**UML-diagram:**

**Vilka det finns:**

1. Use case diagram
2. Klassdiagram
3. Sekvensdiagram
4. Flow diagram

**Vad är UML?**

Är industristandard för **NOTATION** och **BESKRIVNING** av kravmodellering, analys, design och implementation av IT-system

* **UML 2.0:** Möjliggör ”automatgenererad” kod från diagram
* **Olika diagramtyper:** Totalt 13 st (6 st som är nya)

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**USE CASE DIAGRAM:**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**DEFINITION:**Är sambanden/beroenden mellan olika användningsfall (use cases). **Beskriver samband mellan olika use case**

**GENERELLT:**

* Projektledare gillar Use Case diagram
* Tidigare beskrev vi ett use case i taget, textuellt, men det räcker inte alltid
* Use case blir mer effektiva om man:
  + BESKRIVER SAMBANDEN MELLAN OLIKA USE CASE

**EXEMPEL:**Inloggning (vilka andra use cases använder inloggning) – Olika aktörer

**FÄLLA:**

Beskriver för detaljerat även i diagrammet vilka systemdelar det finns. Use casediagram SKA INTE HA EN SERVER I SIG.

* De ska beskriva VAD ett system ska göra, INTE HUR
* Allt med realisering ska man ta bort från sina användningsfall, inte databas (implementeringsdetalj)

**BRA VID:**Arkitektuella beslut, hur tänker vi konstruera detta? 🡪 Blir lättare att visualisera, får bra översikt

**KOMPONENTER:**

* **AKTÖRER:** 4 st
* **ANVÄNDNINGSFALL**: Kan ärva från något annat, beställa vin/beställa mat = samma process men olika menyer.
  + **Conditions:** När ska detta användningsfall användas?
    - Kan inte dricka vinet om jag inte fått det än
  + **Etiketter:** För att visa vilken typ av action det är. Blir lite som ett AKRIVITETSDIAGRAM (ser lite vad som händer i systemet). Man kan sitta med en kund och beskriva hela processen

**DOMÄNDRIVEN UTVECKLING (pratar med kund):**

* **Context:** Skapa sin context, sin gemensamma ordlista (Use case diagram & use case driver mycket av den ordlistan)
  + **FRÅGA:** Är Servitör är samma som den som står i köket? Nej inte lämpligt

**KAN HA FLERA USE CASE SOM INTERAGERAR:**

**Inte ovanligt att ha flera olika use case diagram för att se hur de går ihop:**

1. **Klientens användning**
2. **Administrativa:** Specialfall för detta?
3. **Någon gemensam vy:** Kan göra paket av användningsfall som interagerar med varandra, men npgon typ av context emellan.

1 sånt här per context, och sen ett integrsationsdiagram, 1 för ren översikt

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**KLASSDIAGRAM**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Definition:**Visar hur klasser relaterar till varandra. Beskriver vad man har för modeller, hur ser vår klassmodell ut?

**VILL KOMMUNICERA KLASSENS INTENTION!!! STRUNTA I DETALJERNA!!!!!!!!**

**ATT HA I KLASSDIAGRAM:**

1. Namn på klass
2. Metoder
3. Attribut
4. Ove gör bara det som är PUBLIKT, inte privat (endast om det är viktigt för implementeringen)
   1. **EX:**
      1. 3, 4, 5 st publika
      2. Jättemånga privata

**TÄNK PÅ:**

* Skriv så LITE SOM MÖJLIGT, bara det som måste kommuniceras ut, det UNIKA för vad faktiskt klassen gör
* EX: Skriv inte ut ToString() för det är sånt som finns i alla projekt

**ARV:**

* Klassdiagram visar även arv: Hur en klass relaterar till en annan, hur de interagerar

**VILKA KLASSER HAR VI:**

* Ofta första steget för design: Baserat på användningsfallen tänker man ut vilka klassen vi ser att vi behöver ha

**EXEMPEL BOWLING:**

1. Spelare
2. Bowlingbanor (inte klot: Systemet hanterar inte kloten)

Genererar ofta kod från klassdiagram och är närmast kopplad mot den kod vi skriver.

**TIPS:** Se till att inte gå in och manuellt ändrar i koden, det ska vara model first

**HUR MAN SKRIVER KLASSDIAGRAM:**

1. Titel
2. Fälten högst upp
3. + / - publikt/privat
4. Metodnamn + Returtyp (om det gör det)

**DEKORATÖRER/LABLAR (eventuellt):**

* I objektorientering skiljer vi inte på en klass och ett interface, modellerar de på exakt samma sätt (bara en annat typ av objekt), men bra att VISA att det är ex ett interface och då kan man använda LABLAR

**PACKETERA KLASSERNA:**

* Packar in klasserna i ett paket:
  + **Publikt:** I ett api
  + **Annat paket:** Visar implementationen

**EXEMPEL BIL:**

* + - Interface som någon kan använda: Köra(), Stanna(), Tanka()
    - Implementationen är oviktig för den som anropar det (vill bara ha något som är av typen…)
    - **I systemet kanske det hanterar annat men för den som använder är det bara viktigt att veta att det går att**

**EXEMPEL PERSON:**

* + - KLOTTRIGT: GetName(), GetBirthday()
    - GÖR: GetAge()
      * + Man får inte reda på när den är född, men man får reda på åldern
        + Implementationsdetalj (att man måste vara född för att kunna fylla år)

**VILL KOMMUNICERA KLASSENS INTENTION!!! STRUNTA I DETALJERNA!!!!!!!!**

**NOTERA PARAMETRAR:**

Parameter i en metod skriver jag inget namn, bara typen (det får de som kodar göra senare). Det handlar bara om METODSIGNATUREN!

**ATT TÄNKA IGENOM:**

* Vilka metoder man kan göra?
* Vilka attribut?
* Kan ha hänglås (eller +/-)

**KOMPOSITER:**

Berättar hur klasser hänger ihop.

**EXEMPEL BANK:**

* MÅSTE HA: Konton (röd ifylld diamant)
* KAN HA NOLL eller HUR MÅNGA SOM HELST: Kunder (gul diamant)
  + 1 bank kan ha många kunder
    - * **MULTIPLICITET och RELATIONER de har:** Ska kommuniceras
      * **LABEL:** CustomerList (vad är det vi pratar om)
      * **HUR KOMMER DE KOMMUNICERA:** Behöver inte vara banken, kan vara någon annan
      * **DIAMANTERNA:**
        1. **Fylld:**

Är en komposit: Den består av: **Den måste bestå av**… Är inte ett fullvärdigt objekt om den inte har detta

* + - * 1. **Tom:**

**KAN BESTÅ AV**: EX Fordon. En bil MÅSTE ha en ratt, alltså måste man ha ett objekt som vi skickar in en bil när vi ska ha ratten.

**Kunderna kan skapas senare.** Få banker som startar med kunder

* + - * 1. **Pannkakor:**

KOMPOSITER: Alla ingredienser är kompositer och skulle ha notering med komposit

INTE KOMPOSITER: Grädde, sylt: Är tom diamant

* + - * **TOMT STRÄCK:**
        1. Är en relation mellan en customer: Customer har ett Account.
        2. Olika typer av beroenden
      * **ARV:** Triangel som pekar, den som specialicerar den som pekar på den som är generell, vem är det som gör implementering av den?
        1. **EX:** Dog går med rakt sträck utåt och pekar med triangel på Animal
      * **EX:** Account abstrakt, SparKonto.cs (limit 100 000kr), CheckKonto.cs (fast ränka på noll). Båda är konton och då kan man ha arvsymbol mot Account.