Dokumentace k projektu z předmětu KIV/ZSWI

MediTab

Daniel Švarc David Pivovar Jan Beneš
Lenka Rychtářová Zdeněk Valeš

10.5. 2015

1 Zadání

Aplikaci budou používat zdravotní sestry, kterým se zjednoduší manipulace s daty o pacientech. Po přihlášení do Oracle databáze se objeví hlavní část programu, kde bude zobrazen aktuální datum a hospitalizovaní pacienti. Do softwaru spadají tři části aplikace.

První část produktu se zabývá bilancí tekutin konkrétního pacienta. Zdravotní sestra bude v průběhu dne zaznamenávat množství tekutin, konkrétně se jedná o přísun výživy, diurézu, sondy, drény, ultrafiltrace a jiné ztráty a přísuny. Na konci dne budou jednotlivé hodnoty sečteny a zapsány do databáze. Výhodou je automatické sčítání bilancí tekutin a usnadněný zápis dat. Informace se nebudou muset zaznamenávat na papír, ale přímo do tabletu. Program bude propojený s databází, kam se informace z tabletu automaticky zapíší. Předejde se tím dvojitému zápisu na papír a do databáze.

Druhou částí je seznam invazivních přístupů zavedených pacientovi. Patří k nim jednotlivé typy drénů a katetrů. Je zde možné vybrat typ invazivního přístupu, umístění, datum zavedení a hloubku. Na obrazovce bude vidět počet dní od zavedení. Zdravotní sestra bude mít možnost stisknout tlačítko vyměnit nebo zrušit, čímž se počet dní automaticky resetuje. V aplikaci bude také možné zadat nový invazivní přístup. Zmíněné aktivity se zapíší do databáze.

Třetí a nejdůležitější částí softwaru budou záznamy o ordinovaných lécích. Na tabletu budou informace o předepsaných medikamentech z databáze. Zdravotní sestra uvidí název léku, množství a časy užití. V případě jiného podání se odlišnosti zpětně promítnou do databáze, ve které je doktor zhlédne.

Produkt je vytvořen s cílem ušetření času při zápisu informací o invazivních přístupech a bilancích tekutin. Dojde k okamžitému propojení dat s databází, předejde se ručnímu zadávání. Omezí se chybovost při přepisování a bude lepší dostupnost dat.

2 Analýza

Program je určen do ostrého provozu, během kterého vznikne velké množství požadavků na změny a případná rozšíření, je tedy nutné aplikaci tvořit co možná nejvíce rozšiřitelnou a udržovatelnou. Z těchto důvodů jsme se při analýze snažili co možná nejvíce oddělit jádro programu a GUI.

Jak je popsáno v zadání, systém bude mít tři stěžejní části - bilanci tekutin, invazivní přístupy a medikační kartu. Kvůli přehlednosti jsou tyto části rozděleny do třech zvláštních (a vzájemně nezávislých) balíků a jediné pojítko mezi nimi je vazba na část s databází. Tento přístup umožňuje jednotlivé části vyvíjet a udržovat samostatně, což je při rozsahu aplikace nutností.

Aplikace ke svojí funkčnosti potřebuje několik konfiguračních údajů. Pro tyto údaje jsme zvolili načítání z config souboru, což je formát založený na XML, pro který je v .NETu nativní podpora.

Diagram tříd je obsažen v příloze.

2.1 Přístup k aplikaci

Do systému nemůže mít přístup každý a je nutné ověřovat totožnost uživatele. Každý zaměstnanec v nemocnici má svůj login a heslo, které bude sloužit k přihlášení do aplikace. Tyto údaje jsou následně použity při komunikaci s databází. Tablety jsou navíc vázány na pracoviště a přístup mají pouze sestry, které mají na tomto pracovišti pracovní úvazek.

První část ověření probíhá samotným připojením k databázi, pokud jsou zadané údaje špatné, připojení selže. Druhou část tvoří dotaz na databázi, který ověří, zda má zadaný uživatel pracovní úvazek na správném pracovišti.

2.2 Bilance tekutin

Bilance tekutin má uchovávat příjmy a výdaje tekutin pacienta. Tekutin k uchování je celkem 12. Příjem a výdej tekutin se odečítá během dne několikrát, ke každé tekutině tedy během dne bude několik hodnot. Námi zvolená implementace představuje pole s 12 dynamickými seznamy.

Bylo by možné volit pouze pole 12 double hodnot, ke kterým by se vždy přičetl naměřený údaj. Nicméně tento přístup neumožňuje detailnější sledování příjmů a výdajů během dne, které by v budoucnu mohlo být potřeba.

V databázi celková bilance tekutin odpovídá právě jednomu záznamu, který má klíč složen z ID pacienta a data dne, kdy byl stav tekutiny změřen. Při poslání do databáze se tedy udělá 12 součtů (z každého seznamu 1) a ty se pak společně s ID pacienta a datem změření odešlou do databáze.

2.3 Invazivní přístupy

Každý pacient může mít zaveden libovolný počet různých invazivních přístupů. Každý přístup má určitě vlastnosti (jméno, umístění, hloubka, datum zavedení..).

Pro invazivní přístup jsme tedy zvolili implementaci podle návrhového vzoru přepravka. Tyto přístupy jsou pak uchovávány v dynamickém seznamu.

Invazivnímu přístupu v databázi odpovídá právě jeden záznam v tabulce. Tyto záznamy o invazivních přístupech v databázi zůstávají i po vyjmutí, takže při vyjmutí přístupu se do databáze pouze zapíše datum, kdy k vyjmutí došlo.

2.4 Medikační karta

Medikační karta je ze třech funkčních částí nejdůležitější a má největší prioritu. Oproti ostatním je také o něco složitější.

Medikační karta slouží k evidenci podávaných léku u pacienta. Navíc je k každému léku potřeba rozepsat dávkování s přesností na hodinu. Na opravdové medikační kartě je tabulka s 24 sloupci - pro každou hodinu jeden.

Pro implementaci léku jsme opět zvolili návrhový vzor přepravka. Medikační karta pak obsahuje dynamický seznam těchto léků.

Dávkování je v databázi, i mezi doktory předepisováno pomocí určitých výrazů (2-1-1, 1-1, po 12 hod..). Dávka může může v každou hodinu nabýt právě jednoho ze tří stavů - lék není na danou hodinu předepsán, lék je předepsán, lék byl podán. Mimo to, je třeba ke každé dávce ukládat množství, pokud nějaké je.

K implementaci těchto tří stavů jsme zvolili třídu založenou na výčtovém typu - samotný výčtový typ nám právě kvůli potřebě držet informaci o množství nevyhovoval. Dávkování je po rozparsování výrazu uloženo v poli délce 24, takže je v GUI lehce zobrazitelné.

V databázi existují na podané léky dvě tabulky. V jedné jsou uloženy naordinované léky ke každému pacientovi, v druhé pak jednotlivá konkrétní podání.

2.5 Databáze

Ve fakultní nemocnici používají Oracle databázi. V .NET sice existuje knihovna, která slouží ke komunikaci s MySQL i Oracle databází, nicméně oficiální stanovisko Microsoftu doporučuje použití knihovny, kterou Oracle dodává. Jedná se o knihovnu Oracle Data Access Components a její instalace je nutná i na koncovém zařízení (tedy na tabletu).

Původní návrh databáze předpokládal předávání odkazu na existující instanci. Postupem času jsme však zvolili pohodlnější způsob. Třída komunikující s databází je vytvořena jako veřejná statická proměnná v hlavní třídě (inicializována v metodě `main`).

Práce s databází s sebou nese nutnost rozsáhlého ladění. Aby bylo toto ladění a pozdější hledání chyb co nejlehčí, zvolili jsme univerzální objekt, který vrací každá metoda, která s databází komunikuje. Tento výsledný objekt pak obsahuje informaci o úspěšnosti komunikace, případné znění chyby a výsledek, pokud má nějaký být. Toto řešení se nám v praxi osvědčilo.

Komunikaci s databází bude obstarávat pouze jedna třída, která bude vlastně jen jakousi knihovnou metod. Pro přehlednost jsme se rozhodli přidat rozhraní, ve kterém jsou popsány všechny veřejné metody pro komunikaci s databází - není třeba procházet celý kód, ostatním kolegům z týmu stačí pouze název metody a její popis.

2.6 GUI

Grafické uživatelské rozhraní je možné řešit dynamicky, nebo staticky. Vzhledem k tomu, že aplikace má vyplňovat celou plochu tabletu a měla by obsahovat minimum prázdného prostoru, rozhodli jsme se pro dynamické řešení, kdy se komponenty GUI samy přizpůsobují velikosti obrazovky.

Aplikace je určena pro tablety a ovládat ji budou zdravotní sestry ve stresovém prostředí. Proto by tomu měla být úměrná i velikost komponent.

3 Popis tříd a metod

Námi vyvíjená aplikace není příliš algoritmicky náročná, to ale vynahrazuje rozsahem a nároky na grafické rozhraní, které musí být uživatelsky velmi přívětivé. Zásadní částí

vývoje tohoto produktu byl návrh UML diagramu (viz. příloha). Kvůli přehlednosti musel být systém rozdělen do pěti základních částí/balíků.

3.1 Bilance tekutin

Bilance tekutin je část systému, která zařizuje práci s denní bilancí tekutin pacienta. Ke spojení s databází využívá instanci třídy Dtb, která implementuje rozhraní IDtb. Denní bilance tekutin je pojem, který zahrnuje veškeré příjmy a ztráty tekutin pacienta za celý den. Mezi tyto údaje patří například parenterální či enterální příjmy a ztráty odvedené přes drény, sondu nebo diurézou.

3.1.1 Diureza

Třída řídící správu informací o denní bilanci tekutin pacienta.

Uchovávání záznamů Záznamy o bilanci tekutin uchováváme v poli seznamů. Rozhodli jsme se tak proto, že víme předem počet seznamů, který budeme potřebovat(tj. předem daný počet způsobů příjmu či ztrát tekutin), ale nevíme počet hodnot přiřazených k jednotlivému způsobu příjmu či ztráty tekutin.

Zavrženou variantou bylo dvourozměrné pole, které by se dynamicky zvětšovalo, avšak pak by pole pro každý příjem či ztrátu tekutin mělo stejnou délku a to by znamenalo mnoho nevyužité paměti. Další zavržená varianta byla seznam seznamů, to je ale zbytečné, jelikož přesně známe množství způsobů příjmu či ztrát tekutin.

public Diureza(String pacientID) Konstruktor přijme String pacientID, tato informace je postačující k volání funkce getTekutiny, která je v kontraktu rozhraní IDtb. Dále už jen inicializuje každý jednotlivý seznam v poli, jeden pro každou možnost příjmu či ztráty tekutin.

public String getPrijem() Metoda sečte všechny příjmy tekutin, kromě položky ERY_RES, která v součtu nemá být zahrnuta. Vrací součet převedený na řetězec, aby bylo v GUI co nejméně funkčního kódu.

public String getVydej() Metoda sečte všechny výdaje tekutin. Vrací součet převedený na řetězec, aby bylo v GUI co nejméně funkčního kódu.

public String getHodnota(Tekutiny idSeznamu) Metoda najde v poli seznam na pozici idSeznamu a v něm najde poslední uloženou hodnotu, kterou vrátí ve formě řetězce, opět z důvodu extrakce funkčního kódu z GUI. Tekutiny jsou výčtový typ jednotlivých příjmu či výdajů, jejich přetypováním na int se jednoduše získá pořadí.

public String zadejHodnotu(int idSeznamu, String val) Metoda v poli najde seznam na pozici idSeznamu a v něm najde poslední uloženou hodnotu. Řetězec val se převede na int a porovná se s hodnotou, která je v seznamu na poslední pozici. Toto porovnání slouží ke zjištění, jestli hodnota není nižší než hodnota předchozí. Pokud je nová hodnota vyšší, přidá se do seznamu. Vrátí novou hodnotu ve formě řetězce, pokud byla vyšší než předchozí nejvyšší. Pokud se uživatel snažil zadat nižší hodnotu, vrátí předchozí největší hodnotu ve formě řetězce.

private void loadFromDtb() Metoda načte údaje z databáze. Každý databázový dotaz vrací instanci třídy State, ve které je atribut Boolean ok, který značí, jestli databázový dotaz proběhl podle očekávání. Pokud ne, vypíše se chyba. Pokud vše proběhne v pořádku, ze State se načtou údaje o příjmech a ztrátách tekutin pacienta.

public void pushToDtb() Metoda nejprve načte údaje o příjmech či ztrátách tekutin z databáze.

Dále zkontroluje, zda jsou naše data aktuální, to znamená, že poslední hodnota v našem seznamu je vyšší, než poslední hodnota v seznamu načteném z databáze. Tato podmínka musí platit pro každý seznam příjmu či ztráty tekutin. Pokud toto neplatí, vypíše se chyba, pokud podmínka platí, metoda se pokusí uložit námi získané údaje o ztrátách a příjmech tekutin pacienta do databáze.

Dále zkontroluje již dříve zmíněný State.ok a zjistí tak, zda vše proběhlo v pořádku, pokud ne, vypíše chybu.

3.1.2 Tekutiny

Výčtový typ všech způsobů příjmu či ztráty tekutin. Možné způsoby jsou:
DIUREZA,
SONDA,
JINE_ZTRATY,
ULTRAFILTRACE,
DRENY,
PER_OS,
JINY_ENTERALNI,
PARENTERALNI,
JINY_PARENTERALNI,
MLP,
ERY_RES,
OSTATNI_KREVNI_DERIVATY

V tomto pořadí jsou také vybírána a ukládána data do databáze.

3.2 Invazivní přístupy

Invazivní přístupy jsou část systému, která zajišťuje práci s informacemi o invazivních přístupech zavedených pacientovi. Pojem invazivní přístup značí „hadičku“ zavedenou pacientovi. Mohou sloužit k odvádění či přivádění určitých tekutin do/z těla pacienta.

K propojení s databází využívá instanci třídy Dtb, která implementuje rozhraní IDtb.

3.2.1 ZavedenePristupy

Třída řídící správu informací o invazivních přístupech zavedených pacientovi.

Uchovávání informací Informace uchováváme v seznamu typu Pristup, jiný způsob snad ani nemohl připadat v úvahu, jelikož seznam je pro tento problém naprosto ideální.

public ZavedenePristupy(String pacientID) Konstruktor přijme String pacientID, tato informace je postačující k volání funkce getPristupy, která je v kontraktu rozhraní IDtb. Dále už jen inicializuje seznam zavedených přístupů.

public void updatePristup(String poradiPristupu) Po výměně přístupu klikne uživatel na tlačítko aktualizuj a to zavolá tuto metodu.

Metoda najde invazivní přístup, v seznamu invazivních přístupů, podle poradiPristupu, která je unikátní hodnotou a používá se také jako primární klíč v databázi. Vytvoří se nový dočasný invazivní přístup tmp, který bude totožný s vybraným invazivním přístupem ze seznamu. Tomuto invazivnímu přístupu nastaví datum zavedení na dnešní datum, vymen nastaví na konstantu VYMEN_FALSE a počet dnů zavedení nastaví na 1.

Tento dočasný invazivní přístup se pokusíme poslat do databáze a upravit tak informace v uloženém invazivním přístupu. Pokud vše proběhne v pořádku, upravíme informace i v zavedeném invazivním přístupu v našem seznamu. Činíme tak proto, aby za každých okolností informace v databázi a programu korespondovaly.

public void vymenPristup(String poradiPristupu) Pokud má být invazivní přístup vyměněn, klikne uživatel na tlačítko vyměň a to zavolá tuto metodu. Metoda najde invazivní přístup v seznamu invazivních přístupů podle poradiPristupu. Vytvoří se nový dočasný invazivní přístup tmp, který bude totožný s vybraným invazivním přístupem ze seznamu. Tomuto invazivnímu přístupu nastaví char vymen na konstantu VYMEN_TRUE. Tento dočasný invazivní přístup se pokusíme poslat do databáze a upravit tak informace v uloženém invazivním přístupu. Pokud vše proběhne v pořádku, upravíme informace i v zavedeném invazivním přístupu v našem seznamu.

public void delPristup(String poradiPristupu) Pokud uživatel invazivní přístup vyjme, klikne pak na tlačítko smaž a to zavolá tuto metodu. Metoda najde invazivní přístup v seznamu invazivních přístupů podle poradiPristupu. Pokusí se invazivní přístup smazat z databáze, pokud databázový dotaz proběhne v pořádku, invazivní přístup smaže i z

našeho seznamu. Pokud vše proběhne v pořádku, smažeme invazivní přístup i z našeho seznamu.

public void addPristup(String cisloPristupu, String nazev, String umisteni, String hloubka) Po zavedení invazivního přístupu pacientovi uživatel zadá informace o invazivním přístupu a klikne na tlačítko přidej, kterým zavolá tuto metodu. Metoda vytvoří dočasný invazivní přístup se zadanými informacemi, který se pokusí poslat do databáze. Pokud přidání do databáze proběhne v pořádku, přidá se invazivní přístup i do seznamu zavedených přístupů v našem programu.

private void loadFromDtb() Metoda načte údaje z databáze. Každý databázový dotaz vrací instanci třídy State, ve které je atribut Boolean ok, který značí, jestli databázový dotaz proběhl podle očekávání. Pokud ne, vypíše se chyba. Pokud vše proběhne v pořádku, ze State se načtou údaje o invazivních přístupech zavedených pacientovi, ty se dále uloží do seznamu zavedených přístupů.

3.2.2 Pristup

Třída reprezentující jeden zavedený invazivní přístup. Používá k tomu Stringy značící pořadí, číslo, název, umístění, hloubka zavedení, dále dvě proměnné typu DateTime, jedna značí datum vytvoření (tj. datum prvního zavedení), druhá má uložený datum posledního zavedení přístupu. Poslední priměnnou je char vymen, který značí, zda má být invazivní přístup vyměněn pomocí konstant VYMEN_TRUE = 'A' a VYMEN_FALSE = 'N'. Pro značení, zda má být přístup vyměněn, je použit char , abychom se drželi konvence zavedené v databázi zadavatele, i když si uvědomujeme, že jinak by byl Boolean mnohem vhodnější.

public Pristup (String poradi, String cislo, String nazev, String umisteni, String hloubkaZavedeni, DateTime datumZavedeni) Konstruktor přijme všechny potřebné informace o invazivním přístupu a uloží si je do svých proměnných.

public Pristup (Pristup pristup) Konstruktor používaný k vytváření kopií Přístupu, aby se nemusela v každé části kódu vytvářet kopie přístupu pomocí kopírování jednotlivých proměnných.

public String getDnu() Vrací počet dnů, který uběhl od zavedení invazivního přístupu, používá k tomu instanci třídy TimeSpan, která umí jednoduše spočíst rozdíl ze dvou dat uložených v instancích třídy DateTime. Vrací rozdíl + 1 ve formě řetězce, pro snadné použití v GUI.

3.3 Medikace

Medikace je část systému, která zajišťuje práci s informacemi o lécích předepsaných pacientovi. K propojení s databází využívá instanci třídy Dtb, která implementuje rozhraní

IDtb. Medikační karta je tabulka, která pro každý lék ukazuje hodiny, kdy má být daný lék podán a v jakém množství.

3.3.1 MedikacniKarta

Třída řídící správu informací o lécích předepsaných pacientovi.

Uchovávání informací Informace uchováváme v seznamu typu Lek, jiný způsob snad ani nemohl připadat v úvahu, jelikož seznam je pro tento problém naprosto ideální.

public MedikacniKarta(String pacientID) Konstruktor přijme String pacientID, tato informace je postačující k volání funkce getLeky, která je v kontraktu rozhraní IDtb. Dále už jen inicializuje seznam předepsaných léků.

public void addLek(String nazev, String davkovani, String idLeku) Metoda přijme informace o léku zadané v GUI a přidá novou instanci léku do seznamu předepsaných léků.

public void zadejPodani(String idLeku, int cas) Když uživatel podá lék pacientovi, klikne pak na příslušnou hodinu v řádce s daným lékem a tím zavolá tuto metodu. Pomocí idLeku vyhledá metoda daný lék v seznamu a zavolá metodu z Dtb, kterou zadá podání léku do databáze, pokud se podání léku vydaří, zadá podání léku i do instance léku uložené v seznamu předepsaných léků. Pokud databázový dotaz selže, vypíše chybu.

private void loadFromDtb() Metoda načte údaje z databáze. Každý databázový dotaz vrací instanci třídy State, ve které je atribut Boolean ok, který značí, jestli databázový dotaz proběhl podle očekávání. Pokud ne, vypíše se chyba. Pokud vše proběhne v pořádku, ze State se načtou údaje o lécích předepsaných pacientovi, ty se dále uloží do seznamu předepsaných léků.

3.3.2 Lek

Třída reprezentující jeden pacientovi předepsaný lék. Pouýívá k tomu Stringy idLeku, nazev, davkovani a info. V poli typu Davka davkovaniCas jsou uložené údaje o tom, kdy má být lék podán a o počtu pilulek, které má pacient dostat. Dále jsou zde definované regulární výrazy, které akceptují různé způsoby zápisu dávkování léků. Pro regulární výrazy jsme se rozhodli, jelikož řešit takto rozsáhlý problém pomocí if-else konstrukce by bylo extrémně náročné jak na úvahu, tak na realizaci a čas. Různé způsoby zápisu dávkování léků jsou rozebrány v jednotlivých parsovacích metodách.

public Lek(String nazev, String davkovani, String idLeku) Konstruktor přijme informace o názvu, dávkování a id léku. Uloží si je do proměnných a inicializuje pole davkovaniCas o délce 24 – podle počtu hodin. Dále pak zavolá parsovací metodu, která

pomocí regulárních výrazů určí, kdy má být lék podán a kolik pilulek má v daný čas dostat. Jako parametr této metodě předá String davkovani.

private void parsuj(String retezec) Metoda zavolá pět dalších parsovacích metod, pro každý regulární výraz existuje pro přehlednost jedna.

private void parsujReg1(String retezec) Metoda zjistí, zda část řetězce vyhovuje regulárnímu výrazu `\d+(\s*[-]\s*\d+)+`. Tomu vyhovují dávkování zapsané ve tvaru `1-2`, `1-0-1`, `2-1-0-2`, `(1-2)`, `(1-0-1)`, `(2-1-0-2)` apod.

Pokud řetězec vyhovuje, vezme jeho akceptovanou část a rozdělí ji na jednotlivá čísla. Čísla mohou být maximálně 4. Pokud jsou nalezena dvě čísla, znamenají počty pilulek, které mají být podány ráno a večer. Pokud jsou nalezena čísla tři, znamenají počty pilulek, které mají být podány ráno, v poledne a večer. A konečně pokud cyklus nalezne 4 čísla, znamenají počty pilulek k podání ráno, v poledne, večer a v noci.

Výsledné hodnoty uloží do pole davkovaniCas, kde pomocí typu Davka uchováváme ke každé hodině informaci o podání a o počtu.

private void parsujReg2(String retezec) Metoda zjistí, zda část řetězce vyhovuje regulárnímu výrazu `\d+(\s*[,]\s*\d+)+`. Tomu vyhovují dávkování zapsané ve tvaru `08,12`, `10,16,22`, `12, 18, 24, 06`, `(12,24)`, `(06, 12, 18, 24)` apod.

Pokud řetězec vyhovuje, vezme jeho akceptovanou část a rozdělí ji na jednotlivá čísla. Nalezená čísla znamenají hodinu, kdy má být lék podán.

Výsledné hodnoty uloží do pole davkovaniCas, kde pomocí typu Davka uchováváme ke každé hodině informaci o podání a o počtu.

private void parsujReg3(String retezec) Metoda zjistí, zda část řetězce vyhovuje regulárnímu výrazu `a\s*\d+\s*hod`. Tomu vyhovují dávkování zapsané ve tvaru `a12 hod`, `a 12 hod`, `a6hod`, `a 6hod` apod.

Pokud řetězec vyhovuje, vezme jeho akceptovanou část a vybere z něj číslo. Nastaví podání na půlnoc a každou další hodinu, která má modulo nalezeného čísla rovno nule.

Výsledné hodnoty uloží do pole davkovaniCas, kde pomocí typu Davka uchováváme ke každé hodině informaci o podání a o počtu.

private void parsujReg4(String retezec) Metoda zjistí, zda část řetězce vyhovuje regulárnímu výrazu `[1234]\s*[x\*]\s*\d`. Tomu vyhovují dávkování zapsané ve tvaru `2*1`, `2x1`, `3x3`, `3 x 3`, `3 * 1`, `4*2` apod.

Pokud řetězec vyhovuje, vezme jeho akceptovanou část a rozdělí ji na jednotlivá čísla. První nalezené číslo znamená, kolikrát má být lék za den podán a může být maximálně 4. Pokud dvakrát, má to být ráno a večer, třikrát znamená ráno, v poledne a večer a čtyřikrát znamená ráno, v poledne, večer a v noci. Druhé číslo znamená počet pilulek, které má pacient dostat.

Výsledné hodnoty uloží do pole davkovaniCas, kde pomocí typu Davka uchováváme ke každé hodině informaci o podání a o počtu.

private void parsujReg5(String retezec) Metoda zjistí, zda část řetězce vyhovuje regulárnímu výrazu `\d*\s*ml\s*\/\s*hod`. Tomu vyhovují dávkování zapsané ve tvaru `1ml/hod`, `22 ml/hod`, `333 ml / hod` apod.

Pokud řetězec vyhovuje, vezme jeho akceptovanou část a uloží ji do proměnné akceptováno. Do pole davkovaniCas se neukládá, jelikož tento zápis dávkování se používá pro léky podávané nitrožilně.

private void vyberCisla(String retezec) Vybere jednotlivá čísla z řetězce. Uloží je do pole typu int, se kterým dále ostatní parsovací metody pracují.

3.3.3 Davka

Tato třída uchovává informace o podání léku v konkrétní hodině. Má dvě proměnné typu int počet a podání. Podání může nabývat uložených konstant ZADNA=0, ORDINACE=1, PROVEDENA=2 a počet může být jakékoli množství podávaných pilulek.

public Davka(int podani, int pocet) Konstruktor si uloží informaci o podání a počtu pilulek do svých proměnných.

3.4 Databáze

Část systému, které obstarává komunikaci s databází je tvořena čtyřmi třídami – State, Dtb, DtbConfig, IDtb. Obecně každá metoda, která s databází komunikuje vrací objekt State, který obsahuje zprávu o úspěšnosti komunikace, znění případné chyby a výsledek (pokud je nějaký očekáván).

Konfigurace připojení je načítána z konfiguračního souboru třídou DtbConfig.

3.4.1 State

Třída, která uchovává výsledek metody, která komunikuje s databází. Má celkem tři parametry – ok, err, res.
`bool ok` – Informuje, zda komunikace s databází proběhla úspěšně bez chyb. Pokud ano, je nastaven na true.
`String err` – Pokud se při komunikaci s databází vyskytla chyba, je zde její znění a stack trace. Jinak je zde prázdný řetězec.
`List<Object> res` – Parametr, který slouží k ukládání výsledků. Pokud nemá metoda vracet žádný výsledek, je nastaven na null. V opačném případě obsahuje seznam výsledků. Formátování konkrétního výsledku je vždy popsáno v komentáři metody v rozhraní IDtb.

Tento způsob zapouzdření případné chyby a výsledku dotazu na databázi do jedné třídy jsme zvolili z důvodu snadného ladění. Pokud dojde k chybě při komunikaci (špatný dotaz, přerušení spojení...) program nespadne, ale odchytí se výjimka, kterou je možné v zobrazovací části programu vypsat na obrazovku (ideálně MessageBox, nebo do konzole).

3.4.2 Dtb

Nejdůležitější třída. Zde jsou implementovány všechny metody které vybírají, nebo ukládají data do databáze. Pokud z nějakého důvodu selže načtení konfigurace připojení, třída nastaví defaultní hodnoty, které připojují na databázi students.kiv.zcu.cz.

V této třídě je také uloženo ID pracoviště (předáno z hlavní části programu v konstruktoru).

Obecně se každá metoda komunikující s databází skládá ze tří částí. V první části se inicializují objekty potřebné ke komunikaci – OracleConnection, OracleDataReader (pokud je potřeba), OracleCommand.

V další části se sestaví dotaz na databázi a v poslední části probíhá exekuce dotazu a případné ukládání výsledku. Poslední část je umístěna v bloku try-catch-finally z důvodu odchytávání vyjímek a ukládání chybových zpráv do výsledného State.

public State logInPsw(String username, String password) Metoda slouží k připojení uživatele aplikace do systému. Pokud se v databázi nachází uživatel se zadaným username+password, který má pracovní úvazek na pracovišti, na které je tablet nastaven, výsledek je true, jinak false.

Kombinace username, pasword se v tomto případě používá také k přístupu do databáze. V této metodě se tedy nastaví username a password podle zadaných údaju a znovu se volá metodu buildConString(), která správně sestaví připojovací řetězec.

Výsledný State.res pak bude obsahovat List<bool>s true/false na prvním místě.

public State getPacienti() Metoda slouží k získání pacientů na pracovišti, které bylo zadáno v konstruktoru. Výsledný State.res pak bude obsahovat List<Pacient>se seznamem pacientů na zadaném pracovišti.

public State getTekutiny(String pacientID) Metoda slouží k získání bilance tekutin u daného pacienta k dnešnímu dni. Pokud pro zadaného pacienta není v databázi k dnešku záznam, metoda jej vytvoří s hodnotami 0. Výsledný State.res pak bude obsahovat List<int>ve kterém bude uložen seznam tekutin v pořadí, podle Tekutiny.

public State saveTekutiny(String pacientID, List<int>tekutiny) Metoda slouží k uložení bilance tekutin daného pacienta do databáze. Hodnoty v tekutiny musí být ve stejném pořadí jako jsou konstanty ve třídě Tekutiny.

public State getPristupy(String pacientID) Metoda vybere z databáze všechny aktivní (vyjmuté přístupy v databázi zůstavají) invazivní přístupy a ty vrátí. Výsledný State.res pak bude obsahovat List<Pristup>se seznamem invazivních přístupů. Pokud pacient žádné aktivní přístupy nemá, seznam bude prázdný.

public State addPristup(String pacientID, Pristup pristup) Metoda slouží k přidání nového invazivního přístupu k zadanému pacientovi.

public State vymenitPristup(String pacientID, Pristup pristup) Metoda nastaví u zadaného požadavku požadavek na výměnu na true. ID pacienta je v tomhto případě potřeba, protože klíč v databázi se skládá z ID pacienta a ID přístupu.

public State updatePristup(String pacientID, Pristup pristup) Metoda slouží k aktualizaci přístupu v databázi. Primární účel metody sice je pouze nastavit u přístupu dnešní datum, nicméně dotaz je konstruován tak, že se dá změnit celý přístup.

public State vyjmoutPristup(String pacientID, Pristup pristup) Metoda nastavi zadaný invazivní přístup u zadaného pacienta jako vyjmutý. Vyjmutí neznamená smazání z databáze, pouze nastavení data vyjmutí na dnešní.

public State getMaxPoradi(String pacientID, DateTime datum) Metoda slouží k získání maximálního pořadí invazivního přístupu u daného pacienta k dnešnímu dni. Pokud k dnešnímu dni není žádný záznam, metoda vrátí 0. Tato metoda je potřebná při přidávání nového invazivního přístupu jehož ID se nastaví na max+1.

Výsledný State.res pak obsahuje List<int¿, který má na první pozici maximální pořadí.

public State getLeky(String pacientID) Metoda slouží k získání naordinovaných léků u zadaného pacienta. Z databáze se vyberou pouze léky jejichž DATUM_DO je menší nebo rovno dnešnímu datu. Výsledný State.res pak obsahuje List<Lek>se seznamem naordinovaných léků.

public State addLek(String pacientID, Lek lek) Metoda slouží k přidání nového léku pacientovi. Tato metoda má nejkomplikovanější dotaz, protože vzhledem k struktuře databáze je třeba přidávat záznam do dvou tabulek (a vybírat dvě maxima, kvůli ID).

public State addPodani(String pacientID, Lek lek) Metoda slouží k zaznamenání podání léku do databáze.

public State getPrvniPodani(String pacientID, Lek lek) Metoda z databáze zjistí první podání zadaného léku u zadaného pacienta. Bere v potaz jen léky, které jsou v současné době podávány. Údaj o prvním podání je potřeba zjistit v případě léků, které se podávají po určitém čase (např. každých 12 hodin.)

Výsledný State.res pak obsahuje List<DateTime>na jehož prvním místě je datum a čas prvního podání.

public State getKlinUdalost(String pacientID) Metoda z databáze vybere poslední klinickou událost u zadaného pacienta. Poslední klinickou událost je třeba znát při přidávání nového léku.

Výsledný State.res pak obsahuje List<String>na jehož prvním místě je ID klinické události.

public State getMaxOrdlekID(String pacientID) Metoda z databáze vybere nejvyšší ID naordinovaného léku u zadaného pacienta. Toto ID je třeba znát při vytváření nového léku.

Výsledný State.res pak obsahuje List<int>na jehož prvním místě se nachází nejvyšší ID ordinovaného léku.

public State getUmisteni() Metoda vybere z databáze číselník invazivních přístupů. Výsledný State.res pak obsahuje List<String >ve kterém jsou názvy umístění invazivních přístupů. Při vytváření nového invazivního přístupu je možno vybrat z tohoto číselníku, ale není to nutné.

public State getKlinUdal(String pacientID) Metoda vybere z databáze všechny klinické události, které přísluší zadanému pacientovi a ty vrátí. Tuto metodu volá GUI v případě, že chce sestřička přidat nový lék. Každý lék u pacienta je vázaný na klinickou událost a pokud se přidá nový, musí se i tento ně jednu z klinických událostí navázat.

private void loadConf() Metoda načte konfiguraci přístupu k databázi ze souboru Meditab.exe.conf a uloží ji do proměnných dtbPort, dtbHost... Při načítání konfigurace z konfiguračního souboru se využívá třídy DtbConfig.

Pokud se načtení z konfiguračního souboru nezdaří, jsou nastaveny defaultní hodnoty (připojení ke školní databázi).

private void buildConString() Metoda sestaví connectionString pro připojení k Oracle databázi a uloží jej do proměnné connString. Conneciton string se sestaví z proměnných dtbPort, dtbHost... Pokud se má třída připojovat k jiné databázi, než defaultní (DEF_PORT, DEF_HOST..) je nutné před voláním této metody zavolat loadConf() - to se děje v konstruktoru třídy.

3.4.3 DtbConfig

Třída oddděná od ConfigurationSection. Slouží k načítání z konfiguračního souboru, konkrétně ze sekce databáze.

V .NET je konfigurační soubor formátován jako XML a v případě přidání vlastních elementů (=sekcí) je potřeba vytvořit zvláštní třídu, která umí tento element správně načíst.

Třída obsahuje pouze několik property, kterými vrací načtené hodnoty. Třída z konfiguračního souboru načítá port připojení, hostname a jméno databáze. IDtb

Rozhraní, které definuje metody pro komunikaci s databází. Všechny public metody, které jsou ve třídě Dtb jsou implementovány z tohoto rozhraní, proto zde metody nebudu znovu uvádět. Rozhraní v tomhle případě slouží čistě k přehlednosti pro ostatní, kdo se na projektu podílejí.

4 GUI

GUI bylo vytvářeno dynamicky tak, aby komponenty zcela vyplňovaly prostor rodičovského kontejneru. Ovládácí komponenty mají větší velikost pro snazší ovládání na tabletu.

K vytvoření GUI jsme použili knihovnu Windows Forms. Většina použitých komponent je systémových, pro invazivní přístupy jsme vytvořili vlastní grafickou komponentu PristupBox, která je zděděná od GroupBox.

5 Závěr

Tento projekt byl naší první prací pro externího zadavatele a co víc, pro Fakultní nemocnici Plzeň. Pro dobro všech pacientů, a v budoucnu možná i naše, jsme se snažili vyvinout systém co nejkvalitněji a pro zdravotní sestry co nejpříjemněji ovladatelný, aby je při jejich už tak dost náročné práci nezatěžoval. Doufáme, že náš systém bude pro jednotky intenzivní péče aspoň takovým přínosem, jakým byl pro nás.