

# Introduktion till UML, OOAD & OOP, del 3

Data- och informationsvetenskap: Objektorienterad programmering och modellering för IA

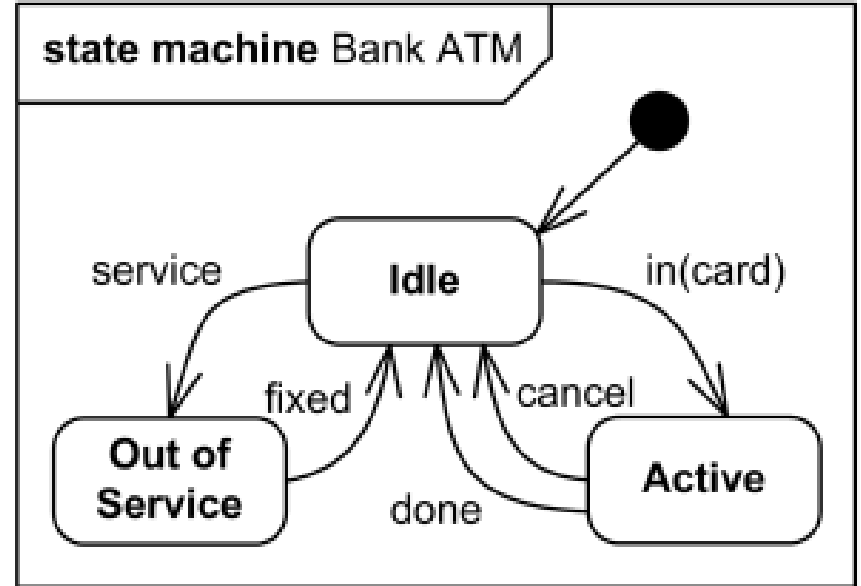
# Dagens agenda

- Förra föreläsningen
- Aktivitetsdiagram
- Use case-diagram

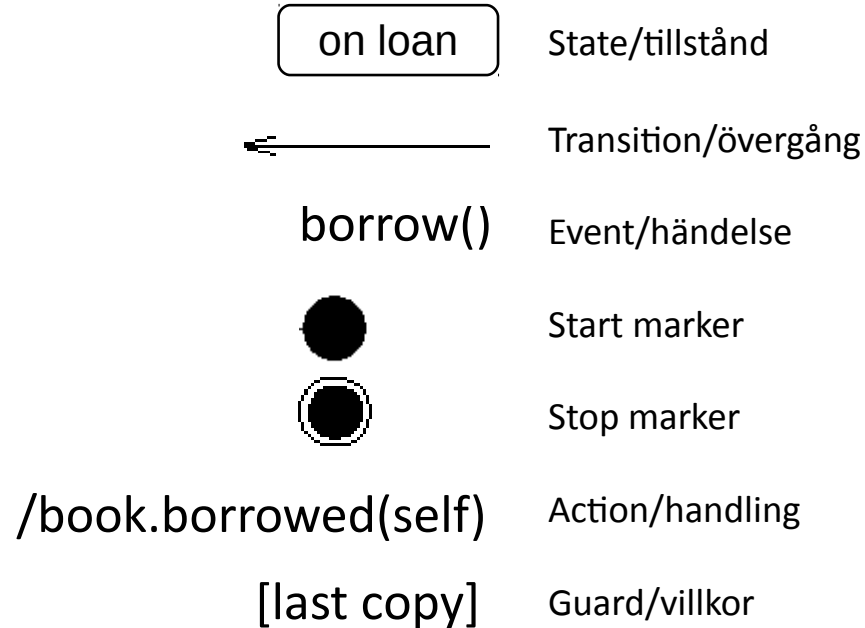
# Tillståndsdigram

Tillståndsdigram beskriver beteendet hos en avgränsad del av ett system.

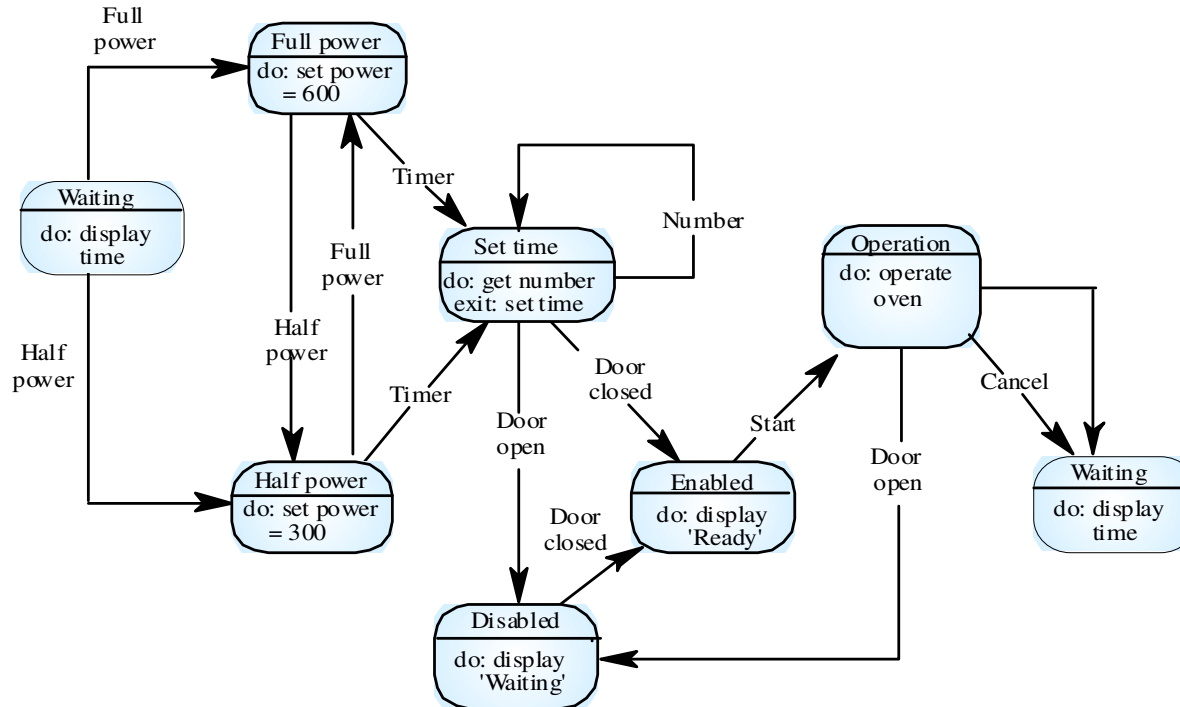
Diagrammet visar hur tillståndet hos (del)systemets ingående objekt kan förändras.



# Tillståndsdigram

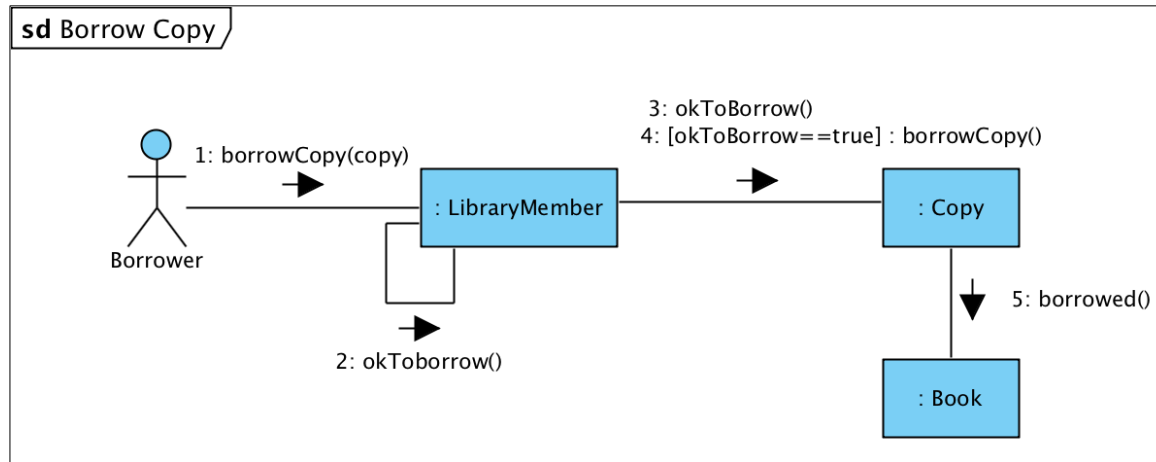


# Exempel: mikrovågsugnen

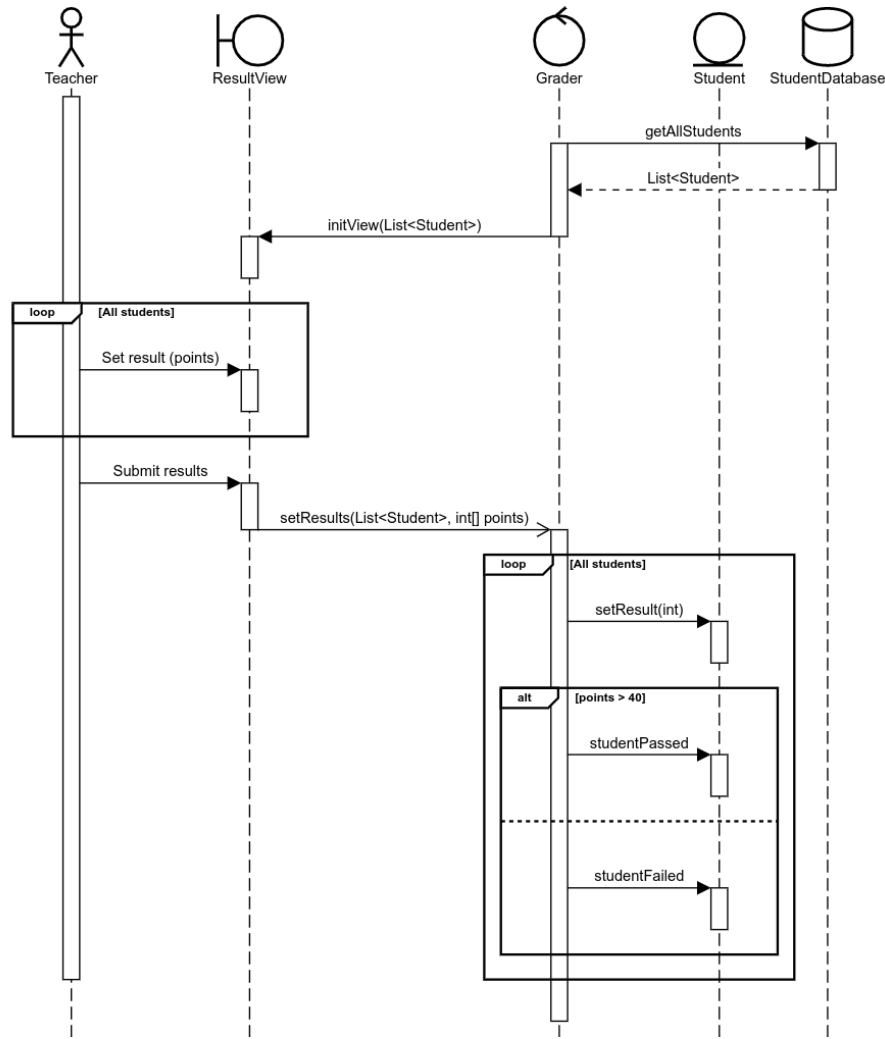


# Först: ett kommunikationsdiagram

En äldre version av ett interaktionsdiagram. Det visar alla ingående objekt och interaktioner. Interaktionerna är numrerade efter den ordning de sker i.



## Register exam results



## Sekvensdiagram

En typ av interaktionsdiagram som visar hur objekt i ett system interagerar med varandra.

Varje diagram visar hur ett enda användningsfall ser ut i systemet

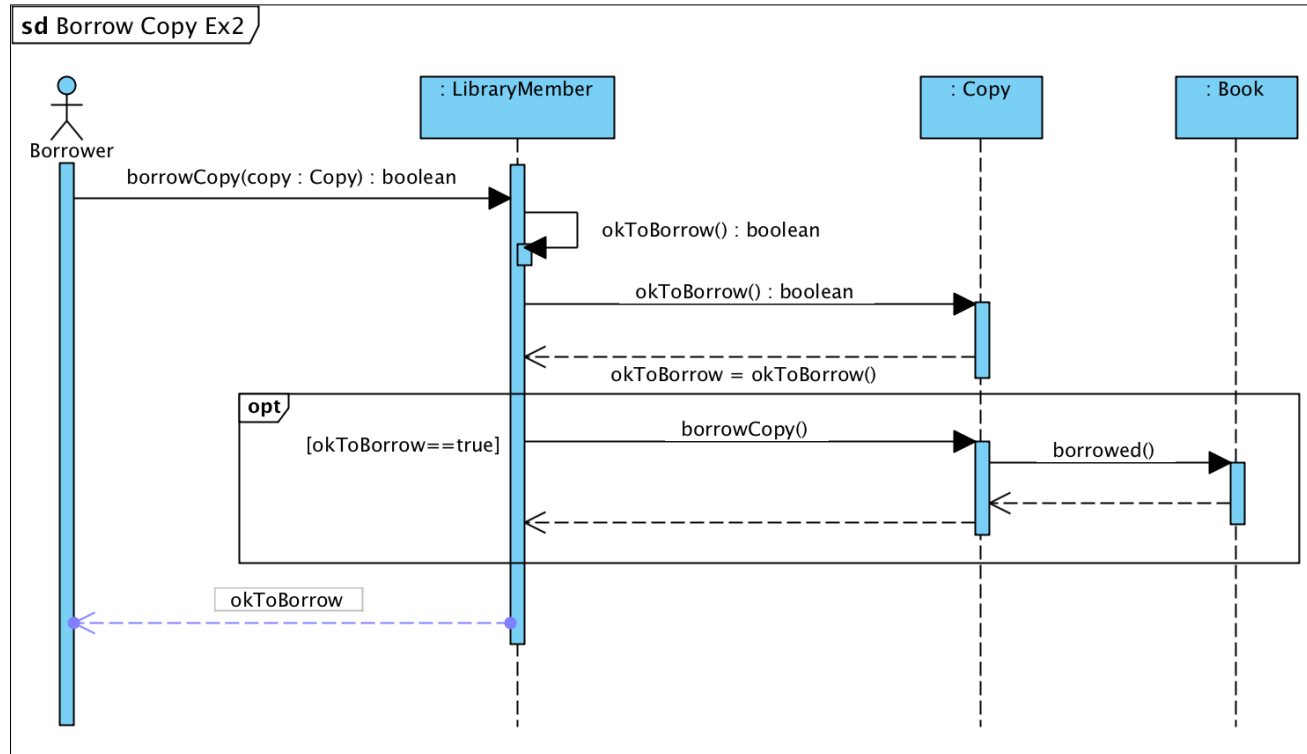
Diagrammet visar vilka objekt som ingår i interaktionen, vilka meddelanden som utbyts och i vilken ordning dessa meddelanden utbyts.

# Sekvensdiagrammet

- Sekvensdiagram visar **objekt**, deras **livstid** och deras **aktivitet** under en viss **uppgift**.
- Meddelanden numreras inte utan ordningen mellan dessa visas genom var på livstidsaxeln de sker. Tiden startar högst upp och rör sig nedåt i diagrammet.



# Användningsfall: låna en bok

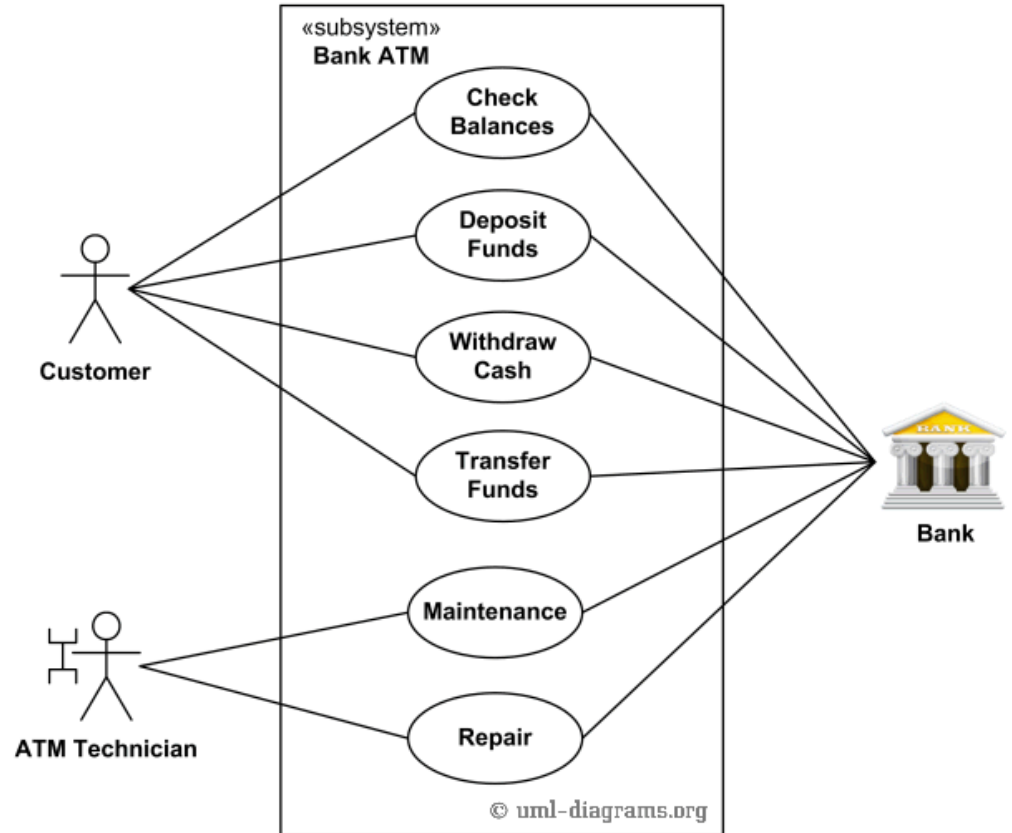


# Use case-diagram

# Use Case-diagram

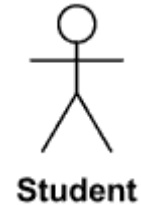
Use Case-diagrammet beskriver funktionaliteten hos ett system från *användarens* perspektiv.

De användningsfall som modelleras i ett Use Case-diagram förtydligas med fördel i ett aktivitets-, kommunikations- eller sekvensdiagram.



# Aktörer

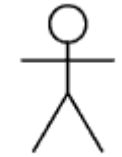
**Aktören** interagerar med ett **subjekt** i en **roll** som specificeras i text. En aktör modellerar inte nödvändigtvis en unik **entitet**, utan kan representera flera olika. På samma sätt kan en unik entitet modelleras av flera olika aktörer, baserat på olika roller.



# Aktörer, andra representationer

En aktör behöver inte symboliseras med en streckgubbe, utan kan symboliseras med valfri ikon.

Aktörer kan även modelleras som klasser med *actor*-stereotyp.



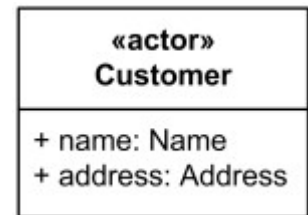
Student



Web Client

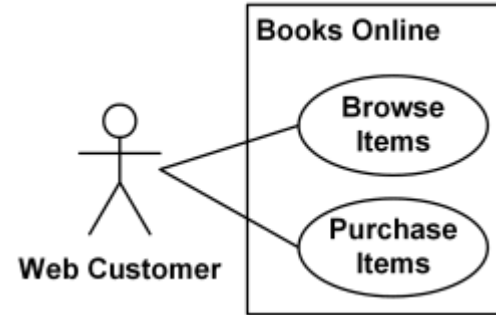


Bank



# Subjekt

**Subjektet** är det system eller delsystem som **aktören** interagerar med. Ett system kan möjliggöra ett eller flera **användningsfall**.

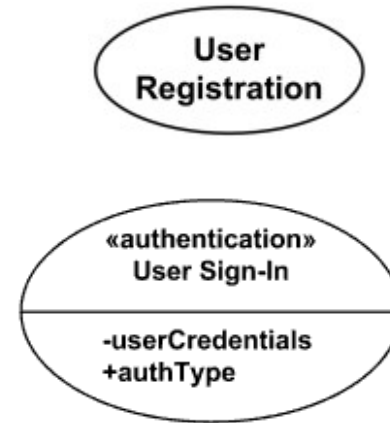


# Användningsfall

## Användningsfallet

betecknar en inkapslad bit funktionalitet. Det namnges i regel efter den **händelsekedja** som ska modelleras.

Ett användningsfall kan, men måste inte, ha **attribut** och **operationer** kopplade till sig.



# Användningsfall, utbyggnadspunkter

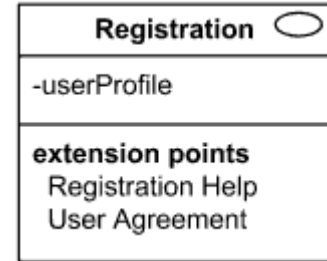
Ett användningsfall kan definiera **utbyggnadspunkter**, vilka används för **utökade användningsfall** (se senare slide). Dessa punkter berättar vilka utökningar som är tillåtna för det aktuella användningsfallet.





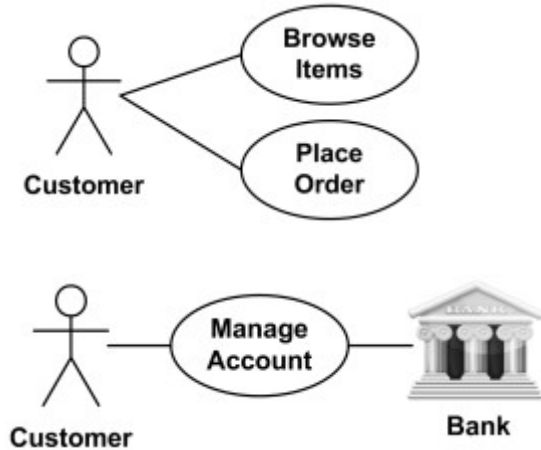
# Användningsfall, klassnotation

Användningsfall kan även modelleras som klasser i klassdiagram. Notera symbolen i övre högra hörnet.



# Associering

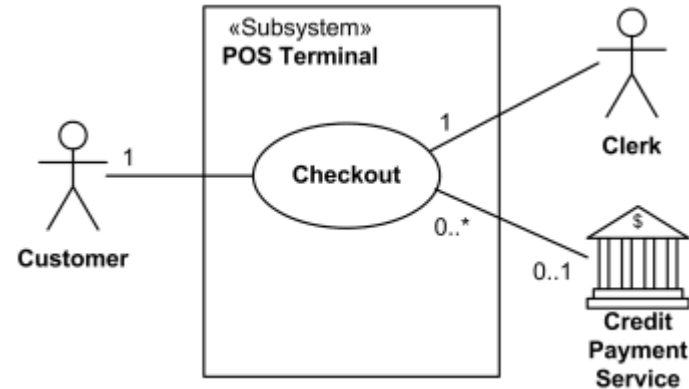
Aktörer kan vara **associerade** med flera användningsfall och vice versa. I Use Case-diagram placeras vanligtvis **externa aktörer** till vänster om användningsfallen och **interna aktörer** till höger om dessa.



# Multiplicitet

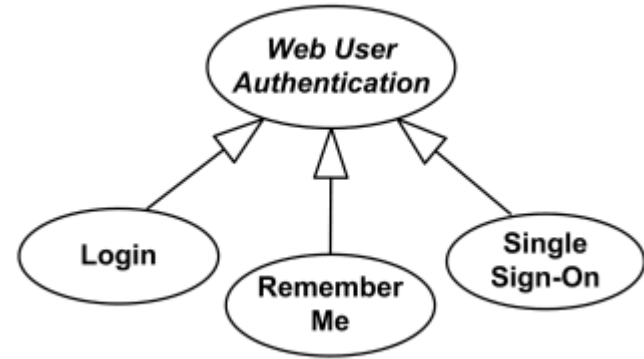
Aktörer kan även ha relationer till flera användningsfall av samma typ och vice versa.

Multipliciteter anges på samma sätt som i klassdiagram.



# Generalisering

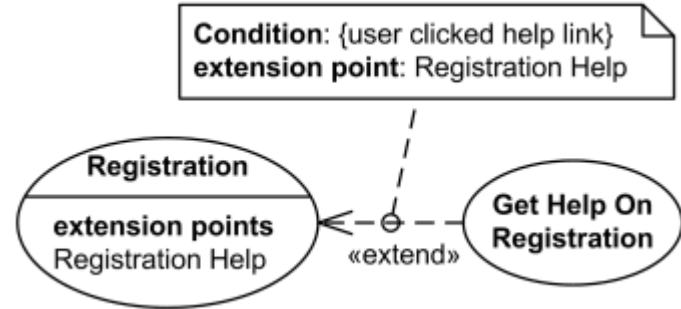
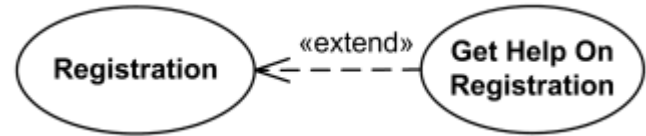
Ett användningsfall kan agera bas åt ett antal andra användningsfall. Detta är användbart när ett flertal externa användningsfall bidrar med liknande funktionalitet, men endast basfunktionaliteten egentligen är intressant.



# Utökade användningsfall

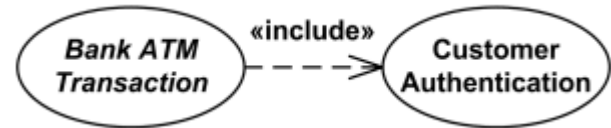
Användningsfall kan utökas med ytterligare funktionalitet när detta önskas.

Det användningsfall som utökas är helt fristående, medan det utökande användningsfallet är beroende av sin “partner”.



# Inkludering

Ett användningsfall kan **inkludera** ett annat användningsfall som definieras på någon annan plats. Detta gör att vi kan återanvända användningsfall.



# Ett gemensamt exempel

Vi utgår från förra laborationens Niagara-beskrivning.

- Vilka användningsfall kan vi hitta i den?
- Vilka hittar vi med följande tillägg?

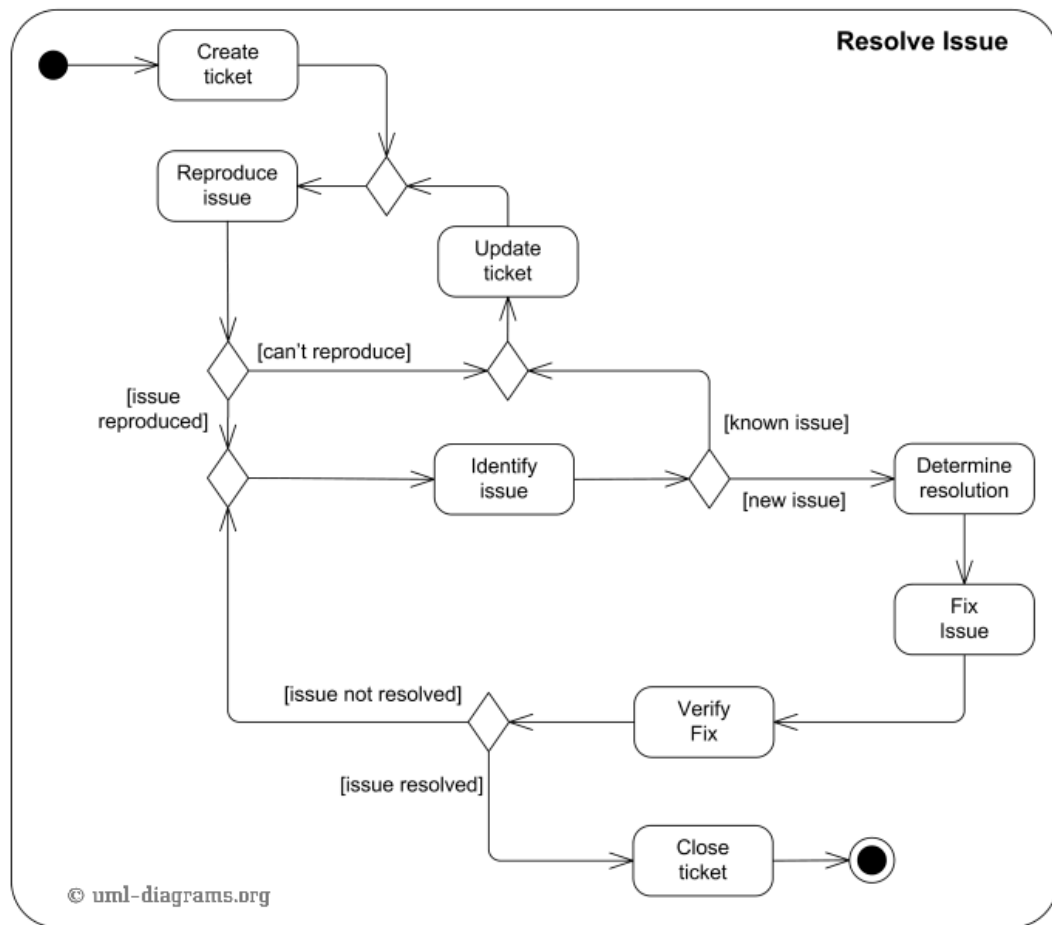
*Schemahandläggaren kan boka lokaler i Niagara. Detta görs tillsammans med lärarna, vilka undervisar i nämnda lokaler.*

*Lokalvårdarna ansvarar för att lokalerna är rena, medan vaktmästarna ansvarar för att lokalerna är hela och funktionella.*

*På taket kan studenterna exempelvis titta på Köpenhamn, äta lunch eller kasta apelsiner på förbipasserande fotgängare.*

# Aktivitetsdiagram





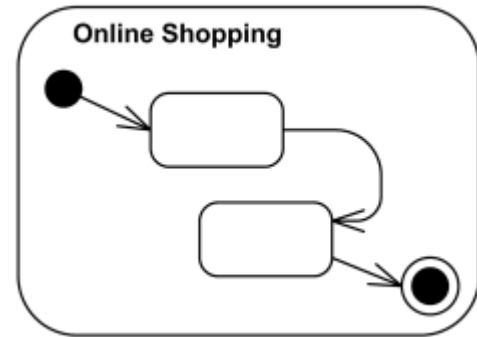
## Aktivitetsdiagram

En typ av beteendediagram som visar hur kontrollflödet i ett system eller delsystem ser ut.

Varje diagram visar hur ett enda användningsfall ser ut i systemet, men kan hänvisa till andra användningsfall.

# Aktiviteter

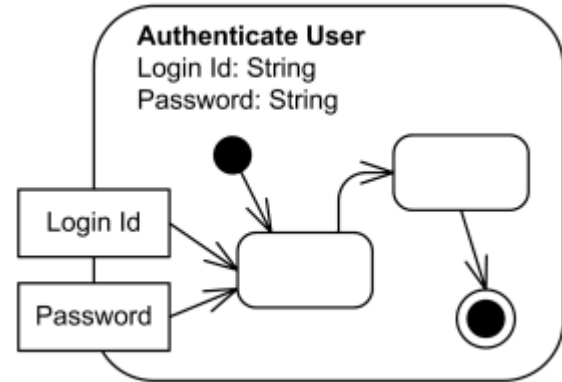
En **aktivitet (activity)** är ett **händelseflöde**. Den identifieras med ett namn, vilket ofta motsvarar ett **användningsfall** i ett Use Case-diagram eller en sekvens beskrivet i ett sekvensdiagram.



# Parametrar

Vi kan ange **parametrar** till en aktivitet. Dessa modelleras som rektanglar på gränsen till aktiviteten.

Parametrar är indata som behövs för att kunna utföra en given aktivitet.



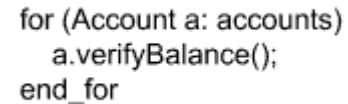
# Handlingar

Den grundläggande komponenten i ett aktivitetsdiagram är **handlingen** (eller **action**). En handling utgör ett atomärt steg i en aktivitet, vilket innebär att den inte kan brytas ner i mindre beståndsdelar.

Handlingar namnges på samma sätt som användningsfall, men undvik att återanvända namnen inom ett givet system.



Process  
Order



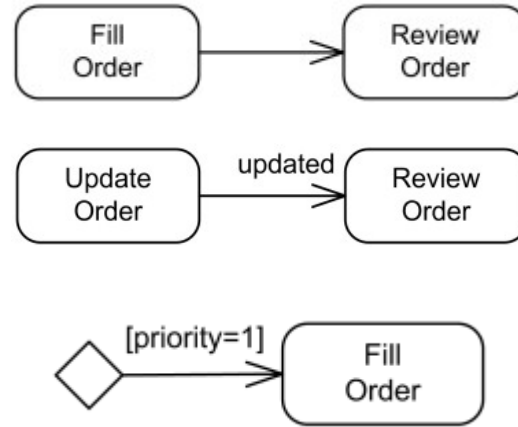
```
for (Account a: accounts)
  a.verifyBalance();
end_for
```

Handling beskriven i ett för applikationen givet handlingspråk.

# Bågar

Vi kopplar samman handlingar med pilar kallade **bågar (edges)**.

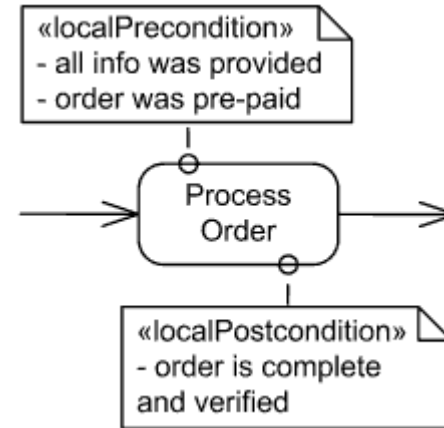
Bågarna är riktade och kan vara uppmärkta med ett **namn** eller en **vakt** ovanför pilhuvudet.



# Lokala förutsättningar

Vi kan ge en händelse **lokala förutsättningar** som måste vara uppfyllda för att händelsen ska genomföras. På samma sätt kan vi även specificera de förutsättningar som krävs för att en händelse ska räknas som slutförd.

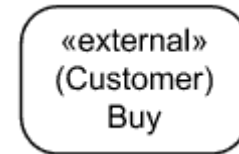
Dessa modelleras som kommentarer med stereotyperna *localPrecondition* och *localPostcondition*.



# Yttre handlingar

**Yttre handlingar** är handlingar som strikt taget inte tillhör det aktuella diagrammet. De specificeras ofta, men inte alltid, i ett annat diagram i systembeskrivningen.

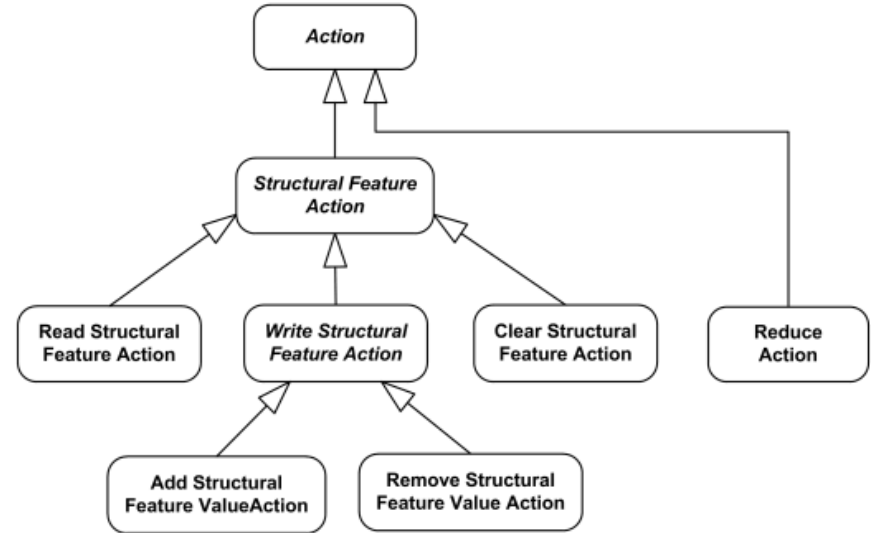
Yttre handlingar markeras med stereotypen *external*.



# Generaliseringar

Vi kan **generalisera handlingar** genom att modellera dessa på samma sätt som arv i klassdiagram.

**Abstrakta handlingar** kan modelleras genom att kursivera namnet.

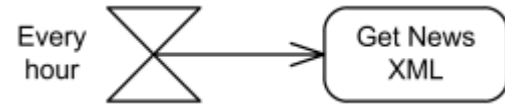




# Periodiska handlingar

**Periodiska handlingar** är handlingar som sker med **jämna mellanrum** och därför inte nödvändigtvis initialiseras på initiativ av en användare.

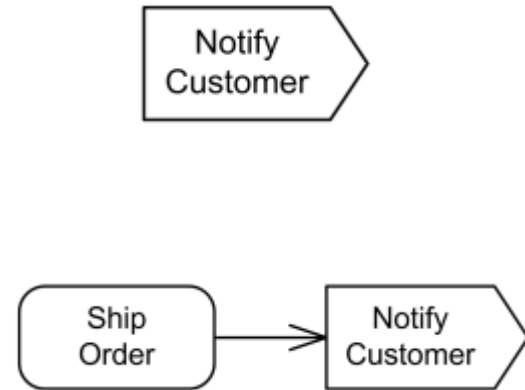
Dessa handlingar modelleras med ett stiliserat timglas.



# Signaleringar, skicka signal

**Signaleringar** används för att koppla samman olika arbetsflöden. Ett vanligt synsätt på detta är att vårt flöde har **sidoeffekter** som modelleras på andra håll.

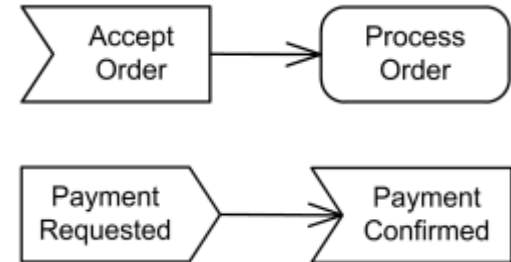
Vi visar att en händelse **skickar en signal** med en spetsig box.



# Signaleringar, ta emot signal

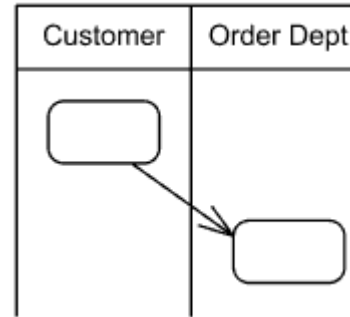
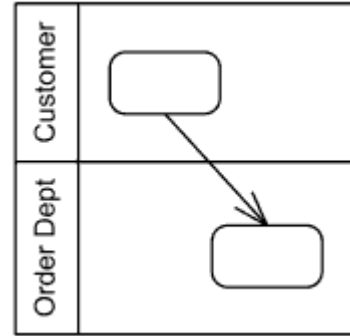
På liknande sätt som vi skickar signaler kan vi även **ta emot signaler** i våra modeller. Vi visar detta med en “avbiten” rektangel.

Vi kan även modellera hopp in och ut ur ett diagram genom att använda en skickad signal och en mottagen signal bredvid varandra.



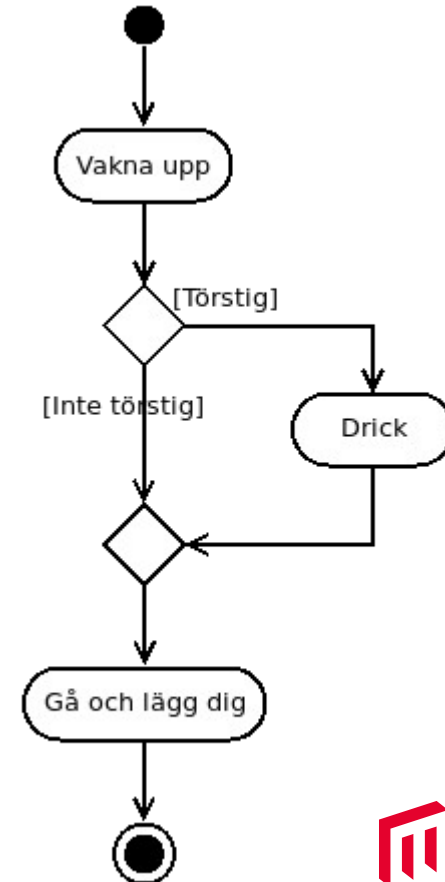
# Partitioner

Vi kan förtydliga var händelser hör hemma med hjälp av **partitioner**. Dessa modelleras som **simbanor (swim lanes)**, vilka kan vara både horisontella och vertikala.



# Kontrollnoder

För att kunna modellera mer komplexa händelsekedjor än rent linjära sådana använder vi **kontrollnoder**. Dessa används för att modellera exempelvis if-satser, loopar och så vidare.

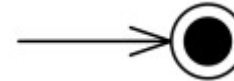
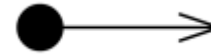


# Start/slut

**Startnoden** visar, precis som i ett tillståndsdigram, var ett flöde **startar**.

**Flödesavslutningsnoden (flow final node)** visar att ett delflöde avslutas. Den kan användas för att modellera delflöden i ett större flöde som kräver exempelvis en signalering eller periodisk händelse för att kunna slutföras.

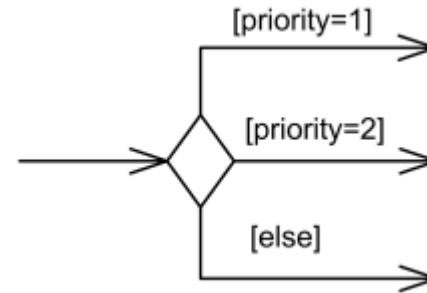
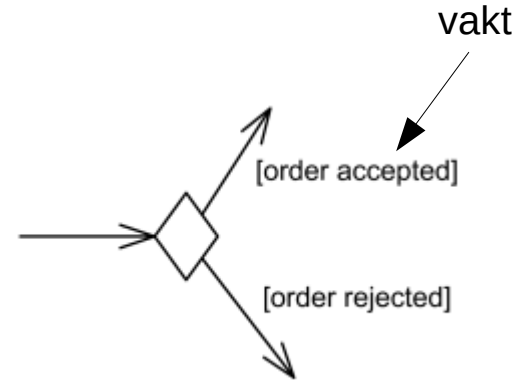
**Avslutningsnoden (activity final node)** visar, precis som i ett tillståndsdigram, var ett flöde avslutas.



# Beslutsnoder

**Beslutsnoder (decision nodes)** används för att modellera **if-satser** och liknande kontrollstrukturer, exempelvis **select/switch-satser**.

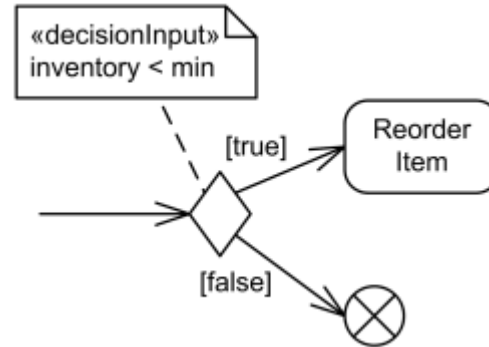
Varje förgrening beskrivs med en **vakt (guard)**. Beslutsnoder kan ha två eller flera förgreningar.



# Beslutsnoder, input

Vi kan modellera **input** (tänk **villkor**) i en beslutsnod med en kommentar kopplad till beslutsnoden med en streckad linje.

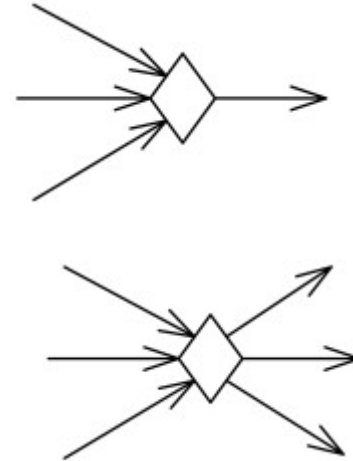
Inputen märks upp med prototypen *decisionInput*.





# Sammanslagningsnoder

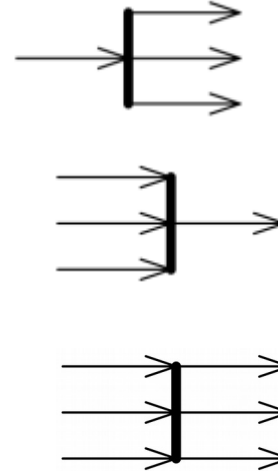
**Sammanslagningsnoder**  
(**merge nodes**) används för att modellera att två eller flera vägar från en beslutsnod når samma plats i flödet. Dessa noder kan även utgöra beslutsnoder för nästa steg i flödet.



# Avgreningar och anslutningar

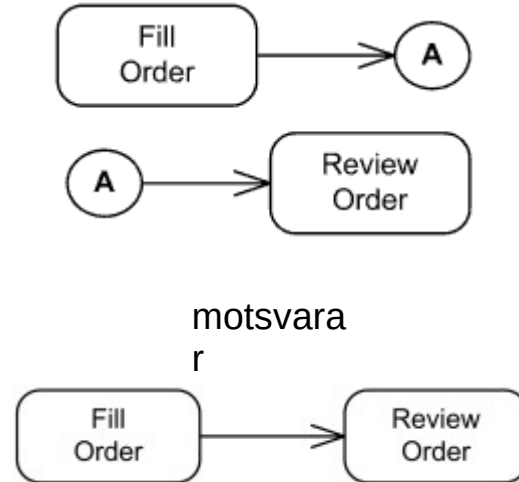
**Avgreningar (forks)** visar att två eller flera **delflöden sker parallellt**. Dessa delflöden är oberoende av varandra och samlas samman i en **anslutningspunkt (join point)**.

Precis som med sammanslagningspunkter kan en punkt även utgöra en ny avgrening.



# Sammankopplare

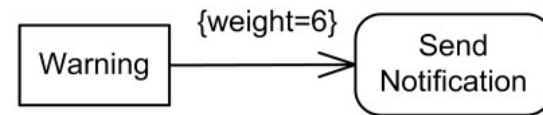
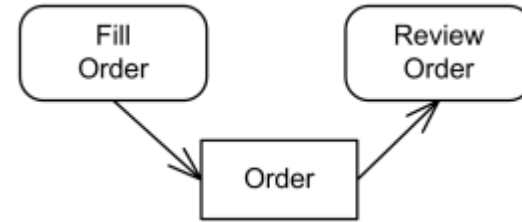
**Sammankopplaren** (**connector**) kopplar samman två delar av flödet. Den har ingen annan betydelse än att du slipper dra streck över hela diagrammet.



# Objektflöden

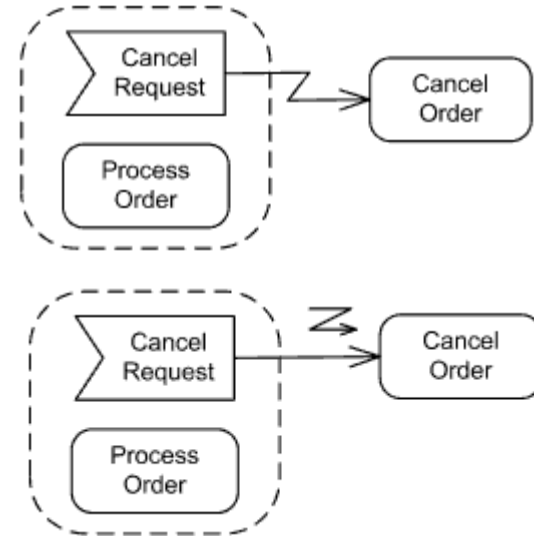
Ett **objektflöde (object flow)** visar att den data som skickas från en handling till en annan utgörs av ett **objekt**. Dessa modelleras med namngivna rektanglar som i ett objektendiagram.

En båge i ett objektflöde kan ha en vikt, vilket innebär att övergången till nästa handling sker när villkoret (dvs antalet mottagna objekt) är uppfyllt.



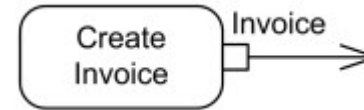
# Interrupts

**Interrupts** är händelser som kan inträffa när som helst i flödet. De avbryter flödet och tar prioritet över skeendet.



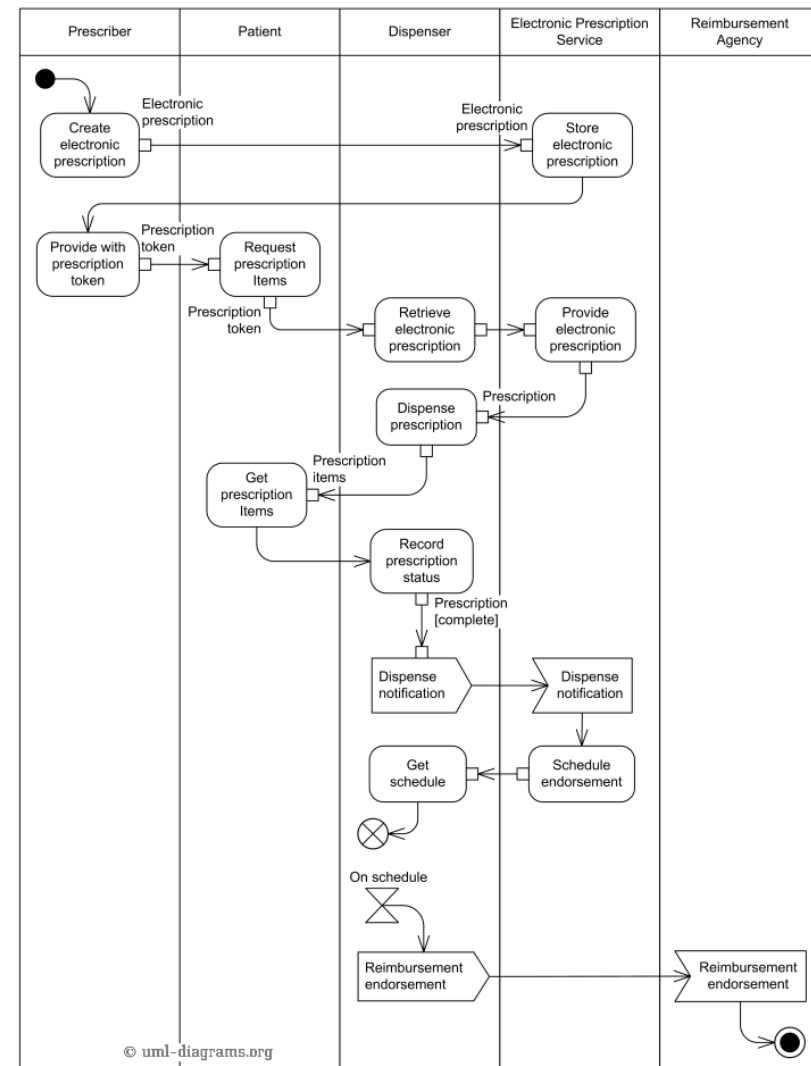
# Pins

En **pin** visar vilken typ av objekt som **förväntas** eller **skapas** av en handling. De kan användas för att ersätta objektflöden och göra ett diagram lite mindre plottrigt.



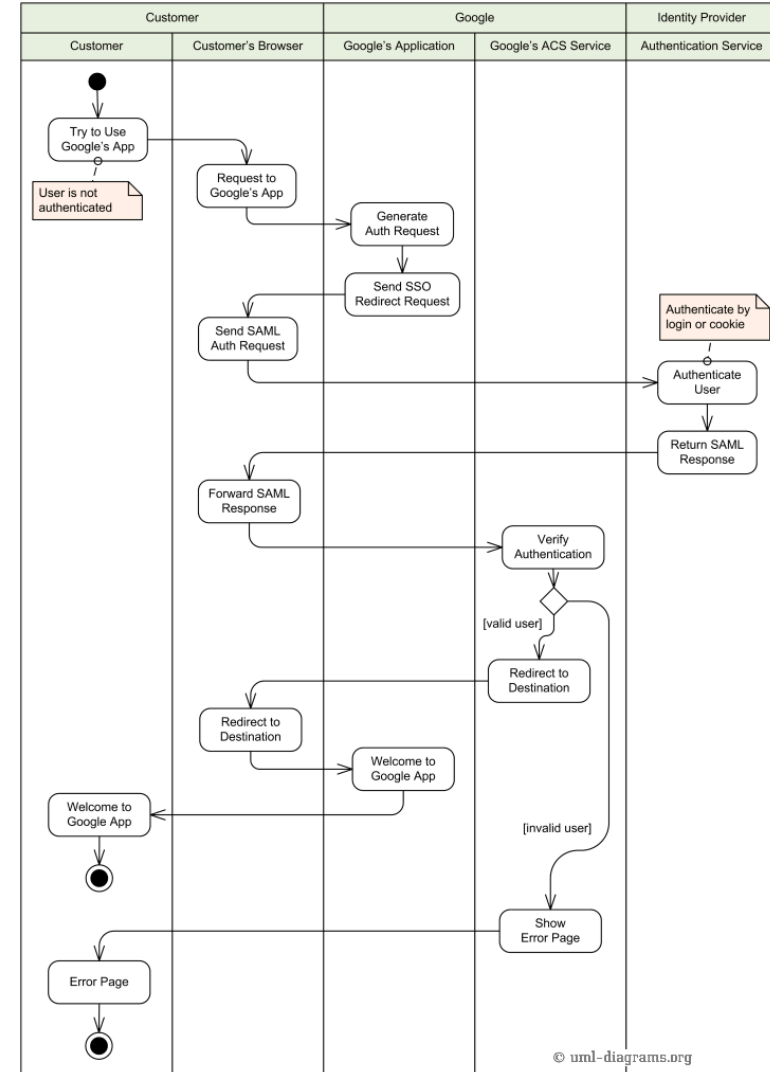
# Exempel

Diagrammet till höger, som är lånat från UML Diagrams.org, visar flödet för hur ett e-recept hanteras inom den brittiska sjukvården.



# Exempel

Diagrammet till höger, som är lånat från UML Diagrams.org, visar flödet för Single Sign On hos Googles olika applikationer. Det här kan vara trevligt att känna till inför vår nästa kurs.





# Slutligen

# Läsanvisningar

## Object-Oriented Systems Analysis and Design Using UML

- 5.2 Modeller och diagram
- 5.3 Aktivitetsdiagram
- 6.6 Use Case-diagram
- Appendix A Notationssammanställningar

## UML Diagrams.org

- <https://www.uml-diagrams.org/use-case-diagrams.html>
- <https://www.uml-diagrams.org/activity-diagrams.html>