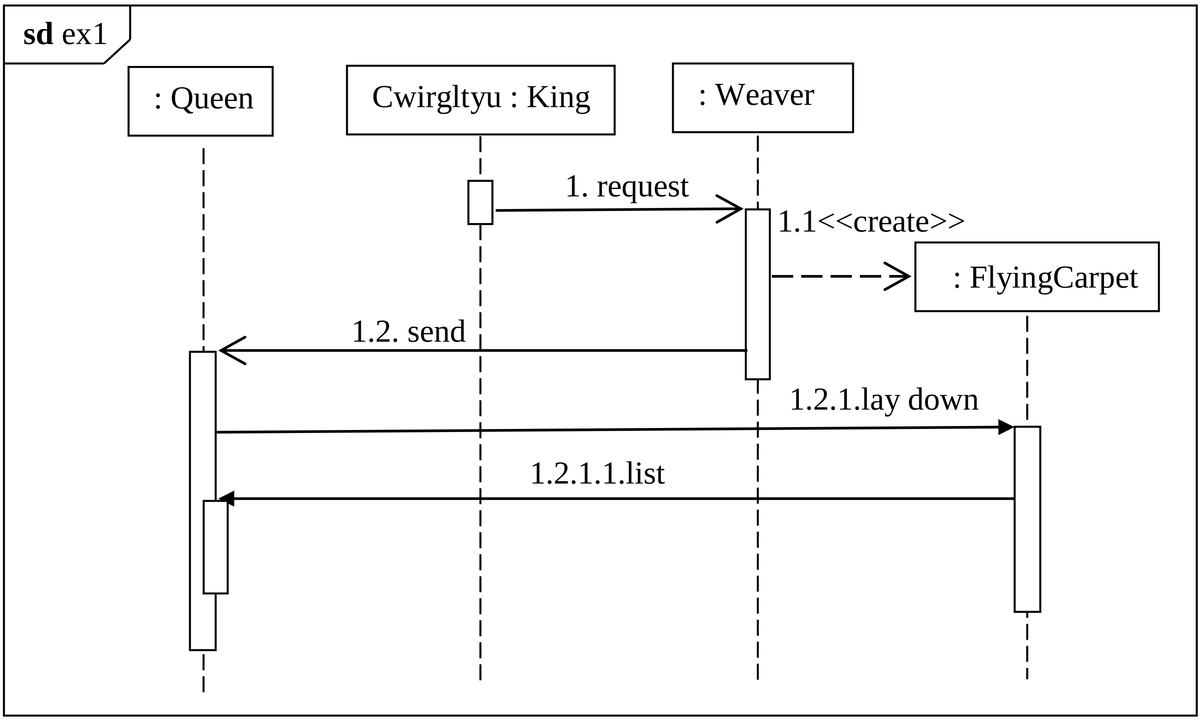
Joe elaborates a business plan, and submits to a bank. Because the plan is not complete, the bank returns it. Joe improves the plan and resubmit. The bank sends the plan to an expert for reviewing. If the expert is not involved in the business, it returns the plan to the bank. The bank signs, and accepts the plan, and returns to Joe. If the expert is involved in the same business, he sends the plan to Joe for modifying the plan according to the expert's needs. Because Joe doesn't accept the expert's needs, he destroys the plan.



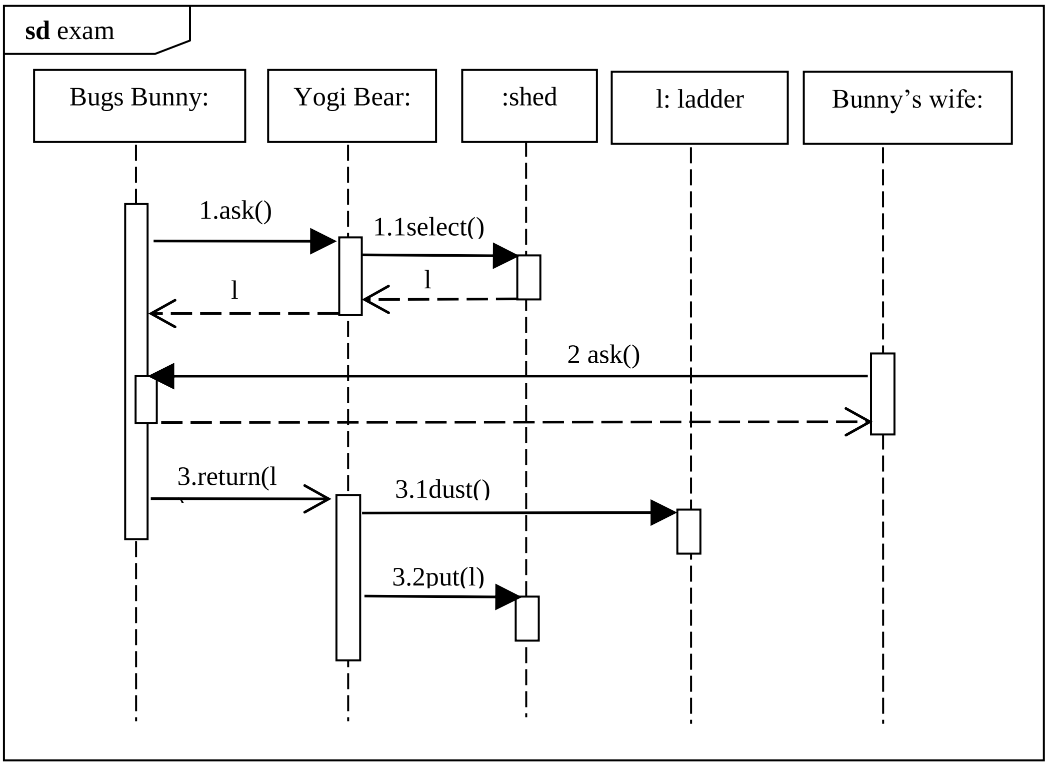
Draw a UML2 sequence diagram based on the story given below!
King Cwrgltyu asks the list of the profitable companys from the Tax Office. He asks the balance from the companies on the list. Moreover the King sends a warning to those companies, which company's profit is larger than the profit of his favourite company. The warned company is obliged to create a gift, and to pass the profit and the gift to the Tax Office. The Tax Office immediately sends the gift to the King, and then reply an acknowledgement to the company. The King destroys the gift, if it is ugly. If the King likes it, he passes to his wife.



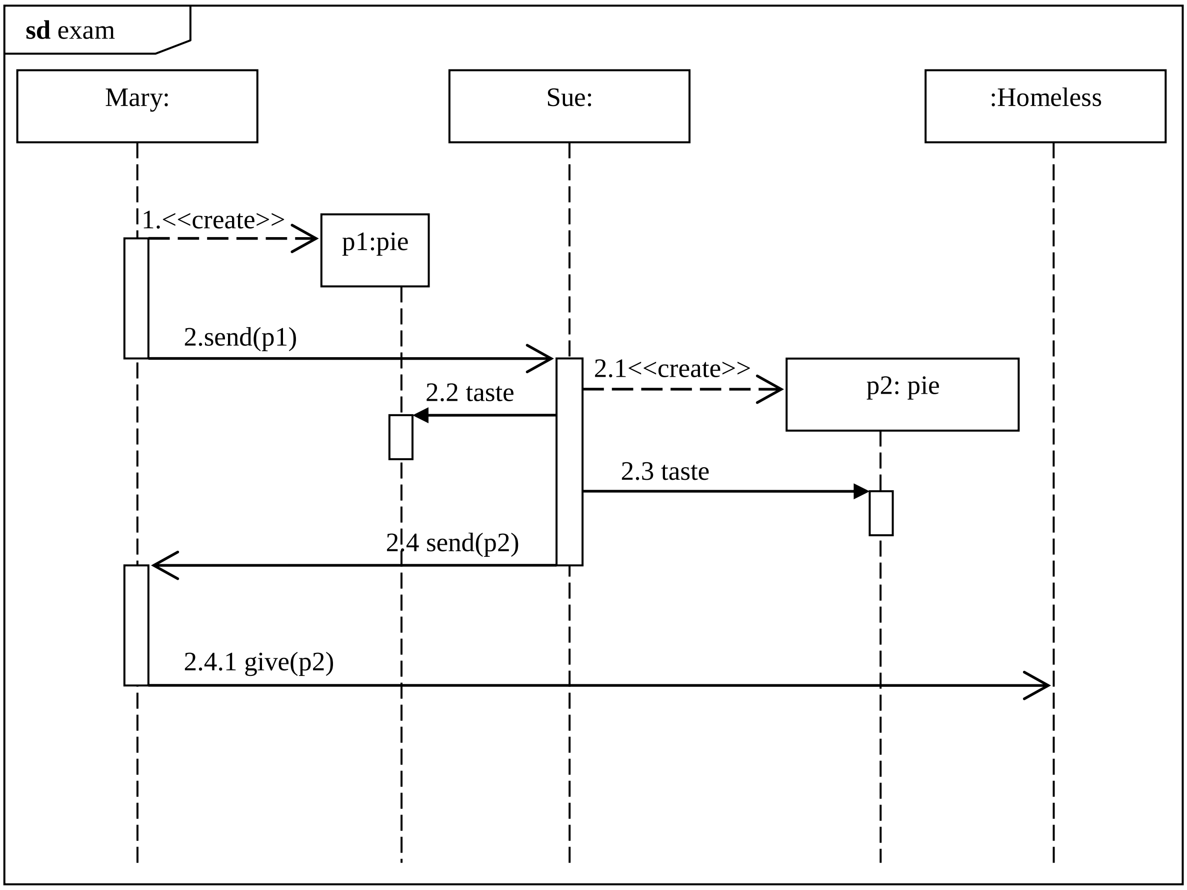
Draw a UML2 sequence diagram !. Apply hierarchical numbering on messages !
King Cwirgltyu sends a request to his weaver to create a flying carpet for his wife, the Queen. (It is a hard work, creating a flying carpet takes a long time). When the carpet is ready, the weaver sends it to the Queen. The Queen lays down the carpet immediatly and the carpet lists the available destinations to the Queen.



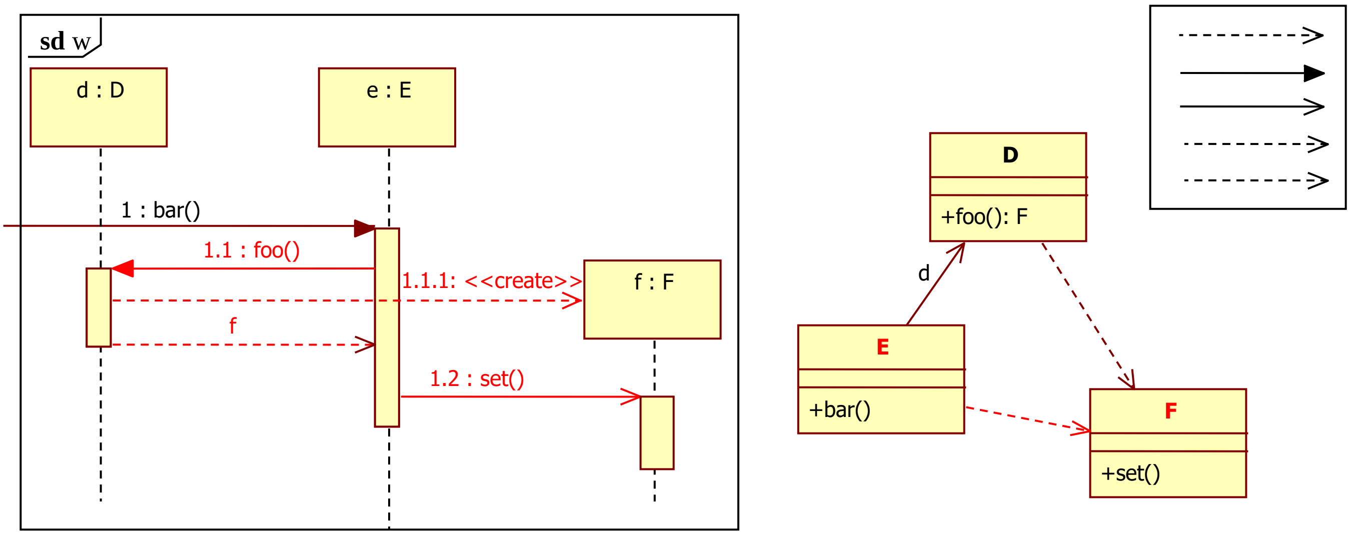
Draw a UML2 sequence diagram ! Apply hierarchical numbering on messages !
Bugs Bunny visits Yogi Bear and asks for a ladder. Yogi Bear selects his most beautiful ladder in the shed and gives it to Bunny. At home Bunny's wife asks his husband whose this ladder is. When Bunny finishes his job, he returns the ladder to the Bear. Yogi Bear dusts the ladder and puts it into the shed.



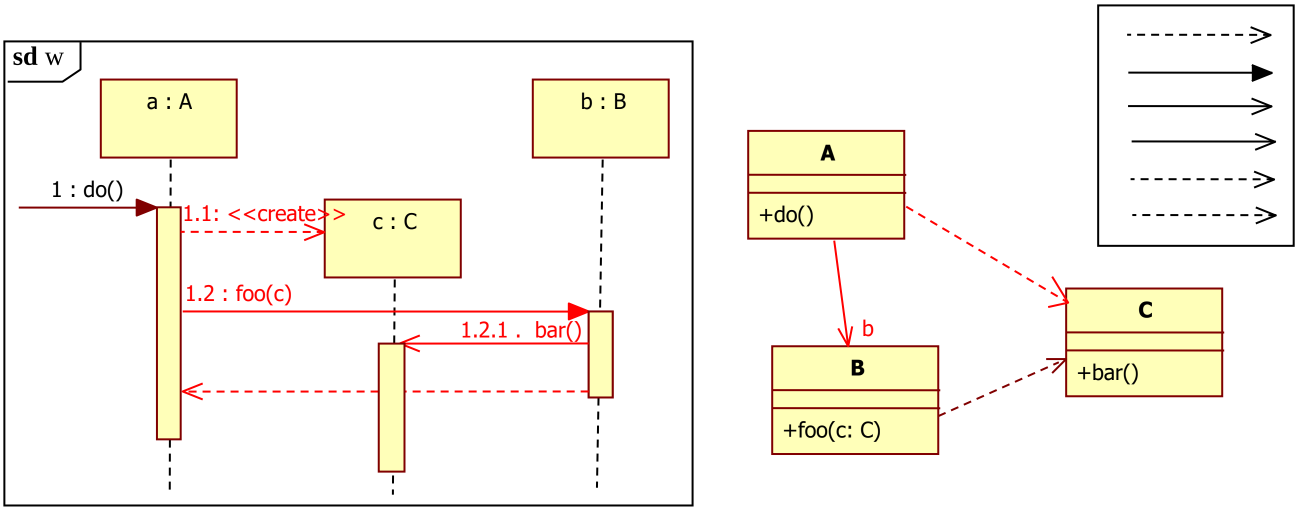
Draw a UML2 sequence diagram ! Apply hierarchical numbering on messages !
Mary bakes an apple pie and sends it to Sue. Sue immediately bakes a banana pie, tastes both pies, and sends her banana pie to Mary. Mary gives the received pie to the first homeless on the street.



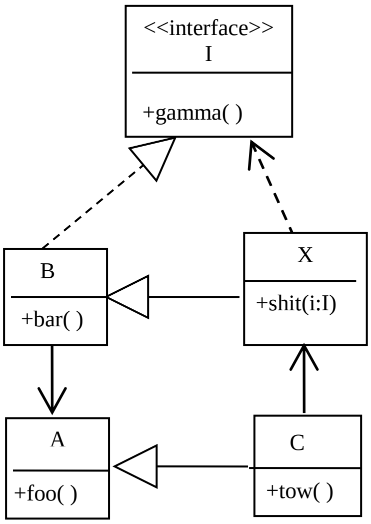
Given a UML2 class diagram and a UML2 sequence diagram below. The diagrams are semantically coherent, but some arrows are missing. The missing arrows can be found in the separate box. Insert each arrow into the diagrams in order to have both syntactically and semantically coherent diagrams! Attach label to the arrows as needed!

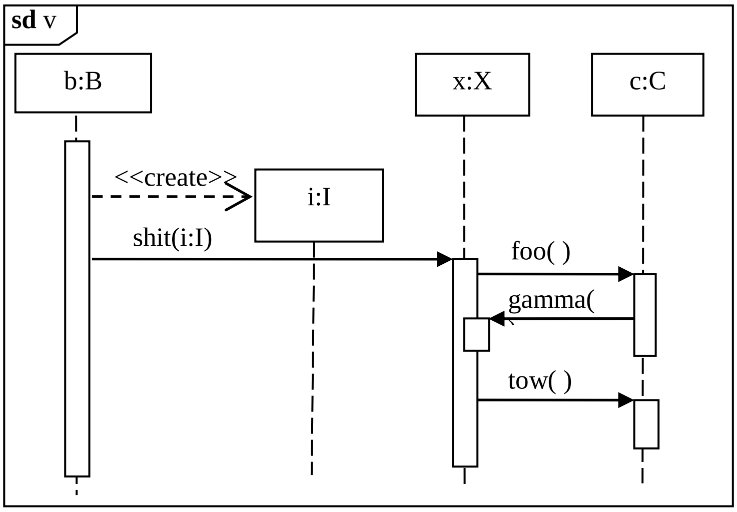


Given a UML2 class diagram and a UML2 sequence diagram below. The diagrams are semantically coherent, but some arrows are missing. The missing arrows can be found in the separate box. Insert each arrow into the diagrams in order to have both syntactically and semantically coherent diagrams! Attach label to the arrows as needed!



Given a UML2 class diagram and a UML2 sequence diagram below. The two diagrams are semantically coherent. Supposing that the class diagram is fully correct, list the semantical and/or syntactical errors of the sequence diagram !





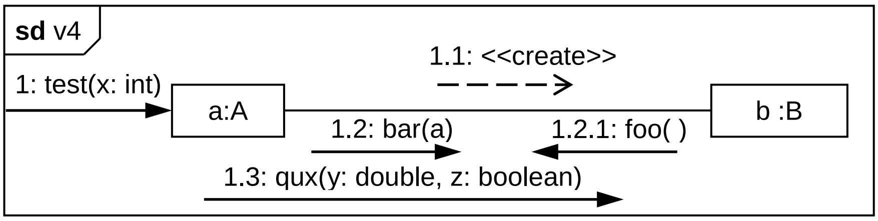
Interface is not instanciable

B doesn’t see X, so B’s shit(i:I) cannot be called

X doesn’t see C’s tow( )

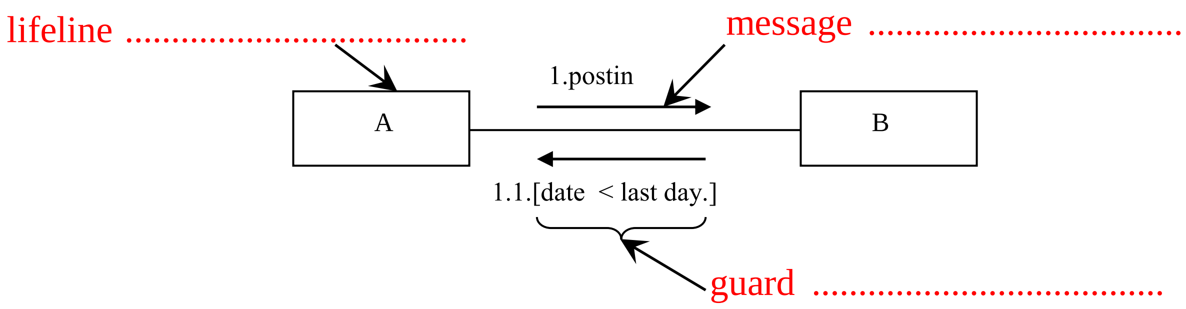
6 Communication diagram

Define the minimum instance attributes of the classes to realize the behavior defined by the UML2 communication diagram given below!



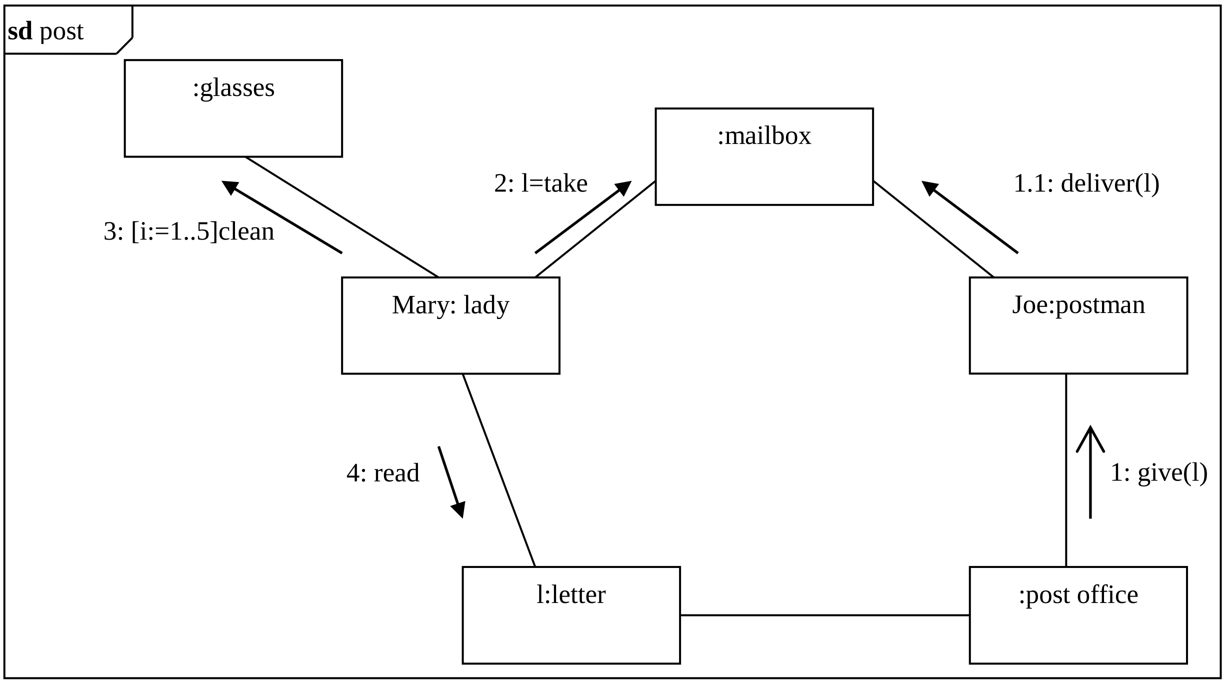
Class	No need attribute	Type of the attribute
A	□	
B	□	

Define the UML2 name of the pointed items !


lifeline ....
message ....
1.postin
A
B
1.1.[date < last day.]
guard ....

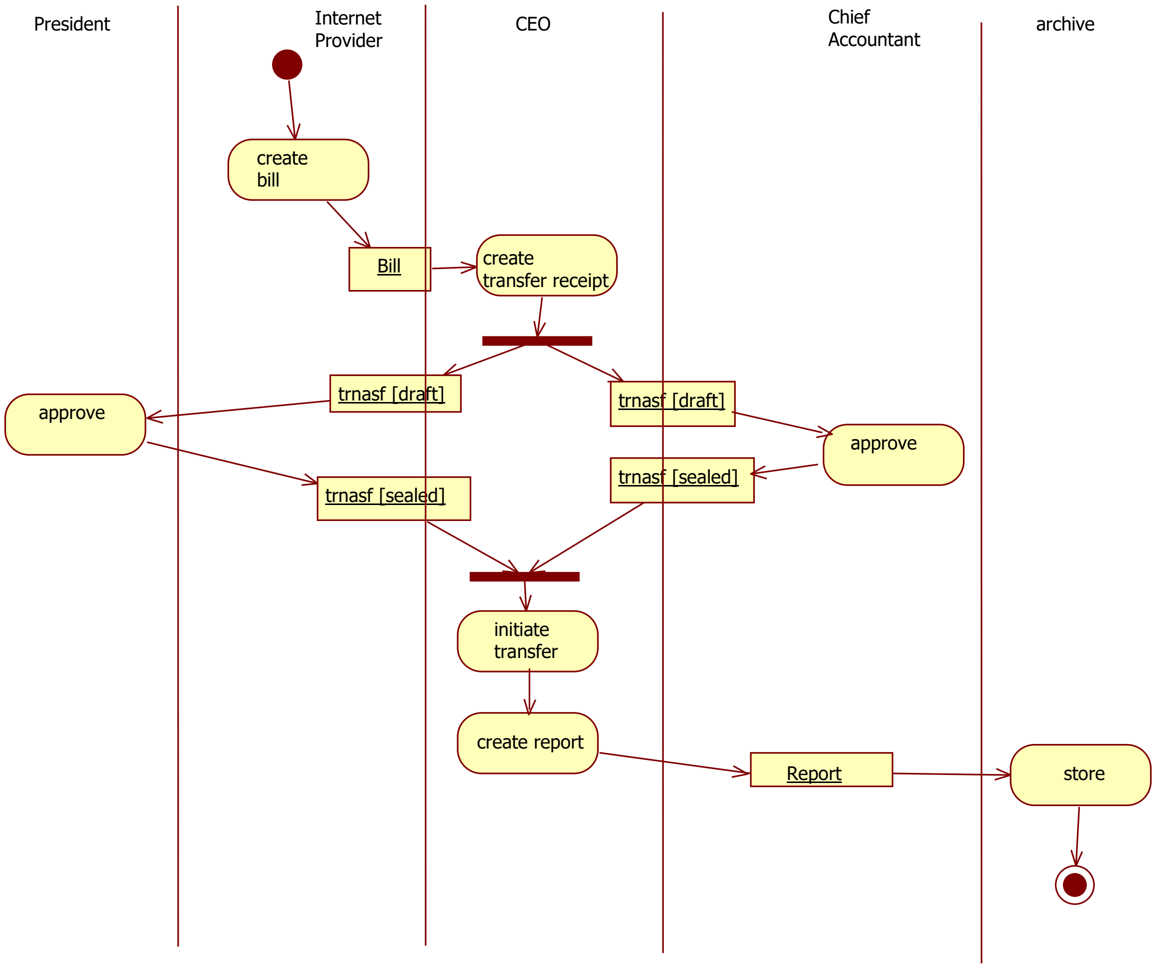
Draw a UML2 communication diagram !

In the post office a letter to be delivered to Mary Smith is given to Joe, the postman. Joe delivers this letter into the mailbox of Mary Smith. In the evening, Mary takes out the letter from the mailbox. She cleans her glasses five times, and then reads the letter.

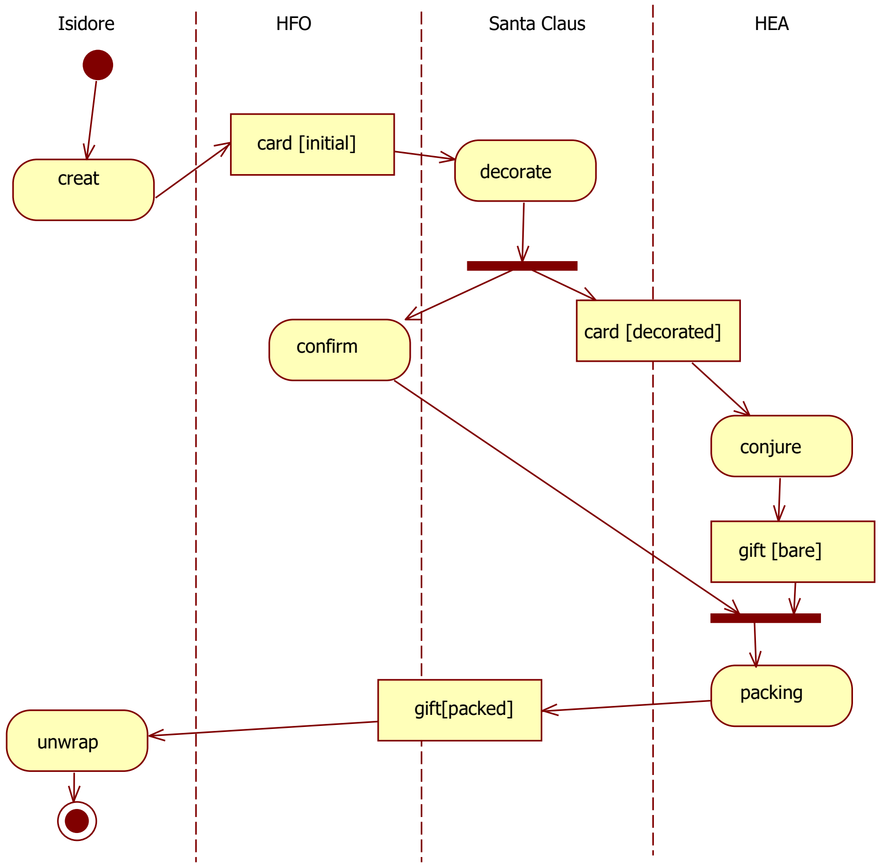


7 Activity diagram

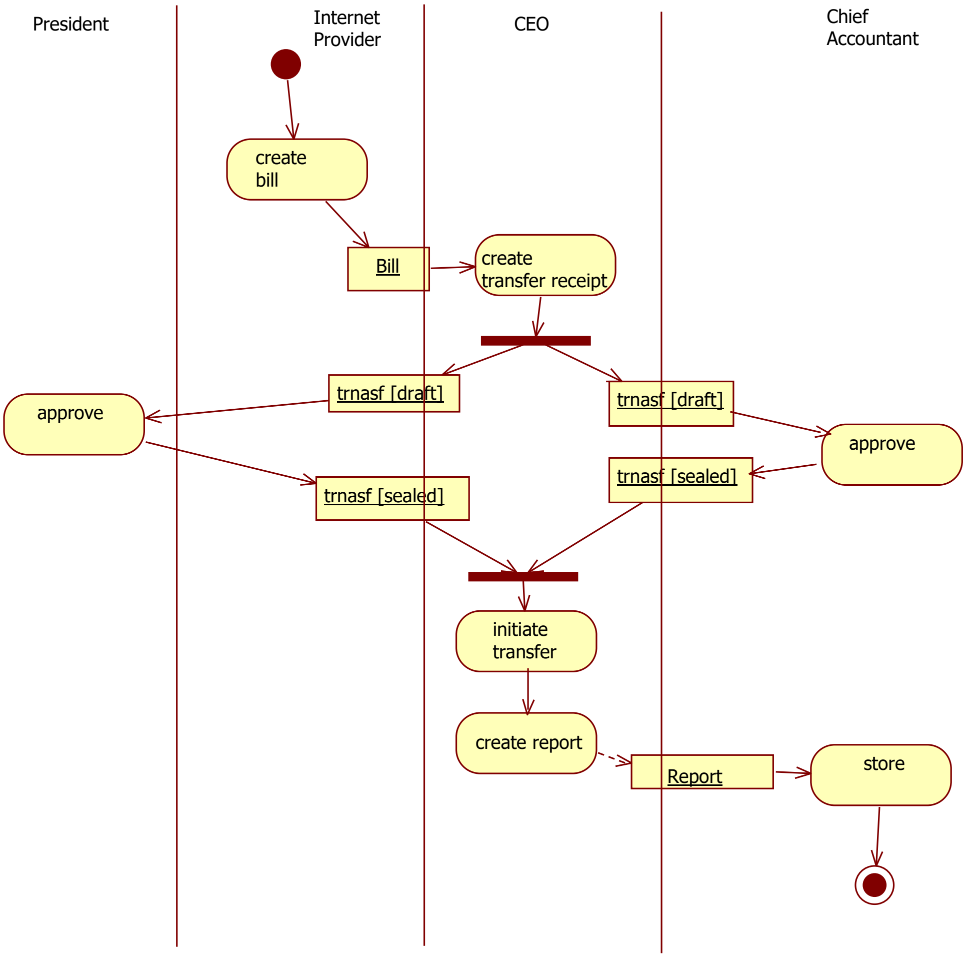
Draw a UML2 activity diagram (extended by object flow) based on the story given below! (8 pts)
The internet provider creates a bill and sends to the CEO of the software company. The CEO creates the draft of the transfer order, and sends it to the President and the chief accountant. They send back the approved and sealed draft to the CEO, who initiates the transfer. Finally the CEO creates a report and sends it to the archive, where it is stored.



Draw a UML2 **activity diagram** (extended by object flow) based on the story given below!
Isidore creates a request card specifying his christmas wish, and sends it to Santa Claus. Santa Claus decorates it and later forwards the wish to the Heavenly Financial Office (HFO) and the card to the Heavenly Export Authority (HEA) in unknown sequence. Receiving the card the HEA conjures the gift. Meantime the HFO makes decision and sends the confirmation to the HEA. When HEA receives the confirmation, packs the the gift and sends the pack to Isidore. Isidore unwraps the present.



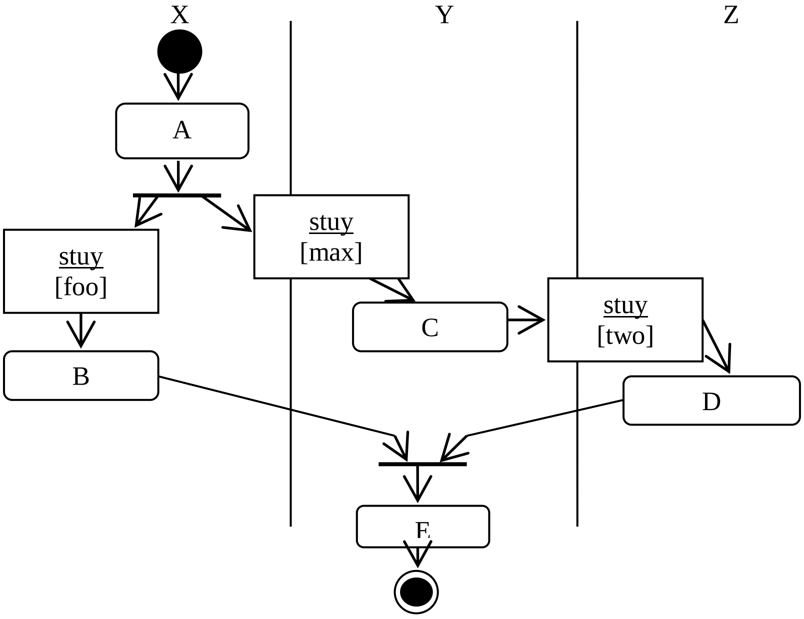
Draw a UML2 activity diagram (extended by object flow) based on the story given below!
The internet provider creates a bill and sends to the CEO of the software company. The CEO creates the draft of the transfer receipt, and sends it to the President and the chief accountant. They send back the approved and sealed draft to the CEO, who initiates the transfer. Finally the CEO creates a report and sends it to the archive, where it is stored.

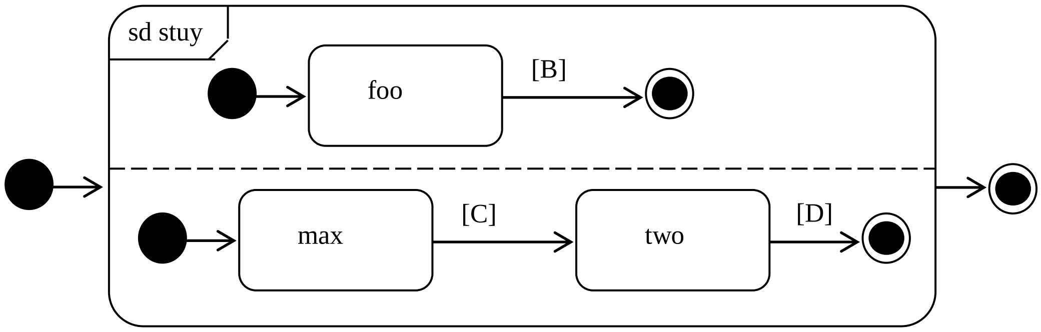


What does an edge in a UML2 activity diagram represent?

- □ only control flow
- □ only data flow
- □ control flow or data flow
- □ control flow or data flow or both

Based on the activity diagram – extended by object flows – draw a UML2 state-chart of such an object, which has more states!



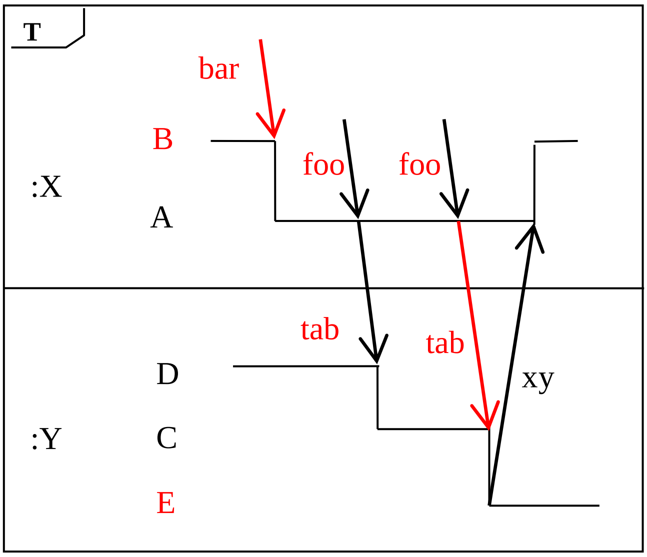


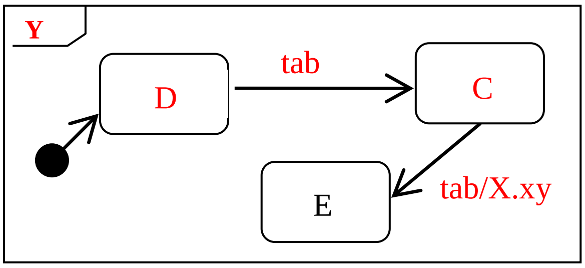
The UML2 Activity diagram can be regarded as a special version of an “other” UML2 diagram. What is this “other” diagram? Compare these two diagrams!

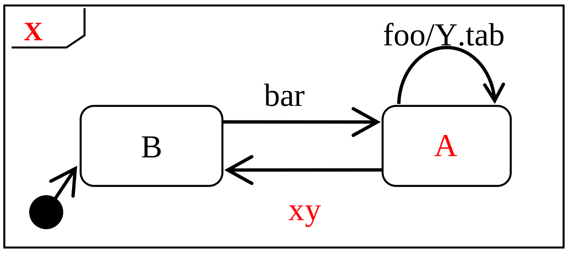
Activity diagram	Statechart……… diagram
state – do activity	state – wait for an event
transition – implicit event when the activity is finished	transition – external event initiates transition.

8 Timing diagram

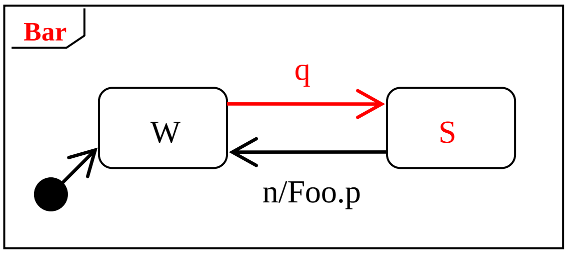
Given three UML2 diagrams. The diagrams are semantically coherent, but some elements (arrow, text, etc.) are missing. Add proper elements to the diagrams to have both syntactically and semantically coherent diagrams!

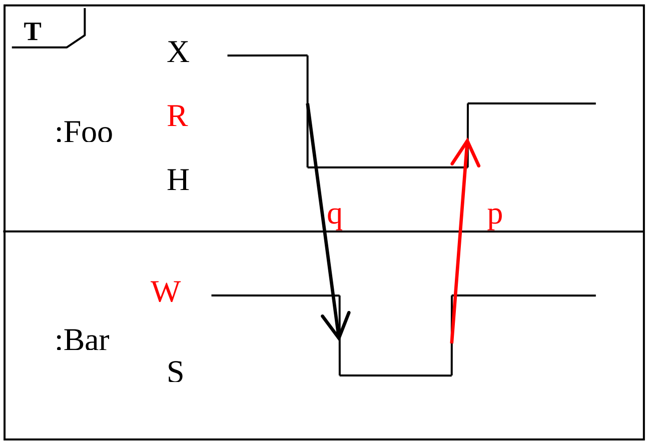


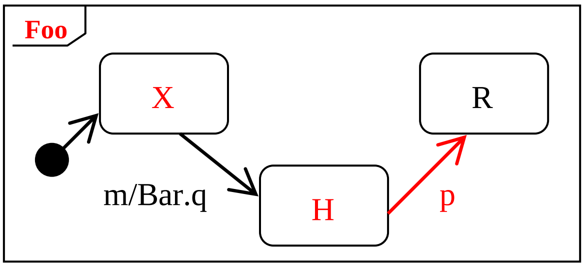




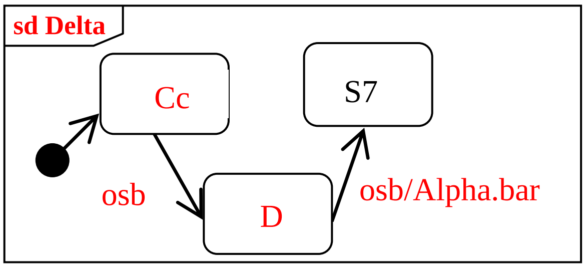
Given three UML2 diagrams. The diagrams are semantically coherent, but some elements (arrow, text, etc.) are missing. Add proper elements to the diagrams to have both syntactically and semantically coherent diagrams!

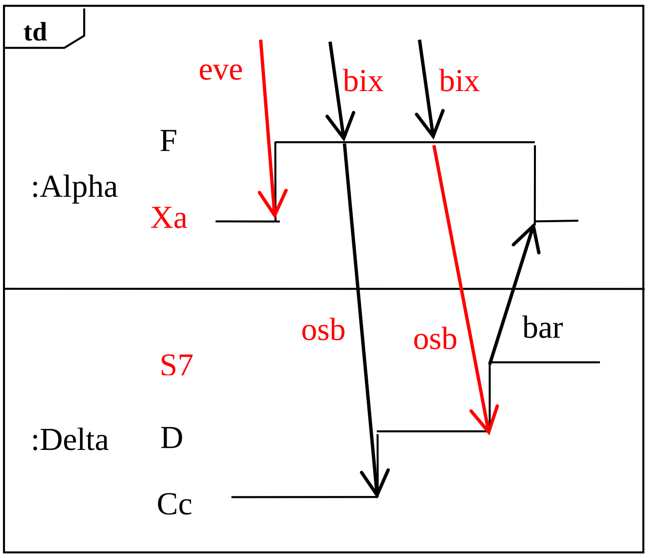


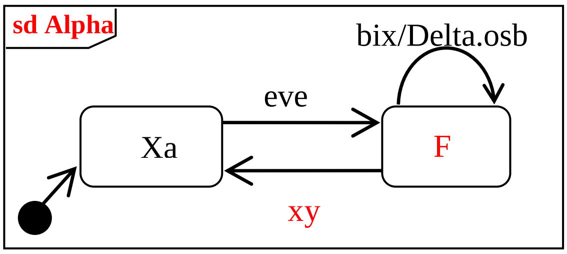




Given three UML2 diagrams. The diagrams are semantically coherent, but some elements (arrow, text, etc.) are missing. Add proper elements to the diagrams to have both syntactically and semantically coherent diagrams!



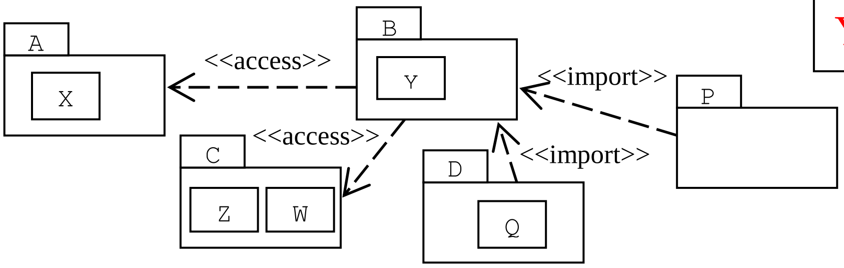




9 Package diagram

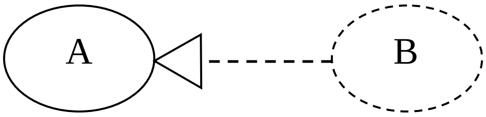
List the element(s) of the package **P** !

Y ....



10 Collaboration

What is **A** and **B** in the UML diagram given below ?



A		B	
□	operation	□	embedded state
□	use-case	□	method
□	state	□	collaboration
□	process	□	conditional use-case

11 Rational Unified Process

List the design decisions in the Rational Unified Process design workflow!

- Organize the system into packages (subsystems)
- Identifying concurrency
- Allocating packages to processors
- Storage and Persistence
- Handling global resources
- Choosing software control
- Handling boundary conditions

List the RUP process workflows and add the proper views!

RUP Process Workflows	Views
Requirement	Use-case
Analysis	Logical
Design	Component
Implementation	Process
Deployment	Deployment

Which is the last phase of the Rational Unified Process (RUP) life cycle model? Which activities can be attached to it?

last phase: .transition ....

activities: manufacturing, delivering, training ....

The RUP (Rational Unified Process) suggests or prescribes to create contracts. In which workflow ? Who are the contracting parties ? List the main points of a contract !

workflow: analysis..

contracting parties: operations vs. users of the operations..

main points of a contract: Responsibilities, Pre-conditions, Post-condition, Types, Crossrefs, ...

Exceptions, Output ....

What kind of use-cases are created in the requirement phase of the Rational Unified Process (RUP) ? What is the most important feature of such a use-case ?

high level, (some expanded), essential

technology- and implementation-independent

Characterize the RUP (Rational Unified Process) by its three main features !

iterative and incremental........ use-case driven..........................

architecture-centric..................

What is the goal of the “transition” phase of the RUP ?

supplying the product to the user community (manufacturing, delivering, and training)

12 Other problems

12.1 Collections

Define the properties of the ***Bag*** collection in UML2!

Yes	No	Not applicable	Property
□	□	☑	delegated
□	□	☑	qualified
□	☑	□	ordered
□	☑	□	unique

The UML2 collections have two important features: ordering and uniqueness. What are the names of the UML2 collections?

ordered	unique	name of the UML2 collection
Yes	Yes	Ordered Set
Yes	No	Sequence
No	No	Bag
No	Yes	Set

Define the properties of the ***Sequence*** collection in UML2!

Yes	No	Not applicable	Property
□	☑	□	unique
□	□	☑	qualified
☑	□	□	ordered
□	□	☑	delegated

What are the two most important properties of the UML2 collections ?

1. ordered...................................... 2. unique ....

12.2 Concurrency

Class ***X*** has ***aaa()*** and ***bbb()*** methods. A client of ***X*** calls the ***aaa()*** method. During the execution of the ***aaa()*** method, an other client calls the ***bbb()*** method. How to manage the concurrency in different UML2 concurrency semantics? Fill the table!

concurrency semantics	policy (how to manage)
Sequential	not allowed
Guarded	waits for finishing aaa(), then starts bbb()
Concurrent	interrupts aaa(), starts bbb()

What is the difference, if the other client calls the ***aaa()*** method also?

nothing ....

Explain the UML2 term „sequential concurrency”

callers must coordinate outside the object so that only one flow is in the object at a time.

List other kinds of the concurrency semantics defined by the UML2!

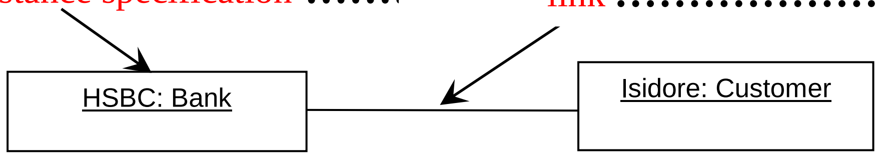
guarded, concurrent

12.3 Metaclass

Define the pointed UML2 meta-model elements !

instance specification ......

link ....



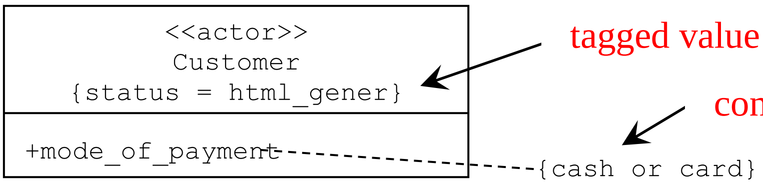
Assign the proper layer of the 4-layer metamodel of UML2 to the terms in the table given below!

	M0	M1	M2	M3
Typed Element				X
Actor			X	
State			X	
Car		X		
Bob	X			
UseCase			X	
Named Element				X
Person		X		

Define the pointed UML2 meta-model elements !

tagged value.............................

constraint.................................



Assign the proper layer of the 4-layer metamodel of UML2 to the terms in the table given below!

	M0	M1	M2	M3
Actor			X	
State			X	
Jerry	X			
UseCase			X	
Car		X		
Tom	X			
Customer		X		
Policeman		X		

12.4 General

Mark the erroneous side if the assertion is violated!

	server	client	none
invariant	[✓ □]	□	□
precondition	□	[✓ □]	□
postcondition	[✓ □]	□	□

Mark the wrong party(ies) in case of violation of the contract terms.

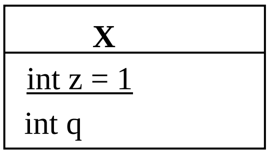
contract term	client	server
invariant		X
postcondition		X

List the elements of the general extension mechanizm of the UML!

constraint, stereotype, tagged value ....

Let `x1` and `x2` two instances of the class `X` given below, and then we execute the next three operations:

```
x1.z = 3;
x2.q = x2.z + 5;
x1.q = X.z – x2.q;
```

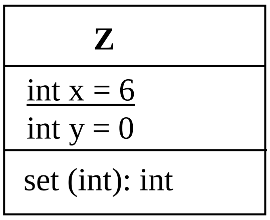


What is the actual value of

`x1.q` = -5............ `x2.q` = 8...........

Let `z1` and `z2` two instances of the class Z given below, and then we execute the next operations:

```
z2.x = 0; z1.x = 3;
z1.y = z2.x + 2;
z2.y = Z.x + z1.y;
```

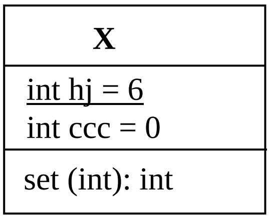


What is the actual value of

`z1.y` = 5 .... `z2.y` = 8...........

Let x1 and x2 two instances of the class X given below, and then we execute the next operations:

```
    x2.hj = 0; x1.hj = 2;
    x1.ccc = x2.hj + 5;
    x2.ccc = x2.hj + x1.ccc;
```

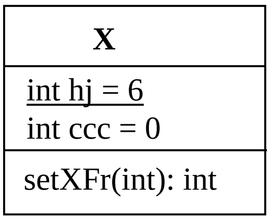


What is the actual value of

x1.ccc = 7 .... x2.ccc = 9...........

Let x1 and x2 two instances of the class X given below, and then we execute the next operations:

```
    x2.hj = 0; x1.hj = -3;
    x1.ccc = x2.hj + 5;
    x2.ccc = x2.hj + x1.ccc;
```

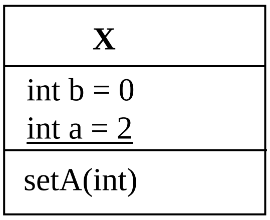


What is the actual value of

x1.ccc = 2 .... x2.ccc = -1

Let x1 and x2 two instances of the class X given below, and then we execute the next operations:

```
    x2.a = 3; x1.a = -3;
    x1.b = x2.a + 5;
    x2.b = x2.a + x1.b;
```

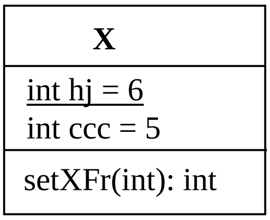


What is the actual value of

x1.b = 2 .... x2.b = -1...........

Let x1 and x2 two instances of the class X given below, and then we execute the next operations:

```
    x2.hj = 3; x1.hj = -3;
    x1.ccc = x2.hj + 4;
    x2.ccc = x2.hj + x1.ccc;
```

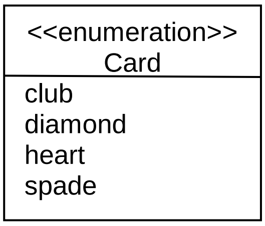


What is the actual value of

x1.ccc = 1 .... x2.ccc = -2...........

Define the following enumeration in standard UML2 !

```
                Card = [club | diamond | heart | spade]
```



Mark the truth of the statements (related to the standard UML) given below!

T [✓ F] The UML standard defines the RUP (Rational Unified Process) too.

T [✓ F] Missing visibility marker means public visibility.

T [✓ F] Use-case description and dataflow diagram are equivalent.

T [✓ F] Specialization is the weakest among the association, composition, dependency and specialization.

[✓ T] F A class can have more independent superclasses.

[✓ T] F The “link” of the object diagram has no multiplicity.

T [✓ F] The swimlanes of the activity diagram are used to define concurrency.

[✓ T] F On the communication diagram the message sequence can be defined by sequence numbering.