# Teoretické základy projektování IS

4.11.2010

Princip rozlišovacích úrovní, princip tří architektur, princip modelování.

Druhy přístupů k analýza a návrhu IS - Strukturovaný přístup, Objektově orientovaný přístup.

## Analýza a návrh IS

- Myšlenkové postupy ABSTRAKCE a KONKRETIZACE využíváme v průběhu celého procesu analýzy a návrhu IS.

Na myšlenkových postupech A a K jsou založeny principy:

- Princip rozlišovacích úrovní;
- Princip tří architektur;
- Princip modelování.

## Princip rozlišovacích úrovní

- Založen na zobrazení vyvíjeného systému na určité podrobnosti rozlišení detailů.
- V případě nejhrubší rozlišovací úrovně je IS znázorněn jako jeden prvek spolupracující s okolím – principem zobrazení je popis vazeb systému s okolím.
- Jde o víceúrovňové zobrazení systému (od rozlišovací úrovně „0“ až po úroveň „N“, na které vidíme detaily rozpracované do potřebné úrovně pro řešení problému.

## Princip tří architektur

- Úzce souvisí s principem rozlišovacích úrovní.
- Veškerá zobrazení vyvíjeného IS jsou rozdělena do tří rozlišovacích úrovní – kategorií (vrstev podrobnosti) = tzv. VRSTVENÁ ABSTRAKCE SYSTÉMU.

1. Vrstva – KONCEPTUÁLNÍ (esenciální)
2. Vrstva – TECHNOLOGICKÁ (logická)
3. Vrstva – IMPLEMENTAČNÍ (fyzická)

# Princip tří architektur

1. Vrstva zobrazení systému – KONCEPTUÁLNÍ

Zobrazení systému na hrubé rozlišovací úrovní – tj. zobrazení systému a jeho vazem na okolní systémy, dále zobrazení systému s viditelnými subsystémy a vazbami mezi nimi.

S takovéhoto zobrazení je patrné, které subsystémy fungují jako rozhraní mezi IS a okolím, resp. Které subsystémy zpracovávají VSTUPY a VÝSTUPY systému vzhledem k okolí.

**ÚČELEM zobrazení 1. vrstvy je odpověď na otázku CO je obsahem systému?**

# Princip tří architektur

2. Vrstva zobrazení systému – TECHNOLOGICKÁ

Zobrazení systému, která zviditelňují, **JAK je obsah systému realizován?**

Je zde zohledněna **organizace dat** (souborový systém, relační databázový systém, apod.) a **architektura systému** (např.: klient/server).

Zobrazení není zatíženo implementačními specifiky (tj. konkrétní realizace za pomoci prostředků ICT).

# Princip tří architektur

3. Vrstva zobrazení systému – IMPLEMENTAČNÍ

Detaily konkrétní implementace (např. vlastnosti plynoucí z konkrétního databázového systému).

**Z této vrstvy je patrné ČÍM je systém realizován.**

**Princip tří architektur** je aplikován v průběhu celého procesu analýzy a návrhu IS – vznikají MODELY vyvíjeného IS na jednotlivých úrovní ABSTRAKCE.

V jednotlivých etapách vývoje IS jde o snahu zpracovat model IS v požadované vrstvě.

# Princip modelování

- V podstatě jde o TVORBU MODELU vyvíjeného IS.
- MODELOVÁNÍ = účelové zjednodušené zobrazení systému za pomoci vhodných (např. grafických) prostředků.
- MODELOVÁNÍ = ABSTRAKTNÍ obraz reality.

- MODEL = formalizovaný prostředek pro znázornění vyvíjeného IS, prostředek komunikace mezi odborníky, analytiky a uživateli IS. Znázorňuje strukturu systému (strukturu procesů, dat, atd.) na zvolené rozlišovací úrovni. Umožňuje optimalizaci struktury systému vzhledem ke zvoleným kritériím. Dále umožňuje simulaci s studium provedených změn systému a jejich vliv na subsystémy a okolí systému.

## Charakteristika modelu

- Model formulován jako systém (tzn. znázorňuje prvky a jejich vzájemné vazby),
- Hraniční prvky realizují vazby s okolím systému (VSTUPY,VÝSTUPY),
- Obsah modelu je objektivní – každý prvek modelu odpovídá objektu reálného světa (tzv. pomocný prvek),
- uspořádání prvků modelu odpovídá uspořádání prvků části reálného světa, který model znázorňuje.

## Analyzované dimenze IS

- **funkční**

(popis procesů, datových toků a vazeb mezi subsystémy)

- **datová**

(popis druhů dat, se kterými bude IS pracovat)

- **řídící**

(popis časových souvislostí systémových akcí)

- **organizačně-technologická**

(popis a znázornění organizace práce s IS, popis zamýšlených provozních technologií, které bude IS realizovat)

- **systémově-technologická**

(popis realizace implementačně závislých systémových funkcí a jejich časové návaznosti)

# Druhy přístupu k analýze a návrhu IS

V průběhu historického vývoje se vyprofilovaly dva základní přístupy k analýze a návrhu IS:

- STRUKTUROVANÝ přístup (70. léta 20. stol.),
- OBJEKTOVĚ ORIENTOVANÝ přístup (90. léta 20. stol.).

# Strukturovaný přístup

- Pojem „strukturovaný přístup“ odráží myšlenkový postup „strukturování“ (problematiky i předmětu zkoumání).
- V průběhu analýzy a návrhu IS potřebujeme zobrazit dva hlavní aspekty vyvíjeného IS:

1) PROCESY probíhající v systému
2) DATA, se kterými systém pracuje a která produkuje.

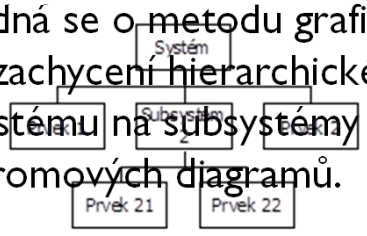
# MODELY

STRUKTUROVANÝ PŘÍSTUP – charakteristické (na rozdíl od přístupu objektového) relativně samostatné zobrazení datových struktur systému v jednom modelu; datových toků a procesů zpracovávajících data v jiném modelu.

Metody strukturované systémové analýzy jsou plně v souladu s principy obecné teorie systémů

# Funkční struktura

- Metoda analýzy funkční struktury je jednou ze základních metod strukturované analýzy používaná především k popisu struktury systému. Jedná se o metodu grafickou, která slouží k zachycení hierarchické dekompozice systému na subsystémy a prvky pomocí stromových diagramů.



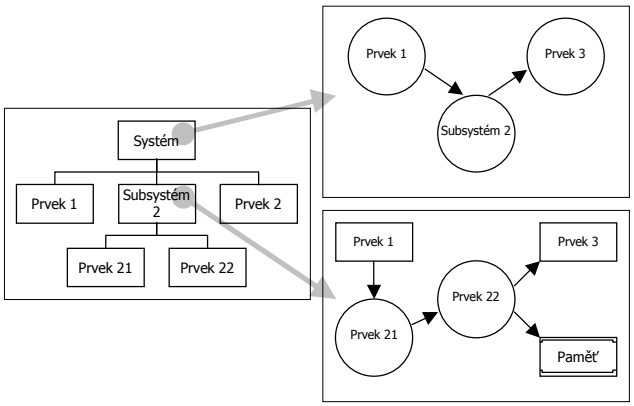
# Informační toky

- Grafická metoda, která formou hierarchicky uspořádaných síťových diagramů vyjadřuje dekompozici systému na subsystémy a prvky a současně dovoluje zachytit informační vazby mezi těmito prvky.
- Vhodná metoda pro studium strukturálních vlastností systému.
- Základními aktivními prvky jsou funkční prvky neboli funkce, prvky zajišťující transformaci vstupní informace na výstupní.
- **Aktivní prvky** lze dále rozlišovat na prvky příslušné k popisovanému systému a prvky, které lze považovat vzhledem k popisovanému systému za vnější.

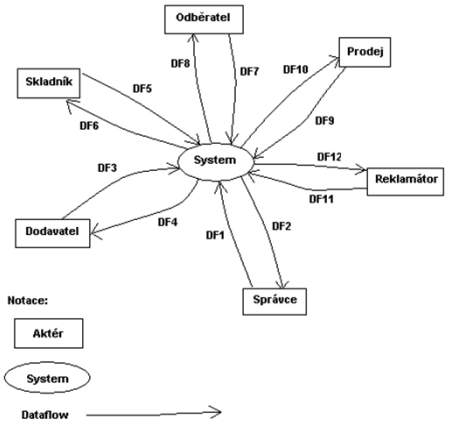
# Informační toky

- **Pasivní prvky** představují paměti. Jedná se o prvky, které jsou schopny uchovat uloženou informaci. V případě softwarově orientovaných systémů mohou být realizovány například soubory nebo databázemi, v oblasti nesoftwarových systémů například protokoly, seznamy nebo záznamovými knihami.
- Významnou složkou diagramu informačních toků jsou **informační vazby mezi prvky systému** – informační toky. Obsah informačního toku nemůže být s ohledem na požadavek přehlednosti diagramu vyjádřen zcela detailně.
- **Metoda informačních toků** vyjadřuje formou hierarchicky uspořádaných síťových diagramů dekompozici systému na subsystémy a prvky a současně zachycuje informační vazby mezi těmito prvky.
- Na vrcholu této hierarchie stojí **tzv. kontextový diagram,** který vyjadřuje začlenění systému do souvislostí okolního světa.

# Informační toky




# Kontextový diagram



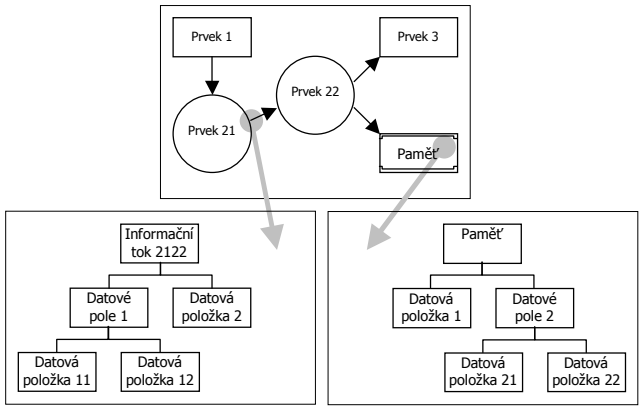
## Datové struktury

- Metoda analýzy funkční struktury a metoda informačních toků neposkytují vhodné a dostatečné prostředky pro analýzu datové složky systému. Jak vazby mezi prvky systému popsané informačními toky, tak i pasivní prvky systému mohou představovat složité a rozsáhlé datové struktury.
- Jedním z prostředků datové analýzy je **popis datových struktur.** Analýza datových struktur umožňuje pomocí hierarchických stromových diagramů postupné rozčlenění informačních toků nebo paměťových prvků. **Všechny prvky tohoto diagramu představují obecně chápané bloky informace – datové položky.**

## Datové struktury

- Na analýzu struktury dat bezprostředně navazuje detailní datová analýza, která je prostředkem k podrobnému popisu elementárních datových položek – **listových prvků hierarchického stromového diagramu.**
- Detailní datová analýza umožňuje přiřadit každé elementární položce datové struktury datový element, který určuje formu uložení informace. Pro datové elementy je možno specifikovat například typ, rozsah nebo výčet možných hodnot.

# Datové struktury

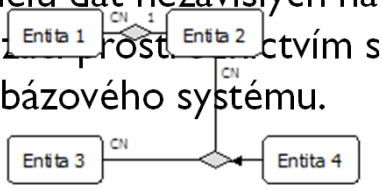



# ER model

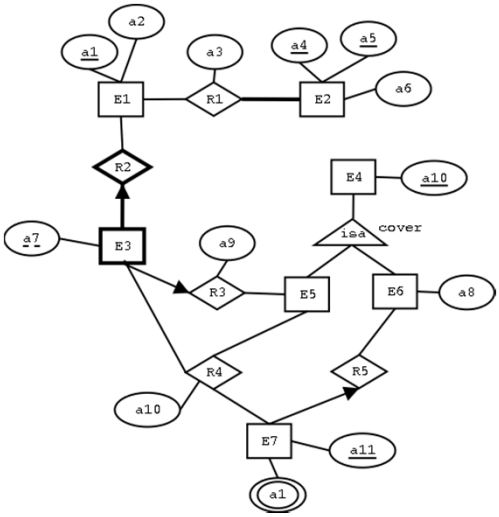
- ER model – model entit a jejich vzájemných vztahů.
- Vhodný pro analýzu systému v případě, kdy složitost systému spočívá spíše ve složitosti struktury dat než ve složitosti jeho funkčních složek.
- ER model zachycuje formou síťového grafu objekty reálného světa a vztahy mezi nimi. Množiny objektů reálného světa mající shodné vlastnosti se nazývají entitami, vztahy mezi nimi pak relacemi. Entity mohou být blíže specifikovány množinou atributů, které mají shodný význam jako datové elementy užívané při detailní datové analýze.

# ER model

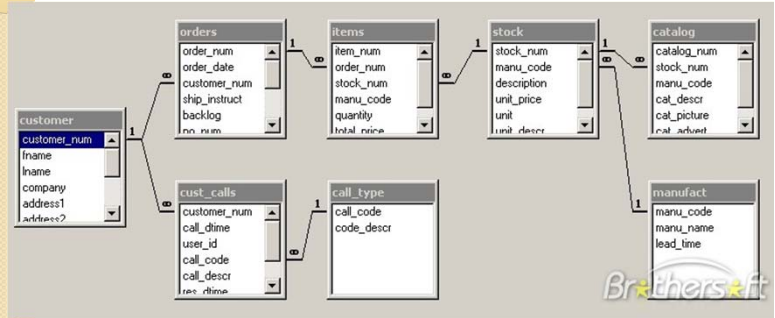
- Relace mezi entitami jsou specifikovány kardinalitou a těsností vazby. ER modely se využívají pro tvorbu modelů dat na logické neboli konceptuální úrovni, tedy modelů dat nezávislých na jejich fyzické reali[illegible]prost[illegible]ctvím specifického databázového systému.




# ER model



# ER model



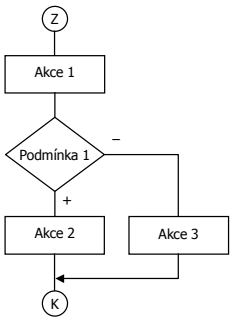

# Metody popisu chování

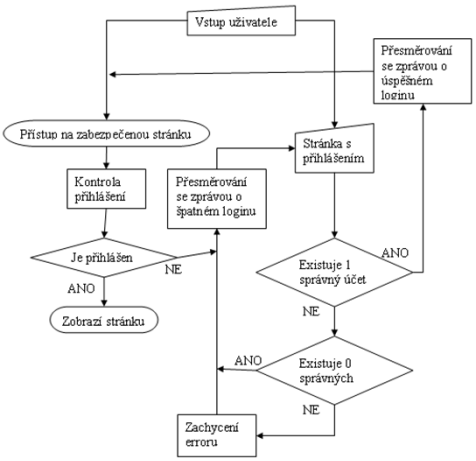
- Popis chování systému je v obecném slova smyslu jeho algoritmizací.
- Algoritmus chování je však v určitých případech nutné nebo účelné podrobněji popsat již ve fázi analýzy. Metody popisu chování umožňují ve fázi analýzy zachytit algoritmickou složku systému, avšak abstrahují od konkrétního způsobu realizace, který je předmětem fáze návrhu.
- Metody jsou založeny na grafickém vyjádření nebo na vhodné kombinaci grafického vyjádření a formalizovaného textového popisu. Obvykle je pomocí grafických prostředků zachycena základní struktura algoritmu, která je pro specifikaci algoritmu na detailní úrovni doplněna relativně krátkými sekvencemi příkazů zapsanými formalizovaným jazykem.

# Metody popisu chování

- Klasickým grafickým prostředkem zápisu algoritmu je vývojový diagram. Díky své jednoduchosti získal oblibu v nejrůznějších oblastech, které daly vzniknout jeho různým modifikacím. Nejjednodušší varianta zachycuje základní strukturu algoritmu pomocí vzájemně propojených elementů představovaných bloky a podmínkami. Detailní specifikace algoritmu je obsahem příslušných elementů a je vyjádřena formalizovaným jazykem.

# Vývojový diagram

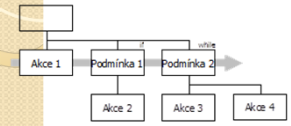


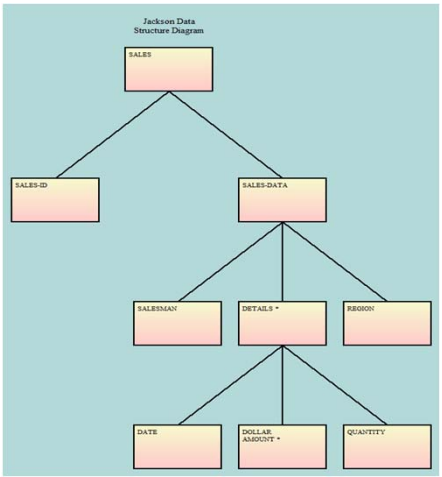



# Metody popisu chování

- Metody popisu chování jsou obvykle velmi příbuzné metodám užívaným při tvorbě programů.
- Při nedodržení určitých pravidel mohou konstrukce zachycené pomocí vývojových diagramů odporovat zásadám strukturovaného programování, a tím mohou znesnadňovat případnou následnou programovou realizaci.
- Mezi metody, které podporují a dodržují zásady strukturovaného programování, patří metoda grafického zápisu algoritmů podle Jacksona, označovaná jako Jacksonovy diagramy. Základní struktura algoritmu je popsána hierarchickým stromovým diagramem, detailní specifikace je obsahem příslušných elementů a je vyjádřena formalizovaným jazykem.

# Jacksonův diagram

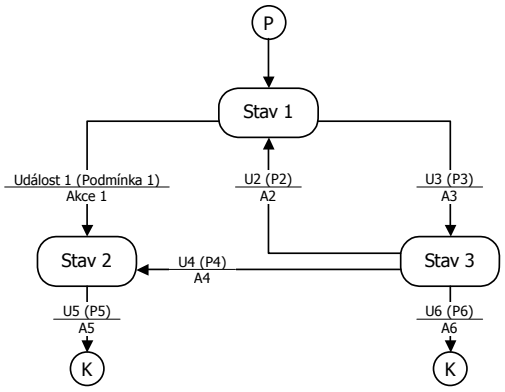


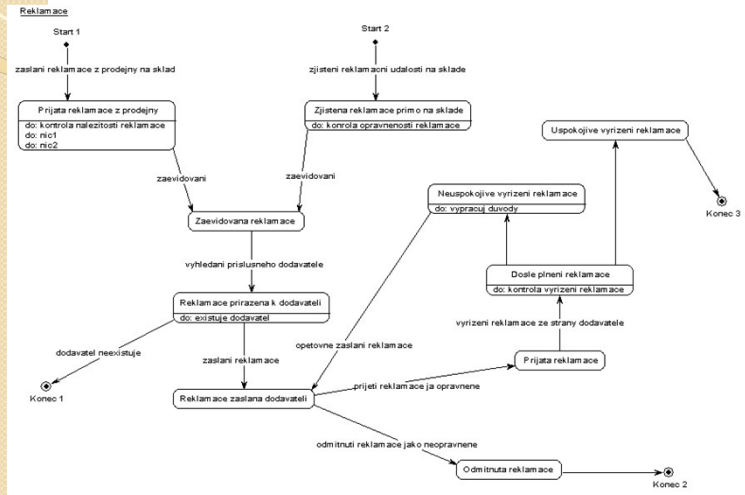
# Metoda stavových diagramů

- grafická metoda popisu chování použitelná pro vyjádření jednodušších algoritmů nebo pro popis algoritmů na hrubé úrovni.
- Systém je popsán SÍŤOVÝM GRAFEM, jehož **uzly znázorňují stav systému a hrany naznačují možné přechody mezi stavy** s vyjádřením podmínek přechodu a akcí s přechodem spojených.
- Akce a stavy systému pouze symbolicky označují funkce a jejich účinky. Algoritmickou náplň akcí je však nutné popsat s využitím jiných prostředků.

# Stavový diagram



# Stavový diagram



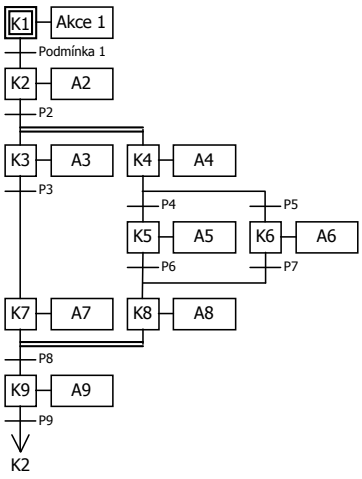

# Metoda sekvenčních funkčních grafů

- Metoda sekvenčních funkčních grafů představuje grafickou metodu popisu chování systému vyhovující nejobecnějším nárokům na popis algoritmu. Základy metody jsou postaveny na principech Petriho sítí.
- Metoda funkčních kroků a přechodů byla poprvé uveřejněna v roce 1977 pod názvem GRAFCET (Graphe Fonctionnel de Commande Etape-Transition). V současné době se však s touto metodou lze setkat častěji pod názvem SFC (Sequential Function Chart).

# Metoda sekvenčních funkčních grafů

- Metoda sekvenčních funkčních grafů využívá k popisu systému obdobně jako metoda stavových diagramů síťový graf.
- Zásadní rozdíl však spočívá v tom, že uzlem síťového grafu není stav systému, ale krok algoritmu, respektive akce s daným krokem spojená. Přechod z kroku do kroku je vázán podmínkou přechodu a aktivitou kroků této podmínce bezprostředně předcházejících. Síťový graf tedy zachycuje základní strukturu algoritmu, detailní popis je pak obsahem akcí spojených s dílčími kroky.

# Sekvenční funkční graf



# Teoretické základy projektování IS

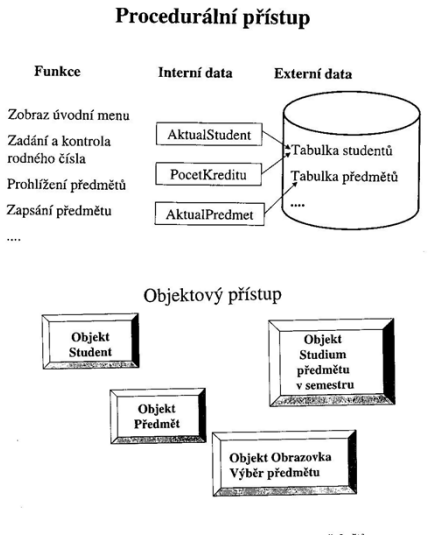
11.11.2010

Objektově orientovaný přístup - Diagram Use case, Sekvenční diagram, Diagram spolupráce, Diagram tříd, Stavový diagram, Diagram aktivit, Diagram nasazení

## Objektově orientovaný přístup

- Historicky mladší technika navazující na strukturovaný přístup.
- Přístup je založen na OBJEKTECH, jakožto strukturách, které mají definované vlastnosti (ATRIBUTY) a své chování (operace, které daný objekt může provádět).
- Vlastnosti i operace jsou „zapouzdřené“ v jednotlivých objektech.
- IS je chápán jako MNOŽINA spolupracujících OBJEKTŮ.
- Každý OBJEKT je schopen reagovat na události, které na něj působí jako IMPULS.

# Objektově orientovaný přístup




# Objektově orientovaný přístup

- umožňuje lepší využití kódu než knihovny procedur.
- možnost znovupoužití.
- lze využít prostředky pro okenní rozhraní systémů.
- dávkové úlohy jsou interaktivní (objekt reaguje na více podnětů)

## Objekt

Struktura objektu:

1) VNITŘNÍ PAMĚŤ
2) METODY OBJEKTU
3) JINÉ OBJEKTY
4) SCHOPNOST PŘIJMOUT A ZPRACOVAT ZPRÁVU ZVNĚJŠKU

## Objektově orientované modelování

- způsob nazírání na IS pomocí abstrakce reálného světa. Základním stavebním prvkem je objekt (entita), která v sobě zahrnuje jak datovou strukturu popisující určitou entitu, tak i pravidla chování této entity.

# Vizuální modelování (OOM)

**Vlastnosti**

- "Informace v obrázcích"
- Prostředek komunikace (pohledy, modely, diagramy)
- Zachycuje obchodní procesy, věcnou problematiku, architekturu systému
- Je podporováno silnou notací a objektovými metodologiemi

**Metodologie**

- **Booch**, Coad, Yourdon
- OMT (Object Modeling Technik), **Rumbaugh**
- OOSE (Object-Oriented Software Technik)
- **Jacobson** (scénáře)
- EFEM (Extrémní efektivní modelování)

# Vizuální modelování (OOM)

- Modely vytváříme pro pochopení komplexnosti systému jako celku.

Za pomocí modelů lze:

- vizualizovat stávající nebo vytvářený systém,
- specifikovat strukturu nebo chování systému,
- získat základ pro konstrukci systému,
- dokumentovat rozhodnutí o systému.

# Objekty

Charakteristické vlastnosti objektů:

- **Jedinečnost** – znamená, že každý objekt je rozlišitelnou entitou, i když má jinak stejné kvalitativní i kvantitativní charakteristiky. K rozlišení se používá identifikační číslo.
- **Zatřiditelnost** – objekty se stejnou datovou strukturou (atributy) a chováním (operacemi) jsou seskupovány do tříd.
- **Mnohotvárnost** – znamená, že stejně označená operace se může chovat rozdílně a dávat rozdílné výsledky pro různé třídy objektů.
- **Dědičnost** – je sdílení atributů a operací mezi třídami založenými na hierarchických vztazích. Třída může být definována poměrně široce a potom ji lze následně zjemňovat do podtříd. Každá podtřída pak zahrnuje vlastnosti své nadřazené třídy.

# Objektově orientované modelování

- Založeno na principech, které vyjadřují podstatu objektového přístupu.

**4 principy OOP:**

- **Zobecnění** – představuje zaměření na podstatné vnitřní aspekty objektů a ignoruje jeho nepodstatné vedlejší vlastnosti. To znamená, že klade důraz na to, co objekt je, a co by měl dělat, a ne na to, jak by to měl dělat.
- **Zapouzdření** – znamená oddělení externích aspektů objektu, které jsou dostupné ostatním objektům, od interních implementačních detailů objektu, zakrytých ostatním objektům. To znamená, že implementace objektů může být změněna, aniž by se to dotklo aplikací, ze kterých je objekt použit.
- **Sdílení** – je umožněno dědičností datových struktur a chováním mezi několika podobnými třídami bez redundance. To umožňuje opakované použití již navržených a ověřených programů v dalších aplikacích.
- **Spolupůsobení** – jednoznačnost, zatřiditelnost, mnohotvárnost a dědičnost charakterizují hlavní proud objektově orientovaných jazyků. Každý z těchto aspektů může být izolován, ale dohromady působí symetricky.

## Výhody a nevýhody OOP a tradiční analýzy

Výhodou tradičních popisů IS oproti OOP bylo, že jejich osvojení bylo relativně snadnější, neboť jejich rozsah od zadání softwarové úlohy až po bezprostřední zadání programových dat a procedur byl řešen nezávisle na pochopení funkcí reálného podnikového systému jako celku a jeho potřeb nejen informačních.

## Nevýhody objektového přístupu

- Zavedení nového vyjadřovacího prostředku a nového způsobu myšlení, což je dosti náročné pro lidi, zúčastněné na procesu modelování, a později pak i na programování.
- Používání počítačové podpory pomocí vhodného CASE, bez níž nelze provést kvalitní analýzu i relativně malého systému. To opět vyžaduje zvládnutí práci s daným CASE (Computer Aided System Engineering).
- Technologickou kázeň s tím nutně spojenou.

## Výhody OOP

- Možnost vysledovat na modelu vnitřní vztahy prvků v podnikovém systému a vytvářet software pro „poskytovatele služeb“, tj. pro prvky podniku, které poskytují své služby jiným podnikovým prvkům.
- Takto vytvářený software může být opakovatelný, zejména při použití knihoven standardizovaných tříd.
- Možnost vytvářet software podstatně pružnější vzhledem k proměnlivým potřebám podniku.
  Z hlediska programů je to dáno tím, že dříve byly strukturované programy vytvářeny podle pevné struktury potřeb podniku v době jejich vzniku, bez snadné možnosti změn.

## Historie OOP

- O OOM se začíná hovořit v 2. pol. 80. let 20. stol.
- Na počátku 90. let již existovala řada metod, nástrojů a technik založených na OOP.
- Snaha o integraci jednotlivých metod a o unifikaci OOP → Grady Booch & James Rumbaugh publikovali v r. 95 metodu s názvem Unified Method (vycházela z metod Booch a OMT).

## Historie OOP

- V r. 95 se k autorům Unified Method Jacobson (autor metody Objectory/OOSE).
- Vznikají další verze Unified Method.
- Ukazuje se, že se jen těžko podaří prosadit jednotné postupy a doporučení a ucelenou OO metodu vývoje IS.

## Historie OOP - UML

- Přejmenováním Unified Method na Unified Modeling Language (UML) vznikl nástroj – grafický jazyk pro tvorbu modelů IS použitím OOP.
- UML byl přijat sdružením OMG (Object Management Group – sdružení usilující o unifikovaný rozvoj OOp) jako doporučený standard notace pro tvorbu modelů IS.

## UML

- V současnosti je UML nejrozšířenější objektovou notací.
- Zahrnuje několik druhů modelů (diagramů) IS a grafických vyjadřovacích prostředků pro jejich konstrukci.
- UML dnes podporují všechny CASE nástroje pro objektovou analýzu a návrh IS, OOM začleňují UML mezi své výrazové prostředky.

## UML

V současnosti UML poskytuje:

- pravidla pro pojmenování, rozsah platnosti, rozsah viditelnosti, omezení, prezentaci modelu,
- různé specifikace,
- rozšiřitelnost jako jsou stereotypy, dodané hodnoty.

# UML

**Definice:**

*UML je standardní jazyk pro vizualizaci, specifikaci, konstrukci a dokumentaci prvků projektu, ve kterém hraje významnou roli vývoj software.*

**Stavební kameny:**

- artefakty (prvky),
- vztahy,
- diagramy.

# UML – skupiny ARTEFAKTŮ

**Týkající se struktury systému**

- tvoří statickou část modelu,
- třídy, rozhraní, use case, komponenta.

**Týkající se chování systému**

- tvoří dynamickou část modelu,
- interakce, stavy, aktivity.

**Týkající se organizace systému**

- package.

**Týkající se vysvětlení účelu**

- popis, anotace, poznámka.

# UML - VZTAHY

**Agregace (závislost)**

- jeden prvek závisí na druhém.

**Asociace**

- propojení prvků.

**Dědičnost**

- specializace/generalizace

# UML - DIAGRAMY

**Grafická reprezentace obsahu modelu**

- zachycení prvků a jejich vztahů.

**Pohled na systém z různých perspektiv.**

**Různé typy diagramů**

- Diagram užití (use case), tříd, objektů, sekvenční, spolupráce, stavový, aktivit, komponent, nasazení, balíčků (package).

# UML - DIAGRAMY

1. Diagram Use case
2. Sekvenční diagram
3. Diagram spolupráce
4. Diagram tříd
5. Stavový diagram
6. Diagram aktivit
7. Diagram nasazení

# Diagram Use case

- jeho vypracování je obsahem use case analýzy, zachycuje funkcionalitu systému z pohledu uživatele, popisuje chování systému z hlediska uživatele.

**Prvky diagramu use case:**

**Aktor** (vymezením aktorů specifikujeme okolí systému a vymezíme jeho hranice).
**Use case** (případ užití, typ jednání, funkcionalita,systému, kterou využívá aktor),
**Vztahy** (mezi aktorem a use case, mezi use case, výjimečně mezi aktory).

## Diagram Use case

**Aktor** = kdokoliv nebo cokoliv mimo systém, kdo nějak komunikuje a interaguje se systémem.

- zachycení okolí systému,
- prvky aktivně komunikující se systémem:
  - uživatelé
  - jiné softwarové systémy
  - čas

## Diagram Use case

**Use case** (typ jednání, případ užití)

jakákoliv funkčnost, která dává měřitelnou hodnotu uživatelům tohoto systému.

reprezentuje ucelenou funkcionalitu problémové domény

## Diagram Use case

**Typy vztahů v UCD:**

1) Vztah mezi AKTOREM a USE CASE – aktor komunikuje se systémem (vyvolává a účastní se USE CASE)
2) Vztahy mezi USE CASE

Use case mohou vzájemně spolupracovat (slouží pro zjednodušení modelu a podobá se dekompozici).

## Vztahy mezi USE CASE

Dva typy vztahů mezi use case:

- **Vazba include (dříve uses)**

povinný vztah, oba spojené use case se musí povinně provést, použijeme tam, kde se část use case v navrhovaném systému může opakovat.

- **Vazba extends**

rozšiřující vztah, za jistých podmínek se vykonávají oba use case, použijeme pro volitelné chování, pro chování za specifických podmínek, pro chování podle volby aktora.

# Popis USE CASE - scénář

- Stručná charakteristika (1 - 3 věty)
- Scénář:
  - nutné podmínky před spuštěním,
  - nutné podmínky po ukončení,
  - tok událostí - sekvence akcí.
- Jeden scénář "HAPPY DAY" (obsahuje základní tok událostí a subtoky).
- Ostatní scénáře jsou alternativní, chybové.

# Scénář (tok událostí) pro Use Case Př.: Evidence rezervace (půjčovna CD)

**Předpoklady**

Tento Use Case začne, když člen nemůže být uspokojen, protože dané CD není momentálně na skladě, nebo daný titul není v půjčovně k dispozici.

**Hlavní tok**

Tento Use Case začíná, když člen předloží asistentovi svoje identifikační číslo a název titulu, který si chce zarezervovat. Asistent zkontroluje existenci člena v databázi členů (**A-1**), zkontroluje, zda titul existuje v databázi titulů (**A-2**) a zkontroluje, zda jsou všechny kopie daného titulu zapůjčeny (**A-3**).

Pokud má někdo kopii zapůjčenu déle než 10 dní, je upomenut o navrácení (**S-1**). Je založen záznam o rezervaci této kopie pro daného člena. Je vytištěn doklad o rezervaci titulu.

**Subtoky**

S-1: Asistent vyhledá všechny členy, který mají půjčený daný titul a zkontroluje délku jejich půjčky. Pro ty, kteří mají půjčku delší než 10 dnů, vytiskne upomínku.

**Alternativní toky**

A-1 : Je vloženo špatné ID člena, nebo člen neexistuje. Asistent může opakovat vstup ID nebo vložit údaje o členu (bude řešeno v Use Case Přidání nového člena), nebo ukončit Use Case.

A-2 : Je vložen špatný titul, nebo titul neexistuje v půjčovně. Asistent ukončí Use Case (není založena rezervace) a vytiskne objednávku na daný titul (Use Case Objednání materiálu).

A-3 : Asistent zjistí kdo má půjčené kopie a vloží rezervaci pro člena.

# Use case diagram

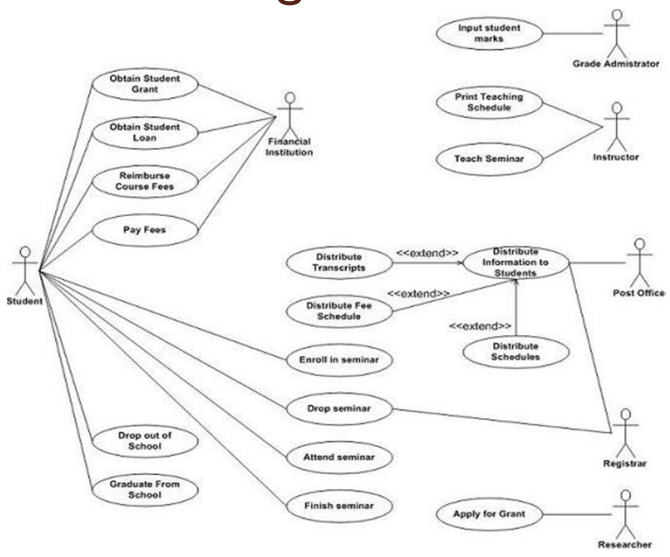
**Informace o use case:**

- jde o vizualizaci funkcionality, další informace musím spravovat v textové podobě,
- use case musí mít tyto náležitosti (jméno, popis – účel, kdo a co jej používá, související use case, hlavní a alternativní scénář, nepovinně poznámky).

**Význam use case diagramu**

- zachycení požadavků na systém,
- vizualizace a organizace požadavků ve standardní formě,
- pro nalezení objektů, tříd a zodpovědností z popisu scénářů.

# Use case diagram



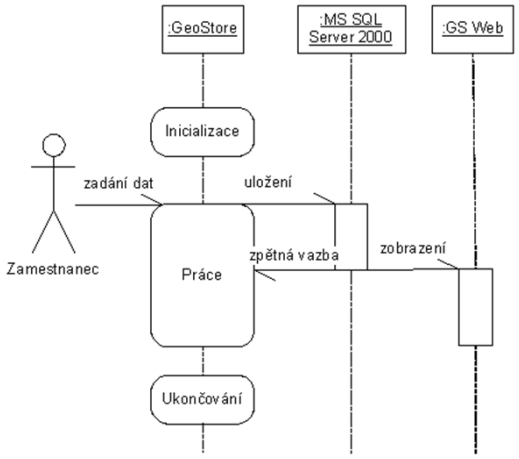
# Sekvenční diagram UML

- Zachycuje interakci mezi objekty, zachycuje zasílání zpráv mezi objekty v rámci systému.
- Zachycuje dynamické chování s orientací na čas.

**Vlastnosti sekvenčního diagramu:**

- Objekty sekvenčního diagramu spolu komunikují pomocí zasílání zpráv.
- Popisuje jeden průchod zpráv systémem.
- Nemá přímé výrazové prostředky pro smyčky, větvení a podmínky.
- Pro jednoduché případy použiji poznámky.
- Složité případy řeším separátními diagramy.

# Sekvenční diagram

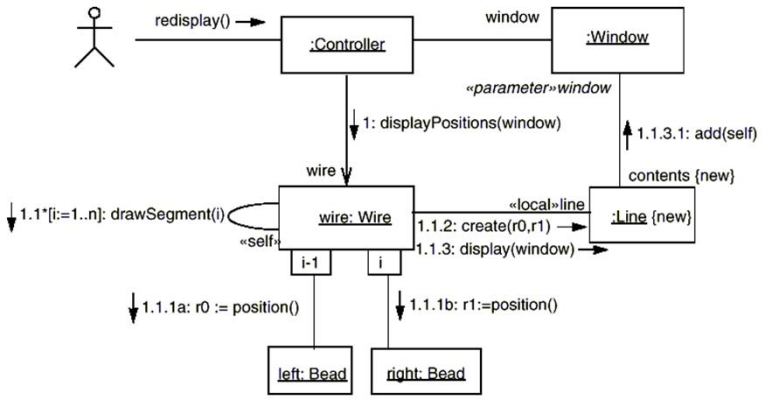


# Diagram spolupráce UML

Vlastnosti diagramu spolupráce:

- Ukazuje tytéž informace jako sekvenční diagram s ohledem ne na čas, ale na propojení mezi objekty.
- Slouží pro pozdější určení vztahů mezi objekty:
  - datové vztahy - dány distribucí informací,
  - komunikační vztahy - dány spoluprací mezi objekty

# Diagram spolupráce UML



## Diagram tříd UML

- Třída je abstrakce objektů, které mají společné chování a o kterých nás zajímají stejné informace.
- V OOP je to šablona pro instance objektů.
- Statický pohled na modelovaný systém.
- Vytváří se v etapě analýzy a postupně se zpřesňuje, je základem pro implementaci a nástrojem pro dokumentaci.

## Diagram tříd UML - Třída

Třídy vyhledáváme analýzou problémové domény (podstatná jména ze scénářů).

Třída je zapouzdřením určitého chování a určitých informací.

Zapouzdření je koncept, který dává ve třídě dohromady to, co spolu souvisí a dává nějaký smysl.

Obsahem třídy je:

**Jméno,**
**Atributy,**
**Operace.**

# Diagram tříd UML –Operace, atributy

**Operace třídy**

- Operace, které třída definuje, představují její chování nebo také zprávy, kterým třída rozumí.
- Zdrojem pro hledání operací jsou především scénáře use case analýzy.

**Atributy třídy**

- Atributy třídy jsou informace, které o třídě uchováváme.
- Zdrojem pro atributy třídy jsou věcné znalosti o dané problematice a analýza podrobných požadavků uživatelů.
- Atributy třídy by měly být atomické a nedělitelné.

# Vztahy mezi třídami

Třídy nejsou v systému osamocené, jejich objekty ke svému chování potřebují využít schopností jiných objektů. Třídy mezi sebou sdílí informace.

**Asociace** ("Slabá" vazba mezi třídami), např. čtenář a kniha

- Neříká nic jiného, než to, že dvě třídy mají mezi sebou vztah, tedy že o sobě vědí.
- Defaultně obousměrná vazba.
- Může být definováno jméno asociace, role a násobnost (kolik instancí třídy existuje vůči jiné třídě).

**Agregace** (volná vazba mezi třídami), např. počítač a periferní zařízení

- Představuje vztah skládání celku z částí, celek odpovídá za vytvoření a zrušení částí, je to vztah celku k jedné části, definujeme násobnost, jméno a role ne.

# Vztahy mezi třídami

**Kompozice** (nejsilnější forma asociace, velmi pevná vazba mezi třídami), např. faktura a řádek faktury, třída se skládá z jiných závislých tříd

- Třídy tvoří hierarchii

**Dědičnost** (nejsilnější forma vazby mezi dvěma a více třídami).

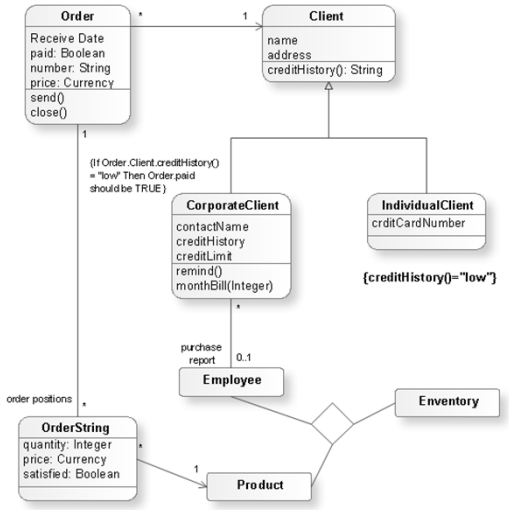
- Generalizace/specializace
- Potomek dědí celou specifikaci svých předků (atributy i operace).
- Viditelnost prvků určuje, jak jsou děděny (public a protected jsou v potomkovi přístupné, private ne).
- Vztah mezi třídou a speciálním případem této třídy.
- Rozlišuje, co je stejné a co jiné.

# Vztahy mezi třídami

- **Rekurzivní asociace** (asociace na sebe sama).

- **Asociativní třída** (atributy asociace)
  - Pokud sama asociace nese určité informace, které nemohou být atributy ani jedné z asociovaných tříd.
  - Hovoříme o „link class“.
  - Může mít atributy i operace.
  - Např. vztah mezi osobou (jméno, rcislo), firmou (název) je vazba s asociativní třídou pracovní poměr (datum nástupu, funkce, plat).

# Diagram tříd



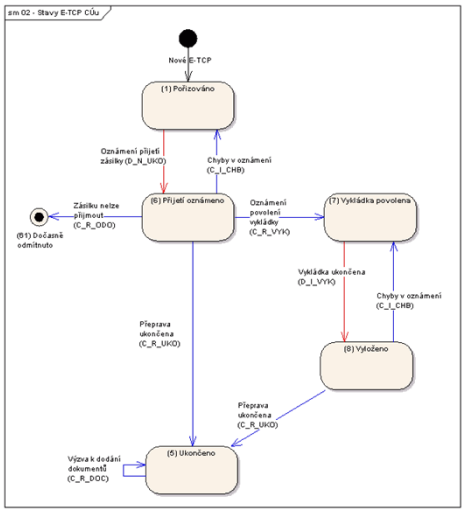

# Stavový diagram UML

Dynamické chování systému je modelováno pomocí diagramů aktivit i stavových diagramů

**Stavový diagram**

- Používá se k modelování životního cyklu jednoho objektu. Hovoříme o objektech s výrazným dynamickým chováním nověji **reaktivní objekty.**
- Stavový diagram modeluje chování systému **napříč všemi use casy.** Znázorňuje, jak se stavy objektu mění v závislosti na událostech, které se ho dotýkají
- Stav objektu je dán hodnotami jeho atributů.
- Stav objektu může ovlivňovat jeho chování.
- Stav objektu je zachycen na stavovém diagramu jako stav jednoho objektu jedné třídy bez vazeb na jiné objekty nebo jiné třídy.

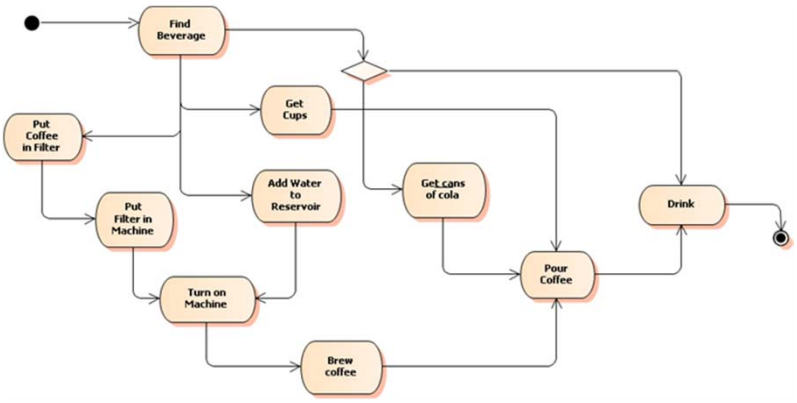
# Stavový diagram




# Diagram aktivit UML

- Použití pro modelování systémů pracujících v reálném čase, systémů pro řízení technologických procesů, nebo paralelních procesů a jejich synchronizaci.
- Další použití pro znázornění složitého scénáře a doplnění sekvenčního diagramu.
- Jsou zvláštním případem stavových diagramů, kde stavy jsou vyjádřeny jako akce a kde přechody jsou spouštěny automaticky po ukončení předchozích akcí nebo aktivit. Používají obvykle pouze malou podmnožinu bohaté syntaxe stavových diagramů UML.
- Lze používat symbolů rozhodování (tzv. hodnocení přechodů), symbolů rozvětvení (jeden vstup několik výstupů), spojení (více vstupů jeden výstup), plavecké dráhy – swimlanes pro specifikace osob, oddělení nebo tříd zodpovědných za aktivitu.

# Diagram aktivit




# Diagram nasazení

- Etapa Nasazení (implementační etapa) je proces přiřazení artefaktů (soubory, skripty, DB tabulky, modely UML) uzlům (např. PC, server, prostředí zpracování).
- Diagram nasazení umožňuje modelovat distribuci SW systému na fyzickém HW. Ukazuje fyzický hardware na němž bude softwarový systém (komponenta) spuštěn a také způsob, jak je SW na tomto HW nasazen.

**Diagram nasazení ukazuje:**

- **Uzly** – typy HW, na nichž bude systém spuštěn (osobní PC, server).
- **Relace** – typy spojení mezi uzly (komunikační kanál, který slouží k přenosu informací, např. HTTP).
- **Komponenty** – typy komponent nasazených na určité uzly (modul IS, MS Word).

# Diagram nasazení