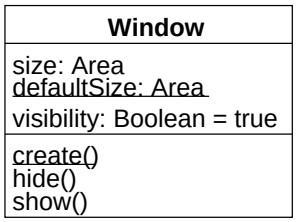
KIV/ZSWI 2003/2004
Přednáška 5

Diagramy tříd (pokračování)

Poznámka (atribut patřící třídě)

* atributy a operace patřící třídě (nikoli instanci třídy) se znázorňují podtržením
  - odpovídají statickým atributům a metodám v jazycích C++, Java a C#.



[]

Poznámka (názvy asociací a názvy rolí)

* názvy rolí podstatná jména, např. zaměstnavatel, manager, zaměstnanec, adresa pro fakturaci apod.
  - před jméno role můžeme připojit indikátor viditelnosti, většinou se používá '+' (asociace je ve směru k roli viditelná)
* názvy asociací bývají slovesa, např. zaměstnává, řídí, pracuje pro apod.
  - asociace je třeba nazývat konkrétně a vyhýbat se příliš obecným názvům jako např. má, je součástí apod. (lepší je "učitel učí předmět" než "učitel má předmět")
  - neumíme-li asociaci rozumně pojmenovat, můžeme totéž vyjádřit přiměřeným jménem role (asociaci pak nemusíme pojmenovávat)
    . například místo "počítač má monitor" pojmenujeme roli monitoru - monitor je "zobrazovací zařízení"

* pokud jsou mezi stejnými třídami dvě asociace, znamená to, že objekt dané třídy může být pomocí každé z nich spojen s jiným objektem (instance asociace se nazývá "link" - uvidíme je např. v diagramech objektů)
  - pak je musíme rozlišit asociace rozlišit názvem asociace nebo názvem role

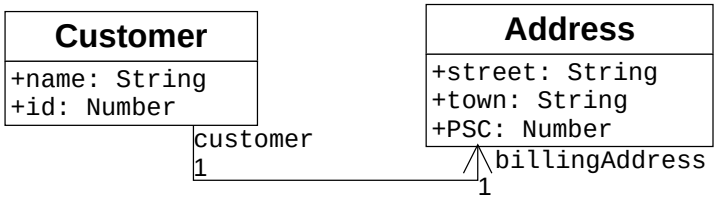
[]

Uspořádání

* pokud je násobnost větší než 1, pak množina prvků může být neuspořádaná nebo uspořádaná
  - není-li v diagramu uvedeno jinak, je množina neuspořádaná
  - pokud je uspořádaná, uvádíme to klíčovým slovem {ordered}

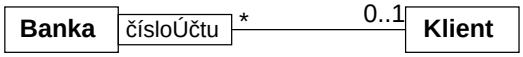
Průchodnost

* průchodnost (navigability) říká, že se pomocí asociace můžeme dostat z dané třídy do cíle
  - znázorňuje se šipkou směřující k cíli
  - pro větší přehlednost se v CASE nástrojích pro asociace s průchodností oběma směry často potlačuje zobrazení šipek (tj. šipky se zobrazují pouze pro asociace s průchodností jedním směrem)



Kvalifikované asociace
.....

* kvalifikátor je jeden nebo více atributů, jejichž hodnoty slouží pro určení množiny instancí, se kterou je objekt sdružen pomocí asociace
 - kvalifikátor musí rozdělovat instance do disjunktních množin
 - v UML se zobrazuje jako malý obdélníček připojený ke konci asociace, je umístěn u zdrojového konce asociace (kvalifikátory nepatří ke třídě, ale k asociaci)
  . atributy kvalifikátoru se znázorňují uvnitř obdélníčku (ve stejném tvaru jako atributy třídy)
  . násobnost umístěná u cílového konce asociace značí, jak velká bude množina cílových instancí (možné kardinality)

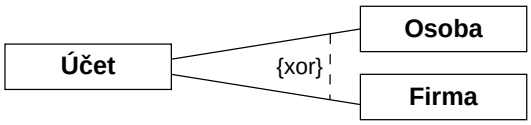


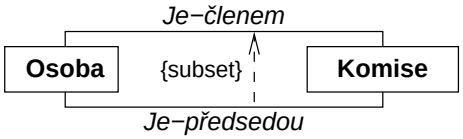
Pokud je na jenom konci asociace více symbolů (tzv. ozdob), např. označení pro kompozici i pro průchodnost, zobrazují se v tomto pořadí: navigační šipka, symbol agregace/kompozice, kvalifikátor.

Omezení
.....

* omezení (constraint) je sémantický vztah mezi prvky, který musí být splněn aby byl model platný (tj. musí ho splňovat každá implementace modelu)
 - v UML můžeme specifikovat pomocí podmínky zapsané v {}
 - může se týkat libovolného elementu, např. třídy, asociace, atributu třídy
 - pokud se týká jednoho prvku, kreslí se u prvku
 - pokud se týká dvou prvků, přerušovaná šipka nebo čára spojující prvky

* například následující asociace má omezení {xor}, tj. bankovní účet může být buď osobní nebo firemní





Poznámka (properties)

 Pro zmatení veřejnosti se v UML stejným způsobem znázorňují i libovolné další vlastnosti, které nemají grafické vyjádření.

 Například operace může mít vlastnost {query} značící, že operace nemění stav systému nebo {concurrency=cuncurrent} značící paralelní vykonávání. Konce asociace mohou mít vyznačeno např. {frozen}, tj. po vytvoření a inicializaci objektu u konce asociace nebude možné přidávat, rušit nebo měnit spojení tvořící tuto asociaci.

[]

Praktická poznámka (diagram tříd)

 Při návrhu systému obvykle vytváříme více diagramů tříd, každý z nich zachycuje jeden aspekt statického pohledu na systém - obsahuje pouze prvky potřebné pro pochopení příslušného aspektu. Každý diagram by měl proto mít

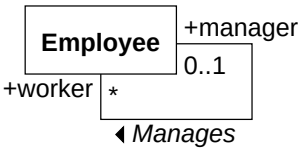
jméno, které vypovídá o jeho účelu.

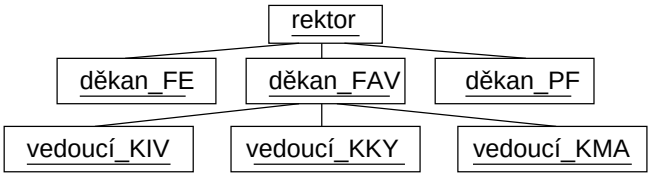
Úplný statický pohled na systém je tedy tvořen všemi diagramy tříd společně.

[]

Diagramy objektů
.....

* instance místo tříd, ukazuje stav systému v daném časovém okamžiku
* používá se poměrně zřídka, vhodné pro vysvětlení malých částí systému se složitými (zejména rekurzivními) vztahy





* instancí asociace je ”link” (propojení)
 - link je n-tice (nejčastěji dvojice) odkazů na objekty
 - znázorňuje se čarou (podobně jako asociace)

Poznámka pro zajímavost (UML a diagramy objektů)

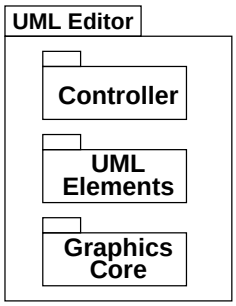
V diagramech tříd je dovoleno uvádět objekty a jejich vztahy (používá se především pro uvedení příkladů datových struktur, viz výše). Z hlediska UML je diagram objektů takový diagram tříd, ve kterém nejsou uvedeny žádné třídy ale pouze instance (tj. neexistuje samostatný typ diagramu ”diagram tříd”).

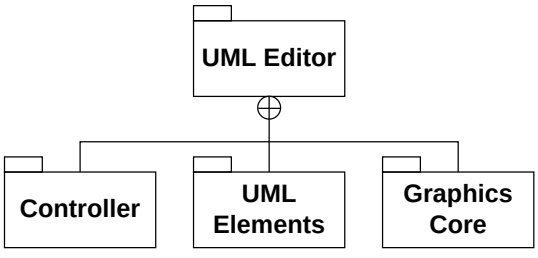
[]

Balíčky (packages)
.....

* slouží pro zjednodušení složitých diagramů
 - sdružují množinu libovolných prvků diagramu (uvnitř balíčků mohou být další balíčky)
 - každý prvek může být nejvýše v jednom balíčku

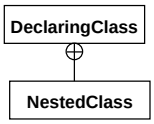
* zakreslují se jako obdélníky s malým držátkem na levé horní straně
 - prvky obsažené v balíčku lze kreslit buď do balíčku nebo je možné nakreslit strom obsahu balíčku
 - viditelnost prvků vně balíčku je možné označit obvyklým způsobem (např. ’+’ pro ”public”)





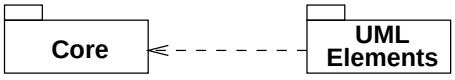
Poznámka pro zajímavost (označení vnořených podtříd)

V diagramu tříd pro označení vnořených podtříd používá stejná se notace jako pro označení stromu obsahu balíčku, tj. kroužek s křížkem:



[]

* závislosti mezi balíčky je možné znázornit přerušovanou čarou s otevřenou šipkou (A závisí na B jako šipku od A k B)
 - balíček A závisí na balíčku B pokud změny v B mohou způsobit změny v A v prvním
 - notace pro závislost se používá i jinde, např. třída závisí na rozhraní



* speciálním druhem balíčku z hlediska UML je podsystém
 - podsystém úplně zapouzdřuje svůj obsah (tj. své chování zpřístupňuje pouze prostřednictvím rozhraní podsystému)
 - tj. dokud zůstane stejné rozhraní, obsah podsystému se může libovolně měnit
 - znázorňuje se jako balíček, v pravém horním rohu velkého obdélníka je ”vidlička”

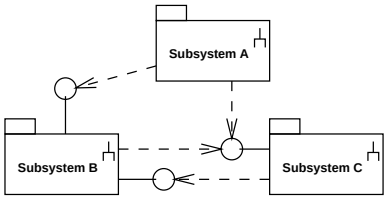
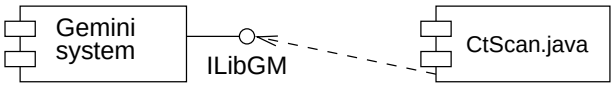


Diagram komponent
---

* diagram komponent je fyzická analogie diagramu tříd
 - komponenta zapouzdřuje implementaci a zveřejňuje množinu rozhraní
 - komponenta může být implementována např. jedním nebo více spustitelnými soubory, zdrojovými texty nebo objektovými moduly, knihovnami, příkazovými soubory apod.

* komponenta se znázorňuje jako obdélník, po jeho straně dva menší obdélníčky
 - prvky se znázorňují umístěné uvnitř komponenty
 - diagram komponent znázorňuje také závislosti komponent (přerušovaná šipka od závislého obvykle k rozhraní jiné komponenty)



Základní rozdíl mezi balíčkem a komponentou: balíček je čistě koncepční mechanismus pro organizaci modelů v UML, zatímco komponenty existují skutečně.

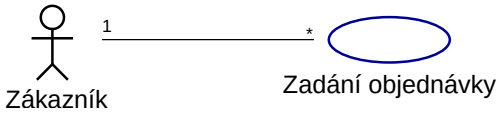
Nyní přejdeme od statického popisu systému k diagramům popisujícím chování systému nebo jeho částí. Nebudu samozřejmě popisovat všechny typy diagramů (na to je UML příliš rozsáhlé), ale popíšu nejdůležitější diagramy pro analýzu a návrh.

Diagramy případů použití
---

* popisují co systém dělá z hlediska vnějšího pozorovatele (nikoli jak to

dělá - k tomu slouží jiné typy diagramů, např. stavový diagram, viz dále)
* všechny diagramy případů použití obsahují

- aktéry (nejčastěji znázorněni jako panáčci, ale používají se i jiné ikony)
  . aktér = cokoli co potřebuje komunikovat se systémem
  . představuje roli nebo množinu rolí uživatele při komunikaci s entitou
- případy použití (znázorněny elipsou)
  . představují dialog nebo transakci vykonávanou systémem, podsystémem nebo třídou
- asociace (čáry mezi aktéry a případy použití znázorňující komunikaci)
  . spojuje aktéry s případy použití
  . konce asociace mohou mít označení násobnosti

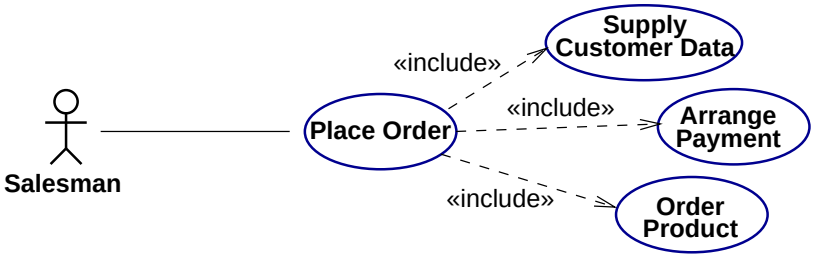


* případy použití mohou být spojeny se scénáři
- scénář = příklad co se stane pokud někdo komunikuje se systémem
- scénář je posloupnost kroků (v diagramu je nezobrazujeme)
- popisuje se obyčejným textem, stavovým automatem apod.
  . např. bankomat - aktér vloží kartu, zadá PIN, zadá typ operace, zadá částku, bankomat předá peníze, předá částku, vrátí kartu

* diagram případů použití může dále obsahovat
- zobecnění případů použití a aktérů
- vztahy "include" (zahrnout, vložit) a "extend" (rozšířit)
- balíčky sdružující prvky modelu do větších celků
- poznámky

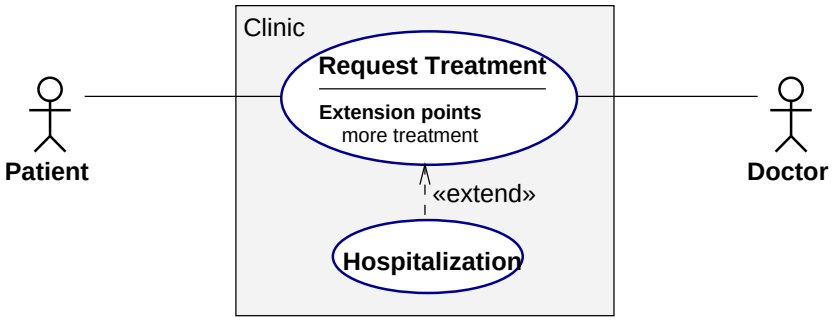
* zobecnění případů použití a aktérů (generalization)
- ukazuje že jeden případ použití nebo aktér je zobecněním jiného (podobně jako rodičovská třída je zobecněním svých potomků)
- znázorňuje se šipkou s trojúhelníkovou hlavou

* vztah "zahrnout, vložit" (include)
- rozložení případu použití na části, užitečné pokud stejnou část můžeme použít ve více případech použití (např. "převod částky" může být součástí případu použití "platba za službu" i "platba za zboží")
- vkládaný případ nemůže existovat samostatně, je vždy součástí jiného
- notace přerušovaná čára označená klíčovým slovem <<include>>



* vztah "rozšířit" (extend)
- označuje že bázový případ použití může být upraven podle obsahu jiného, tím vznikne varianta (rozšíření) bázového případu
- používá se pokud jsou části případu použití volitelné, používají se zřídka apod.
- notace přerušovaná čára označená <<extend>>, šipka k bázovému případu použití

. v bázovém případu použití se uvedou ”body rozšíření”, tj. kdy se použije rozšířený případ
. ”body rozšíření” mají nejčastěji podobu obyčejného textu



* někdy se v diagramu uvádí hranice systému = obdélník oddělující systém od externích aktérů, název systému se uvádí uvnitř obdélníku

* diagramy případů použití se používají:
- pro sběr požadavků
- pro komunikaci s klienty - jsou jim srozumitelné
- pro generování testovacích případů - množina scénářů spojená s případem použití může být odrazovým můstkem pro vytvoření testovacích případů pro scénáře

* nejběžnější modely:
- model kontextu systému nebo podsystému, tj. určení co leží uvnitř systému (znázorňujeme ohraničením systému, viz výše), popis aktérů a jejich rolí
- modelování požadavků - popis kontraktu mezi systémem a aktéry

Poznámka (zápočtová úloha)

V rámci zápočtové úlohy použijete diagram případů použití minimálně pro znázornění kontextu vytvářeného systému.

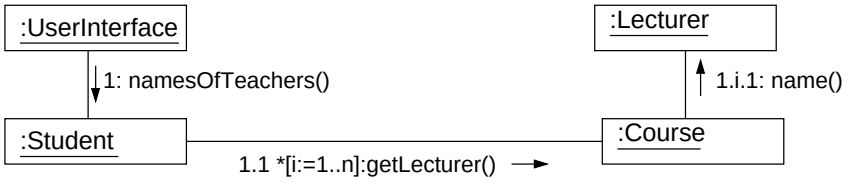
[]

Diagramy spolupráce
---

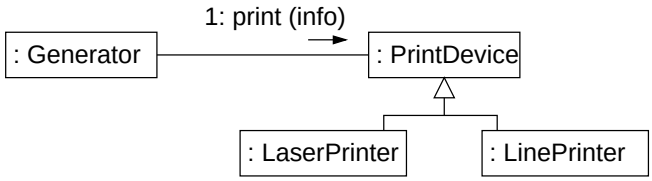
* diagramy spolupráce (collaboration diagrams) popisují spolupráci mezi objekty nebo jejich rolemi
- mohou být připojeny např. k případu použití jako jeho popis
. popisují které objekty nabízejí chování popisované případem použití
. jak objekty případ použití vykonávají
- mohou mít i vyšší úroveň podrobnosti, lze je použít např. pro popis chování operací třídy apod.

* v diagramu spolupráce se vyskytují
- objekty účastnící se interakce, případně role objektů v interakci
- spojení pro přenos zprávy - kreslí se jako plná čára
. často se kreslí s šipkou ukazující průchodnost (v příkladech jí neuvádím)
- zprávy (stimuly) - kreslí se jako šipečka blízko čáry
. šipečka s plnou hlavou znamená volání procedury, volající pokračuje až po návratu z procedury

- každá zpráva má pořadové číslo, končící dvojtečkou
. zprávy na nejvyšší úrovni jsou číslovány postupně 1, 2, 3 atd.
. zprávy na další úrovni vnoření během stejného volání 1.1, 1.2, 1.3 atd.
- za pořadovým číslem může být uvedeno: zpráva(argumenty) nebo případně návratová_hodnota := zpráva(argumenty)
. popisuje zaslanou zprávu, její argumenty a návratovou hodnotu




```
* v diagramu spolupráce lze lze znázornit také iterace a podmínky
 - iterace - jako pořadí zprávy uvedeme hvězdičku (pokud nechceme uvádět
   podrobnosti) nebo např. *[i := 1..n] (pokud chceme iteraci popsat)
 - podmíněná zpráva - jako prefix uvedeme podmínku. např. [x>0]
  . co má být uvnitř hranatých závorek UML nespecifikuje; obvykle se používá
    pseudokód nebo syntaxe cílového programovacího jazyka
  . například: 4[x<0]: display(x)
 - zprávy je možné také "číslovat" identifikátorem

* obdélníčky v diagramech spolupráce jsou buď objekty (název je podtržený)
  nebo role (název není podtržený)
 - plný zápis objektu "objekt/role : Třída", plný zápis role "/role : Třída"
* lze kombinovat se vztahy z diagramu tříd, např. zobecnění, agregace apod.
```

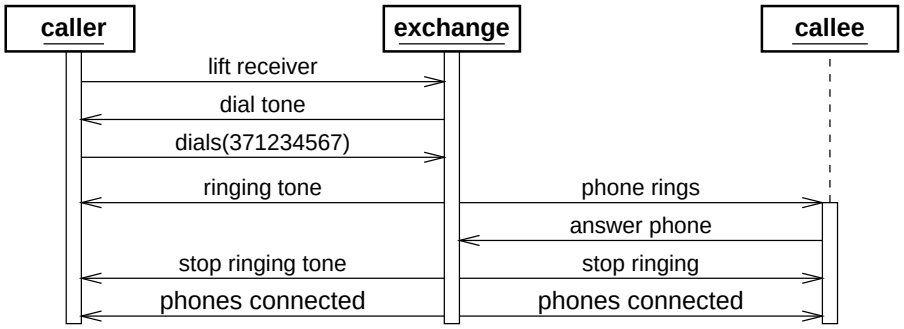



```
Pomocí diagramů spolupráce se obvykle znázorňují jednoduché posloupnosti
zpráv. Pro složitější chování se obvykle používají (významově ekvivalentní)
sekvenční diagramy, ve kterých se lépe zobrazují časové závislosti.


Sekvenční diagramy
------------------

* sekvenční diagramy (sequence diagrams)
 - popisují stejnou informaci jako diagramy spolupráce, ale soustředí se na
   časové závislosti
 - v některých metodikách (např. OOSE) se k případům použití vytvářejí
   sekvenční diagramy místo diagramů spolupráce

* sekvenční diagramy znázorňují aktéry, objekty v systému se kterými
  interagují a posloupnost vyměněných zpráv
 - čas plyne shora dolů, pro každý objekt svislá přerušovaná "čára života"
   (lifeline) reprezentující čas kdy objekt existuje a hraje určitou roli,
   doba aktivity objektu se znázorňuje úzkým obdélníkem
 - horizontální uspořádání není podstatné a má být zvoleno s ohledem na
   srozumitelnost diagramu
 - vlevo od diagramu sloupec popisující interakci s okolním světem
   ("hranice systému")
```

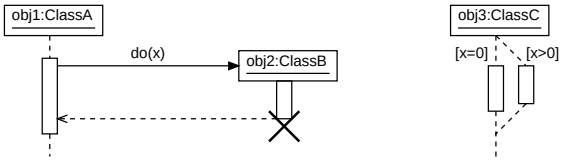

```
* šipky podle typu komunikace:
 - volání procedury nebo jiný vnořený tok řízení (tj. vnější sekvence
   může pokračovat až po dokončení vnitřní sekvence) - plná šipka, viz (a)
 - asynchronní komunikace - šipka tvořená čarami, viz (b)
 - návrat z procedury - přerušovaná šipka, viz (c)
  . pokud řízení toku procedurální (viz "návrh mechanismu řízení"), můžeme
    vynechat šipku pro návrat z procedury
```

- stejné tři typy šipek lze použít i v diagramech spolupráce
- horizontální šipka = atomické zaslání zprávy
- šipka směřující šikmo dolů = během zaslání zprávy může nastat další událost, např. zaslání zprávy opačným směrem
- vlevo od diagramu bývá posloupnost akcí okomentována např. pseudokódem

* příklad - dva telefonní účastníci a ústředna:



* další možnosti (uvádím pouze pro zajímavost)
- zpráva může zapříčinit vznik nebo zánik objektu (šipka směřuje k symbolu objektu resp. symbolu zániku objektu X); objekt může provést autodestrukci
- neexistuje-li objekt po celou dobu pokrytou diagramem, čára života začíná a končí na místě vzniku a zániku objektu (objekt se kreslí na její začátek)
- čára života se může rozdělit na dvě pro znázornění podmínky (podmínka se uvádí v hranatých závorkách), čáry se později mohou opět spojit
- podmínkou se mohou označovat i zprávy



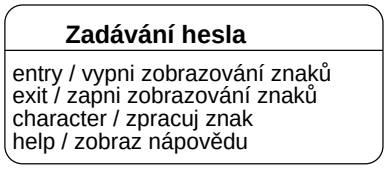
Zatímco sekvenční diagramy se používají pro vyjádření časových závislostí, diagramy spolupráce slouží spíše pro znázornění struktury systému a vztahu mezi instancemi.

Stavové diagramy

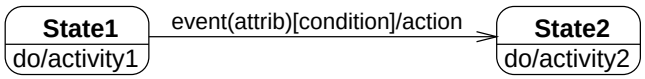
* typicky se používají pro popis chování instance třídy, někdy také pro případy použití nebo pro celý systém nebo podsystém
- popisují možné posloupnosti stavů, kterými objekt prochází v důsledku reakcí na události (událostí může být např. vyvolání operace, uplynutí času apod.)
- oproti klasickým ”plochým” stavovým diagramům umožňují stavové diagramy v UML strukturování (které nebudu příliš podrobně popisovat, ale pro rozsáhlejší diagramy je nutné)

* stav = situace během života objektu, kdy objekt splňuje nějakou podmínku, provádí nějakou akci nebo čeká na událost
* událost = výskyt stimulu, který může spustit přechod do jiného stavu
* přechod = změna stavu způsobená událostí; nový stav závisí na původním stavu a na události

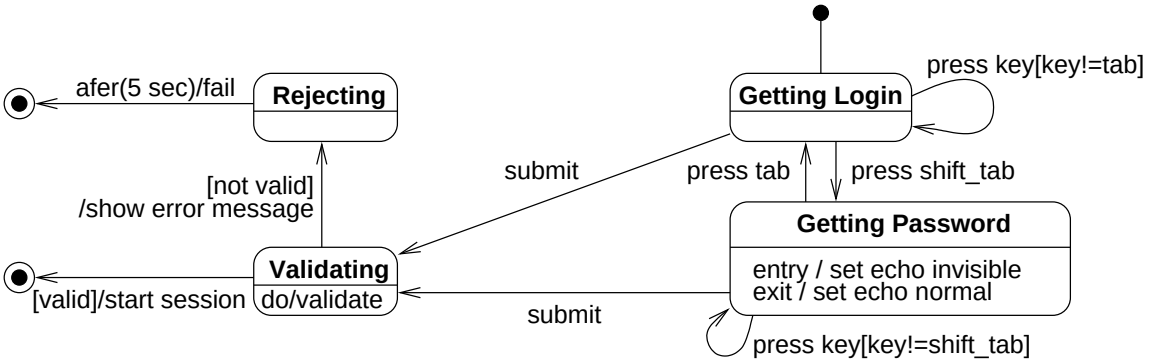
* stavový diagram je graf, kde
- uzly grafu = stavy, znázorněny jako obdélníky s kulatými rohy, uvnitř volitelně část s názvem stavu a s posloupností akcí
  . entry/akce - akce prováděná při vstupu do stavu
  . do/akce - akce prováděná během stavu
  . exit/akce - akce prováděná při opuštění stavu
  . mohou být uvedeny další uživatelem definované akce



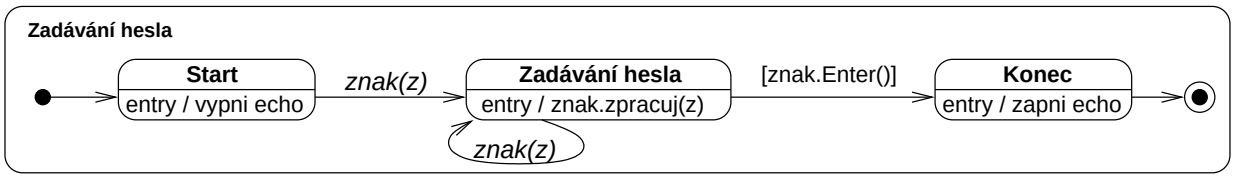

```
 - orientované hrany grafu = přechody mezi stavy
  . popis hrany = událost která způsobí přechod a akce provedené jako
    důsledek přechodu ve tvaru: událost/akce, případně událost[podmínka]/akce
  . událost má tvar: jméno_události(parametry)
```




```
 - počáteční stav - znázorněn jako černé kolečko
 - koncový stav - znázorněn jako černé kolečko v kroužku (”býčí oko”)
```




```
* činnost v daném stavu lze popsat opět pomocí stavového automatu - vnoření
 - počáteční a koncový pseudostav je znázorněn stejně jako počáteční a
   koncový stav
```




```
Poznámka (volně šířený nástroj Fujaba)

 Stavové diagramy lze vytvářet i ve volně šířeném nástroji Fujaba, který z
 nich umí vygenerovat kód v jazyku Java. To se může hodit např. pro
 implementaci síťových protokolů apod.

[]
```

❇