Klassendiagram:

Methodiek: UML

Uitgangspunt zijn de requirements uit het PVE.

Stappen:

1. Verzamelen de zelfstandige naamwoorden. De zelfstandige naamwoorden worden gewogen door ze te noteren en daarna te wegen. Noteer ieder zelfstandignaamwoord maar 1 keer.

Wegen:

\* Kandidaat klasse: het znw heeft eigenschappen (attributen of velden). Aan een klasse kan je iets vragen. Bv. Student heeft eigenschappen(achternaam, studentnr etc.) en je kan aan hem vragen: wat is je naam? => public function getNaam() : string.  
  
\* attribuut van …: bv: achternaam is een attribuut van Student.

\*synoniem: zelfde betekenis van twee verschillende woorden: student en leerling.

Gooi er 1 weg.

\*het systeem: wat je gaat bouwen. Dit moet je opslaan.

Stel je bouwt voor een school een registratiesysteem, dan moet je school opslaan

\*te ruim of te vaag: denk aan dingen zoals info of omgeving.

Verhaaltje:

Voor een bioscoop moeten we bouwen: de verkoop van tickets. De klant koopt een ticket voor een voorstelling. Op het ticket komt een ID, datum, tijd, zaal, en plaats.

Tevens staat op ticket de film. De klant bestelt via internet, de klant betaalt dan ontvang zijn tickets op zijn e-mail

|  |  |
| --- | --- |
| ZNW | Weging |
| Bioscoop | Kandidaat klasse |
| Verkoop | Kandidaat klasse |
| Klant | Kandidaat klasse |
| Ticket | Kandidaat klasse |
| Voorstelling | Kandidaat klasse |
| Film | Kandidaat klasse |
| ID | Attribuut ticket |
| Datum | Attribuut ticket |
| Tijd | Attribuut ticket |
| Zaal | Kandidaat klasse |
| Plaats | Kandidaat klasse |
| ~~Internet~~ | Te vaag of te ruim |
| e-mail | Attribuut van klant |

1. Relaties:

We gaan uit van de twee klassen: auto en eigenaar:

Een relatie gaat ALTIJD ÉÉN richting op.

Dus

1. Iedere auto heeft 1..n(1 of meer) Eigenaren.
2. Iedere eigenaar heeft 1..n(1 of meer) auto’s
3. Dit zijn 2 relaties

Er zijn drie belangrijke relaties:

* 0..n (0 of veel relatie) en 1..n ( 1 of veel) relaties vb: in iedere klas zitten 0 of meer studenten. De fk gaat van de veel kant naar de 1 kant. De fk klas zit bij student.
* 1..1(1 op 1) of 0..1(0 of 1). Iedere pc heeft 0..1 muis. Moeilijke relaties: ieder huis heeft 0..1 hoofdbewoner. Bij verwijderen: een hoofdbewoner kan niet bestaan zonder huis. Dus huis is de parent en hoofdbewoner is het child. Let op: stel huis brand af, dan moet je ook de hoofdbewoner verwijderen.
* Veel op veel(N..n) de auto kent 1 of meer eigenaren en een eigenaar kan 1 of meer auto’s hebben. Oplossing: tussentabel. Wordt ook hulptabel genoemd. Hier is dat AutoEigenaar : klasse.
  + De klasse AutoEigenaren heeft 2 references
    - 1 naar auto
    - 1 naar eigenaar
  + Hier is de classifier of pk is auto + eigenaar
  + PK : samengestelde sleutel van auto en eigenaar.
* Zijn specialce relaties: een paar bij dansen zijn er precies twee: een danspaar bestaat uit 2..2

Of een voetbalteam 8..11

* Een bijzondere relatie: stel je wil registreren, welke personeelsleden. Dus de relatie gaat van persoon naar persoon : heet een eigen relatie.
  1. Persooneel:
     1. Bsn (pk) : int
     2. WoontSamenMet -> int en deze wijst naar Bsn van een ander persooneelslid, kan nul zijn

Oplossing van relaties:

We gaan uit van de auto’s en eigenaren.

1. Iedere auto heeft 1..n eigenaren. Stel ik heb een auto met 2 eigenaren: truus en jan. Kan in auto opslaan in een Array van eigenaren.`

` Array Eigenaren[] : Eigenaar

Probleem: de programmeur Emma gaat klagen: als jan overlijdt, dan verwijder je de eigenaar en moet je de eigenaar uit de array verwijderen. Vergeet je vaak. Array: liever niet als variabelen

1. Iedere eigenaar heeft 1..n auto’s.

Weer een array: nu in Eigenaar van Auto’s

Array autos[] : Auto.

Weer een probleem.

1. Als beide relaties een veel relatie zijn (0..n) of (1..n) dan een extra klasse: AutoEigenaren. De relaties tussen AutoEigenaren en Auto en Eigenaren gaat via references.

Opdracht:

* Maak een gebruiker GarageeUser die toegang heeft tot de database Garage. Mag geen root zijn.
  + Geeft de juiste rechten: select, insert,delete,en update.
* Maak een workspace aan Garage
* Maak een db.ini met GarageUser instellingen
* Maak verbinding met de DB
* Test deze
* Maak de klas Auto met:
  + Constructor(kenteken)
  + setType
  + Gettype
  + findAuto: ga in de database opzoek naar de auto en vul het type in.maak deze methode private en start hem vanuit de constructor zodat type gevuld wordt
  + toString
* Maak de klasse Eigenaar
  + Constructor(bsn,naam,tel)
  + SetTel
  + getTel
  + toString
* maak de klasse AutoEigenaar met
  + constructor(auto,eigenaar)

hallo bitch