UNIVERZITA PALACKÉHO V OLOMOUCI

PEDAGOGICKÁ FAKULTA

KATEDRA TECHNICKÉ A INFORMAČNÍ VÝCHOVY

Využití grafického programovacího jazyka UML při návrhu software školní třídní knihy

Bakalářská práce

Petr Felner

OLOMOUC 2014 Vedoucí práce: Mgr. Jan Kubrický, Ph.D

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a použil jsem pouze uvedené informační zdroje.

V Olomouci 23. 4. 2014 …….

Petr Felner

Poděkování:

Děkuji Mgr. Janu Kubrickému, Ph.D. za odborné vedení a cenné rady při vypracovávání bakalářské práce. Dále děkuji své rodině za podporu a trpělivost po dobu zpracovávání mé práce.

Obsah

Úvod

Grafický programovací jazyk Unified Modeling Language, většinou se používá zkráceně UML[1], je nástroj softwarového inženýrství. Jeho důležitost vyniká především při návrhu složitějších aplikací, které jsou založeny na objektově orientovaném programování. Objektově orientovaný přístup má oproti původnějšímu strukturovanému programování mnoho výhod. Mezi ty nejzásadnější patří znovu použitelnost kódu. Hlavní úlohou UML je graficky zobrazit návrh vyvíjené aplikace i s jejím průběhem. Pro tyto účely má jazyk integrované mnohé nástroje. Mezi ně patří především různé diagramy, jako je například diagram tříd a diagram případu využití. První verze UML byla vydána v roce 1997, vývoj stále pokračuje a v prosinci minulého roku byla schválena již verze 2.5 [13].

S rozvojem a rostoucí oblibou objektově orientovaného programování, se zvětšuje nutnost zabývat se architekturou návrhu software, proto se ve své práci zabývám jazykem UML, který je k tomuto účelu vytvořen. Hlavním cílem práce je demonstrovat přínos a jednoduchost jazyka UML a jeho využitelnost v praxi. Přitom bych chtěl poukázat rovněž na jeho využitelnost ve výuce. Během mé práce bych chtěl zhodnotit i nástroje, které slouží k vytváření UML diagramů. Všechny diagramy a teoretické poznatky budu demonstrovat na jednom příkladu, který se bude týkat vytváření třídní knihy.

[1] Unified Modeling Language

1 Požadavky na třídní knihu

Mým záměrem je prolínání jednoho příkladu celou prací. Pro tento účel jsem si zvolil demonstraci návrhu elektronické třídní knihy. Každý vývoj softwaru začíná zjištěním požadavků od zadavatele, proto i já uvedu, jaká kritéria by měla třídní kniha splňovat.

Půjde o jednoduchou webovou aplikaci, ke které budou mít přístup jak zaměstnanci školy, tak její žáci. Každá osoba bude mít přiřazená určitá práva, ke kterým se budou vázat i činnosti, které daná osoba bude moci vykonávat.

Nejnižší práva k třídní knize bude mít role **žáka**. Ten po přihlášení bude moci pouze prohlížet záznamy tykající se třídy, ve které je evidován.

Jen o jedno právo navíc má role **ředitele**. Ten navíc k prohlížení záznamů, ovšem na rozdíl od žáka u všech ročníků na jeho škole, bude moci zapsat do třídní knihy údaj o hospitaci na dané hodině.

Nejdůležitější bude role **učitele**. Ten po přihlášení bude moci provést zápis hodiny. Řádek zápisu hodiny bude obsahovat automaticky doplněné číslo hodiny, po něm bude následovat téma vyučovací hodiny. Po vyplnění a potvrzení tématu hodiny bude učitel vyzván k zadání počtu přítomných žáků. Tento počet se porovná s počtem žáků zapsaných v evidenci třídní knihy pro daný ročník. Pokud se tyto dva počty rovnají, nechybí žádný žák a zápis řádku bude ukončen. V případě, že se počty nerovnají, bude učitel donucen vybrat se seznamu žáků ty chybějící. Poslední úkon, který učitel bude moci vykonávat, je zapsání poznámky k dané hodině. Samozřejmě kromě těchto speciálních možností bude moci prohlížet záznamy tříd, ve kterých vyučuje.

Nejvyšší pravomoc bude mít **třídní učitel**, ten má veškeré možnosti jako normální učitel. K těm mu přibývá navíc možnost, po předložení omluvenky, omlouvání absencí u jednotlivých žáků. Ke každé třídní knize bude přiřazen seznam žáků, kteří náleží do této třídy. Tento seznam tvoří a upravuje taktéž **třídní učitel.**

Všechny poznatky z předchozího textu jsou shrnuté v tabulce 1. Jsou tu vypsány všechny role, které budou v softwaru vystupovat, a k nim přiřazené pravomoci a funkce.

role/funkce	***prohlížení***	***zápis hodiny***	***zápis hospitace***	***zápis absence***	***omlouvání absence***	***úprava seznamu***
Žák	ANO					
Ředitel	ANO		ANO			
Učitel	ANO	ANO		ANO		
Třídní učitel	ANO	ANO		ANO	ANO	ANO

Tabulka 1 Třídní kniha

2 Základy UML

2.1 Případy užití a diagramy případů užití

2.1.1 Případy užití

Případ užití je jeden ze základních prvků UML. Zachycují počáteční nároky software. Vytváříme pomocí nich vlastně jakési scénáře, které mohou v softwaru nastat. Vystupují zde především jednotliví aktéři, kteří komunikují se softwarem. Vznikají hlavně při počáteční analýze vyvíjeného software. Společně s diagramy případů užití slouží k uspořádání požadavků zadavatele a případné komunikaci s ním [4].

2.1.2 Pravidla tvorby případů užití

Případy užití se v UML zobrazují různými způsoby. Každý příklad užití musí mít uveden svůj název. Pro ten platí pravidlo, že při víceslovném spojení, by měl být uváděn pomocí velbloudího zápisu. To znamená, že víceslovné spojení uvedeme bez mezer a slova od sebe oddělíme použitím velkého písmena na začátku každého slova. Základním způsobem zobrazení případu užití je jednoduchá tabulka, ve které se vytvoří záznamy jednotlivých kroků, které vykoná buď systém, či aktér. Každý případ užití má svého hlavního aktéra. To znamená, že ten vytváří většinu kroků, které se v případu utvoří, nebo hraje roli prvotního spouštěče případu. V jednoduchém zápisu případu užití se již s ničím jiným nesetkáme.

Základní tabulka lze ovšem podle standardů rozšířit. Toto rozšíření není povinné, ale jeho využívání se většinou doporučuje. Jako například v knize UML a unifikovaný vývoj aplikace [1]. První věc, o kterou můžeme tabulku rozšířit, je vstupní hodnota. Tu udáváme jako předpoklad, který by měl aktér nebo systém splňovat před vstupem do případu užití. Další rozšíření je potom očekávaný výstup. To je naopak předpoklad, který musí být splněn při ukončení případu užití. Poslední důležitou věcí, která bývá uváděna, je alternativní scénář. Ten přichází na řadu v případě nějaké chyby, nebo neočekávané události. Někdo k těmto doplňkům přidává rovněž jednoznačné identifikační číslo.

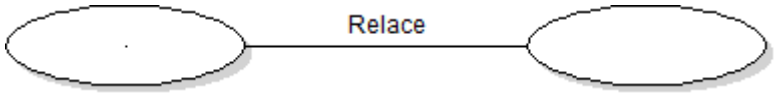
Styl zobrazení tabulky není v UML verze 2 jednoznačně definován. Proto se v knihách od různých autorů velice často liší. Někteří k zobrazení dokonce nepoužívají ani tabulky, ale prostou posloupnost textu.

2.1.3 Struktura diagramu případů užití

Jednotlivé případy užití se skládají do diagramu případu užití. Někdy se zkráceně používá diagramy užití. Slouží k názornému zobrazení toho, kdo vlastně v systému vystupuje a jaké úkony v něm může vykonávat. Tento diagram by měl být první, který při návrhu vznikne.

2.1.3.1 Komunikační relace

Jsou hlavním prvkem, který se v diagramech užití vyskytuje. Znázorňují se jako jednoduché čáry bez šipky na konci. Je to typ přiřazení, který v diagramech případů užití znázorňuje komunikaci mezi dvěma objekty. Mohou to být jak dva aktéři, tak aktér a případ užití, a dokonce i případy užití mezi s sebou [4].



Obrázek 1 Komunikační relace

2.1.3.2 Aktér

Aktéra jsem zmiňoval již při tvorbě případů užití. Jako aktéra označujeme cokoliv, co má možnost komunikovat se systémem. Slovo cokoliv jsem použil záměrně, protože aktéra nemusí zastupovat pouze fyzická osoba. Jako aktér může někdy vystupovat i podsystém či čas [4].

V diagramu případů užití se aktér zobrazuje jako jednoduchá postavička. Pod postavičku se umisťuje označení aktéra. Toto označení by mělo co nejlépe popisovat, kdo aktéra zastupuje a jaké bude mít v systému funkce. Každý aktér by měl mít své vlastní jedinečné označení.

Obrázek 2 Aktér

2.1.3.3 Případ užití

Označuje jeden případ užití, který se v celém diagramu případů užití nachází. Znázorňuje se jako elipsa, do které je vepsán název případu.

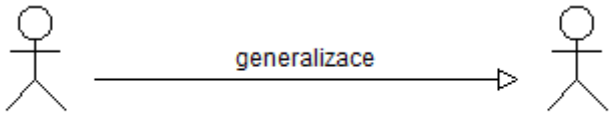
Obrázek 3 Případ užití

2.1.3.4 Hranice systému

Hranice systému ohraničuje, kam až systém svými funkcemi může zasahovat. V diagramu užití se značí jako obdélník, či čtverec. Aktéři stojí vždy mimo hranice systému. Uvnitř hranice systému se znázorňují jednotlivé případy užití, ze kterých se systém skládá. Každá hranice systému může mít v UML svoje jedinečné označení. To se umisťuje doprostřed k hornímu okraji hranice systému. Pro pojmenování hranice nejsou uvedeny, žádné další definice.

2.1.3.5 Zobecnění aktéra

Zobecnění, neboli generalizace, aktéra slouží ke zjednodušení čitelnosti diagramu [1]. Musím přiznat, že ne všichni autoři jeho používání doporučují [4]. Používá se tam, kde dva aktéři používají stejné případy užití. Jeden z nich ovládá ale navíc jeden nebo více případů užití. Když takový případ nastane, začne se mnoho komunikačních relací křížit a zaniká tím snadná čitelnost diagramu. Pro usnadnění nám tedy pomůže generalizace, která se znázorňuje plnou, ale někde také čárkovanou čarou zakončenou uzavřenou nevyplněnou šipkou. Ta vede od potomka k rodičovskému aktérovi. Potomek je ten z dvojice aktérů, který ovládá o případ užití více. Tímto naznačíme, že potomek dědí případy užití po svém rodiči a může je tedy také využívat. Správnost generalizace lze ověřit metodou záměny. Pokud zaměníme potomka za svého rodiče, musí bezpodmínečně zvládat všechny jeho případy užití [1].

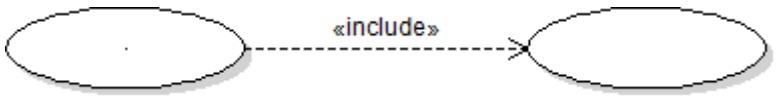


Obrázek 4 Generalizace aktéra

2.1.3.6 Relace

Include

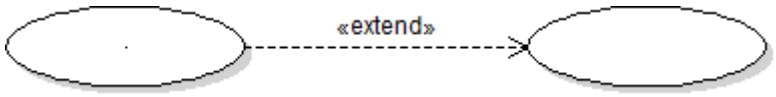
Relace include se používá k redukci opakujících se úkonů. Když se tedy ve více případech užití opakují ty samé kroky, vyčlení se zvláštní případ užití, který tyto kroky provede a s původními případy se propojí relacemi include. Zaznamenává se jako komunikační relace, s tím rozdílem, že při této relaci musíme nad spojující čáru napsat ve dvojitých francouzských uvozovkách slovo include a na konci čáry vedoucí k novému případu užití se nachází otevřená šipka. Zápis tedy vypadá takto: <<include>> [5].



Obrázek 5 Relace include

Extend

Relace extend slouží k rozšíření původního scénáře. Pokud nastane situace, kdy je potřeba rozšířit scénář o kroky, které se nebudou provádět standardně, použijeme tuto relaci. Extend není na rozdíl od include soběstačná. To znamená, že nemůže bez původního scénáře pracovat. V diagramu ji značíme jako nový případ užití. Ten se na původní odkazuje čárou, na jejímž konci je otevřená šipka. Nad čáru se uvádí slovo extend ve dvojitých francouzských uvozovkách. Zápis tedy vypadá takto: <<extend>> [5].



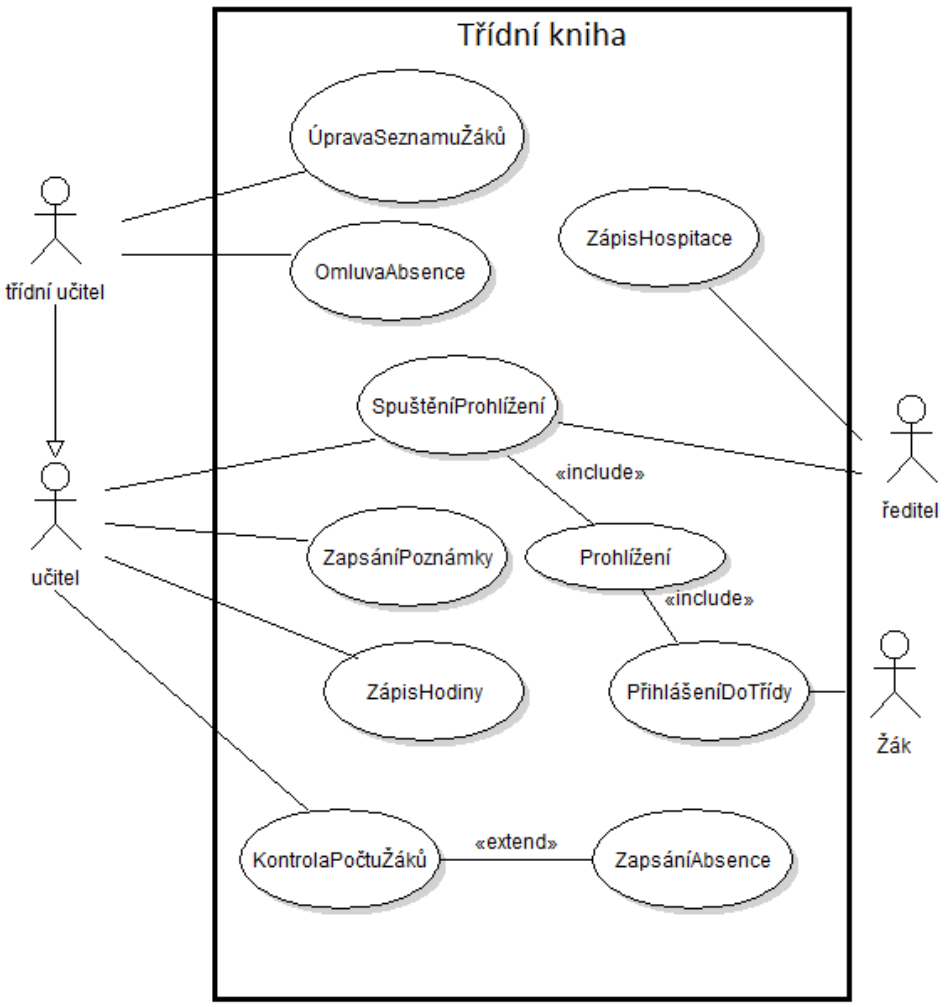
Obrázek 6 Relace extend

2.1.4 Aplikace v návrhu třídní knihy

2.1.4.1 Diagram případů užití pro příklad třídní knihy

Diagram případů užití je první, který jsem při návrhu třídní knihy vytvořil. Ostatní diagramy budou již vycházet z tohoto diagramu. Celkový obraz diagramu se nachází na obrázku číslo 7.

Ukázkový diagram obsahuje jedinou hranici systému. Ta byla nazvána **Třídní kniha** a obsahuje všechny případy užití, které se v systému využívají. Aktéři, kteří v diagramu vystupují, jsou znázorněni mimo hranici systému. Všichni aktéři vznikli podle zadání příkladu. Je tu tedy zastoupen **učitel**, **třídní učitel**, **ředitel školy** a **žák**. V diagramu byla použita generalizace aktérů. Generalizování byli **třídní učitel** a **učitel**. Tato generalizace byla použita, protože třídní učitel je zároveň i učitel, proto má stejná práva jako normální učitel. Navíc k těmto právům má svoje výhradní funkce. Z popisu **třídního učitele** je jasné, že pravidlo záměny zde bude platit. Po zaměnění potomka (**třídní učitel**) za rodiče (**učitel**), bude třídní učitel moci vykonávat všechny případy užití, které normálně provádí učitel.



Obrázek 7 Diagram případů užití

Aktér **žák** má k sobě komunikační relací připojen jediný případ užití s názvem **PřihlášeníDoTřídy**. Tento scénář bude zajišťovat přihlášení žáka do správné třídy a bude na něj navazovat případ užití **Prohlížení**, který je připojen pomocí relace include. Relace include jsem tady použil z toho důvodu, že **Prohlížení** bude využívat více aktérů, kteří ho budou volat přes různé případy užití. Ty se budou navzájem lišit, ale část zastoupená v **Prohlížení** by se několikrát zbytečně opakovala.

Na aktéra **učitel** potažmo tedy **třídní učitel** jsou navázány komunikační relací, případy užití **ZápisHodiny**, **ZapsáníPoznámky**, **SpuštěníProhlížení** a **KontrolaPočtuŽáků**. **SpuštěníProhlížení** bude podobné **PřihlášeníDoTřídy** u aktéra **žák.** S tím rozdílem, že systém vybere učiteli třídy, ve kterých vyučuje a v nich si poté bude moci procházet záznamy. Tento případ je taktéž navázán pomocí relace include na **Prohlížení**. Dále spouští případ **KontrolaPočtuŽáků**, kde dojde ke kontrole přítomných žáků. Tento scénář je rozšířen pomocí relace extend. Ta přijde ke slovu v případě, nerovná-li se počet žáků přítomných s počtem

zapsaných ve třídní knize. Poslední jsou dva jednoduché případy **ZapsáníPoznámky**, který slouží k zapsání poznámky k dané hodině a **ZápisHodiny** pro provedení zapsání tématu hodiny.

Třídní učitel, potomek **učitele**, ovládá všechny jeho případy užití. Navíc spouští **OmluvaAbsence**, ve které může na základě omluvenky od žáka omluvit již zapsanou absenci. A také má právo upravovat seznam žáků ve třídě, a to pomocí **ÚpravaSeznamuŽáků**.

Ředitel školy opět uplatňuje přes relaci include **ProhlíženíZáznamů**, ta je v jeho případě navázána na případ spouštění prohlížení. A kromě toho může zapsat absolvovanou hospitaci díky **ZápisHospitace**.

2.1.4.2 Případ užití

Všechny případy užití jsou zobrazeny na předchozím diagramu. Tabulka číslo 2 obsahuje případ užití **KontrolaPočtuŽáků**.

Případ užití: KontrolaPočtuŽáků
ID: 1
Hlavní aktér: Učitel
Vstupní podmínky: Učitel je přihlášen
Hlavní scénář: 1. Učitel zvolí zapsání absence 2. Systém vyzve učitele k zadání počtu žáků 3. Učitel zadá počet žáků 4. Systém porovná počty žáků Bod rozšíření: ZapsáníAbsence 5. Učitel ukončí a uloží zapsání absence
Výstupní podmínky: Zadání počtu žáků
Alternativní scénář: ZadejČíslo

Tabulka 2 Hlavní scénář

První položkou je název případu, následovaný unikátním identifikačním číslem. U hlavního scénáře jsem zvolil číslo celé, naproti tomu u rozšiřujícího a alternativního scénáře jsem použil číslo desetinné. Z důvodu větší názornosti a odlišení postavení. Hlavním aktérem je zde **učitel**, k tomuto se vztahuje vstupní podmínka, a to že učitel musí být před spuštěním případu užití do systému třídní knihy přihlášen. Po splnění vstupní podmínky přichází na řadu hlavní scénář, kde už je krok po kroku rozepsáno, jak bude program postupovat při průchodu scénářem. První anomálie nastává u bodu číslo 4, na který je navázán alternativní scénář **ZadejČíslo**, který je zpracován v tabulce číslo 3.

Případ užití: ZadejČíslo
ID: 1.1
Hlavní aktér: Učitel
Vstupní podmínky: Učitel zadá špatný údaj
Alternativní scénář: 1. Scénář začíná krokem 4 hlavního scénáře 2. Systém upozorní učitele na chybu
Výstupní podmínky:
Alternativní scénář:

Tabulka 3 Alternativní scénář

Ten má za úkol upozornit učitele na špatně vyplněný počet (například písmena, nebo větší počet žáků, než je ve třídě zapsáno). Po upozornění se vrátí ke kroku číslo 2 hlavního scénáře, odkud dále pokračuje. Další zvláštnost následuje po bodu číslo 4, a to bod rozšíření zobrazený v tabulce číslo 4.

Případ užití: ZapsaníAbsence
ID: 1.5
Hlavní aktér: Učitel
Vstupní podmínky: Systém vyhodnotí, že chybí jeden či více žáků
Hlavní scénář: 1. Systém nabídne učitele seznam žáků 2. Učitel vybere nepřítomné žáky
Výstupní podmínky:
Alternativní scénář:

Tabulka 4 Rozšiřující scénář

Ten odkazuje na případ užití spojený s hlavním scénářem relací extend. Pokud je tedy vyhodnoceno, že chybí jeden či více žáků, přepneme se do rozšiřujícího případu. V rozšíření dá systém učiteli vybrat se seznamu žáků ty, kteří jsou nepřítomní. Po provedení kroků z rozšiřujícího scénáře se opět vrátím do hlavního scénáře. Zde bude proveden poslední krok a případ užití končí.

2.1.5 Využití diagramů případů užití

Pokud jsou diagramy vypracovány správně, jsou velice dobře čitelné. Jsou postaveny tak, aby jim rozuměl i člověk, který se návrhem softwaru nezabývá. Proto se požívají i pro komunikaci se zákazníkem. Při předložení tohoto návrhu, v mém příkladu řediteli školy, mu bude z diagramu patrné, kdo může co vykonávat. Může vznášet své připomínky a návrhy a vlastně vám pomoci k vylepšení systému, aby byl přesně podle jeho představ, ještě před samotným začátkem vytváření. Tímto se můžeme vyhnout nepříjemnému předělávání až po dokončení systému. I samotné případy užití nejsou některak složité. Ředitel by díky nim viděl srozumitelně jednotlivé kroky, které bude zapotřebí vykonat, k provedení libovolného úkonu. Může tedy opět vznést svoje námitky a připomínky, kterými můžeme systém vylepšit. Diagramy jsou vhodné i pro aplikaci ve výuce. Svou názorností se hodí pro vysvětlování struktur systémů žákům. Další jejich užitek je pro softwarového architekta, který pomocí těchto diagramů rozvíjí software dále, za použití již jiných diagramů, které se zabývají funkčností a přesnou stavbou softwaru.

2.2 Diagramy tříd

Diagramy třídy patří k nejpoužívanějším diagramům vůbec. Tento fakt je způsoben rostoucí oblibou objektově orientovaného stylu programování, pro který je tento diagram vhodný. Slouží k zobrazení tříd, které systém obsahuje, jejich objektů a komunikace mezi nimi.

2.2.1 Objekty

Objekty jsou označovány jako instance třídy. To znamená, že každý objekt přebírá souhrn metod a vlastností po své třídě. Každá třída může obsahovat žádný nebo více objektů, ale každý objekt musí náležet pouze do jedné třídy. Objekty je třeba od sebe jednoznačně odlišit například názvem [5].

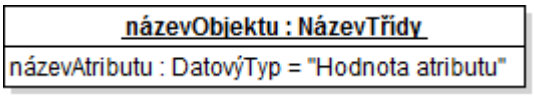
2.2.1.1 Zobrazování objektů

Objekty se v jazyku UML zobrazují jako tabulka rozdělená na dvě části. V první části tabulky se udává název objektu.

Názvy objektů musí být mezi ostatními instancemi jedné třídy jedinečné, aby nedošlo ke zmatkům a záměně. K zápisu názvu se požívá velbloudí zápis, ve kterém se začíná malým písmenem. Za samotným názvem následuje dvojtečka, za kterou se udává název třídy, do které objekt náleží. Celá tahle soustava názvu se pro odlišení od třídy podtrhává. Existují i takzvané anonymní objekty. U nich se vynechává část před dvojtečkou a udává se pouze název třídy,

před kterým se ovšem ponechává dvojtečka. Tento zápis lze použít pouze tehdy, má-li třída jen jedinou instanci [1].

V druhé části tabulky se zobrazují vlastnosti objektu. Tato část je na rozdíl od první nepovinná. Jednotlivé vlastnosti se skládají z několika částí. Prvně se udává název vlastnosti. Zápis názvu by měl být nejlépe opět zapsán velbloudím zápisem s prvním malým písmenem. Za dvojtečkou se nachází datový typ vlastnosti. Za ním už následuje samotná hodnota vlastnosti, které předchází znaménko rovná se [1].



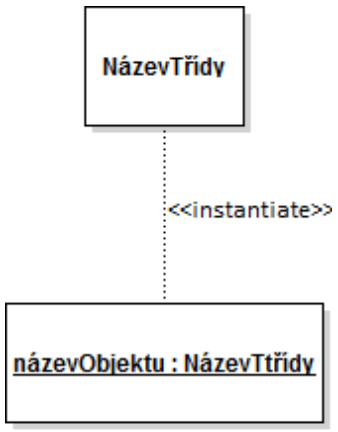
Obrázek 8 Objekt

2.2.2 Třídy

Třídy se dají brát jako předlohy pro tvorbu objektů. Jsou to vlastně předci objektů, které můžeme brát jako její potomky. Třídu lze definovat jako ukazatel objektů, které spolu sdílejí stejné atributy a operace, chování a metody [10].

2.2.2.1 Vztah třídy a objektu

Vztah třídy a objektu se znázorňuje pomocí relace instance. Tato komunikační relace, nám říká, že objekt je potomek dané třídy. Vyjadřujeme pomocí ní závislost objektu ke třídě. V diagramu se tato relace zobrazuje tečkovanou čarou. Čára je ukončena šipkou, která vede směrem od objektu ke třídě. K čáře se přidává slovo instantiate, které se umisťuje do dvojitých francouzských uvozovek [5].



Obrázek 9 Instantiate

2.2.2.2 Zobrazování třídy

Zobrazování třídy se velice podobá zobrazení objektu. V tomto případě je, ale tabulka rozdělena na tři části. V první části se nachází název třídy, ve druhé se popisují vlastnosti neboli atributy. Poslední část obsahuje metody a operace, které třída používá.

Část s názvem třídy je jediná povinná. Zápis, který nyní budu prezentovat, pochází z knihy UML2 a unifikovaný proces vývoje aplikací [1]. Obsahuje samotný název třídy. Vyjadřujeme ho opět velbloudím stylem zápisu. Tentokrát ovšem název nepodtrháváme. Při vymýšlení názvu bychom se měli vyvarovat speciálních znaků (např. @, &, # apod.), ale také požívání zkratek. Zkratky by mohly vnést zmatek do následné implementace.

První z nepovinných částí je tedy pole s atributy neboli vlastnostmi. Sem zadáváme jednotlivé vlastnosti, které třída obsahuje. Každý řádek s vlastností se skládá s několika údajů.

Na začátek se umisťuje viditelnost vlastnosti. Ta má celkem čtyři možnosti. První možnost je, že je vlastnost přístupná všem součástím systému. Takové vlastnosti potom označujeme jako veřejné a v zápisu se značí znaménkem plus. Další typ viditelnosti oznamuje, že vlastnost je použitelná pouze operacemi definovanými přímo v dané třídě. Tento typ viditelnosti se nazývá soukromý a v zápisu bývá značen znaménkem mínus. O něco mírnější je chráněná viditelnost. Ta umožňuje přístup k vlastnosti operacím dané třídy a i operacím, které byli definovány až u potomků této třídy. Tuto viditelnost značíme znakem křížku. Posledním typem viditelnosti je balíček. Ten označujeme znakem vlnovky a umožňuje přístup k vlastnosti operacím tříd ze stejného balíčku.

Na viditelnost navazuje název vlastnosti. Tento údaj se musí uvést vždy. Zápis se opět provádí velbloudím stylem začínajícím malým písmenem.

Další části je typ atributu. Slouží k určení, jaká hodnota se bude v atributu očekávat. Typy jsou shodné s většinou objektově orientovaných programovacích jazyků. Máme tedy například možnost integer pro celá čísla, date pro datum, string pro označení řetězce znaků, boolen na nabývání hodnot true nebo false a tak dále.

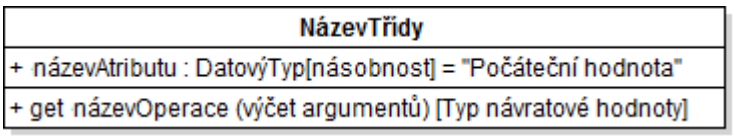
Za typ se může umístit do hranatých závorek násobnost. Je to hodnota omezující počet očekávaných údajů. Může se udávat buď jednoduché číslo, které dává jasný údaj o počtu, nebo rozmezí mezi dvěma čísli. V rozmezí se od sebe čísla oddělují dvěma tečkami.

Jako poslední máme možnost uvést počáteční hodnotu atributu. Pokud tento údaj vyplníme, tak každý nový objekt bude obsahovat atribut již s vyplněnou hodnotou a ta se bude rovnat právě počáteční hodnotě.

Poslední část třídy je úsek obsahující operace vykonávané třídou. Třída tento úsek nemusí nutně obsahovat a může proto existovat třída, na kterou nejsou navázány žádné operace. Operace mají svůj styl zápisu. Stejně jako u atributu lze začínat viditelností, která nabývá i stejných hodnot jako bylo popsáno výše (veřejný, skrytý, chráněný a balíček). Pokračujeme povinným názvem operace. Za název se do jednoduchých závorek uvádí výčet argumentů. Nakonec se udává typ návratové hodnoty, možnosti typů jsou rovněž shodné jako u atributů.

S celého zápisu operace je nejzajímavější výčet argumentů. Ten má totiž svoje další pravidla pro zápis. Kromě samotného názvu argumentu, přidáváme ještě jeho typ a takzvaný směr. Směr nám udává, zda je argument vstupní, označujeme jako in, výstupní označený out nebo může být obojí, poté jej označujeme jako inout. Implicitní hodnota směru je in, pokud tedy neuvedeme jinak, bude se argument považovat za vstupní.

Každá operace má v sobě zabudovánu dotazující metodu Query. Pokud tuto metodu použijeme, operace bude pracovat jako vyhledávací. Nebude tedy atributy upravovat, ani s nimi nikterak manipulovat. Pouze nalezne požadovaný atribut a zobrazí jeho hodnotu. Query má dva stavy, true a false. Pokud je nastaven na false operace nepracuje jako dotazovací, ale jako klasická. False je nastavena jako implicitní. Pokud chceme hodnotu query nastavit na true, musí to být v operaci uvedeno. Nejčastěji se udává před název operace slovo get, pokud chceme, aby fungovala jako vyhledávací [4].

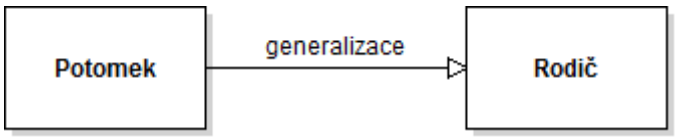


Obrázek 10 Třída

2.2.2.3 Generalizace tříd

Zobecňování tříd se velice podobá zobecňování aktérů v diagramu aktivit. Vystupují zde dvě a více tříd. Z nichž je vždy jedna rodičovská, zbytek tříd jsou její potomci. U generalizace se uplatňuje princip dědičnosti. Všichni potomci jednoho rodiče, po něm dědí všechny atributy a operace. Potomci by měli obsahovat i atributy, či operace na víc. Ovšem platí zde opět pravidlo zastupitelnosti. Takže každý potomek musí být schopen zastoupit rodiče

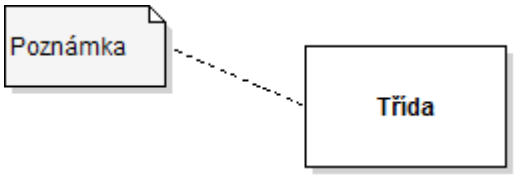
při každé situaci, která může nastat. Stejně jako u aktérů se generalizace značí nepřerušovanou čárou, která je ukončená uzavřenou nevyplněnou šipkou směřující k rodiči [4].



Obrázek 11 Generalizace

2.2.2.4 Komentáře

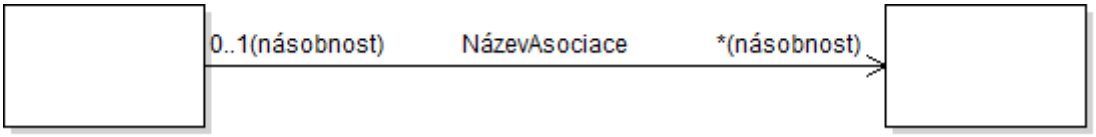
Komentáře používáme ke komentování tříd nebo pro napsání poznámky. Kromě diagramů tříd se ve stejné podobě využívají i u diagramů případů užití. Mohou stát jak volně v diagramu, tak mohou být navázané přímo k určité třídě či objektu. V takovém případě je navazujeme pomocí čárkované čáry. Samotný komentář respektive poznámka se poté vkládá do obdélníku s horním rohem přehnutým. Mimo této možnosti jde také vkládat poznámky přímo do prvku diagramu. U této možnosti oddělujeme poznámku od standartního textu dvěma pomlčkami.



Obrázek 12 Komentář

2.2.2.5 Asociace

Asociace nám znázorňují relace mezi třídami a jejich atributy. Zobrazují se plnou, čarou mezi dvěma třídami. Mohou se k ní přidávat následující informace. Název asociace. Měl by popisovat vztah mezi třídami ve slovesném tvaru. Dále mohou být obsaženy názvy rolí. Ty se přidávají nad, čáru blíže ke třídě a popisují jakou má roli, vzhledem k třídě druhé. Měli by být zobrazeny vždy dvě role, každá u jedné třídy. Názvy rolí a název asociace by se nikdy neměly objevovat u jedné asociace společně, role totiž vyplynou z názvu asociace a naopak. Násobnost vyjadřuje počet objektů, které mohou za danou třídu vystupovat v určitém čase. Vyjadřuje se podobně jako u atributů číslem nebo rozsahem čísel. Hodnotu od určitého čísla výše nezobrazujeme znakem nekonečno, ale pomocí hvězdičky. Údaj se zapisuje na čáru ke třídě, které násobnost odpovídá [4].



Obrázek 13 Asociace

2.2.2.6 Agregace a kompozice

Agregace je typ vazby, který říká, že druhá třída je součástí té první. Kompozice je velice podobná, s tím rozdílem, že třída, která je ve vztahu kompozice, nemůže stát v diagramu samotná. Kdežto třída z vazby agregace samotná stát může. Příkladem agregace může být vztah třídní knihy a seznamu žáků. Seznam žáků může existovat i bez třídní knihy. Příkladem kompozice je jedna hodina a absence v ní. Absence by bez hodiny, u které vznikla, nedávala samostatně smysl [4].

V diagramu značíme agregaci plnou čarou, s nevyplněným kosočtvercem na straně blíže ke třídě nadřazené. Nad konce čar můžeme umístit násobnost vazby. Kompozici značíme podobným způsobem a to včetně použití násobnosti. Jediný rozdíl je použití vyplněného kosočtverce na straně nadřazené třídy [4].

Obrázek 14 Agregace a kompozice

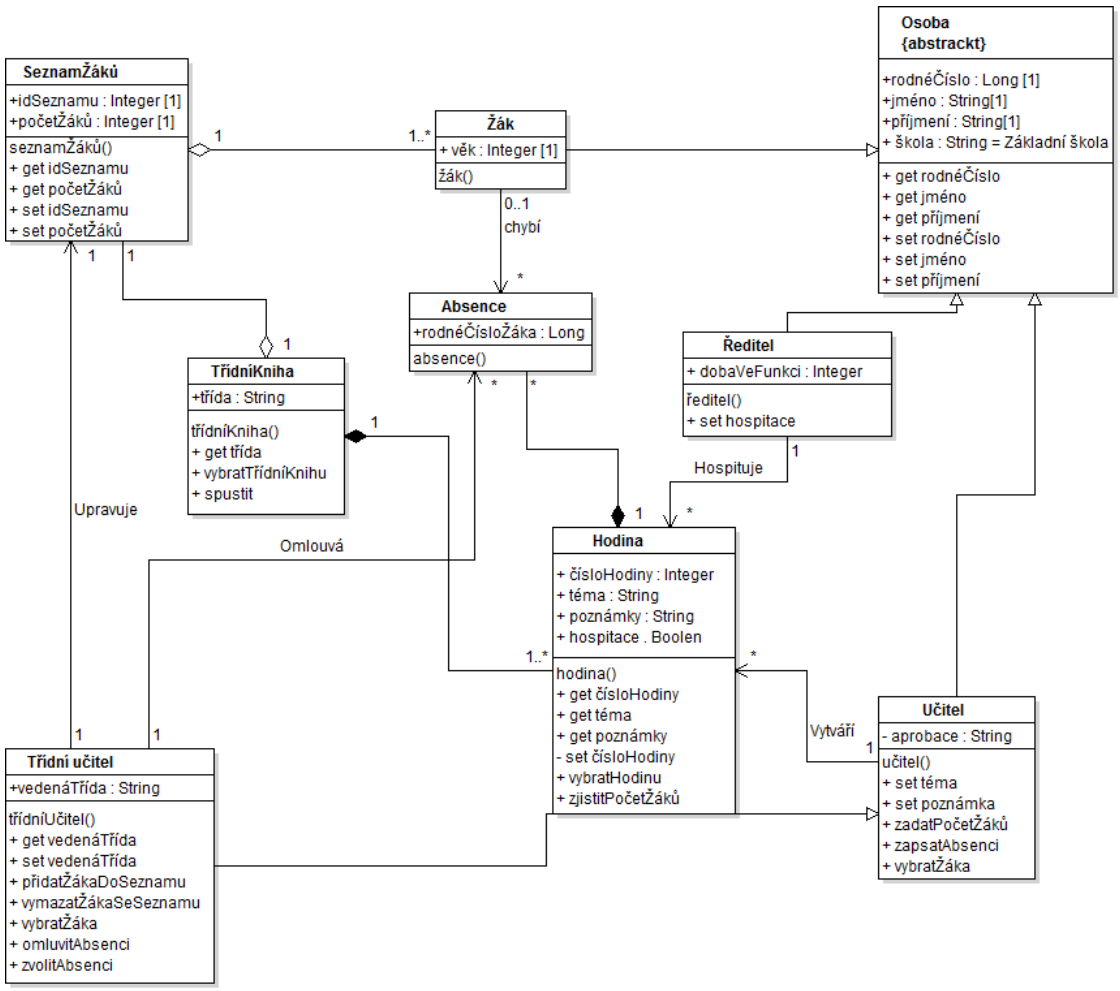
2.2.2.7 Abstraktní třída

Krom klasické třídy, můžeme v diagramu vytvořit i takzvané třídy abstraktní. Jsou odlišné tím, že je později neaplikujeme při vytváření konečného programu. Taková třída nemůže mít ani žádné svoje instance. Většinou slouží pouze k tomu, aby po ní ostatní třídy mohli efektivně podědit atributy a operace. Takovou třídu označíme v části názvu třídy, slovem abstrakt ve složených závorkách [14].

2.2.2.8 Konstruktory

Konstruktory jsou speciální metody s jediným účelem. Slouží k vytváření nových objektů třídy, pro kterou jsou definované. Definujeme je s ostatními metodami a operacemi v poslední části třídy. Pojmenováváme je stejným názvem jako třídu. Neudáváme jim žádnou návratovou hodnotu, ani argumenty. Naproti konstruktorům stojí destruktory, které slouží k uvolněním paměťového místa po objektu.

2.2.3 Aplikace v návrhu třídní knihy



Obrázek 15 Diagram tříd

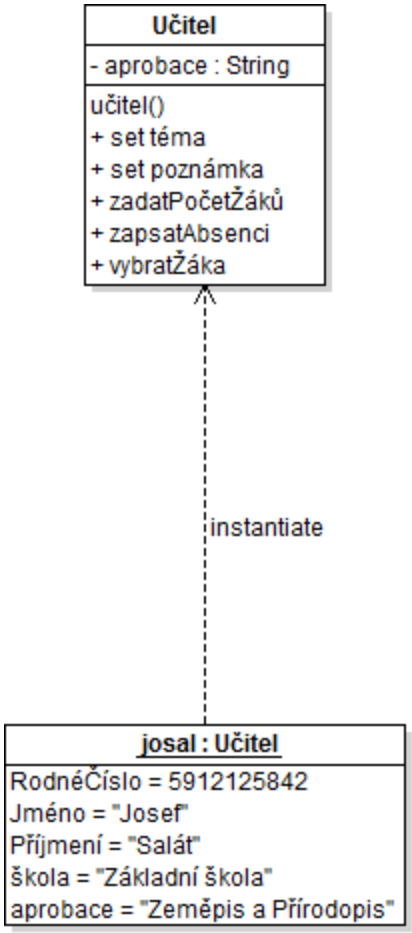
Na obrázku číslo 15 je zobrazen diagram tříd pro příklad třídní knihy. Vytvořil jsem několik tříd, některé z nich se shodují s aktéry s předchozího diagramu případů užití. Z toho diagramu je převzat **Učitel**, **TřídníUčitel**, **Žák** a **Ředitel**. Naproti diagramu případů užití zde vznikly i nové třídy. Jsou to třídy, které zastupují samotnou strukturu třídní knihy. Patří mezi ně **Třídníkniha**, **Hodina**, **Absence** a **SeznamŽáků**. Kromě těchto klasických tříd je tu i jedna třída abstraktní. **Osoba** je abstraktní třída, která obsahuje všechny základní atributy a operace, jako je jméno a příjmení a operace pro jejich zobrazení a nastavení. Všechny tyto položky po **Osobě** dědí pomocí vztahu generalizace třídy **Učitel**, **Ředitel**, **Žák** a potažmo i **TřídníUčitel**. **TřídníUčitel** je totiž navázán pomocí generalizace na **Učitele**. To znamená, že **TřídníUčitel** dědí všechny položky po **Učiteli** a to i ty, které sám podědil od **Osoby**.

U každé třídy vidíme několik atributů. Například u osoby je atribut **jméno.** V tomto zápisu znamená, že je to atribut veřejný, tudíž jeho hodnotu může upravovat kterákoliv třída. To je potřebné z důvodu budoucího dědění tohoto a ostatních atributů této třídy. Po jménu atributu za dvojtečkou je očekáváný datový typ s multiplicitou ve hranatých závorkách, ta je v tomto případě jedna. Je tu tedy očekáván jeden řetězec znaků. U atributu **škola** byla navíc použita počáteční hodnota. Pokud tedy vytvoříme instanci této třídy, počáteční hodnota atributu **škola**, bude nastaven na hodnotu „základní škola“. Tato hodnota není pevně daná a její změna je tedy možná.

Každá třída, kromě abstraktní, má na prvním místě operací konstruktor, pro vytvoření objektů dané třídy. Například u třídy žák je to **žák ()**. Po konstruktorech následují ostatní operace. Nejčastěji se objevují operace s metodami get a set na začátku. Například u seznamu žáků operace **get idSeznamu** zobrazí identifikační číslo daného seznamu žáků. Tato operace má status public, tudíž může být vyvolána i jinou třídou. Oproti této operaci stojí operace **set idSeznamu**. Pomocí této operace budeme nastavovat identifikační číslo seznamu. Je označena jako veřejná z toho důvodu, že kromě samotného seznamu, může hodnotu tohoto atributu upravovat i **TřídníUčitel. TřídníUčitel** má mimo to i svoje operace, kterými může upravovat a vytvářet různé objekty v diagramu. Například operace **přidatŽákaDoSeznamu**, která má status veřejný, bude sloužit k přidání nového objektu **Žák** do instance třídy **SeznamŽáků.**

Mezi jednotlivými třídami jsou znázorněny asociace, které jsou popsány slovesným tvarem. Například mezi **TřídnímUčitelem** a **SeznamemŽáků** je asociace **upravuje**. Tím se vyjadřuje, že třída **TřídníUčitel** může upravovat a vytvářet **SeznamŽáků**. Krom názvu je u této asociace znázorněna i násobnost. Zde konkrétně je jedna ku jedné. To znamená, že v jeden okamžik může jedna instance třídy **TřídníUčitel** spravovat pouze jednu instanci třídy **SeznamŽáků**. Jiná násobnost je mezi **Učitelem** a **Hodinou**. Na straně hodiny je násobnost označená hvězdičkou, to znamená, že může nabývat libovolnou hodnotu mezi nulou a nekonečnem. **Učitel** tedy může vytvářet více hodin, ale každá **Hodina** je vytvořena pouze jedním učitelem. Mimo asociace jsou v diagramu použity i agregace a kompozice. Agregace je například mezi třídou **Žák** a **SeznamŽáků.** V tomto případě znázorňujeme, že žák náleží do seznamu žáků. Aby každý žák náležel jenom do jednoho seznamu, zajišťujeme pomocí násobnosti. Tou ošetřujeme i potřebu mít v každém seznamu alespoň jednoho žáka. Žák může ovšem stát v systému i samostatně a to v případě vyloučení žáka ze třídy. Jako kompozice je spojena třída **TřídníKniha** a **Hodina. Hodina** totiž náleží do třídní knihy a tam musí náležet vždy. Třída hodina nikdy nemůže stát mimo třídní knihu.

Na diagramu zobrazeném na obrázku číslo 3, je znázorněna spolupráce třídy a jejího objektu. Jako počáteční třídu jsem vybral **Učitele.** K němu jsem vytvořil jeho instanci, která nese název **josal.** Jsou spolu spojeni závislostí instantiane. Název je oddělen dvojtečkou, od názvu třídy **Učitel**. Všechny atributy objektu **josal** jsou poděděné po **Učitelovi,** s tím rozdílem že u každého, je již vyplněna konkrétní hodnota. U atributu **škola** byla ponechána počáteční hodnota, nastavená již v mateřské třídě.



Obrázek 16. Třída a její objekt

2.2.4 Využití diagramů tříd

Diagramy tříd se využívají k návrhu struktury softwaru. Neřešíme zde jeho funkčnost, ale pouze si vytváříme představu o tom, jak by mohlo vše mezi sebou pracovat. Diagram je například vhodný i ke komunikaci se zadavatelem projektu, v našem případu tedy opět ředitel školy. Jsou na něm názorně vidět objekty, které budou v systému vystupovat, včetně jejich vlastností. Ředitel po předložení tedy může argumentovat, že mu například chybí u žáků datum nástupu do školy, či jiný atribut u někoho jiného. Také má v této fázi pořád šanci přidat aktéra. Například si při tomto návrhu může uvědomit, že mu chybí ještě uklízečka, která by mohla provádět zápis do poznámky o poškození věci. Mimo přidávání a upravování třídy, může ředitel

zasáhnout i do vztahů v systému. Třeba si může uvědomit, že jako ředitel vůbec nechce mít přístup k hodině. Pro programátora má v této fázi diagram tříd význam hlavně při tvorbě databáze. Tu vytváří podle vzoru, který je v diagramu navržen. Softwarový architekt může po vytvoření tohoto diagramu pokračovat v návrzích dalších částí, jako je například sekvenční diagram, kde už je potřeba znát třídy a jejich vztahy.

2.3 Sekvenční diagramy

Sekvenční diagram je jeden z takzvaných diagramů interakcí, tedy spolupráce. Zobrazují se pomocí nich spolupráce objektů, při postupu jednotlivými scénáři. Tento postup je seřazen v diagramu časově. Jak scénáře, tak i objekty jsou navrhnuty již v dřívějších fázích vývoje a my už jen navrhujeme jejich komunikaci. Tento diagram byl prosazován jedním z prvních autorů standardů UML Ivarem Jacobsonem.

2.3.1 Struktura sekvenčních diagramů

Sekvenční diagramy se mohou umisťovat do hranice diagramu. Ta se podobá hranici případů užití. Opět se jedná o obdélník, ve kterém jsou uzavřeny všechny prvky diagramu. Název diagramu se tentokrát vkládá do levého horního rohu. V UML druhé verze je zvykem přidávat před název sekvenčního diagramu označení sd. To se používá pro jednoznačné odlišení diagramů interakcí. Název se vpisuje do obdélníku s levým spodním koncem zkoseným. Další podobností s předešlými diagramy jsou poznámky. Jejich zápis se provádí stejně jako u diagramu tříd. Tudíž v obdélníku s levým zahnutým rohem, který je spojen s místem náležícím poznámce čárkovanou čarou [5].

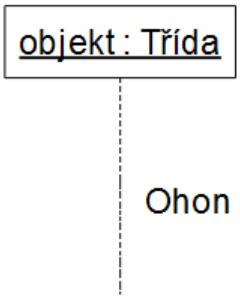
2.3.1.1 Čára života

Samotný sekvenční diagram lze poté rozdělit na dva základní oddíly. První oddíl, který se většinou zobrazuje na vrchní části diagramu, obsahuje objekty a aktéry, kteří v diagramu vystupují. Druhý oddíl se skládá z jejich interakcí.

Pokud ve scénáři případu užití, který budeme realizovat sekvenčním diagramem, vystupuje aktér, můžeme ho zobrazit stejně jako v diagramu případů užití. Jeho značka je tedy obyčejný panáček, s názvem pod sebou. Toto značení však není povinné. Vystupující objekty se označují obdélníkem, ve kterém se mohou objevit tři informace. První je samotný název objektu. Ten můžeme vynechat u objektu anonymního a zobrazení komunikace třídy. Další, opět nepovinou částí, je selektor. Pomocí něho můžeme nasměrovat interakci s daným

objektem, přesně podle hodnoty zvoleného atributu. Poslední části je tradičně název třídy, jejíž je objekt instancí. Celý zápis se pro odlišení od třídy podtrhává [5].

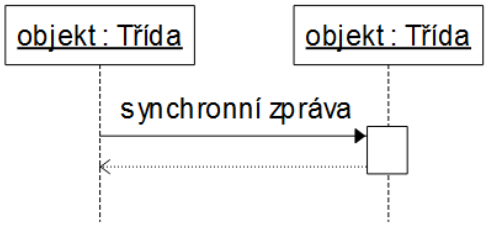
Tyto objekty a aktéři jsou první částí takzvané čáry života. Pomocí ní se znázorňuje způsob, kterým jsou objekty zapojeny do interakcí. Další částí čáry života je takzvaný „ohon“. Zobrazuje svislou čárkovanou čarou, vedoucí od označení objektu dolů. Od ní poté vedou všechny zprávy, které aktér či objekt vysílají a naopak k ní vedou odpovědi, které jsou přijímány od ostatních objektů či aktérů.



Obrázek 17 Čára života

2.3.1.2 Zprávy

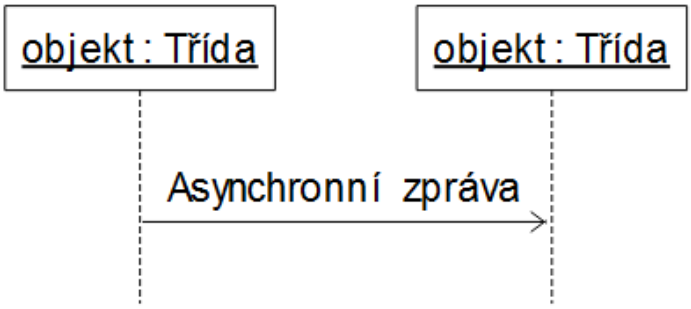
Čáry života spolu komunikují pomocí takzvaných zpráv. Máme dva typy základních zpráv. Zprávy synchronní a zprávy asynchronní. Po vyslání synchronní zprávy objekt čeká na odpověď a sám nepokračuje dál v činnosti. Kdežto u asynchronní na nic nečeká a ihned po odeslání dále pokračuje.



Obrázek 18 Synchronní zpráva

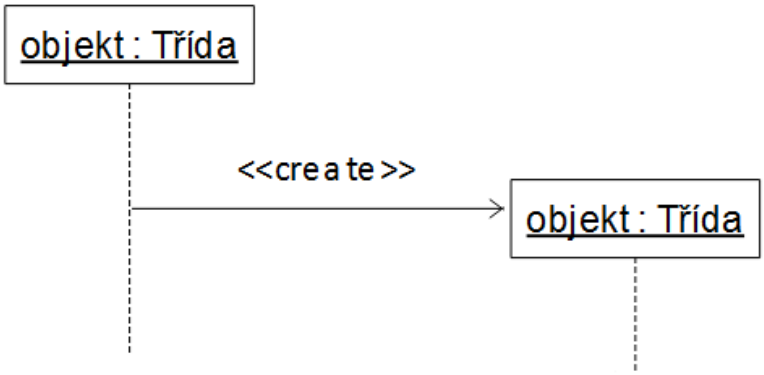
Zprávy se zobrazují šipkou od ohonu vysílajícího objektu, k ohonu objektu, který zprávu přijímá. Synchronní zpráva se značí na konci šipkou vyplněnou, asynchronní šipkou nevyplněnou. Nad šipku se umisťuje název zprávy, který by měl vyjadřovat co má řízený objekt

vykonat. Zprávy se mohou shodovat s operacemi vytvořenými v diagramu tříd. Do závorky za název se může udat parametr, se kterým bude zpráva odeslána. Odpověď objektu se poté značí níže pod samotnou zprávu, až poté co volaný objekt provede všechny vyžadované operace, otevřenou šipkou, která probíhá čárkovanou čarou. Nad odpověď nepřidáváme název, ani žádný parametr.



Obrázek 19 Asynchronní zpráva

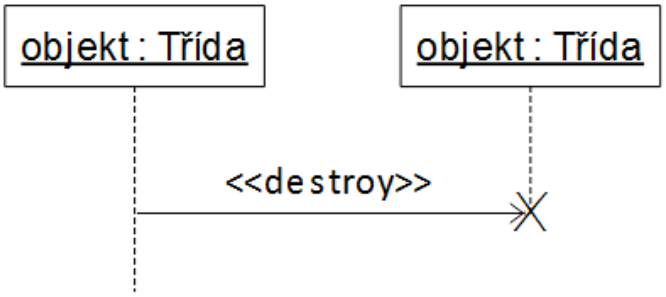
Zvláštním typem zpráv jsou poté zprávy na vytvoření nebo zánik čáry života. Zpráva pro tvorbu, slouží k vytvoření nového objektu a potažmo i k vytvoření jeho čáry života. Vyvolání zániku způsobí naopak uvolnění objektu, na který je zpráva vedena, to znamená, že se na něj již nadále nedá odkazovat. Zpráva pro tvorbu se značí stejně jako obyčejná asynchronní zpráva. S tím rozdílem, že se nad šipku přidává stereotyp create ve dvojitých francouzských uvozovkách. Tato zpráva nevede k ohonu čáry života, ale přímo k nově vzniklému popisu objektu [1].



Obrázek 20 Vytvoření čáry života

Zpráva pro zánik se opětovně značí totožně jako obvyklá asynchronní zpráva. Tentokrát nad ní přidáváme stereotyp destroy ve dvojitých francouzských uvozovkách. V místě kam zprávu přivedeme, naznačíme konec čáry života velkým křížkem. Objekt může zrušit svou čáru

života i sám po tom co vyhodnotí, že již není nadále potřebný. Tato situace se opět zaznačí velkým křížkem na konci čáry života [1].

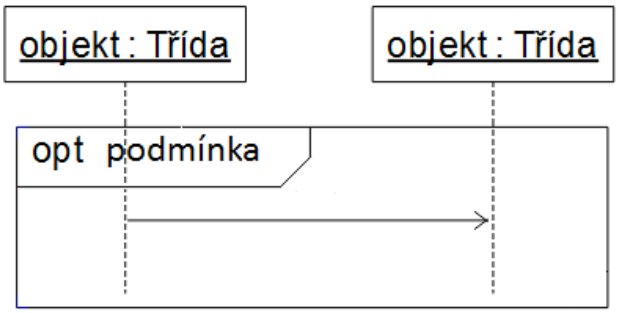


Obrázek 21 Zánik čáry života

Na čárách života je možné vyobrazit i kdy je daná čára aktivní. To je, když odeslala synchronní zprávu a čeká na odpověď, nebo když provádí jiné operace. I při odeslání asynchronní zprávy se čára života na chvíli zaktivuje. Aktivita se zobrazuje úzkým obdélníkem, který překrývá část čáry života, která má být aktivní.

2.3.1.3 Větvení

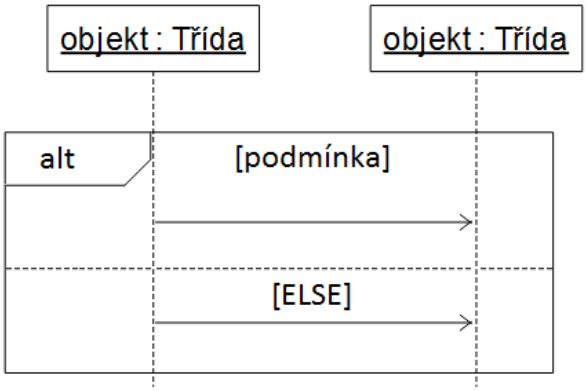
Další možností zabudovanou v definicích sekvenčních diagramů je větvení. Při větvení v sekvenčním diagramu máme možnost použít dva operátory. Oba se zobrazují obdélníkem, takzvaným fragmentem, který zahrnuje část diagramu, který obsahuje samotné větvení. To jsou zprávy, které budou odeslány v jednotlivých větvích. V levém horním rohu je úsek, obsahující název operátoru. První operátor je opt. Opt se dá považovat za ekvivalent if používaného ve většině programovacích jazyků. U opt se k názvu operátoru přidává ještě podmínka. Při jejím splnění jsou zprávy obsažené ve fragmentu odeslány [11].



Obrázek 22 Opt

Druhým operátorem je alt. Alt lze přirovnat například k větvení select case používaného v jazyce Visual basic. V tomto případě je fragment rozdělen na další dílčí fragmenty. Každý dílčí fragment má v hranatých závorkách zapsanou vlastní podmínku. Při splnění podmínky

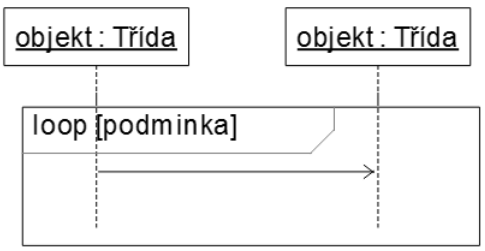
se provedou operace v tomto fragmentu. V každém větvení alt se může podmínka splnit pouze u jediného dílčího fragmentu. Vícenásobné splnění není v UML definováno. Jako poslední lze přidat else větev, která by byla aktivována v tom případě, že podmínka nebude splněna ani v jedné z předchozích větví. Zajímavou alternativou k těmto operátorům je další operátor par. Ten totiž provede všechny své větve v činnost současně, nezávisle na podmínce [1].



Obrázek 23 Alt

2.3.1.4 Cyklus

Kromě operátorů větvení existují i operátory pro tvorbu cyklů. Jejich zápis probíhá stejně jako u větvení. Operátor pro cyklus se nazývá loop. Při uvedení samotného názvu operátoru, vytvoříme nekonečnou smyčku. Pokud k loop přidáme do hranatých závorek podmínku, tak se bude cyklus opakovat jen po tu dobu, po kterou bude podmínka splněna.



Obrázek 24 Loop

Při tomto zápisu můžeme přidat před závorky tečku a hvězdičku, tím naznačíme, že cyklus má být proveden nejméně jednou a poté opět tak dlouho dokud, je podmínka splněna. Poslední zajímavá možnost je zadání přesného počtu opakování při každém průchodu. Tuto možnost znázorníme tečkou za operátorem loop, za kterou bude následovat počet opakování. Cyklus lze přerušit dříve pomocí operátoru break. Ten se značí vlastním fragmentem, který

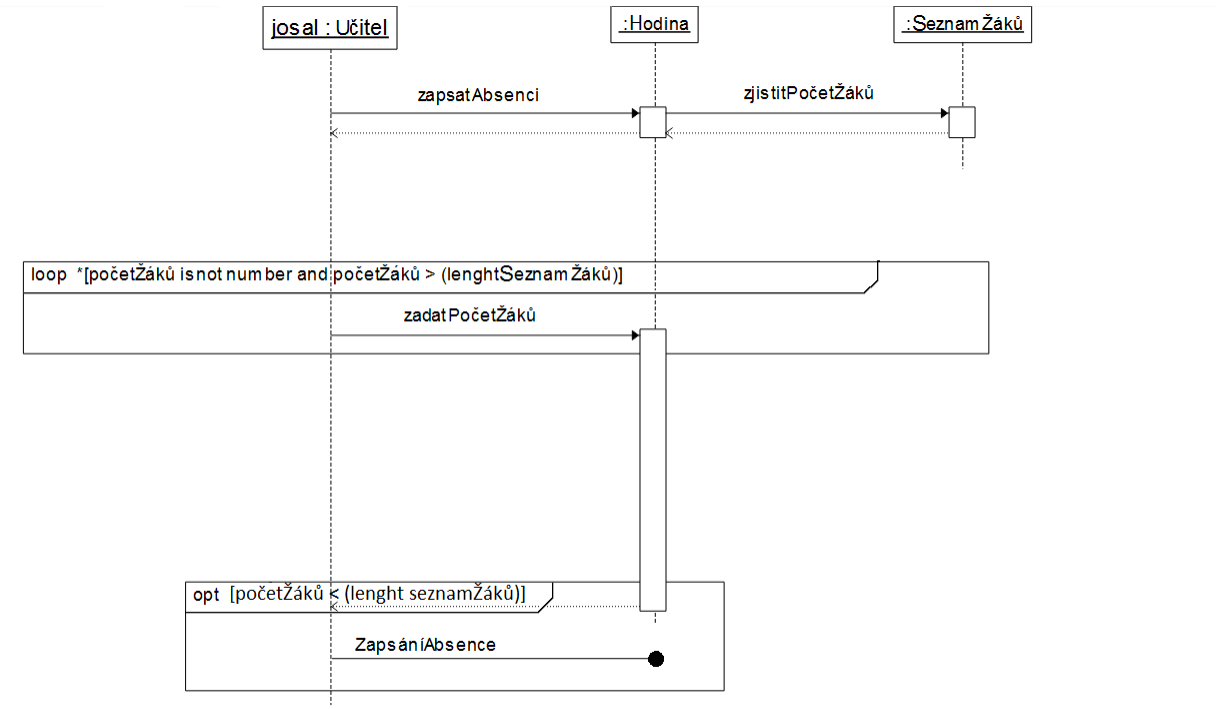
je vůči operátoru loop globální. K break se přidává do hranatých závorek podmínka. Po jejím splnění se vykonají operace ve fragmentu break a cyklus se ukončí [1].

2.3.1.5 Sondy

Pro znázornění relací extend a include se v sekvenčním diagramu požívají sondy. Sondou naznačíme, že diagram pokračuje dále v diagramu jiném. Sonda se znázorňuje čarou vedenou směrem doprava, která je zakončena buď vyplněným, nebo prázdným kolečkem. Prázdné je pro relaci include a vyplněné pro extend. K sondě se přidává ještě název případu užití, kterým diagram pokračuje [5].

2.3.2 Aplikace v návrhu třídní knihy

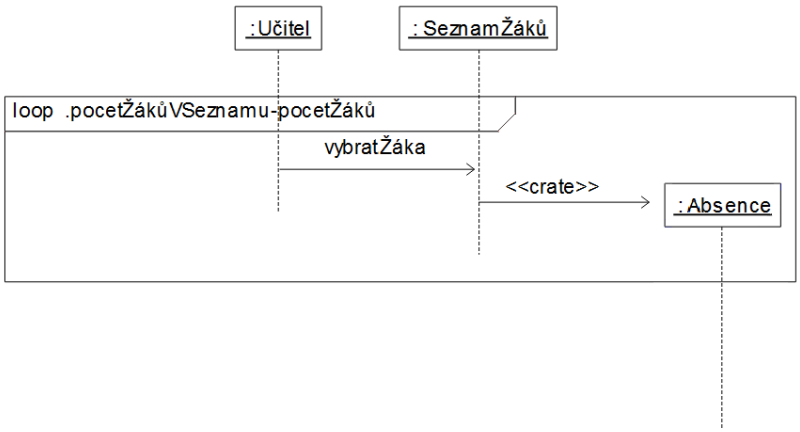
Základní principy sekvenčního diagramu, budu prezentovat na případu užití, který jsem navrhl již dříve. V tomto případu užití je použita relace extend, tudíž jsou diagramy dva. Pro alternativní scénář, jsem další diagram vypracovávat nemusel a zapracoval ho do těla hlavního diagramu.



Obrázek 25. Sekvenční diagram

Všechny akce vykonané v diagramu vyvolává původně aktér **učitel**. Což je v našem případě instance třídy **Učitel**, která nese název **josal**. Ten prvně pošle žádost k **hodině** o provedení zápisu absence. **Hodina** okamžitě pošle žádost o vrácení počtu žáků uvedených

v seznamu. Po návratu počtu žáků, hodina pošle návratovou zprávu **učiteli. Učitel** poté bude vyzván k zapsání počtu aktuálně přítomných žáků. Po tomto bodě přichází na řadu zpracování alternativního scénáře. Ten má za úkol ohlídat, aby **učitel** zadal odpovídající počet. Proto je odeslání zprávy s počtem žáků uzavřené do cyklu. V cyklu je daná podmínka, která říká, aby pokračoval tak dlouho, dokud je počet větší než počet žáků v seznamu třídy, anebo není počet žáků číslo. Program tedy bude požadovat počet tak dlouho, dokud nebude zadané správné číslo. Po skončení cyklu pokračuje program k poslední zprávě. Ta je ve fragmentu s operátorem větvení. Vyvolá se tedy pouze, je-li splněna podmínka, že je počet přítomných žáků menší, než počet žáků v seznamu. Pokud je splněna, zavolá se pomocí sondy, případ užití **ZapsáníAbsence**. Protože je s případem užití **KontrolaPočtuŽáků** spojena relací extend, byla použita sonda s vyplněným koncem. Pokud podmínka splněna nebude, program zprávu přeskočí a případ užití se ukončí.



Obrázek 26 Sekvenční diagram

Na druhém diagramu, zobrazeném na obrázku číslo 26, je znázorněn průběh rozšiřujícího případu užití **ZapsáníAbsence**. Je o poznání jednoduší než hlavní scénář. Je tu opět cyklus, tentokrát má ale předem definovaný počet průchodů. Počet průchodů tělem cyklu se rovná počtu chybějících žáků. Ten jsem zjistil odečtením počtu chybějících žáků od počtu žáků uvedených v seznamu. V těle se vždy vykonají tyto úkony. Prvně bude nabídnut seznam žáků a z něj se vybere chybějící žák. Po vybrání žáka se vytvoří nová instance třídy **Absence**, která ponese rodné číslo vybraného žáka. Po projití cyklů se případ užití ukončí a program se vrátí k hlavnímu scénáři.

2.3.3 Využití sekvenčních diagramů

Sekvenční diagramy požíváme při zobrazení spolupráce objektů, nebo tříd, kteří spolu v části programu komunikují. Část programu většinou odpovídá případu užití. Diagramy této

úrovně jsou již mířeny přímo na programátory a softwarové architekty. A vznikají až v pozdější části návrhu softwaru. Programátor z diagramu jasně vyčte jak má komunikace mezi třídami, či objekty probíhat a ulehčí mu implementaci případu užití do programu.

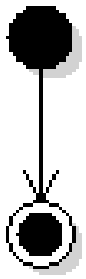
2.4 Diagramy aktivit

Diagram aktivit je velice podobný známějšímu vývojovému diagramu, ten ovšem nepatří do standardů UML. Nejzásadnější rozdíl mezi těmito dvěma diagramy je možnost paralelního běhu dvou větví programu v jednom diagramu. Tuto možnost máme právě u diagramu aktivit. Diagramy aktivit zobrazují posloupnost aktivit, které jsou v programu prováděny. *„Diagramy aktivit modelují procesy jako kolekce aktivit a přechodů mezi nimi“* [3, str. 91].

2.4.1 Struktura diagramu aktivit

2.4.1.1 Koncový a počáteční uzel

Každý diagram má naznačen schematicky svůj konec a začátek. Jedním ze způsobů naznačení počátku a konce je počáteční a koncový uzel. Začátek se značí vyplněným bodem. Konec diagramu se poté značí kruhem, který má vyplněný pouze prostřední část. Všechny ostatní elementy leží mezi těmito dvěma body. Toto značení není povinné, ale musí být zastoupené jiným spuštěním diagramu. To může být například časová událost. Diagram aktivit může mít i více než jeden začátek. V takovém případě začíná diagram ve všech bodech v jeden čas [1].



Obrázek 27 Začátek a konec diagramu

2.4.1.2 Aktivity

Jednou z nejdůležitějších součástí diagramů aktivit jsou samotné aktivity. V některých zdrojích jsem narazil spíše na označení akční uzly, nebo stavy akcí [1]. Jsou to nedělitelné jednotky, jejich provedení je velice rychlé a není možné je přerušit. To znamená, že když už jednou bude aktivita spuštěna, musí se dojít k jejímu dokončení. Aktivitu značíme obdélníkem se zaoblenými rohy. Do obdélníku se vpisuje stav aktivity. Ten se popisuje

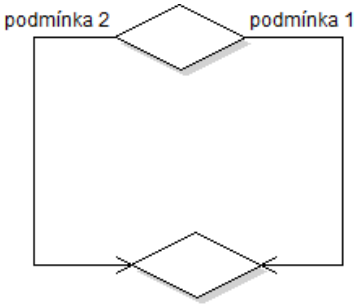
nejčastěji slovesným tvarem a vyjadřuje, co se vlastně v daném uzlu stane. Do každé aktivity smí vstoupit i vystoupit pouze jediná hrana.

Obrázek 28 Aktivita

Postup mezi jednotlivými aktivitami a ostatními elementy se znázorňuje pomocí hran neboli toku. Hrany jsou vyobrazeny jako otevřené šipky mezi elementy. Šipka vždy směřuje k následující aktivitě [5].

2.4.1.3 Rozhodovací uzel

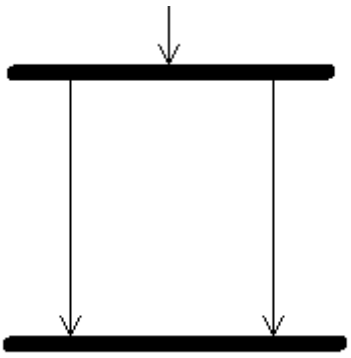
Kromě akčního uzlu je dalším důležitým prvkem uzel rozhodovací. Můžeme ho přirovnat k větvení v programovacích jazycích, i když v tomto případě ho tak neoznačujeme, protože v diagramu aktivit má větvení trochu jiný význam. Rozhodovací uzel se znázorňuje jako kosočtverec, do kterého vstupuje vždy jedna hrana. Vystupovat mohou dvě a více hran. Ke každé se přidává podmínka, která jednoznačně určí, po jaké větvi se bude dále postupovat. Nikdy nesmí nastat situace, při které by byla podmínka splněna na více větví naráz, takový stav neumí rozhodovací uzel řešit. Máme možnost použít i takzvanou else větev. Po ní se bude postupovat v případě, ve kterém by nebyla splněna žádná s ostatních podmínek. Podmínky se píší přímo nad začátek šipky, znázorňující hrany. Někteří autoři se přiklání i k zapsání podmínky ve formě jakési poznámky umístěné nad hranou. Po rozvětvení musíme větve zase sloučit do hromady a tím ukončit platnost podmínek. K tomuto účelu se požívá stejný symbol. Tentokrát do něj vstupuje více hran a vystupuje pouze jedna. Mezi rozhodnutím a ukončením rozhodnutí se musí provést všechny aktivity, které pro podmínku mají být vykonané. Rozhodovací uzly můžeme využít i pro tvorbu cyklů. Vytvoříme ho tak, že jednu hranu nasměrujeme do bodu, do kterého chceme, aby se cyklus vracel. Cyklus tedy bude probíhat tak dlouho, dokud bude podmínka platná [1].



Obrázek 29 Rozhodování

2.4.1.4 Větvení

Výše jsem se zmínil, že větvení má v tomto typu diagramu jiný význam. Díky němu můžeme znázorňovat paralelní chování. Symbol pro rozvětvení je tučná čára. Po rozvětvení běží všechny větve paralelně a vykonají se tudíž aktivity ve všech větvích. Počet větví není v UML verze 2 omezen, nejméně by měli být ovšem dvě. Do větvení může vstupovat jen jedna hrana. Větvení, ale i rozhodování, se může ve svém těle dále větvit. Ovšem musí dojít ke splynutí všech větví, ještě před sloučením původních větví. Větve se opět slučují v symbolu tlusté vodorovné čáry. V tomto místě dojde k synchronizaci všech paralelních dějů [4].



Obrázek 30 Větvení

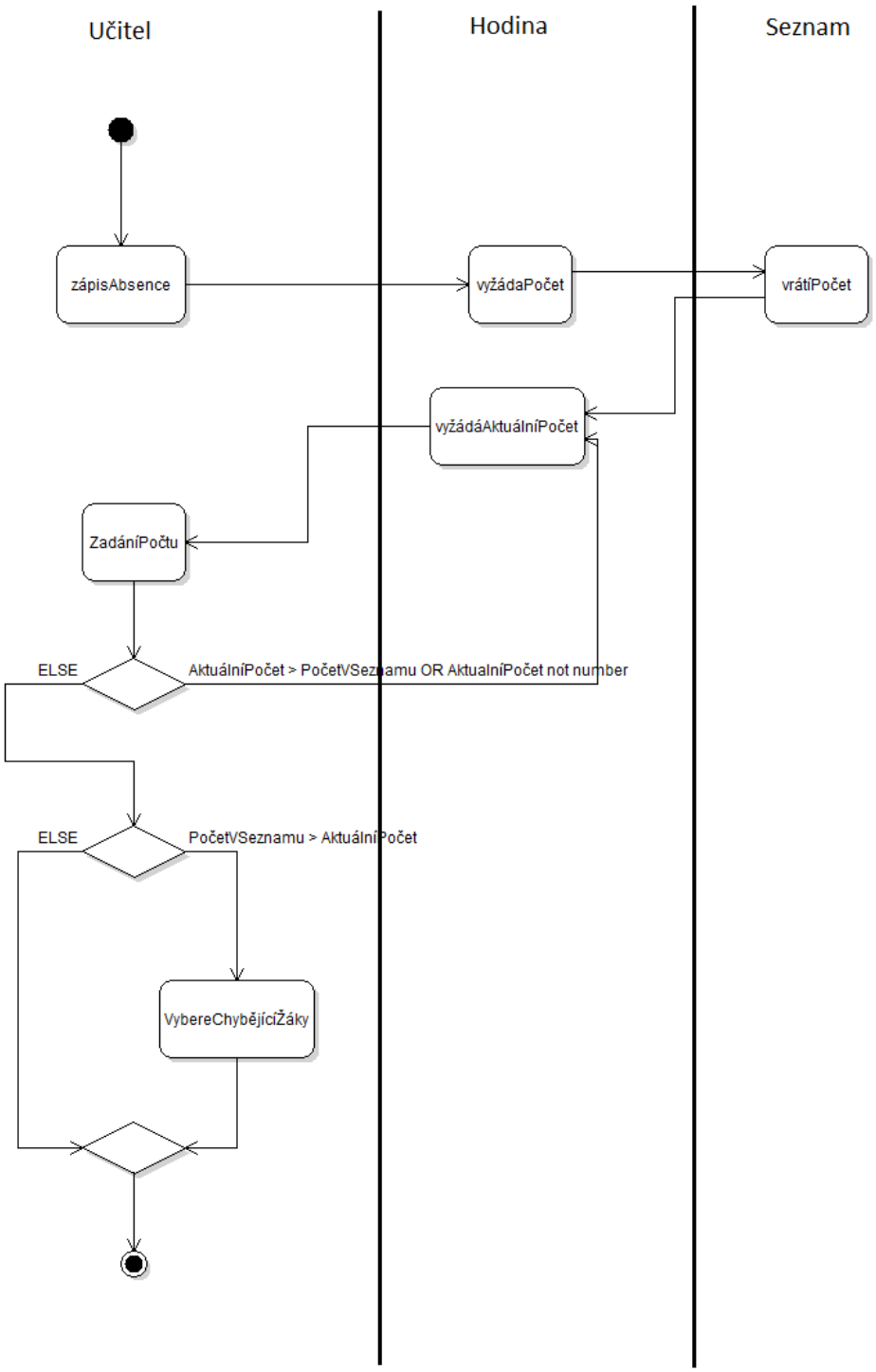
2.4.1.5 Oddíly

V tomto místě jsem již popsal základní elementy diagramu aktivit. Doposud se řešilo jenom, co se bude dít, ale nikde nebylo naznačeno, kdo je zodpovědný za jakou aktivitu. Na přiřazení kdo bude zodpovídat za jakou aktivitu je více způsobu. První možností, ne zcela přehlednou, je označit aktivitu zodpovědným objektem nebo třídou. Přehlednějším způsobem je použití takzvaných plaveckých drah. Pro použití plaveckých drah musíme všechny elementy řadit bezpodmínečně přímo pod sebe. Plavecké dráhy nám totiž umožní pomocí svislých čar rozdělit diagram na libovolný počet částí. Počet částí se bude odvíjet od počtu aktérů, objektů a tříd, kteří budou odpovědní za nějakou část diagramu. Nad diagram se doprostřed rozdělení pomocí plavecké dráhy, umisťuje název třídy nebo objektu. Hrany mohou prostupovat volně mezi plaveckými drahami. Stejně tak i větvení a rozhodování může začínat v jedné plavecké dráze a jeho větve mohou pokračovat v dráhách jiných. Sloučení větví může proběhnout opět v jiné dráze [5].

2.4.2 Aplikace v návrhu třídní knihy

Jako modelový případ opět poslouží případ užití **ZapsáníAbsence**. Tentokrát jsou vloženy všechny tři scénáře pouze do jediného diagramu. Rozšiřující scénář s vytvořením nové absence je tu vložen pouze jako jedna z aktivit.

Diagram je rozdělen pomocí plaveckých drah celkem na tři části. Diagram je pojat obecněji, proto jsou za jednotlivé dráhy zodpovědné třídy a ne přímo jejich objekty. Počáteční uzel, tedy i začátek aktivit leží u třídy **Učitel**. Díky tomu můžeme usoudit, že třída **Učitel** diagram spouští. Hned po první aktivitě, kdy vyvolá učitel spuštění zápisu absence, přechází hrana mezi plaveckými drahami do dráhy třídy **Hodina**. Ta vyžádá celkový počet žáků. Hrana tedy putuje do třídy **SeznamŽáků**, kde se provede vrácení počtu žáků a hrana se vrátí nazpět do třídy **Hodina**. Ta teď požádá o zadání počtu žáků učitele. Po zadání je pomocí prvního rozhodování ověřeno, zda bylo zadané odpovídající číslo. Pokud hodnota není číslo, nebo pokud je číslo větší jak počet žáků v seznamu, tok se navrátí k aktivitě třídy **Hodina** a znovu se vyžádá zadání počtu žáků. Pokud podmínka splněna nebude, program půjde po ELSE větvi přímo k dalšímu rozhodování. Zde bude rozhodnuto, zda je počet přítomných žáků menší než počet žáků vedených v seznamu žáků. V takovém případě půjdeme po větvi, kde bude provedena aktivita zápis absence chybějících žáků. A hrana poté bude přivedena na slučovací uzel. Sem přijde ihned po rozhodovacím uzlu i else větev. Ve slučovacím uzlu se obě hrany sloučí a pokračují do koncového uzlu, kde celý diagram končí.



Obrázek 31 Diagram aktivit

2.4.3 Využití diagramu aktivit

Diagramy aktivit vznikají, jako prezentace toku programu jednotlivými případy užití. Programátor z něj získá informaci o toku programu. Vidí aktivity, které se při průchodu části programu musí vykonat v přesném pořadí. Architekt může pomocí diagramu aktivit,

zjednodušit tok programem. Vidí tu názorně celý program a může tedy dojít ke zrušení zbytečných kroků, které byly, zavedeny ve scénáři případu užití. Protože se jedná o typ diagramu, který můžeme využít i pro sekvenční návrh programu, lze ve výuce diagram využít pro zobrazování algoritmů. Žák z něho pochopí, co se v algoritmu odehrává a může posloužit i pro identifikaci prvků, které budou hrát roli ve výpočtu časové složitosti algoritmu.

3 Nástroje pro tvorbu diagramů

Nástroje určené pro tvorbu diagramů nazýváme CASE nástroje. Jsou to programy, které mimo vytváření diagramů zvládají různé funkce týkající se modelování softwaru. Pokročilejší umí sami převést diagram do zdrojového kódu, odtud pramení označení UML jako programovacího jazyka. Anebo, naopak takzvané reversní inženýrství, pomocí něho můžeme již vytvořený zdrojový kód převést do diagramů [8].

Snažil jsem se najít program, který by mohl nejlépe vyhovovat jak mým nárokům, tak i pozdějšímu nasazení do škol. Zaměřil jsem se hlavně na nástroje s FREE licencí. Placené jsou z většiny určeny pro komerční využití a jejich pořízení je dosti nákladné. Krom toho jsou i dost složité a obsahují funkce, které by nenašly v mém případě využití. Nakonec, po prostudování nástrojů, jsem výběr zúžil na tři programy:

- Agro UML,
- Umbrello,
- Violet UML editor.

Agro UML je dobře vybavený nástroj. Obsahuje všechny hlavní diagramy a nastavení. Bohužel, mi nesedělo rozvržení pracovní plochy, příliš velká část obrazovky je zabrána panelem s nastavením, který lze sice zmenšit, ale protože ho potřebujete prakticky pořád, není to také šťastné řešení. Bohužel, ale poté zbývá velmi malá plocha na tvorbu samotných diagramů. Přidání základních prvků je jednoduché a intuitivní, ale například zadání atributu ke třídě už je zbytečně komplikované. Vše se musí zadávat přes panel s nastavením a k jednotlivým položkám se musí složitě dopracovat. Při tom, v ostatních mnou zkoušených programech, stačí dvojklik na objekt, který chceme upravit. Při tvorbě sekvenčních diagramů došlo ovšem k opačnému problému. Základní tvorba byla jednoduchá, některé prvky se doplňovaly automaticky. Bohužel, zde naopak spoustu důležitých prvků scházelo a po automatickém doplnění se nedaly upravovat. Proto mi konečná podoba, mnou vytvořeného diagramu, nevyhovovala a opět jsem šáhl po jiném nástroji. Největší slabinu tohoto nástroje vidím tedy v jeho přílišné složitosti. Ta sice neplatí u všech typů diagramů, ale zbylé jsou zase příliš ořezané a přijdou mi nepoužitelné. Agro UML má ve specifikacích uvedeno, že je zaměřen hlavně pro výukové účely, ale s tímhle tvrzením já nesouhlasím, při osvojování UML mi lépe posloužily jiné nástroje.

Umbrello je o poznání přehlednější. Rozvrhnutí pracovní plochy je mnohem méně chaotické. Nastavení je jednoduché a přehledné a při tom lze nastavit všechny důležité věci. A to včetně barevného schématu diagramů a fontu písma. Obsahuje možnosti tvorby všech

základních UML diagramů. Při tvorbě mi chybělo pouze pár nástrojů. Největší problém tohoto programu je v jeho stabilitě a složitější instalaci. Program je tvořen na unixové jádro. Díky tomu je na rozšířenějších Windows jeho instalace sice možná, ale komplikovaná. Při testování došlo několikrát k pádu programu a tím ke ztrátě neuložené práce. Tento problém se dá vyřešit nastavením automatického ukládání, ale přesto docházelo ke komplikaci práce. To byl jediný důvod, proč jsem od používání tohoto nástroje nakonec odstoupil. Vrátil jsem se k němu při tvorbě sekvenčních diagramů. Z mnou zkoušených nástrojů nabízel, co se sekvenčních diagramů týče, největší počet prvků a nejlépe se s nimi zacházelo. Nakonec se projevil ovšem další problém a to je nefunkční export diagramu do obrázku.

Violet UML je z těchto tří programů nejjednodušší. Jedná se o prostý editor diagramů. Jeho uživatelské prostředí je velice intuitivní. Veškeré prvky jsou na panelu k rychlému zvolení a jejich nastavování se provádí dvojklikem na objekt. Jediné nástroje, které mi při vytváření diagramů aktivit, případů užití a tříd chyběly, byly vytvoření hranice systému v diagramu případů užití a plavecké dráhy v diagramu aktivit. Ty konkrétně chyběl i v Umrello. Největší problém byl s vytvořením sekvenčního diagramu. Zde má Violet UML velkou slabinu. Chybí zde většina prvků a mezi nimi i ty úplně základní, jako je například asynchronní zpráva. Violet UML editor je sympatický v tom, že nevyžaduje instalaci. Jedná se pouze o spustitelný exe soubor. Violet je tedy jednoduchý, snadno obsluhovatelný program, který pracuje bez problémů. Proto je velice vhodný pro tvorbu jednodušších diagramů a myslím, že pro případné nasazení do školství se jedná o jeden z nejlepších nástrojů. Pouze při tvorbě sekvenčních diagramů je potřeba šáhnout po nějakém z konkurenčních nástrojů.

V tabulce číslo 5 jsem shrnul nejdůležitější aspekty, kterých jsem si u nástrojů všímal a obodoval je. Nejvyšší známku dostal vždy, z mého pohledu, nejlepší nástroj. Nejvyššího počtu bodů celkem dosáhl Violet UML.

	AgroUML	*Violet UML*	*Umbrello*
Instalace	2	3	1
Stabilita a rychlost	2	3	1
Uživatelské prostředí	1	3	2
Množina diagramů	2	1	3
Úprava diagramů	1	3	3
Vybavenost diagramů	2	1	3
Export do obrázku	3	3	1
Celkový počet bodů:	13	17	14

Tabulka 5 Porovnání nástrojů

Závěr

V práci jsem popsal čtyři nejdůležitější a podle mého názoru nejužitečnější diagramy. Byla popsána jejich struktura, která se následně demonstrovala na modelovém příkladu třídní knihy. Příklad byl zvolen tak, aby na něm bylo možné popsat většinu mnou rozebíraných funkcí. Vytvořil jsem celý diagram tříd a případů užití pro třídní knihu, ze kterého jsem vybral nejzajímavější případy užití. Ty byly v práci popsány a dále použity při tvorbě sekvenčního diagramu a diagramu aktivit. Zbylé scénáře, z diagramu případů užití, jsem vložil spolu s diagramy aktivit a sekvenčními diagramy do příloh.

U jednotlivých typů diagramů jsem popsal jejich možné využití v praxi při tvorbě návrhu a implementace software. Diagramy jsem vylíčil přesně v tom pořadí, ve kterém by vznikali ipři běžném návrhu softwaru, tedy kromě diagramů aktivit a sekvenčních diagramů, které mohou vznikat souběžně. Během tvorby diagramů jsem průběžně testoval tři nástroje. Byli velice rozdílné, ale z mého porovnání nejlépe vyšel Violet UML, ve kterém je vytvořena finální podoba většiny diagramů. Violet UML mi vyhovoval hlavně díky poměru jednoduchosti ovládání a funkcí, které nástroj ovládá. I když k vytvoření složitějších diagramů by nebyl příliš vhodný, kvůli absenci některých funkcí, jako je například sdílení tříd a operací mezi diagramy.

V práci se podařilo ukázat, že UML je ve své podstatě jednoduchý nástroj, který může být velice přínosný v jednotlivých fázích návrhu software. Sám jsem si ověřil, že základy tohoto jazyka lze ovládnou poměrně lehce. Jsem tedy toho názoru, že jazyk UML by měl více vstoupit do povědomí odborné veřejnosti a jeho využívání by se vzhledem k jeho nesporným výhodám mělo těšit větší oblibě, než je tomu dnes.

Seznam literatury

[1] ARLOW, Jim a Ila NEUSTAND. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. 2., aktualiz. a dopl. vyd. Brno: Computer Press, 2007. 567 s. ISBN 978-80-251-1503-9.

[2] BOOCH, Grady. *The unified modeling language user guide*. 2nd ed. Upper Saddle River: Addison-Wesley, 2005. ISBN 03-212-6797-4.

[3] CIMBAĽÁK, Michal, Vašek MARTIN, Olga VRCHOTOVÁ, Radek ZIMÁK, PROCHÁZKA a KUKAČKA. *Přehled nástrojů CASE na tuzemském trhu [online]*. [cit. 2014-04-12]. Dostupné z: http://panrepa.org/CASE/jaro2009/case_jaro2009.pdf

[4] FOWLER, Martin. *Destilované UML*. 1. vyd. Praha, 2009, 173 s. Knihovna programátora (Grada). ISBN 978-80-247-2062-3.

[5] KANISOVÁ, Hana a Miroslav MÜLLER. *UML srozumitelně*. 2., aktualiz. vyd. Brno: Computer Press, 2006. 176 s. ISBN 80-251-1083-4.

[6] KRAVAL, Ilja. *Jak se pozná, dobrý Use Case model?* [online]. 2010 [cit. 2014-04-21]. Dostupné z: http://www.objects.cz/clanky/clanek78/clanek78.pdf

[7] LARMAN, Craig. *Applying UML and patterns: introduction to object-oriented analysis and design and iteractive development*. 3rd ed. New Jersey: Prentice-Hall, 2005, 703 s. ISBN 01-314-8906-2.

[8] PEROUTKA, Lukáš, Jan JUREČKA, Daniel KOBRLE a Václav PODLIPNÝ. *Přehled CASE nástrojů na tuzemském trhu* [online]. [cit. 2014-03-11]. Dostupné z: www.panrepa.org/CASE/podzim2011/CASE_podzim2011.pdf

[9] ROSENBERG, Doug a Kendall SCOTT. *Applying use case driven object modeling with UML: an anotated e-commerce example*. Boston: Addison-Wesley, c2001, xiv, 153 p. ISBN 02-017-3039-1.

[10] RUMBAUGH, James, Ivar JACOBSON a Grady BOOCH. *The unified modeling language reference manual*. 2nd ed. Boston: Addison-Wesley, c2005, xx, 721 p. ISBN 03-212-4562-8.

[11] SCHMULLER, Josef. *Myslíme v jazyku UML: knihovna programátora*. 1. vyd. Praha: Grada Publishing, 2001. 359 s. Myslíme v --. ISBN 80-247-0029-8.

[12] STUMPF, Robert a Lavette C TEAGUE. *Object-oriented systems analysis and design with UML: introduction to object-oriented analysis and design and iteractive development*. 3rd ed. Upper Saddle River, N.J.: Pearson/Prentice Hall, c2005, xviii, 428 p. ISBN 01-314-3406-3.

[13] *Unified Modeling Language (UML)* [online]. ©1997 - 2014 [cit. 2014-04-16]. Dostupné z: www.uml.org

[14] VONDRÁK, Ivo. *Úvod do softwarového inženýrství* [online]. 1. 1. 2002 [cit. 2014-04-17]. Dostupné z: http://vondrak.cs.vsb.cz/download/Uvod_do_softwaroveho_inzenyrstvi.pdf

[15] WARMER, Jos B a Anneke G KLEPPE. *The object constraint language: getting your models ready for MDA*. 2nd ed. Boston: Addison-Wesley, c2003, xxvi, 206 s. Addison-Wesley object technology series. ISBN 03-211-7936-6.

Seznam zkratek

UML - Unified Modeling Language

Seznam příloh

Příloha číslo 1 – Případy užití

Příloha číslo 2 – Sekvenční diagramy

Příloha číslo 3 – Diagramy aktivit

Příloha číslo 1 – Případy užití

Tabulka přílohy 1 obsahuje případ užití **OmluvaAbsance**, ve kterém je popsáno omlouvání absence Třídním učitelem.

Případ užití: OmluvaAbsence
ID: 2
Hlavní aktér: Třídní učitel
Vstupní podmínky: Třídní učitel je přihlášen
Hlavní scénář: 1. Třídní učitel zvolí omluva absence 2. Systém vyzve třídního učitele k výběru hodiny 3. Třídní učitel zvolí hodinu 4. Systém vyzve k vybrání absence 5. Absence se vymaže 6. Učitel ukončí a uloží omluvu absence
Výstupní podmínky:
Alternativní scénář:

Tabulka přílohy 1 ID:2

Tabulka přílohy 2 obsahuje případ užití **ÚpravaSeznamuŽáků**, ve kterém je popsán postup při upravování seznamu žáků třídním učitelem.

Případ užití: ÚpravaSeznamuŽáků
ID: 3
Hlavní aktér: Třídní učitel
Vstupní podmínky: Třídní učitel je přihlášen
Hlavní scénář: 1. Třídní učitel zvolí úpravu seznamu žáků 2. Třídní učitel vybere mezi přidáním a odstraněním žáka 3. Když třídní učitel zvolí odstranění 3. a Vybere žáka ze seznamu 3. b Odstraní žáka 4. Když třídní učitel zvolí přidání 4. a Vyplní údaje a bude vytvořen nový žák 5. Učitel ukončí úpravu seznamu
Výstupní podmínky:
Alternativní scénář:

Tabulka přílohy 2 ID:3

Tabulka přílohy 3 obsahuje případ užití **ZápisHodiny**, ten popisuje postup při zapisování nové hodiny do třídní knihy.

Případ užití: ZápisHodiny
ID: 4
Hlavní aktér: Učitel
Vstupní podmínky: Učitel je přihlášen
Hlavní scénář: 1. Učitel zvolí zapsání hodiny 2. Učitel vytvoří novou hodinu 3. Systém doplní číslo hodiny 4. Učitel vyplní téma hodiny 5. Ukončení
Výstupní podmínky:
Alternativní scénář:

Tabulka přílohy 3 ID:4

Tabulka přílohy 4 obsahuje případ užití **ZápisPoznámky**, ve kterém se popisuje postup při přidávání poznámky k hodině.

Případ užití: ZápisPoznámky
ID: 5
Hlavní aktér: Učitel
Vstupní podmínky: Učitel je přihlášen
Hlavní scénář: 1. Učitel zvolí zapsání poznámky 2. Systém vyzve k vybrání hodiny 3. Učitel vybere hodinu 4. Učitel provede zápis poznámky 5. Ukončení
Výstupní podmínky:
Alternativní scénář:

Tabulka přílohy 4 ID:5

Tabulka přílohy 5 obsahuje hlavní scénář případu užití **SpuštěníProhlížení**, ten ovládá jak učitel, tak i ředitel. Případ užití pokračuje scénářem **Prohlížení**, který je navázán relací include.

Případ užití: SpuštěníProhlížení
ID: 6
Hlavní aktér: Učitel, Ředitel
Vstupní podmínky: Učitel, Ředitel je přihlášen
Hlavní scénář: 1. Hlavní aktér spustí prohlížení 2. Systém vybere třídy, ke kterým má aktér práva 3. Systém vyzve k vybrání třídy 4. Hlavní aktér vybere třídní knihu 5. Dále viz případ užití ID = 10
Výstupní podmínky:
Alternativní scénář:

Tabulka přílohy 5 ID:6

Tabulka přílohy 6 obsahuje případ užití **PřihlášeníDoTřídy**, kde se popisují kroky, které nastanou po přihlášení žáka do třídní knihy.

Případ užití: PřihlášeníDoTřídy
ID: 7
Hlavní aktér: Žák
Vstupní podmínky: Žák je přihlášen
Hlavní scénář: 1. Systém přiřadí třídní knihu 2. Pokračuje scénářem ID=10
Výstupní podmínky:
Alternativní scénář:

Tabulka přílohy 6 ID:7

Tabulka přílohy 7 obsahuje případ užití **ZápisHospitace**, kde se popisují kroky, které povedou k zapsání hospitace k hodině.

Případ užití: ZápisHospitace
ID: 8
Hlavní aktér: Ředitel
Vstupní podmínky: Ředitel je přihlášen
Hlavní scénář: 1. Ředitel zvolí zápis hospitace 2. Systém vyzve k vybrání hodiny 3. Ředitel vybere hodinu 4. Přidání záznamu o hospitaci k hodině 5. Ukončení
Výstupní podmínky:
Alternativní scénář:

Tabulka přílohy 7 ID:8

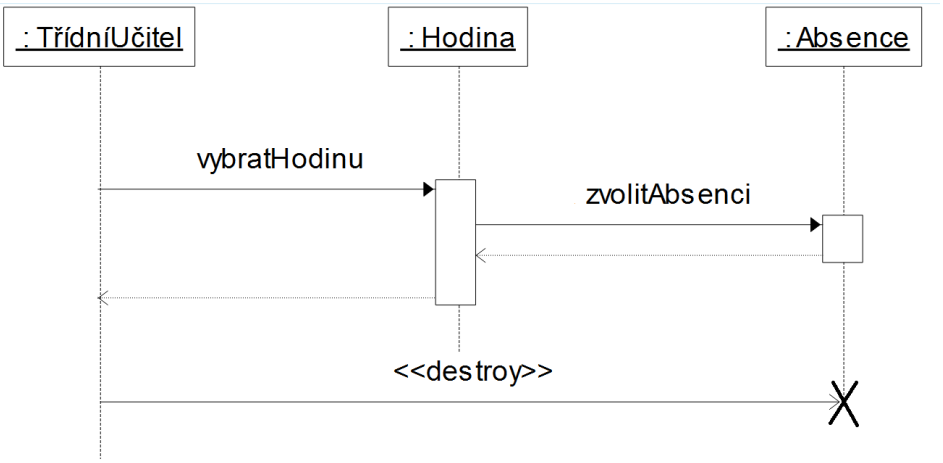
Tabulka přílohy 8 obsahuje případ užití **Prohlížení**. Zde jsou vypsány kroky, které budou provedeny při prohlížení hodin.

Případ užití: Prohlížení
ID: 10
Hlavní aktér: Ředitel/Učitel/Žák
Vstupní podmínky: Ředitel je přihlášen
Hlavní scénář: 1. Aktér vybere hodinu 2. Zobrazí se číslo hodiny, téma, poznámka, absence a hospitace
Výstupní podmínky:
Alternativní scénář:

Tabulka přílohy 8 ID:10

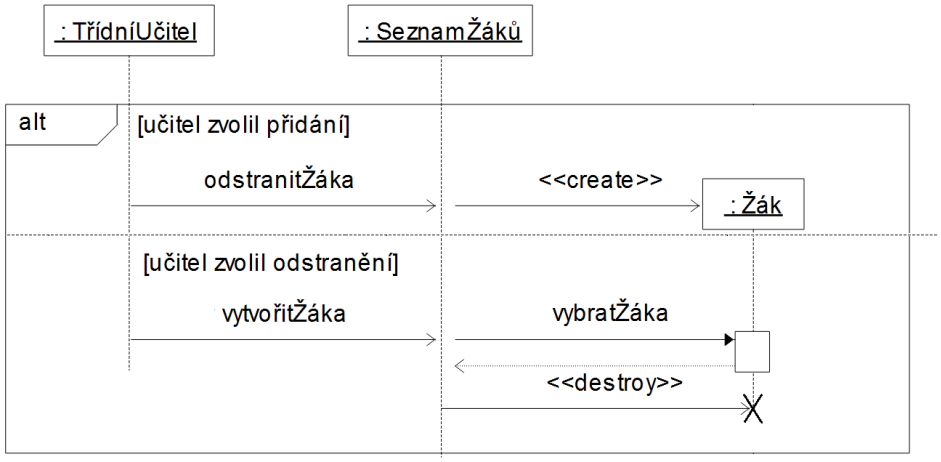
Příloha číslo 2 – Sekvenční diagramy

Obrázek příloh 1 zobrazuje sekvenční diagram pro případ užití **OmluvaAbsence.**



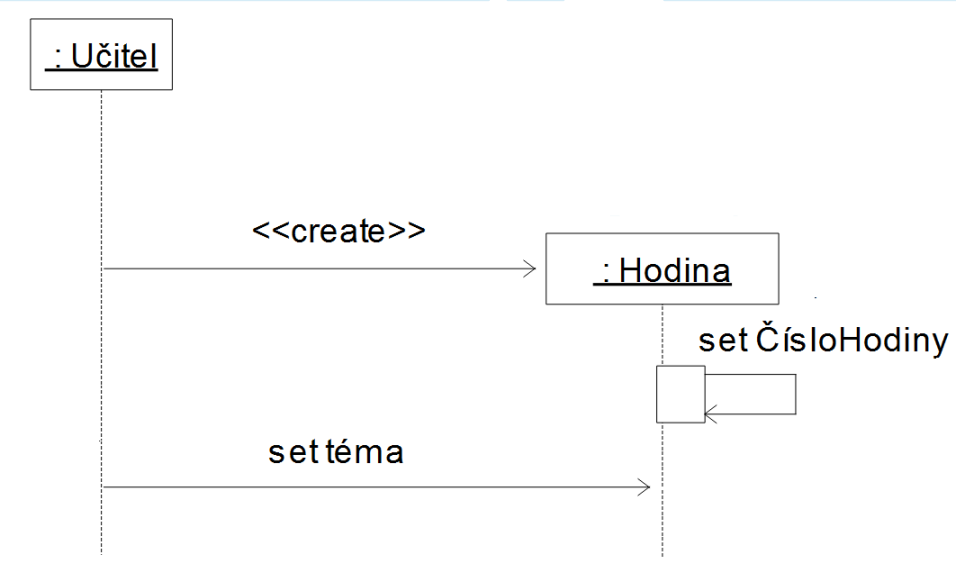
Obrázek příloh 1Pro případ užití ID:2

Obrázek příloh 2 zobrazuje sekvenční diagram pro případ užití **ÚpravaSeznamuŽáků.**



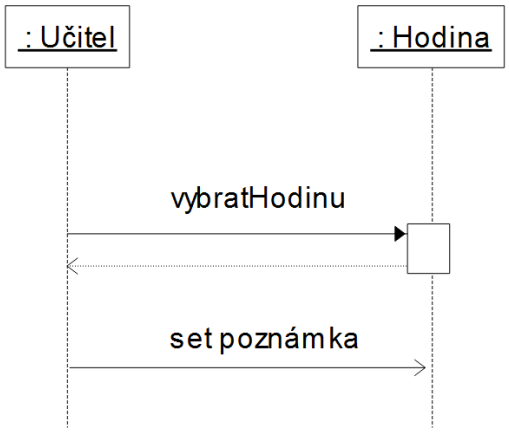
Obrázek příloh 2 Pro případ užití ID:3

Obrázek příloh 3 zobrazuje sekvenční diagram pro případ užití **ZápisHodiny.**



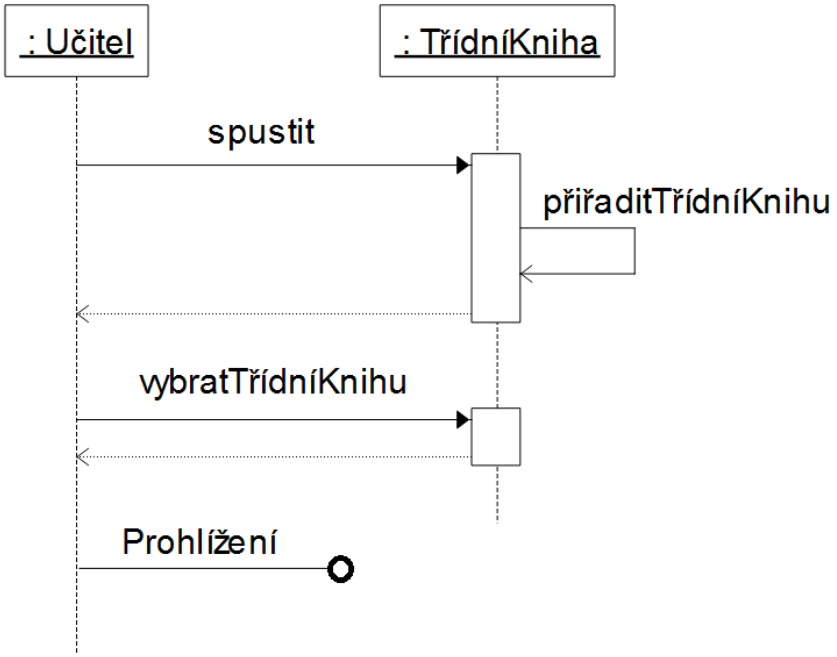
Obrázek příloh 3 Pro případ užití ID:4

Obrázek příloh 4 zobrazuje sekvenční diagram pro případ užití **ZápisPoznámky.**



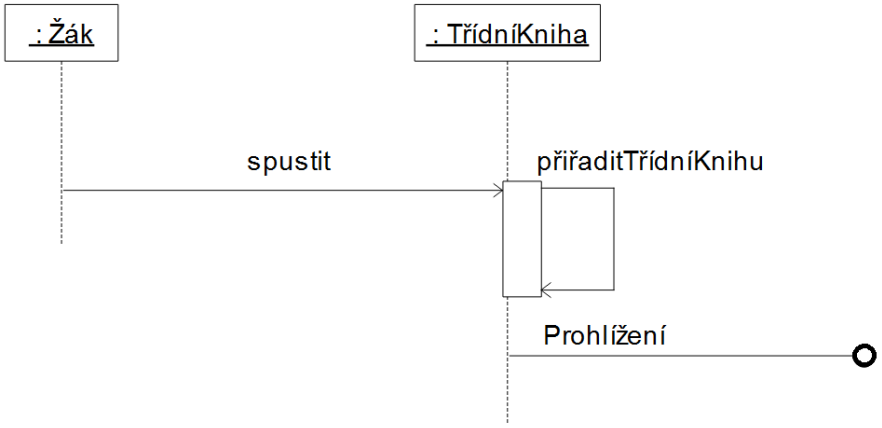
Obrázek příloh 4 Pro případ užití ID:5

Obrázek příloh 5 zobrazuje sekvenční diagram pro případ užití **SpuštěníProhlížení.**



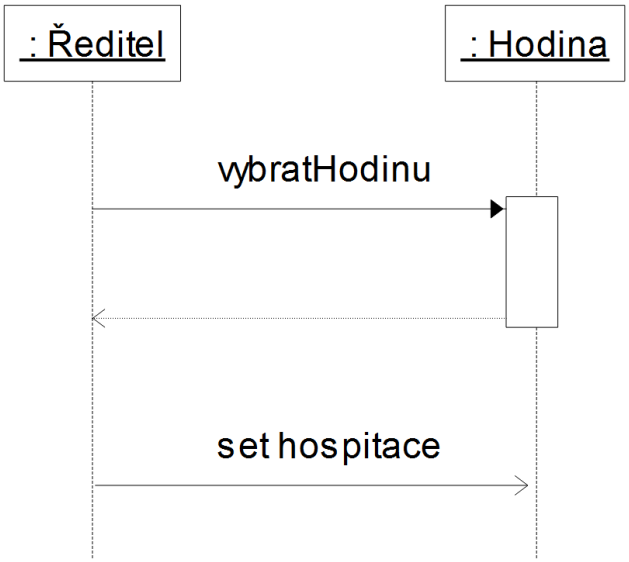
Obrázek příloh 5 Pro případ užití ID:6

Obrázek příloh 6 zobrazuje sekvenční diagram pro případ užití **PřihlášeníDoTřídy.**



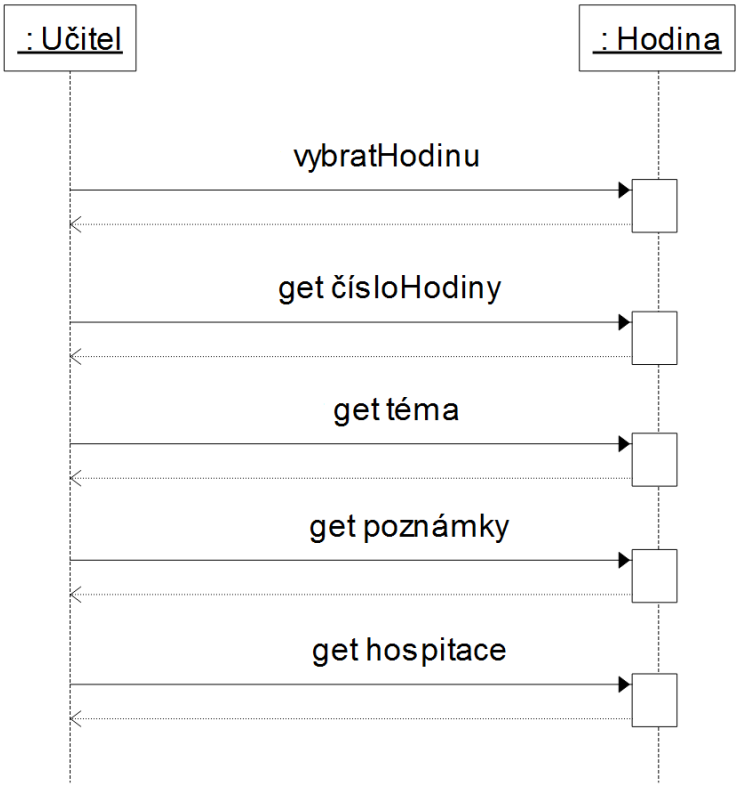
Obrázek příloh 6 Pro případ užití ID:7

Obrázek příloh 7 zobrazuje sekvenční diagram pro případ užití **ZápisHospitace.**



Obrázek příloh 7 Pro případ užití ID:8

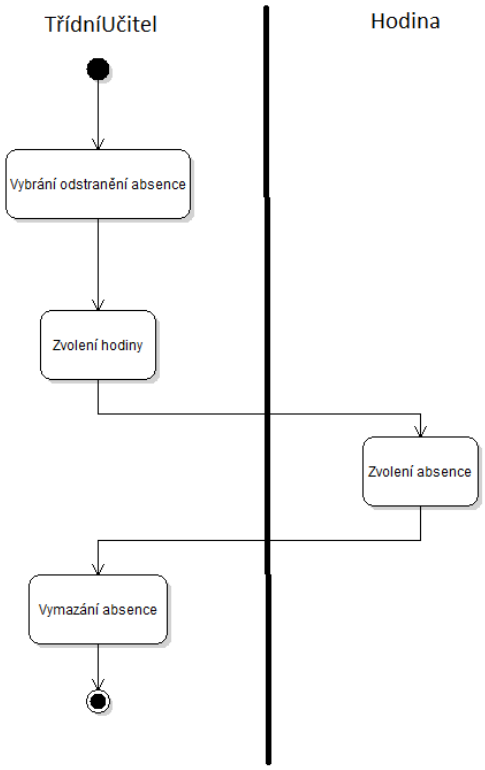
Obrázek příloh 8 zobrazuje sekvenční diagram pro případ užití **Prohlížení.**



Obrázek příloh 8 Pro případ užití ID:10

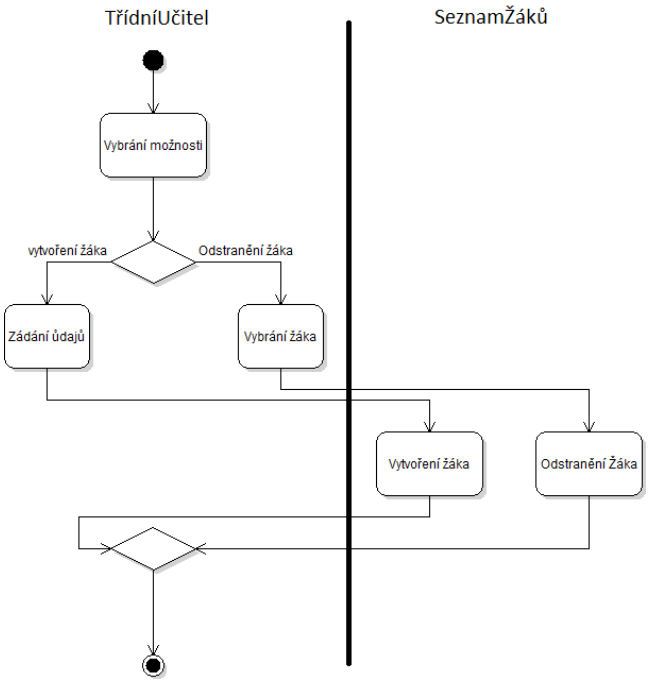
Příloha číslo 3 – Diagramy aktivit

Obrázek příloh 9 zobrazuje diagram aktivit pro případ užití **OmluvaAbsence**.



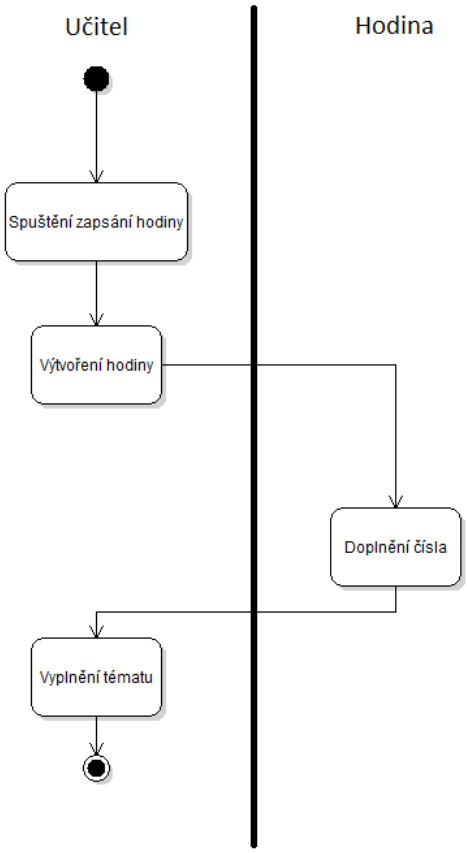
Obrázek příloh 9 Pro případ užití ID:2

Obrázek příloh 10 zobrazuje diagram aktivit pro případ užití **ÚpravaSeznamuŽáků**.



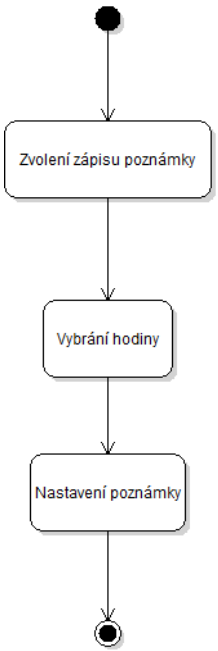
Obrázek příloh 10 Pro případ užití ID:3

Obrázek příloh 11 zobrazuje diagram aktivit pro případ užití **ZápisHodiny**.



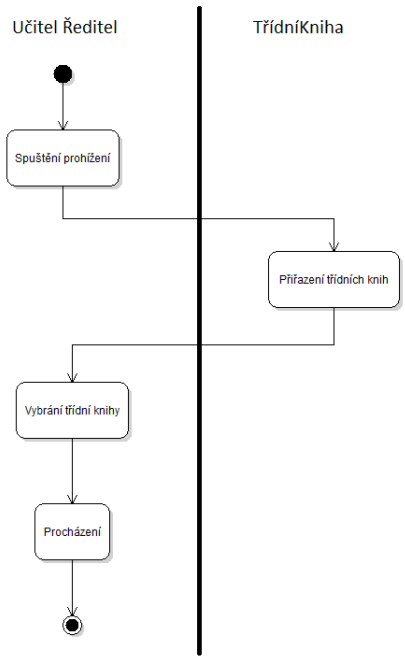
Obrázek příloh 11 Pro případ užití ID:4

Obrázek příloh 12 zobrazuje diagram aktivit pro případ užití **ZapsáníPoznámky**. V diagramu účinkuje jediný aktér a to **učitel**.



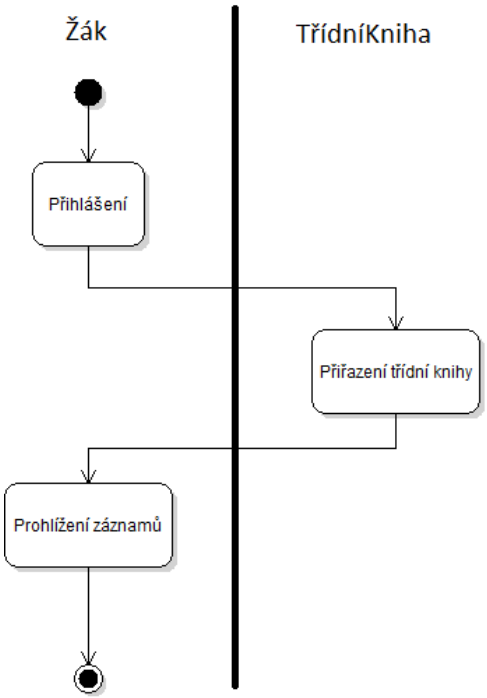
Obrázek příloh 12 Pro případ užití ID:5

Obrázek příloh 13 zobrazuje diagram aktivit pro případ užití **SpuštěníProhlížení**.



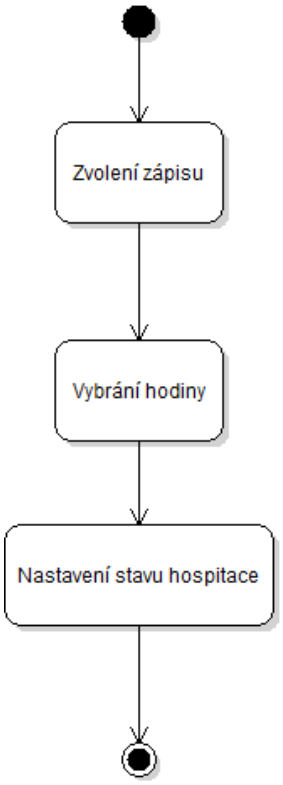
Obrázek příloh 13 Pro případ užití ID:6

Obrázek příloh 14 zobrazuje diagram aktivit pro případ užití **PřihlášeníDoTřídy**.



Obrázek příloh 14 Pro případ užití ID:7

Obrázek příloh 12 zobrazuje diagram aktivit pro případ užití **ZápisHospitace**. V diagramu účinkuje jediný aktér a to **ředitel**.



Obrázek příloh 15 Pro případ užití ID:8

ANOTACE

Jméno a příjmení:	Petr Felner
Katedra:	Katedra technické a informační výchovy
Vedoucí práce:	Mgr. Jan Kubrický, Ph.D.
Rok obhajoby:	2014

Název práce:	Využití grafického programovacího jazyka UML při návrhu software školní třídní knihy
Název v angličtině:	The Use of the Graphical Language UML in the design of software school class books
Anotace práce:	Tato práce se zabývá grafickým programovacím jazykem UML, představují se zde čtyři základní diagramy využíváné při návrhu software. V práci se dále popisuje začlenění jednotlivých diagramů do fází vývoje software a jejich využití v praxi při jednotlivých fází. Všechny diagramy jsou vysvětlovány na jednom příkladu, kterým je elektronická třídní kniha.
Klíčová slova:	UML, objektově orientované programování, OPP, Unified Modeling Language, diagram případů užití, diagram tříd, sekvenční diagram, diagram aktivit
Anotace v angličtině:	This thesis is focused on Graphical Programming Language UML. There are presented four basic diagrams which are used during designing of software. In the thesis there are described integrations of the single diagrams into phases of software development and its use in praxis during single phases. All diagrams are explained by one example - the example of electronic classbook.

Klíčová slova v angličtině:	UML, unified modeling language, object oriented programming, OPP, use case diagram, class diagram, sequence diagram, aktivity diagram
Přílohy vázané v práci:	Příloha číslo 1 – Případy užití Příloha číslo 2 – Sekvenční diagramy Příloha číslo 3 – Diagramy aktivit
Rozsah práce:	55 stran
Jazyk práce:	Český jazyk