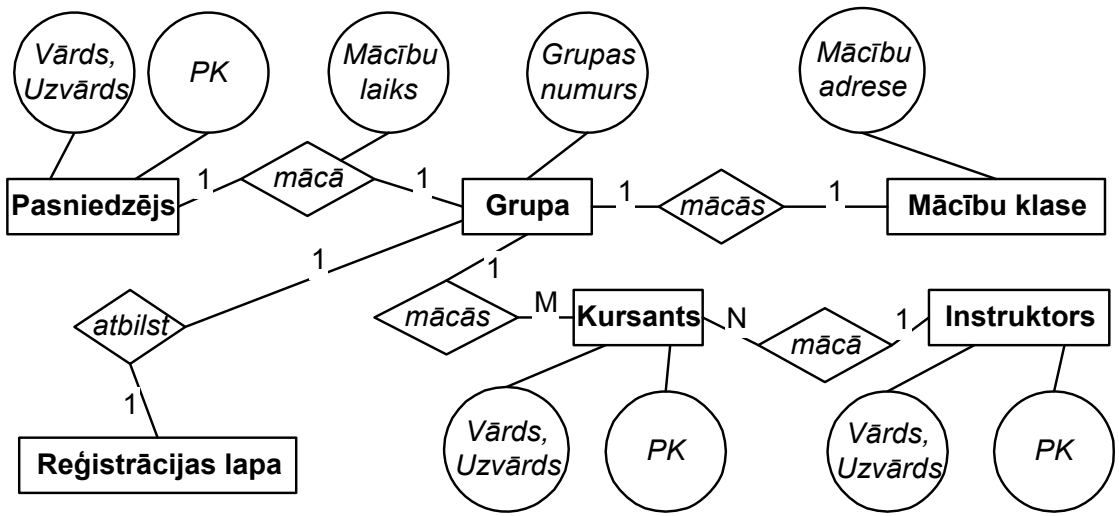
Programmatūras attīstības tehnoloģijas:

Objektorientēta tehnoloģija
Vienota modelēšanas valoda (UML)
Modeļvadāma arhitektūra (MDA)

Dr.sc.ing., asoc. prof. Oksana Ņikiforova
DITF LDI
Lietišķo datorzinātņu katedra
Rīga - LV1048, Meža 1/3, 510.kab., tel.708 95 98
oksana.nikiforova@cs.rtu.lv

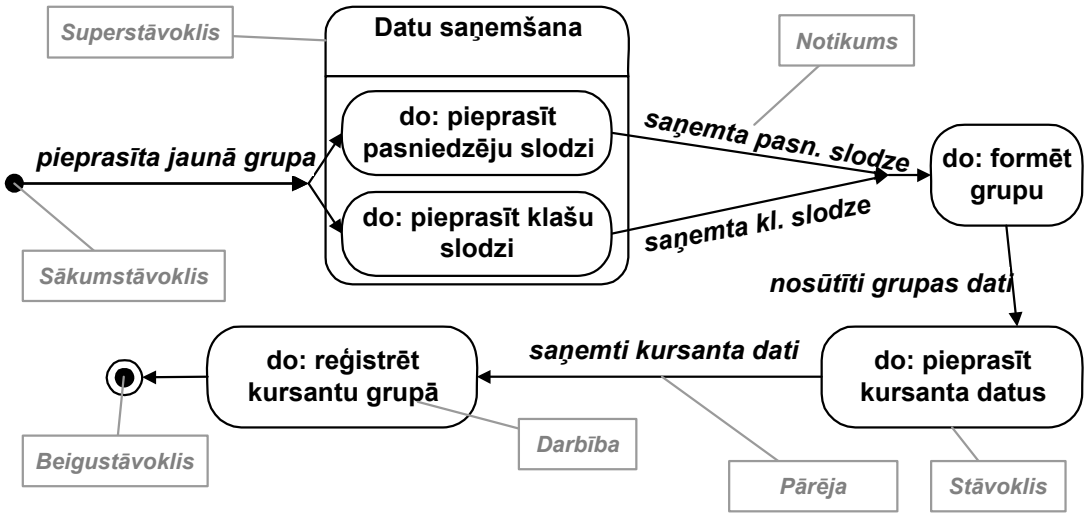
Datu modelēšana

- Relāciju modelis [Codd 1970], Date [1986]
- Relāciju modeļa populārāka notācija – t.s. ER (entity-relationship) diagramma [Chen 1976]
- ER diagramma satur informāciju par entītijām, attiecībām starp tām un to atribūtiem
- Galvenā priekšrocība:
  - instrumentālo līdzekļu relatīvā vienkāršība
- Pamattrūkumi:
  - Netiešā datu tipizācija, kas seko tikai no attiecību nosaukuma
  - Attiecību loģikas sekundārā loma
- ER diagrammas paplašinājums [Hull, King 1987] ar papildus attiecību tipiem



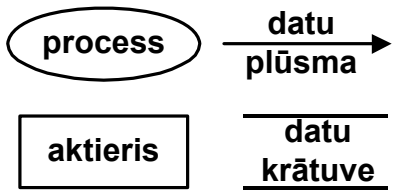
Dinamikas modelēšana

- Finite sate machine, Petri-nets
- Dinamikas modelēšanas populārāka notācija – Stāvokļu diagramma (statechart) [Harel 1987]
- Stāvokļu diagramma satur informāciju par notikumu secību, stāvokļu maiņu un stāvokļa kontekstu.
- Stāvokļu diagrammas paplašinājums – stāvokļu iegremdēšana (nesting), kas veicina hierarhijas ieviešanu diagrammās un ļauj saprast sistēmas uzvedību dažādos abstrakcijas līmeņos

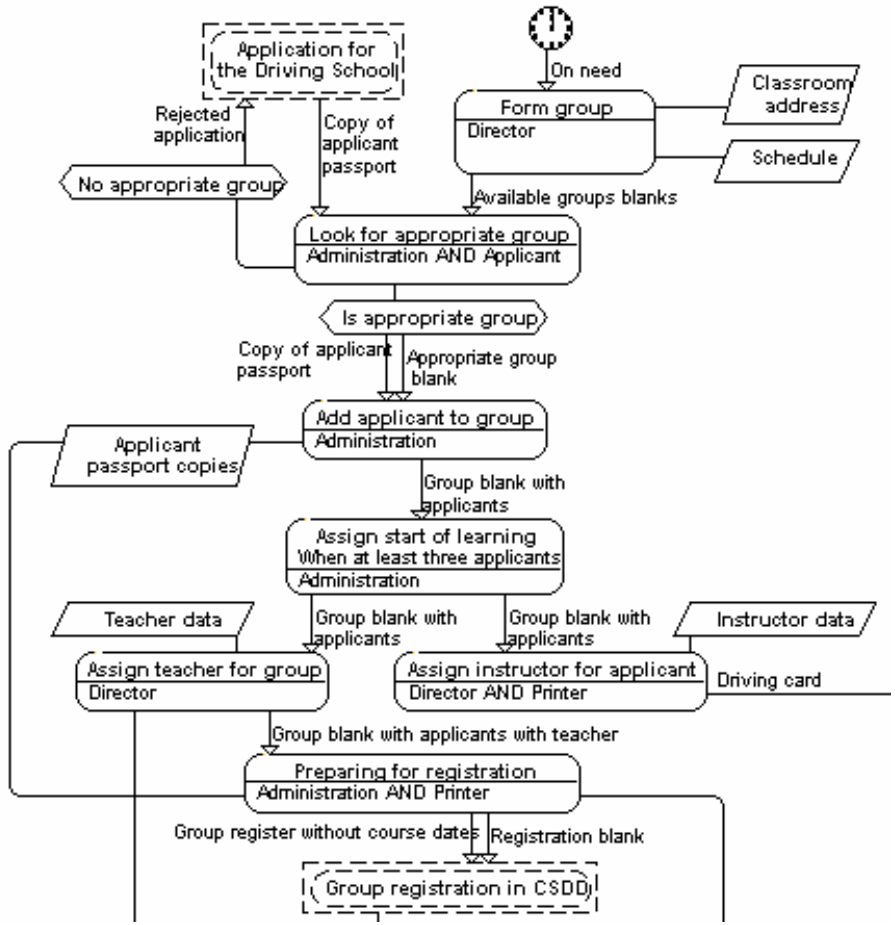


Procesu modelēšana

- Tom Demarco ir strukturālās projektēšanas tēvs [1976] un Ed Yourdon ir datu plūsmu ieviešanas programmatūras izstrādes procesā veicinātājs [1979]
- Procesu modelēšanas populārāka notācija – datu plūsmu (data flow) diagramma
- Datu plūsmu diagramma satur informāciju par procesiem, datu plūsmām, datu glabātuvēm un aktieriem



- Mūsdienās populārāka procesu modelēšanas joma ir biznesa procesu modelēšana



Objektorientētas tehnoloģijas attīstība

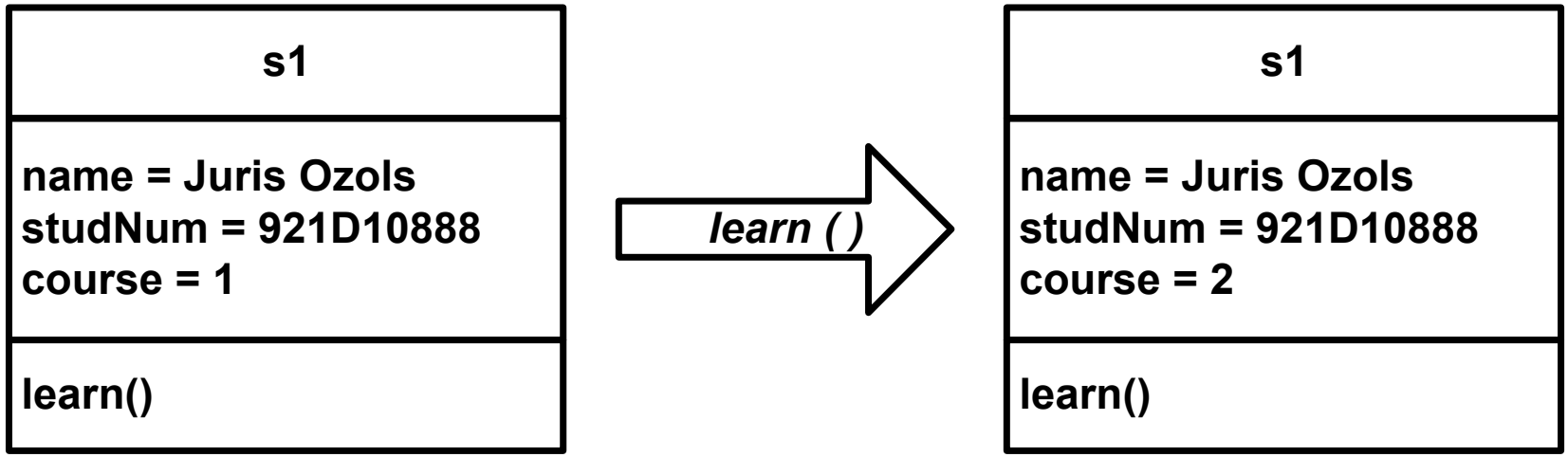
- Objektorientētas pieejas pamatlicēji ir senie grieķi [J.Osis 2001]
- 1968.gads Nygaard un Dahl darbi valodā Simula
- Līdz 1980.gadu vidu objektorientēti darbi tika fokusēti uz objektorientētu kodēšanu
- 1980.gados parādījās objektorientētas programmēšanas valodas Smaltalk, C++, Ada
- 80.gadu beigās tika konstatēts, ka objektorientācija ir tik pat svarīga jau sistēmas analīzē un projektēšanā, kā programmēšanas fāzē
- 1986.gads pirmais raksts par programmatūras izstrādes pilna dzīves cikla objektorientētām iestrādēm, autors - Grady Booch
- 1988.-1994.gadi – objektorientētas sistēmas analīzes un projektēšanas metodes
- Viens no galvenajiem A&P uzdevumiem ir sistēmas modelēšana un lielu metožu skaita dēļ ražošanā nebija saskaņas: kādu no modelēšanas tehnikām uzskatīt par standarta modelēšanas tehniku
- 1994.gadā notika pirmais metožu apvienojums: Grady Booch, James Rumbaugh un Ivar Jackobson (no 1995.gada) apvienoja savus izstrādājumus
- 1996.gads – konsorcijs no 12 kompānijām – lai izveidotu vienotas modelēšanas valodas UML (Unified Modelling Language) specifikāciju
- 1997.gads UML 1.1 pasludināta par standartu programmatūras izstrādē
- 2004.gads UML 2.0 versija – joprojām turpinās attīstība
- 2005.gada beigas UML 2.0 versija beidzot ir oficiāli pieņemta

Objekts

- No cilvēka viedokļa:
  - Sataustāms un (vai) redzams priekšmets
  - Kaut kas, uztverams ar domāšanu
  - Kaut kas, uz ko ir virzīta doma vai darbība
- Objekts ir konkrēts, identificējams priekšmets, vienība vai būtība (reāla vai abstrakta), kurai ir tieši nodefinēts funkcionālais aprīkojums dotajā problēmvidē.
- "Objekts" jeb "Klases eksemplārs"

Objektorientētas pieejas koncepcijas

- **Objekts** ir reālas pasaules būtība. Objektam ir stāvoklis un uzvedība.
- Objekta **stāvoklis** ir objekta īpašības (atribūti) ar to tekošām vērtībām.
- Objekta **uzvedība** ir darbība, ko veic objekts vai ko ar viņu veic citi objekti.



Operācijas **learn()** rezultātā objekts **s1** pāriet citā stāvoklī

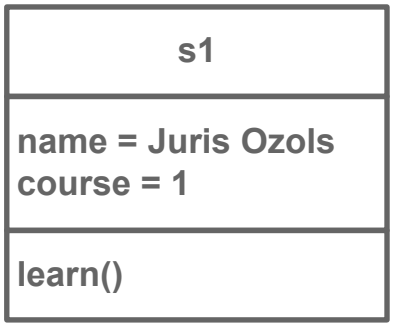
Objekta identiskums ir tā objekta īpašība, kas atšķir viņu no citiem objektiem.

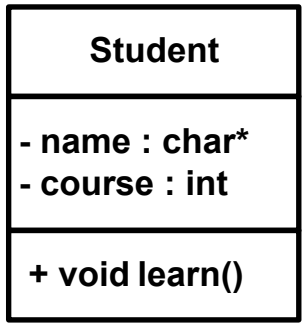
Objektorientētas pieejas koncepcijas

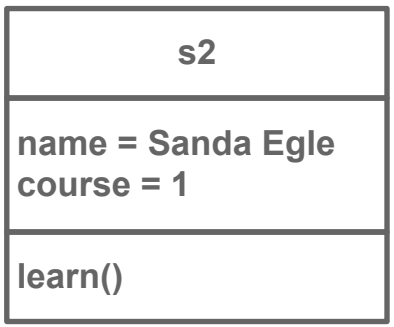
- **Objektu klasifikācija**
  - Objekti ar līdzīgu datu struktūru (atribūtiem) un uzvedību (metodēm) ir apvienoti klasēs.
  - Tā ir **galvenā objektorientācijas filozofija** meklēt objektus ar līdzīgiem atribūtiem un metodēm un apvienot tos klasē.
  - **Iekapsulēšana** – ir klases atribūtu un metožu realizācijas slēpšana klases iekšienē (klases definīcijā).
- **Klašu specificēšana**
  - **Deklarācija** – klases struktūras (atribūtu) un uzvedības (metožu) apraksts.
  - **Definīcija** (jeb **realizācija**) – klases uzvedības (metožu) izpildīšanas apraksts.
  - **Izmantošana** – klases metožu pielietošana mijiedarbībā ar citu klašu objektiem.

Objektorientētas pieejas koncepcijas

- Attiecības starp **objektu un klasi** ir tas pats kā attiecības starp **mainīgo un tipu** klasiskajā programmēšanā.
- Klasei ir nosaukums, atribūti un metodes.
- Objekts (klases **eksemplārs**) ir veidots piešķirot klases atribūtiem noteiktas vērtības (objekta **inicializācija**).



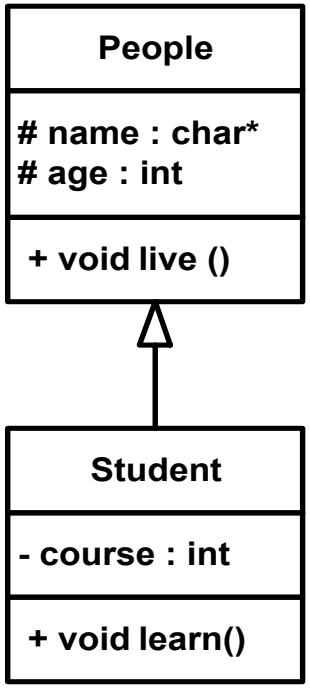





```
class Student {
private:
          char *name;
          int course;
public:
          void set_name (char *n) {name=n;}
          char* get_name () {return name;}
          void set_course(int c) {course=c;}
          int get_course() {return course;}
          void learn() {course+=1;}
};

void main(){
          Student s1;
          s1.set_name("Jānis Ozols");
          s1.set_course(1);
          s1.learn();
}
```

Objektorientētas pieejas koncepcijas

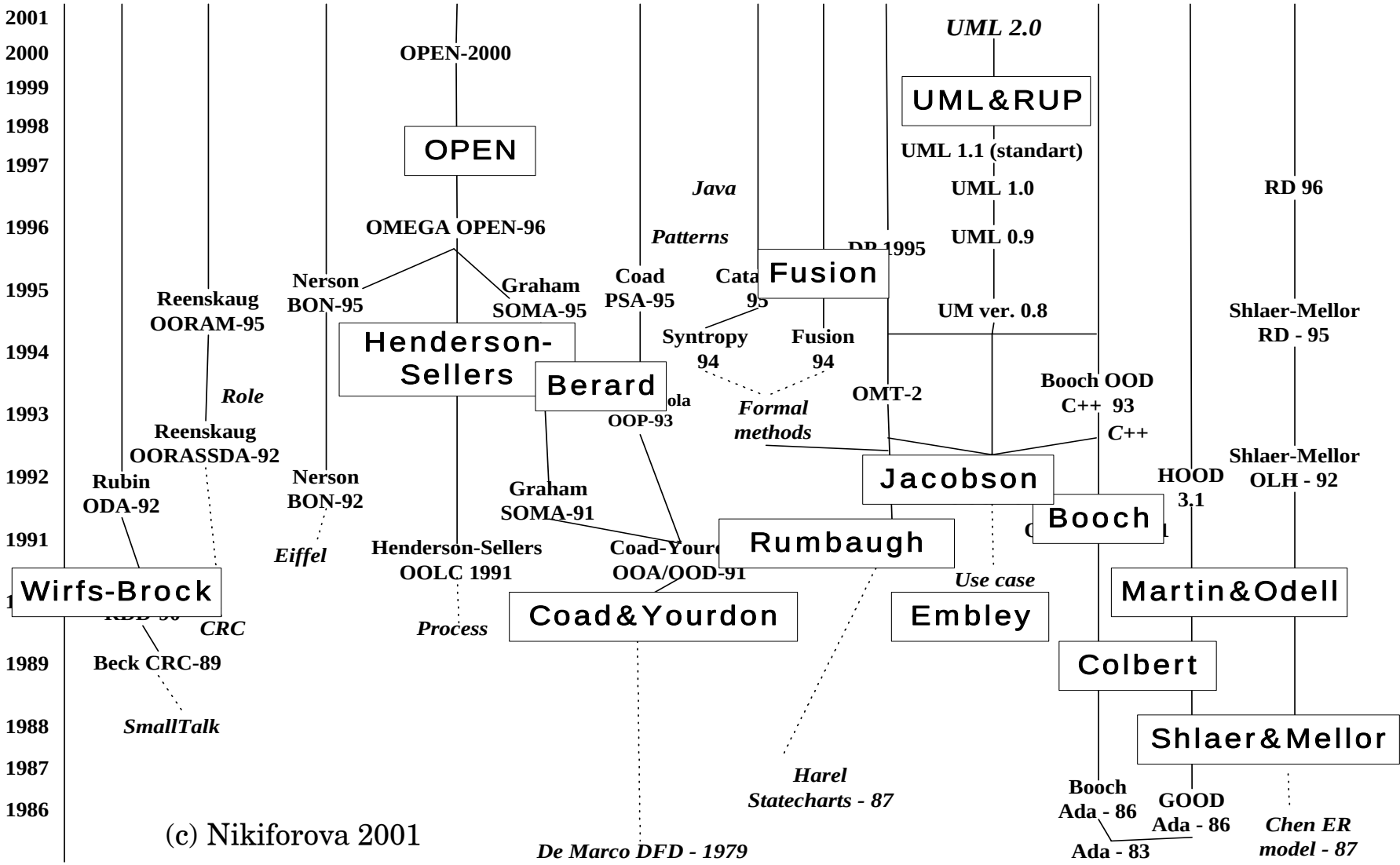
Mantošana – attiecības starp klasēm, kad viena klase atkārto citas klases struktūru un uzvedību. Klase (kas ir saukta **apakšklase**) var mantot īpašības un metodes no citas klases (kas ir saukta **superklase**).



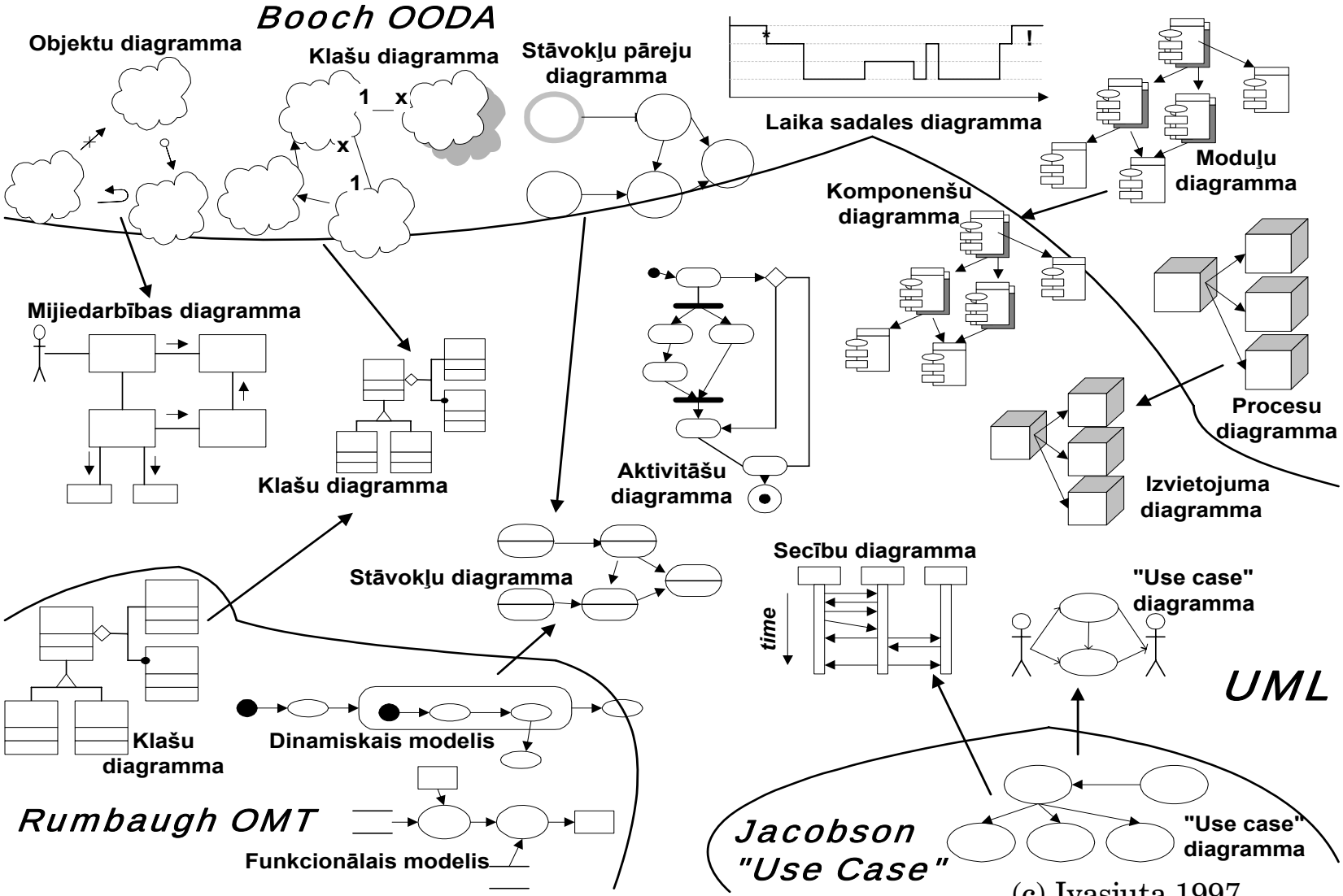

```
class People {
  protected:
      int age;
      char *name;

  public:
      void live() {age += 1;}
      int get_age() {return age;}
      void set_age(int value) {age = value;}
      char * get_name() {return name;}
      void set_name(char *value) {name = value;}
};

class Student : public People {
  private:
      int course;
  public:
      void learn() {course += 1;}
      int get_course() {return course;}
      void set_course(int value) {course = value;}
};
```

Atkārtota lietošana ir viena no objektorientētas pieejas priekšrocībām, kas nodrošina izstrādāto komponentu atkārtotu izmantošanu programmatūras izstrādē.

Objektorientētu metožu attīstība

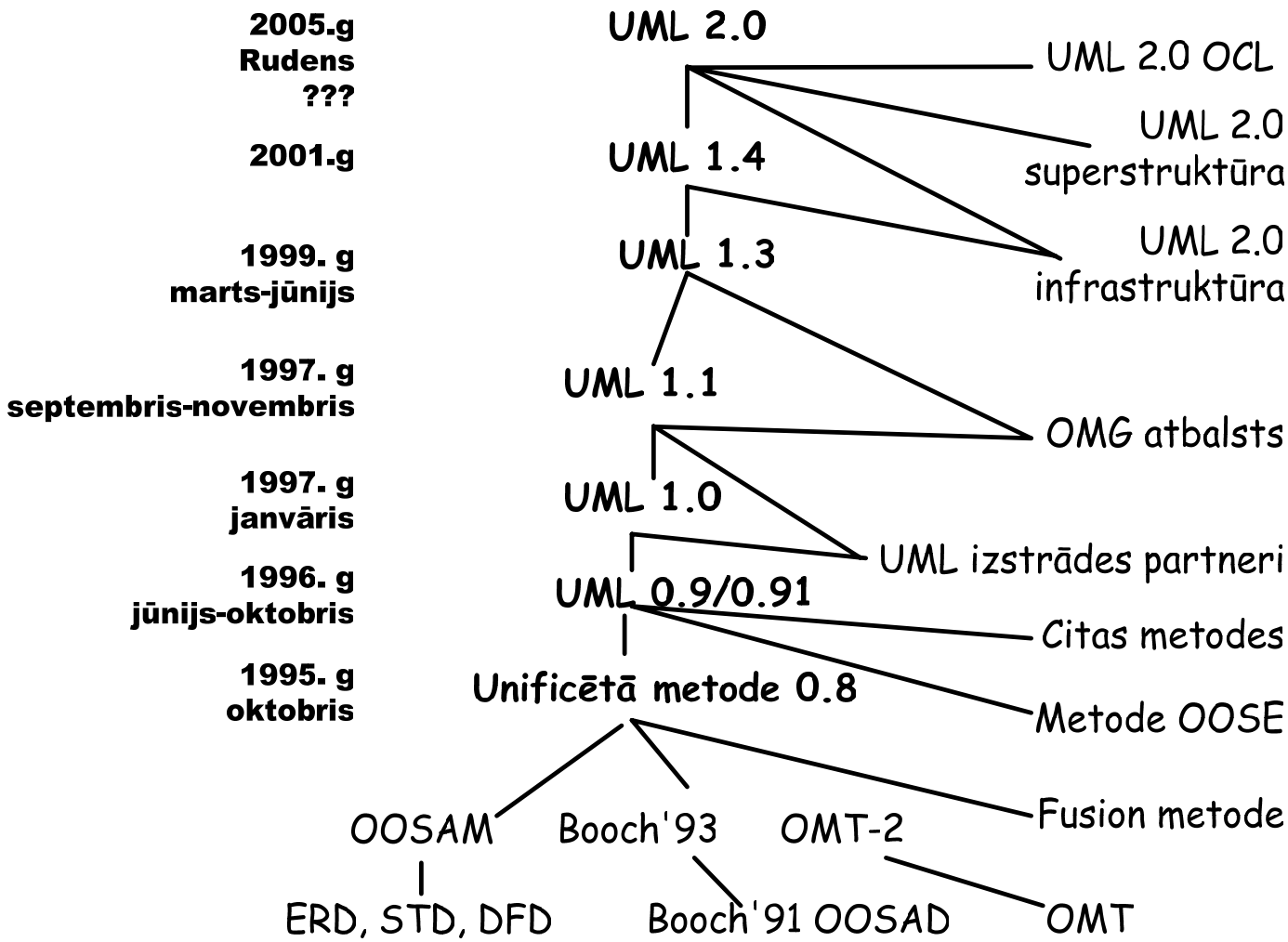


Vienota modelēšanas valoda UML



(c) Ivasjuta 1997

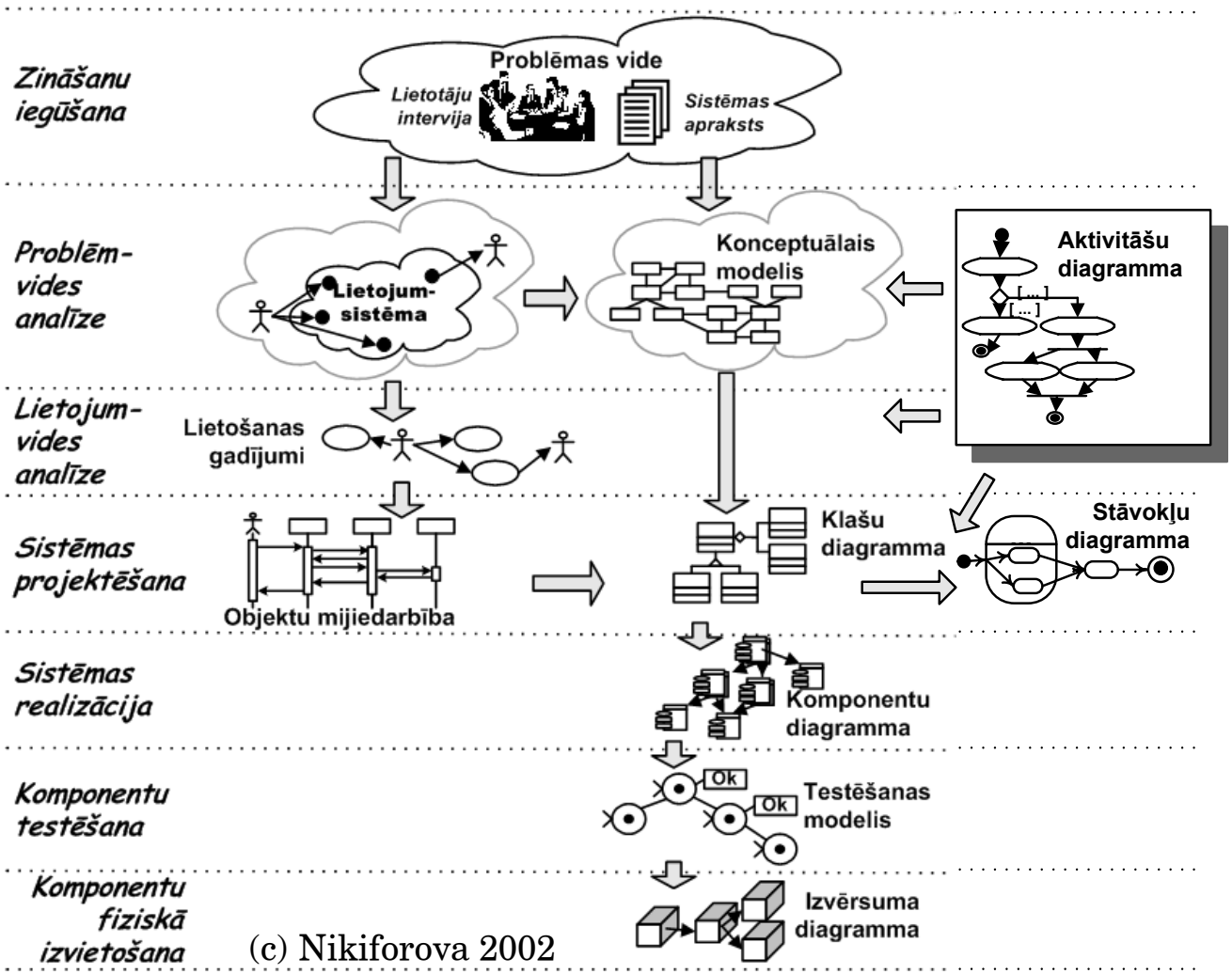
UML rašanās vēsture



Vienota modelēšanas valoda Unified Modelling Language (UML)

- Lietošanas gadījumu diagramma (**use case diagram**)
- Klašu diagramma (**class diagram**)
- Uzvedības diagrammas:
  - Mijiedarbības diagrammas:
    - Secību diagramma (**sequence diagram**)
    - Sadarbības diagramma (**colaboration diagram**)
  - Aktivitāšu diagramma (**activity diagram**)
  - Stāvokļu diagramma (**statechart diagram**)
- Realizācijas diagrammas:
  - Komponenšu diagramma (**component diagram**)
  - Izvietojuma diagramma (**deployment diagram**)

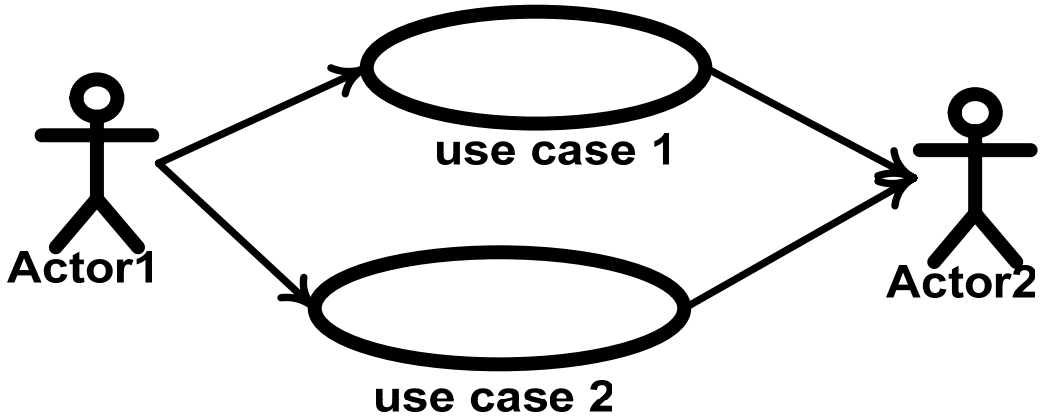
Objektorientētās programmatūras vispārēja uzbūve (UML)



(c) Nikiforova 2002

Lietošanas gadījumu diagramma

Lietošanas gadījumu diagramma attēlo sistēmas ārējo uzvedību: sistēmā paredzētas darbības (lietošanas gadījumus), to apkārtni (aktierus) un attiecības starp tiem.



Lietošanas gadījuma use case 1 apraksts

Lietošanas gadījums:	**Autorizācija**
Aktieri:	**Klients, Kartiņa**
Nolūks:	pārbaudīt iespēju darboties ieliktas kartiņas klientu, pārbaudīt paroli
Īss apraksts:	Klients ieliek kartiņu kartiņas nolasīšanas ierīcē, pēc pieprasījuma ievada paroli un bankomāts pārbauda datus. Beidzot darbu ar autorizāciju, klientam tiek piedāvāts izvēlēties vienu no operācijām "apskatīt kontu" vai "izņemt naudu".
Tips:	Pamatlietojums
Šķērsatsauces:	lietošanas gadījumu "apskatīt kontu" vai "saņemt naudu" pirmsdarbība.

Tipiskā notikumu secība:

	Aktieru darbības		Sistēmas reakcija
1	Klients ieliek kartiņu bankomātā	2	Bankomāts pārbauda, vai var apkalpot šīs bankas kartiņu
		3	Bankomāts pārbauda kartiņas derīguma termiņu
		4	Bankomāts apsveicina klientu un piedāvā viņam ievadīt personālo identifikācijas numuru (PIN)
5	Klients ievada numuru	6	Bankomāts apstiprina ievadīto numuru
		7	Bankomāts izvada pieejamo operāciju sarakstu: apskatīties konta stāvokli vai izņemt naudu
8	Klients izvēlas vajadzīgo operāciju	9	Lietošanas gadījuma beigas

Alternatīva notikumu gaita: **2.:** nav atļauts apkalpot šīs bankas klientu... **3.:** kartiņas derīguma termiņš ir beidzis... **4.:** nepareiza identifikācijas numura ievade...

Kļūdu notikumi: **6.,7.:** kļūda apstiprinot PIN kodu...

Lietošanas gadījuma use case 2 apraksts

Lietošanas gadījums:	**Izņemt naudu**
Aktieri:	**Klients, Kartiņa**
Nolūks:	noteiktas naudas summas izņemšana no bankomāta
Īss apraksts:	Klients ir izvēlējies operāciju "izņemt naudu", nolasītai kartiņai attiecīgajā bankā pēc attiecīga konta stāvokļa tiek pieprasīts apstiprinājums vai izsniegt pieprasīto naudas summu. Ja pieprasījums ir apstiprināts – izsniegt pieprasīto naudas summu, ja nav – paziņot uz ekrāna. Beidzot naudas izņemšanu, klientam atgriezt kartiņu un izdrukāt čeku.
Tips:	Pamatlietojums
Šķērsatsauces:	jābūt sekmīgi izpildītam lietošanas gadījumam "Autorizācija"

Tipiskā notikumu secība:

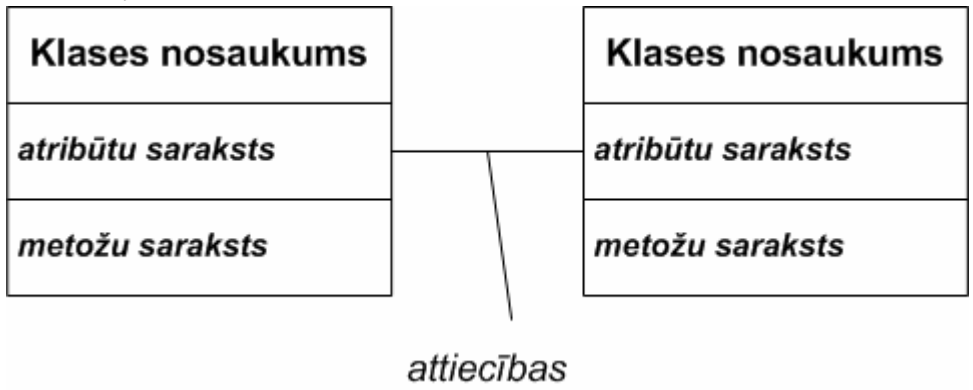
	Aktieru darbības		Sistēmas reakcija
1	Klients ir izvēlējies punktu "Izņemt naudu".	2	Bankomāts pieprasa ievadīt vajadzīgo naudas summu.
3	Klients ievada nepieciešamo summu.	4	Bankomāts noteic, vai kontā pietiek naudas.
		5	Bankomāts atskaita vajadzīgo summu no klienta konta.
		6	Bankomāts izsniedz ievadīto summu skaidrā naudā.
7	Klients izņem naudu.	8	Bankomāts izsniedz klientam viņa kartiņu.
9	Klients izņem kartiņu		Bankomāts drukā kvīti
			Lietošanas gadījuma beigas

Alternatīva notikumu gaita: **4.:** kontā nepietiek naudas, lai izsniegt pieprasīto summu...; **6.:** bankomātā nepietiek naudas, lai izsniegtu pieprasīto summu?; **7.:** klients neizņem naudu ...; **8.:** klients neizņem kartiņu ...

Kļūdu notikumi: **4.,5.:** kļūda apstiprinot ievadīto summu...

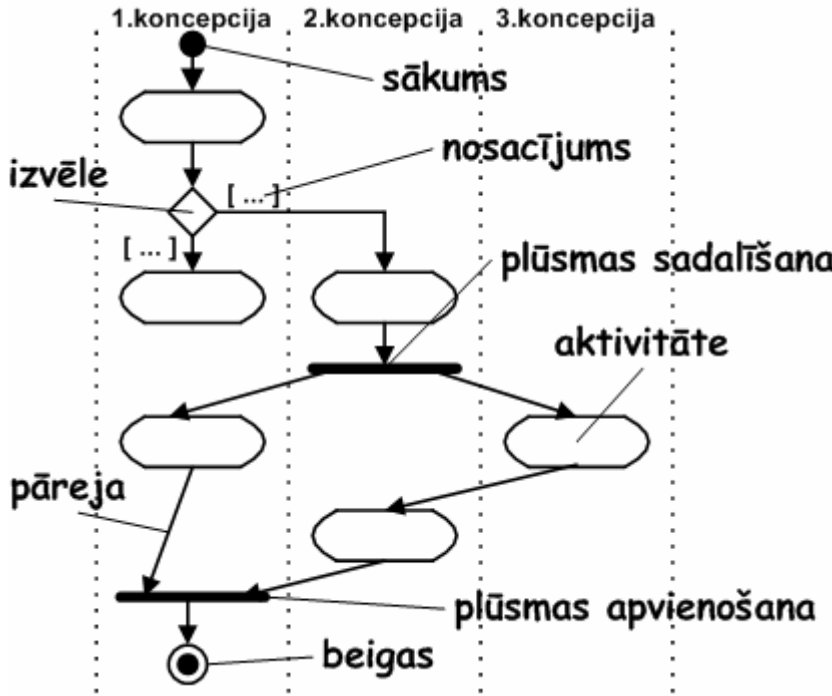
Klašu diagramma

- Klašu diagramma parāda modeļa statisko struktūru: būtības(entītijas), kas eksistē
  sistēmā, to iekšējo uzbūvi un attiecības ar citām sistēmas būtībām.
  - Klases diagrammas pamatelementi ir klases un attiecības starp tām.
  - Klases attēlošanai ir izvēlēta James Rumbaugh piedāvāta konstrukcija – sadalīts trīs daļās taisnstūris:
    - Augšējā daļā ir parādīts **klases nosaukums**.
    - Klases **struktūra** ir parādīta ar atribūtu (īpašību) kopu.
    - Klases **uzvedība** ir parādīta ar metodēm (operācijām).
  - Realizācijas līdzekļi nodrošina to, ka operācijas uzreiz ir piesaistītas pie klasēm (ievērojot objektu grupēšanu) tā, kā secību diagrammas parāda objektu uzvedību.

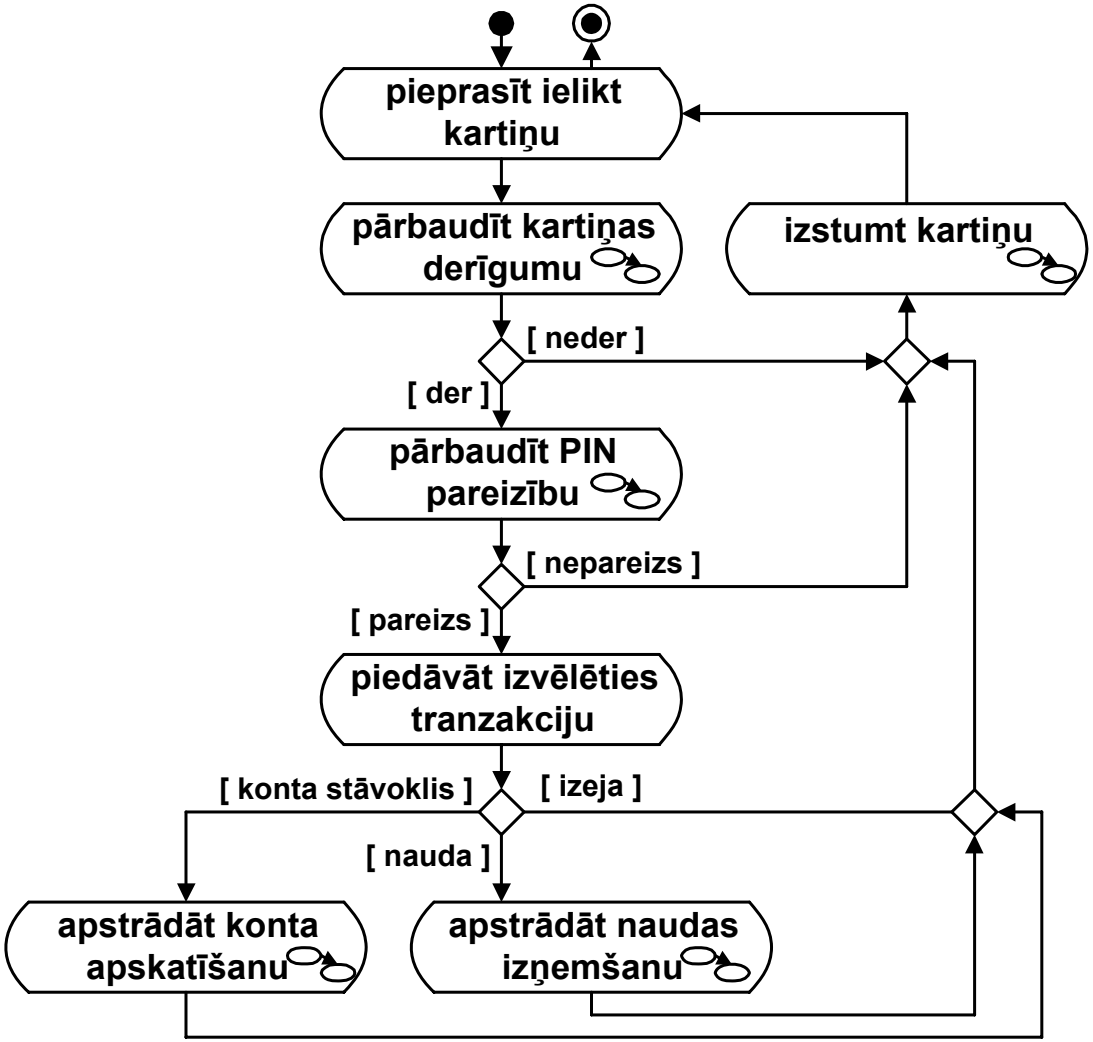


Aktivitāšu diagramma

- Aktivitāšu diagrammas attēlo sistēmas operāciju algoritmisko un loģisko realizāciju, kā arī biznesprocesu modelēšanai.
- Aktivitāšu diagramma ir veidota sistēmas uzvedībai kopumā (attēlo vadības plūsmas starp funkcijām sistēmā) vai noteiktai funkcijai (attēlo plūsmas noteiktas funkcijas iekšienē).
- Aktivitāšu diagrammas elementi ir stāvoklis, pāreja, izvēle un sinhronizācijas līnija.



Aktivitāšu diagramma kopējam procesam

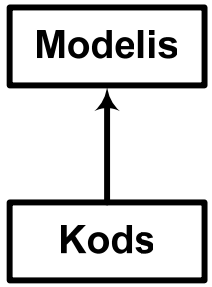


Kodēšana pret modelēšanu

Tikai kods

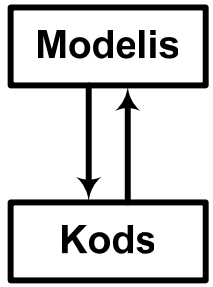
Kas ir modelis?

Koda vizualizācija



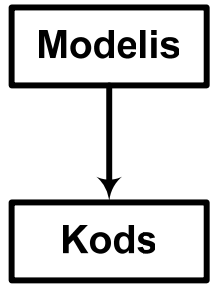
Kods ir modelis?

Reinženierija



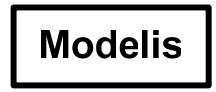
Kods un modelis līdzeksistē?

Ar modeli vadīta



Modelis ir kods?

Tikai modelis



Pagaidam tikai modelē...

Tradicionālā modelēšana un izstrāde