TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky a mezioborových inženýrských studií

Studijní program: B2612 – Elektrotechnika a informatika
Studijní obor: 2612R011 – Elektronické informační a řídicí systémy

Zobrazování geografických dat pomocí J2ME

Displaying of the geographical data by means of J2ME

Bakalářská práce

Autor: **Michal Koťátko**
Vedoucí práce: Ing. Roman Špánek
Konzultant: Ing. Pavel Tyl

V Liberci 18. 5. 2007

Obsah

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Poděkování

V úvodu své bakalářské práce bych rád poděkoval svému vedoucímu inženýru Špánkovi za podnětné rady a metodické pokyny týkající se konceptu a tvorby celé práce.

Anotace

V této bakalářské práci bylo za úkol naprogramovat aplikaci, která bude zobrazovat geografická dat pomocí J2ME. J2ME patří mezi nejpoužívanější programovací jazyky pro programování aplikací pro přenosná zařízení. Je zde uveden stručný popis platformy Java, seznámení s platformou J2ME, třídou MIDlet, používanou pro práci s přenosným zařízením. Dále je zde popsán Geografický informační systém, jehož data zobrazujeme a programovací prostředí NetBeans IDE. V závěru práce jsou popsány jednotlivé třídy používané v aplikaci, včetně popisu základních příkazů a metod.

Klíčová slova: programování, J2ME, GIS

Annotation

The aim of this bachelor paper was to programme such an application that will be able to display geographical data by means of J2ME. This application is one of the most used programming languages used for programming of the applications used for portable appliances. This paper contains a description of JAVA platform, brief introduction to the platform J2ME, rank MIDlet, used for work with portable appliances. Further on we describe Geographical informational system, whose data are displayed, and also the programming environment NetBeans IDE. The conclusion of this paper contains descriptions of the individual ranks used in the application, including description of the basic commands and methods.

Keywords: programming, J2ME, GIS

1. Úvod

V současné době není používání přenosných zařízení, jako jsou mobilní telefony a digitální osobní asistenti výsadou manažerů a bohatých lidí, ale stalo se součástí běžného života každého z nás. Postupným vývojem těchto zařízení došlo k postupnému pronikání aplikací, které byly výsadou počítačů i na tato zařízení. Jedním z používaných programovacích jazyků pro vývoj těchto aplikací je J2ME.

První část aplikace se zabývá jazykem Java, jeho vznikem, popisem základních vlastností, jednotlivých platforem a součásti platformy Java jako jsou Java Virtual Machine a Java Core API.

Druhá a třetí část se věnuje platformě J2ME, stručnému popsání jednotlivých konfigurací, profilů platformy a virtuálních strojů. Dále se zabývá třídou MIDlet, potřebnou pro běh aplikace na mobilním zařízení, a popisem jeho životnímu cyklu.

Čtvrtá část práce obsahuje seznámení s geografickým informačním systémem a jednotlivými reprezentacemi geografických dat.

Pátá část stručně popisuje vývojové prostředí NetBeans IDE, ve které byla aplikace naprogramována, a která obsahuje emulátory mobilních zařízení.

Šestá a poslední část práce se zaobírá strukturou aplikace, jednotlivými třídami pro načítání dat z obou zdrojů, třídou pro vykreslování grafiky na displej mobilního zařízení. Dále je zde popsáno obsluhování událostí, problematiky vláknového zpracování a ošetřování výjimek.

2. Java

Java je nejen programovací jazyk, ale také rozsáhlá technologie a platforma, kterou vyvinula firma Sun Microsystems a představila 23. května 1995. Programy napsané v Javě je možné provozovat na různých platformách, pro které existuje tzv. běhové prostředí Javy, což je jakýsi prostředník mezi operačním systémem a programem. Program je pak kompilován do byte-kódu, aby se dal spustit běhovým prostředím na různých platformách. Toto běhové prostředí se nazývá Virtuální stroj Javy (Java Virtual Machine – JVM). Z užití běhového prostředí vyplývá, že je Java interpretovaný jazyk. Díky tomu, že je jazyk interpretovaný, stačí pro dané platformy pouze vyvinout běhové prostředí. Kdyby Java generovala přímo spustitelný kód, musel by existovat překladač pro každou platformu. Díky své přenositelnosti je používán pro programy, které mají pracovat na různých systémech počínaje čipovými kartami (platforma JavaCard), přes mobilní telefony a různá zabudovaná zařízení (platforma Java ME), aplikace pro desktopové počítače (platforma Java SE) až po rozsáhlé distribuované systémy pracující na řadě spolupracujících počítačů rozprostřené po celém světě (platforma Java EE).

2.1. Základní vlastnosti

jednoduchý – jeho syntaxe je zjednodušenou (a drobně upravenou) verzí syntaxe jazyka C a C++. Odpadla většina konstrukcí, které způsobovaly programátorům problémy, ale na druhou stranu přibyla řada užitečných rozšíření

objektově orientovaný - s výjimkou osmi primitivních datových typů jsou všechny ostatní datové typy objektové, využívá dědičnosti, zapouzdření a polymorfismu

distribuovaný – je navržen pro podporu aplikací v síti (podporuje různé úrovně síťového spojení, práce se vzdálenými soubory, umožňuje vytvářet distribuované klientské aplikace a servery)

interpretovaný – místo skutečného strojového kódu se vytváří pouze tzv. mezikód

(byte-kód). Tento formát je nezávislý na architektuře počítače nebo zařízení. Program pak může pracovat na libovolném počítači nebo zařízení, který má k dispozici interpret Javy, tzv. virtuální stroj Javy - Java Vitrual Machine (JVM)

robustní – je určen pro psaní vysoce spolehlivého softwaru – z tohoto důvodu neumožňuje některé programátorské konstrukce, které bývají častou příčinou chyb (např. správa paměti, příkaz goto, používání ukazatelů). Používá tzv. silnou typovou kontrolu – veškeré používané proměnné musí mít definovaný svůj datový typ

bezpečný – má vlastnosti, které chrání počítač v síťovém prostředí, na kterém je program zpracováván, před nebezpečnými operacemi nebo napadením vlastního operačního systému nepřátelským kódem

nezávislý na architektuře – vytvořená aplikace běží na libovolném operačním systému nebo libovolné architektuře. Ke spuštění programu je třeba pouze nainstalovaný odpovídající virtuální stroj (JVM). Podle konkrétní platformy se dále může přizpůsobit vzhled i chování aplikace

přenositelný – vedle zmíněné nezávislosti na architektuře je jazyk nezávislý, i co se týče vlastností základních datových typů (je například explicitně určena vlastnost a velikost každého z primitivních datových typů). Přenositelností se však myslí pouze přenášení v rámci jedné platformy Javy (např. J2SE). Při přenášení mezi platformami Javy je třeba dát pozor na to, že platforma určená pro jednodušší zařízení nemusí podporovat všechny funkce dostupné na platformě pro složitější zařízení a kromě toho může definovat některé vlastní třídy doplňující určitou speciální funkčnost nebo nahrazující třídy vyšší platformy, které jsou pro nižší platformu neaplikovatelné.

výkonný – přestože se jedná o jazyk interpretovaný, není ztráta výkonu významná, neboť překladače pracují v režimu „Just in time“ a do strojového kódu se překládá jen ten kód, který je opravdu zapotřebí

víceúlohový – podporuje zpracování vícevláknových aplikací, tedy aplikací podporující paralelní běh procedur a funkcí.

dynamický – Java byla navržena pro nasazení ve vyvíjejícím se prostředí. Knihovna může být dynamicky za chodu rozšiřována o nové třídy a funkce, a to jak z externích zdrojů, tak vlastním programem.

elegantní – vyžaduje ošetření výjimek a typovou kontrolu.

2.2. Nevýhody jazyka Java

Proti programovacím jazykům, které provádějí tzv. statickou kompilaci (např. C++), je start programů psaných v Javě pomalejší, protože prostředí musí program nejprve přeložit a potom teprve spustit. Další nevýhodou projevující se hlavně u jednodušších programů je větší paměťová náročnost při běhu způsobená nutností mít v paměti celé běhové prostředí.

2.3. Platforma Java

Platforma Java se dělí na části:

Java ME (dříve J2ME) – je souhrn API (application programming interface - jde o sbírku procedur, funkcí či tříd nějaké knihovny, ale třeba i jiného programu nebo jádra operačního systému, které může využívat programátor, který knihovnu využívá pro vývoj softwaru) pro zařízení s omezenými zdroji, jako jsou mobilní telefony, osobní digitální asistenti (PDA) či jiná malá zařízení.

Java SE (dříve J2SE) – je základní platforma určená pro vývoj desktopových nebo serverových aplikací v jazyce Java. Poskytuje specifikaci jazyka Java, virtuální stroj Java Virtual Machine a sadu základních rozhraní (Java Core API).

Java EE (dříve J2EE) – je programovací platforma – součást platformy Java – pro vývoj a běh distribuovaných vícevrstvých aplikací, založených převážně na modulárních softwarových komponentách běžících na aplikačním serveru.

2.4. Java Virtual Machine

Virtuální stroj Javy je virtuální stroj, který interpretuje a vykonává byte-kód Javy. Tento kód je nejčastěji generován kompilátory Javy, přestože JVM může být také zaměřený na kompilátory jiných programovacích jazyků. Virtuální stroj Javy je klíčová komponenta platformy Java. JVM je dostupný pro mnoho softwarových platforem a druhů hardwaru. Použití jednoho byte-kódu na všech platformách, může být Java charakterizována jako „ zkompiluj jednou, spusť kdekoliv“ jako protiklad k „napiš jednou, zkompiluj kdekoliv“.

2.5. Java Core API

Java Core Application Programming Interface obsahuje značné množství knihovních tříd, které jsou považovány za standardní, tudíž se musí vyskytovat v každém prostředí, kde se používá Java. Pokud používá program metody z API, není jejich kód součástí programu, protože je součástí API. To znamená, že program obsahuje pouze kód napsaný programátorem a proto pak mají poměrně malou velikost.

2.6. Základní principy objektového programování

Objektově orientované programování (dále jen OOP) představuje v dnešní době novou metodu tvorby aplikací oproti klasické metodě procedurálního programování.

Základem OOP je napodobit vzhled a chování objektu reálného světa. Jelikož je programovací jazyk Java objektově orientovaný, jsou níže vysvětleny základní prvky OOP.

Objekt – jednotlivé prvky modelované reality jsou v programu seskupeny do objektů. Objekty obsahují atributy, což jsou jednotlivé vlastnosti daného objektu, a metody, které popisují chování objektu.

Třída – je šablona, která deklaruje metody a atributy skutečných objektů. Každý objekt je instancí některé z tříd.

Zapouzdření – je způsob propojení atributů a metod prostřednictvím objektu. K těmto atributů a metodám můžeme přistupovat, pouze pokud vytvoříme instanci objektu.

Polymorfismus – vlastnost OOP, která umožňuje u jednoho objektu pojmenovat metodu jedním jménem, které ale provádějí různou činnost. Tyto metody se odlišují počtem a typem parametrů.

Dědičnost – objekty jsou organizovány hierarchicky do stromové struktury. Objekty nějakého typu mohou získat vlastnosti a metody děděním od objektu jiného druhu, přičemž přebírají jeho schopnosti, ke kterým přidávají pouze vlastní řízení. Při psaní objektu nemusíme definovat stále dokola ty samé vlastnosti a metody, ale pokud jsme již vytvořili objekt s danými atributy a metodami, můžeme od tohoto objektu tyto schopnosti zdědit a popřípadě doprogramovat atributy a metody specifické pro vytvářený objekt.

Konstruktor – je metoda objektu, která má stejný název jako objekt, ale nevrací žádnou hodnotu. Používá se pro inicializaci objektu.

Specifikátory přístupu – používá se pro ovládání přístupu k atributům a metodám třídy. Je to klíčové slovo uvedené v definici třídy, které určuje, jaká část programu může přistupovat k atributům a metodám této třídy.

- Specifikátor veřejného přístupu – definujeme klíčovým slovem *public* – k prvkům (atributům a metodám) mohou přistupovat ostatní třídy.
- Specifikátor privátního přístupu – definujeme klíčovým slovem *private* – k prvkům není možné přistupovat z vnějšku třídy. Může k nim přistupovat pouze třída, která je definovala.
- Specifikátor chráněného přístupu – definujeme klíčovým slovem *protected* – k prvkům může přistupovat třída, která jej definuje a její podtřídy.

3. Java 2 Micro Edition

3.1. Co je J2ME?

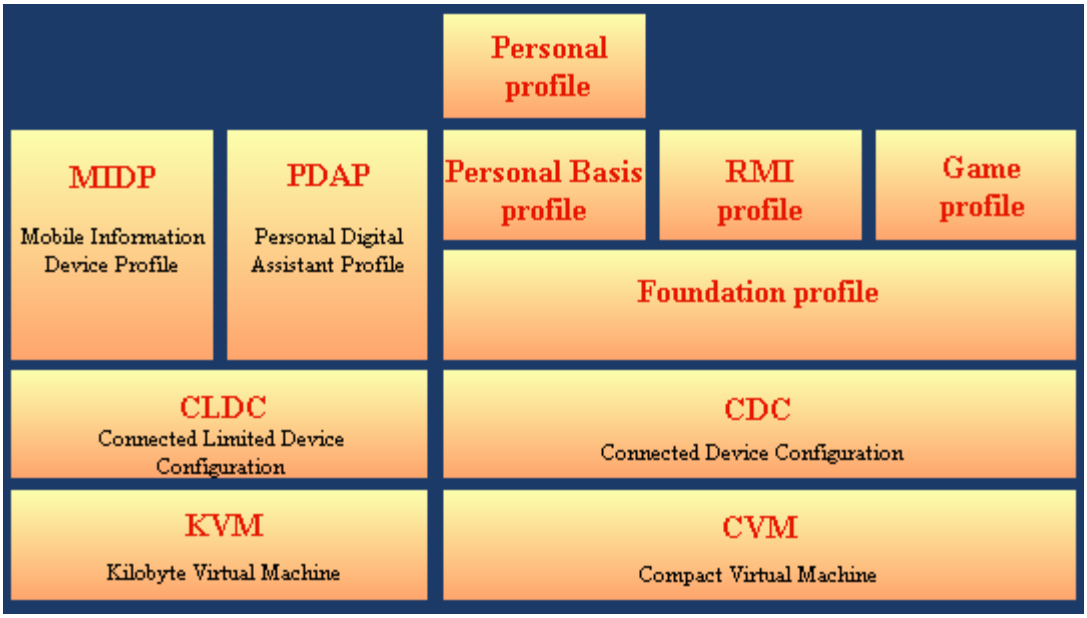
Je verze produktu Java společnosti Sun Microsystems určená pro trh s elektronickými zařízeními, jakými jsou např. mobilní telefony, pagery, osobní digitální asistenti (PDA), přídavná zařízení a další malá zařízení. Vývoj J2ME je zajištěn projektem zvaným Java Community Process (JCP), který umožňuje komukoli s připojením na internet, aby se k tomuto vývoji připojil. J2ME poskytuje kompletní řešení pro tvorbu nejmodernějších síťových aplikací pro malá zařízení. Výrobcům, poskytovatelům a vývojářům těchto zařízení také slibuje možnost vyvíjet nové aplikace pro své zákazníky. Tato možnost však neznamená, že by byly obětovány základní vlastnosti Javy, jejichž význam v současnosti stále roste, jmenovitě kompatibilita různých prostředí a zabezpečení.

3.2. Obecný pohled

Z obecného pohledu J2ME definuje následující komponenty:

- sérii virtuálních strojů, z nichž každý se uplatňuje u jiného typu zařízení
- skupinu knihoven a API, které lze spustit na každém virtuálním stroji, říká se jim konfigurace a profily
- různé nástroje pro vývoj a nastavení zařízení

První dvě komponenty tvoří pracovní prostředí J2ME. Obrázek 3-1 podává přehled vztahů v pracovní prostředí. Jeho centrum tvoří virtuální stroj, který pracuje na hostitelském operačním systému zařízení. Nad ním je specifická konfigurace J2ME, která se skládá z programovacích knihoven. Ty zajišťují základní funkce vycházející z požadavků na zdroje daného zařízení. Vrchol konfigurace tvoří jeden či více J2ME profilů.



Obr. 3-1: Přehled vztahů v pracovním prostředí

3.3. Konfigurace

Mobilní telefony, pagery, organizéry a další malá zařízení se liší formou, funkcemi a vlastnostmi. Často však používají podobné procesory a mají podobné množství paměti. Proto autoři J2ME vytvořili konfigurace. Konfiguracemi se definuje horizontální členění produktů založené na dostupném množství paměti a výkonu procesoru jednotlivých zařízení. Jakmile má tyto informace k dispozici, konfigurace zjistí následující údaje:

- podporované rysy programovacího jazyka Java
- podporované rysy virtuálního stroje Javy
- podporované základní knihovny a API

V současnosti existují pro J2ME dvě standardní konfigurace: CLDC (Connected Limited Device Configuration) a CDC (Connected Device Configuration).

3.3.1. CDC (Connected Device Configuration)

Konfigurace CDC je určena pro výkonná zařízení, která jsou velmi často připojena k síti. Patří sem přídavná zařízení, televize po internetu, domácí spotřebiče a navigační systémy pro vozidla či PDA (Personal digital assistant). CDC obsahuje plnou verzi virtuálního stroje Javy podobné tomu, který se dnes používá u J2SE. Rozdíl spočívá v paměti příslušného zařízení a v zobrazovací schopnosti.
Požadavky na zdroje pro CDC zařízení podle oficiální specifikace J2ME (JSR-36):

- Zařízení je řízeno 32bitovým procesorem
- Zařízení má 2MB a více paměti pro Javu. Tento údaj zahrnuje RAM i paměť flash (paměť, která může být elektricky mazána i programována a uchovává data i po odpojení zdroje elektrického proudu) nebo ROM (z paměti lze pouze číst).
- Zařízení vyžaduje plně funkční virtuální stroj Java 2 „Blue Book“.
- Zařízení je připojitelné k některému typu sítě, často bezdrátovým přerušovaným připojením a omezenou šířkou pásma (často 9600 bps nebo méně).
- Zařízení může mít i poměrně propracované uživatelské rozhraní, ale není to povinností.

3.3.2. CLDC (Connected Limited Device Configuration)

Konfigurace CLDC udává mnohem menší požadavky na zabudovaná zařízení než CDC. Konfigurace CLDC byla poprvé vydána v říjnu 1999 se záměrem vytvořit „nejnižšího společného jmenovatele“ v prostředí Java pro zabudovaná zařízení, dvousměrné pagery a osobní organizéry.

Požadavky pro J2ME CLDC podle oficiální specifikace J2ME (JSR-36)

- Zařízení může mít celkem 160 až 512 kilobajtů paměti pro prostředí Java včetně RAM i paměť flash nebo ROM.
- Zařízení může mít omezený zdroj energie, např. bateriové napájení.
- Zařízení je připojitelné s některým typem sítě často bezdrátovým s přerušovaným připojením a s omezenou šířkou pásma (často 9600 bps nebo

méně).

- Zařízení může mít i poměrně propracované uživatelské rozhraní, ale není to povinností.

3.4. Virtuální stroje

CLDC i CDC stanoví své vlastní skupiny podporovaných rysů z javovského virtuálního stroje. Následně je u každého z nich zapotřebí jiný javovský virtuální stroj. Protože podporuje mnohem méně rysů, je virtuální stroj CLDC mnohem menší než virtuální stroj, který vyžaduje CDC. Virtuálnímu stroji pro CLDC se říká KVM (K Virtual Machine) a virtuální stroj pro CDC se nazývá CVM (C Virtual Machine).

3.4.1. K Virtual Machine – KVM

KVM je kompletní javovské pracovní prostředí pro malá zařízení. Jedná se o skutečný javovský virtuální stroj podle specifikace pro virtuální stroje Javy až na některé specifické odchylky, které jsou nutné ke správnému fungování u malých zařízení. Je od samého základu vytvořen pro malá zařízení s omezenými zdroji a s několika stovkami kilobajtů celkové paměti. Je určený k podpoře postupného standardizovaného rozšiřování vlastností Java virtual machine a Java API zahrnutých v architektuře Java 2 ME

3.4.2. C Virtual Machine – CVM

CVM je vytvořeno pro větší zařízení, např. pro ta, která používají CDC. Podporuje všechny rysy a knihovny virtuálního stroje Java 2 verze 1.3 v oblasti zabezpečení, slabé odkazy, JNI a RMI (Remote Method Invocation). Implementace odkazů, které jsou v současnosti k dispozici od Sun Microsystems, funguje s Linuxem a VxWorks.

3.5. Profily

J2ME umožňuje definovat javovská prostředí pro různé produkty stejné úrovně tím, že zavádí profily. Profil je vlastně sada programových rozhraní (API) tvořících nadstavbu konfigurace. Profil nabízí programu přístup k specifickým vlastnostem pro dané zařízení.

3.5.1. Mobile Information Device Profile – MIDP

Midp je navržen pro práci s CLDC a poskytuje sadu programových rozhraní použitelných v mobilních zařízeních, jako jsou mobilní telefony a obousměrné pagery. MIDP obsahuje třídy pro uživatelské rozhraní, trvalé ukládání a práci v síti. Dále umožňuje „nahrávat“ do koncového zařízení nové aplikace. Malým aplikacím, které běží pod MIDP se říká Midlety.

3.5.2. Profil PDA

Profil PDA je navržen na CLDC a poskytuje API pro uživatelské rozhraní a API pro ukládání dat v příručních zařízeních.

3.5.3. Základní profil

Základní profil rozšiřuje programové rozhraní, které poskytuje CDC, ale nedodává žádné API pro uživatelské rozhraní. Jak naznačuje název „Základní“ (Foundation), tento profil má sloužit jako základní pro další profily, např. Osobní profil a RMI profil.

3.5.4. Osobní profil

Osobní profil rozšiřuje možnosti Základního profilu o grafické uživatelské rozhraní (GUI), na němž lze spustit applety pro Java Web.

3.5.5. RMI profil

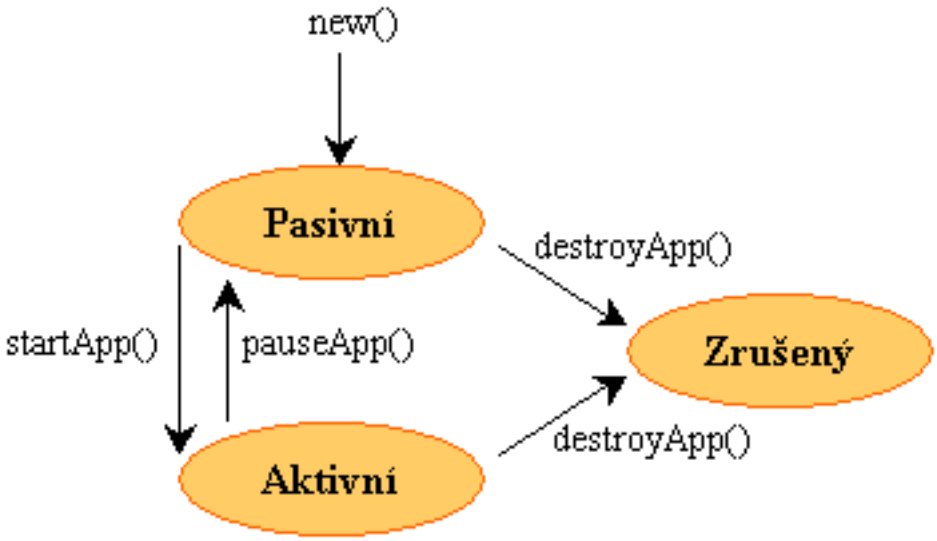
RMI profil rozšiřuje možnosti Základního profilu pro daná zařízení.

4. MIDlet

MIDlet je javovský program pro malá zařízení, přesněji pro virtuální stroj Javy ME. Některými vlastnostmi se podobá appletu. Rozšiřuje speciální třídu *MIDlet* a v zařízení běží v tzv. sandboxu, což je uzavřené prostředí, které nesmí opustit.

4.1. Životní cyklus MIDletu

Základní třída musí rozšiřovat abstraktní třídu *javax.microedition.midlet.MIDlet*. Při spouštění aplikace je tato třída vytvořena zavoláním svého veřejného konstruktoru bez parametrů. Stavy, ve kterých se může MIDlet nacházet, ukazuje následující obrázek. Běh aplikace a její přechod mezi stavy řídí aplikační manažer.



Obr. 4-1: Životní cyklus MIDletu

Po zavolání konstruktoru se aplikace nachází v **pasivním stavu**. V tomto stavu by neměla vlastnit či používat žádné sdílené zdroje. Pro přechod z pasivního stavu do **aktivního stavu** je zavolána metoda *startApp()*. V této metodě aplikace inicializuje zdroje typu *Thread, Connection* ap. Při volání metody *pauseApp()*, kterou se aplikace

vrací do pasivního stavu, by měla aplikace tyto zdroje opět uvolnit. Při ukončení aplikace zavolá aplikační manažer metodu *destroyApp(boolean unconditional)*. Pokud je parametr *true*, bude aplikace ukončena v každém případě, pokud je *false*, může dát vyhozením výjimky *MIDletStateChangeException* najevo, že ještě ukončena být nechce.

4.2. Struktura MIDletu jako třídy

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Midlet extends MIDlet {

   public void Midlet() {
      /* konstruktor třídy Midlet – inicializuje základní proměnné */
   }

   public void startApp() {
       /* metoda je prováděna při spuštění MIDletu */
}

   public void pauseApp() {
       /* metoda je volána buď při spuštění aplikace, nebo při přerušení aplikace
       např. při příchozím hovoru na mobilním telefonu, v tomto stavu by měla
       aplikace uvolnit všechny zdroje, které v tu chvíli používá */
   }

   public void destroyApp(boolean unconditional) {
       /* pokud je volána tato metoda s parametrem unconditional true, bude
       aplikace ukončena */
   }
}
```

5. Geografický informační systém (GIS)

Geografický informační systém je nástroj, který získává, používá a zpracovává údaje, které jsou polohově vázány k povrchu Země. Většina objektů, jevů a činností v reálném světě se vztahuje k určitému místu na Zemi. Zároveň se tyto objekty vyskytují v daném prostoru spolu s dalšími objekty a navzájem se ovlivňují. Znalost umístění a vzájemných prostorových souřadnic potom pomáhá v různých oblastech lidské činnosti. V praxi to znamená, že v počítači jsou zaznamenána data v podobě údajů o objektu (např. v databázích, tabulkách, textových dokumentech, snímcích nebo výkresech) a současně i údaje o jeho poloze dané zeměpisnými souřadnicemi tohoto objektu. Těmto datům se říká geografická a počítačovému systému, který s takovými daty pracuje, říkáme geografický informační systém.

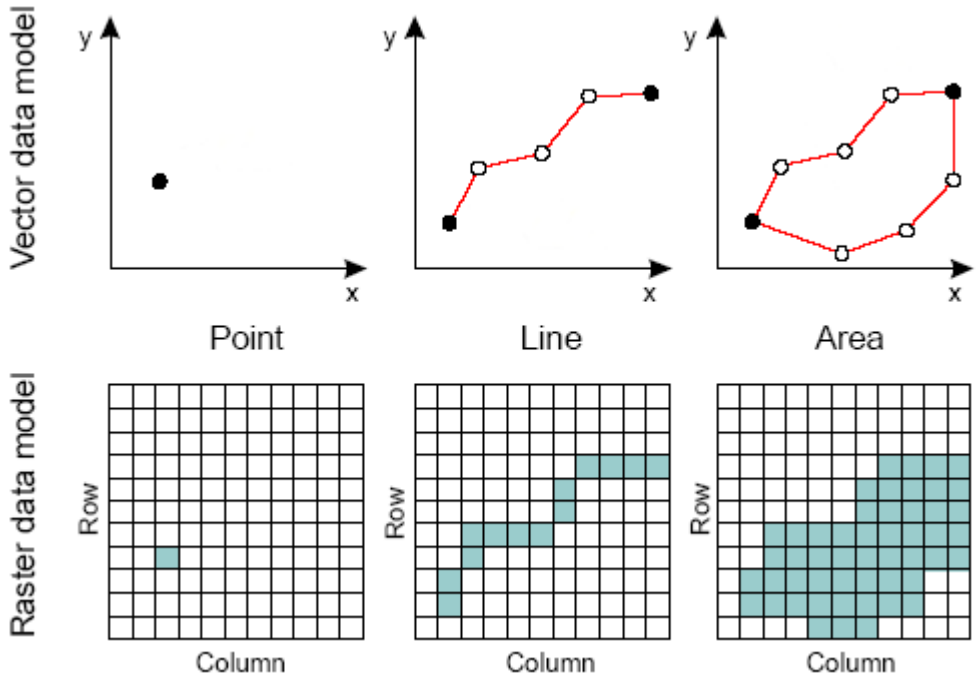
Geografická data mohou být v GIS organizovány dvěma základními modely. Tato reprezentace se následně odráží i v možnostech jejich zpracování, ukládání, analýze a prezentaci.

5.1. Reprezentace dat

Vektorový model dat reprezentuje reálné objekty v prostoru třemi základními typy geometrických prvků. Jsou to:

- body – charakterizovány jednou souřadnicí v prostoru
- linie – uspořádané soubory zřetězených souřadnic
- plochy – uzavřené obrazce např. čtverce, mnohoúhelníky, polygony

Rastrový model dat reprezentuje reálné objekty jako množinu bodů stejné velikosti uspořádaných v rastru, což je síť těchto bodů.



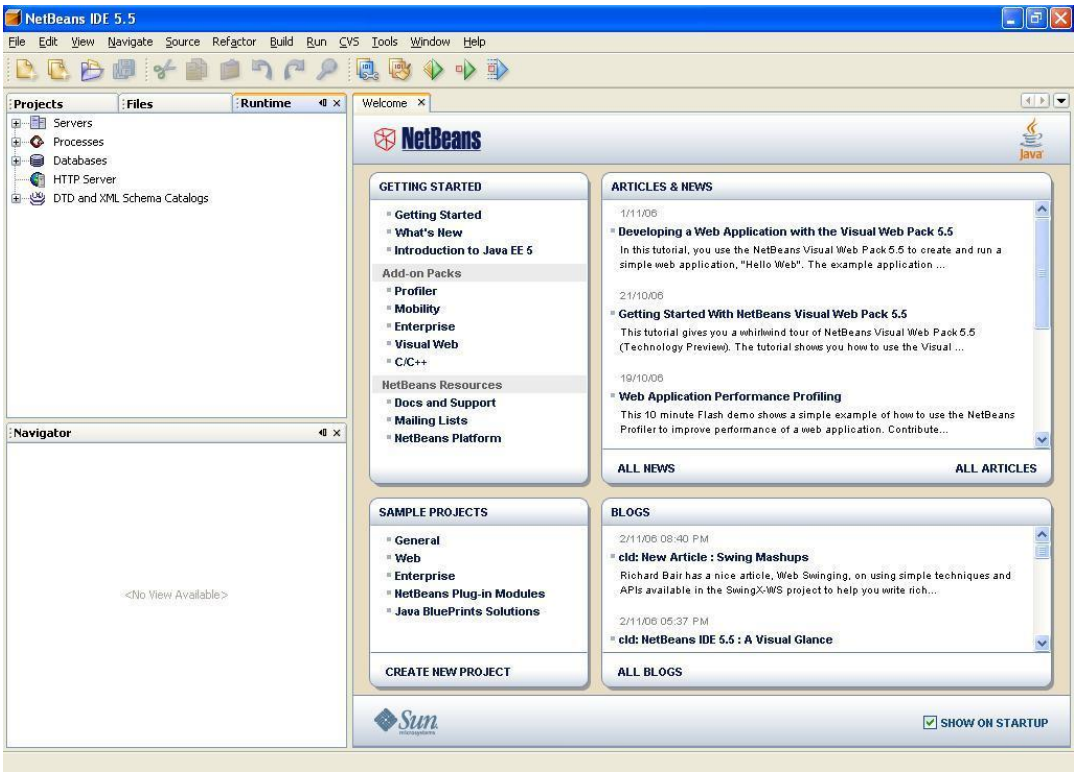
Obr. 5-1: Zobrazení vektorových a rastrových modelů dat

Praktické využití GIS je velmi různorodé. Lze ho využít např.:

- ve státní správě a samosprávě (evidence majetku, obyvatel)
- v kartografii (zpracování map)
- k marketingovým průzkumům (průzkum trhu)
- v ekologii (vývoj krajiny, odpady)
- v lesnictví a zemědělství (půda, hospodaření)
- v integrovaném záchranném systému (policie, hasiči, záchranná služba)
- v armádě (modelování činnosti vojsk)

6. Vývojové prostředí NetBeans IDE

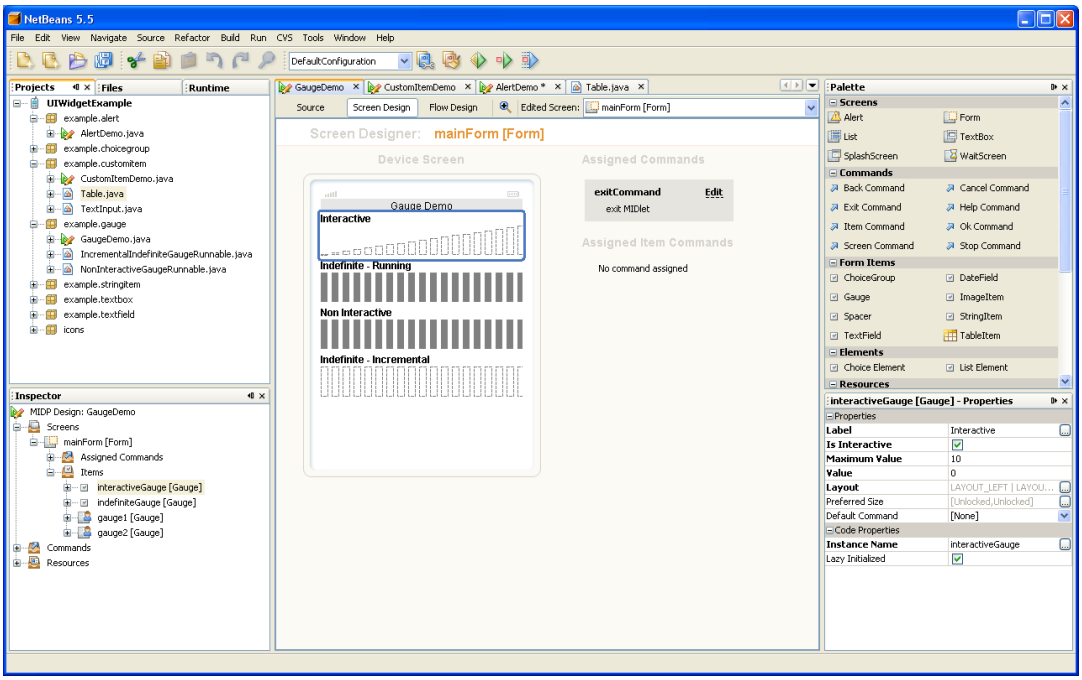
Vývojové prostředí NetBeans IDE, ve kterém byla vytvořena aplikace, je nástroj, pomocí kterého mohou programátoři psát, překládat, ladit a šířit programy. Vývojové prostředí je vytvářeno v jazyce Java, ale nabízí podporu pro **mnoho programovacích jazyků** (např. C++). Mimo to existuje velké množství modulů, které mohou toto vývojové prostředí rozšířit. NetBeans IDE je bezplatně šířený produkt a je ho možné používat bez jakéhokoli omezení. Původně vzniklo na Matematicko-fyzikální fakultě Univerzity Karlovy v roce 1997 jako projekt Xelfi, v roce 1999 bylo koupeno společností Sun Microsystems a v roce 2000 bylo uvolněno pod licencí OpenSource, což přispělo k jeho rychlému vývoji.



Obr. 6-1: Vývojové prostředí NetBeans IDE 5.5

6.1. NetBeans Mobility Pack

Při vývoji v prostředí NetBeans IDE byl použit modul NetBeans Mobility Pack sloužící pro vývoj, testování a ladění aplikací s platformou Java 2 Micro Edition. Mobility Pack integruje podporu Mobile Information Device Profile (MIDP) 2.0 a pro Connected Limited Device Configuration (CLDC) 1.1. Pro testovací prostředí můžeme jednoduše přidávat emulátory různých výrobců mobilních zařízení. Mobility Pack podporuje ladění aplikace přímo v telefonech a umožňuje vyvíjet aplikace i pouhým přetahováním objektů myší, což z něj dělá sofistikovaný nástroj pro vývoj mobilních aplikací.



Obr. 6-2: NetBeans Mobility Pack

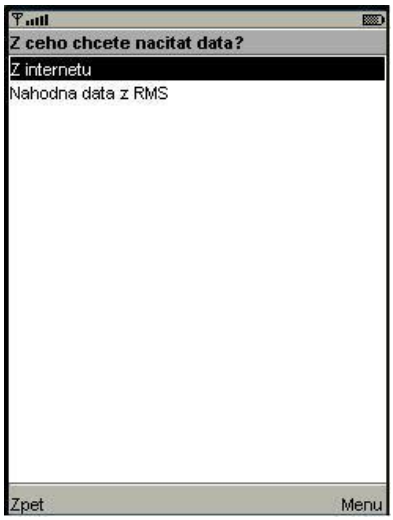
Obr. 6-3: Emulátor mobilního zařízení

Na obrázku 6-3 můžeme vidět emulátor mobilního zařízení použitý pro vývoj aplikace. Tento emulátor je součástí balíčku NetBeans Mobility Pack.

7. Vlastní realizace aplikace

7.1. Struktura aplikace

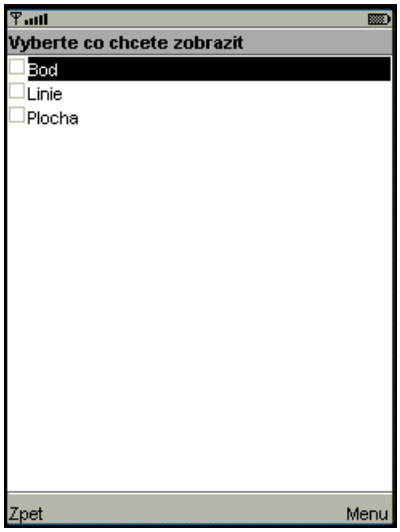
Aplikace je naprogramovaná za pomocí objektově orientovaného programovacího jazyku Java 2 Micro Edition. Po spuštění aplikace máme na výběr ze dvou možností načítání dat. A to buď z internetu stažením libovolného textového dokumentu s definovanou strukturou, nebo náhodně vygenerovaných dat uložených v RMS.



Obr. 7-1: Úvodní obrazovka s možností výběru zdroje dat

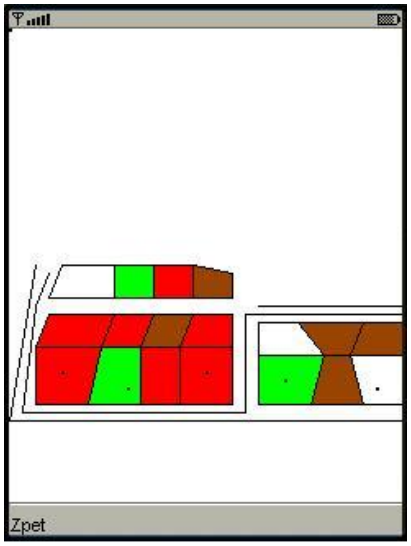
Record Management Systém (RMS) – systém správy záznamů je používaný v J2ME z toho důvodu, že pevné úložiště dat poskytovaných platformou J2SE např. JDBC jsou pro přenosná zařízení příliš velká. Úložiště záznamů se skládá ze seznamu záznamů, které tvoří pole bytů.

Po vybrání zdroje dat a jeho načtení do příslušných proměnných dojde k přechodu na další obrazovku, kde můžeme vybrat, které prvky chceme zobrazit (body, linie a plochy).



Obr. 7-2: Obrazovka s možností výběru zobrazovaných prvků

Jakmile provedeme potvrzení výběru zobrazovaných prvků, zobrazí se obrazovka s vykreslenými daty v podobě zjednodušené mapy.



Obr. 7-2: Obrazovka s vykreslenými daty

Na této obrazovce máme možnost vykreslené prvky posouvat pomocí kurzorových kláves nebo se vrátit zpět na výběr zobrazovaných prvků.

7.2. Způsoby načítání dat

Data jsou v aplikaci načítána ze dvou různých zdrojů, a to z textového souboru, uloženého na webovém serveru na adrese http://www.sweb.cz/dana.kotatkova/data.txt, ve kterém jsou uložena data jednoduché mapy obsahující všechny tři typy prvků vektorově reprezentovaných dat GIS. Další ze zdrojů je databáze typu RMS, do které byla načtena náhodně vygenerovaná data taktéž reprezentující všechny tři prvky vektorových dat GIS.

7.2.1. Třída obsluhující načítání dat z internetu

Třída obsluhující načítání dat z webového serveru je řešena za pomoci vláknového zpracování.

Vláknové zpracování – Sekvenční neboli postupné zpracování není vždy tak efektivní, jak by mohlo být. Proto Java nabízí zpracování vláken, tzv. *threading*, tedy schopnost zpracovat více paralelních procesů v rámci jedné aplikace. Stejně jako víceúlohové zpracování umožňuje běh více programů na jednoprocesorovém počítači, tak vícevláknové zpracování umožní jedné aplikaci zpracovávat více aktivit najednou, což vede k efektivnějšímu a rychlejšímu běhu aplikace. Existuje více způsobů jak vytvořit vlákno. Prvním způsobem je vytvořit třídu, která dědí od třídy *Thread*, dalším způsobem a způsobem použitým v aplikaci je vytvořit třídu, která implementuje rozhranní *Runnable* a posledním způsobem je definovat vlákno jako instanci anonymní třídy, která dědí od třídy *Thread.* Vlákno se může nacházet ve čtyřech stavech. Při spuštění vlákna metodou *start()* se vlákno převede do stavu připravený (*ready*). Po přidělení procesoru přejde vlákno do stavu běžící (*running*). Potřebuje-li vlákno čekat na externí událost nebo na uvolnění nějakého prostředku, je ve stavu čekající (*suspended*). Po ukončení činnosti vlákna je vlákno považováno za mrtvé (*terminated*). Třída obsluhující načítání dat z internetu *pripoj* je deklarována následujícím způsobem, kdy v deklaraci je implementováno rozhraní *Runnable* potřebné pro vytvoření vlákna.

public class pripoj implements Runnable

Připojení se obsluhuje za pomoci rozhraní *HttpConnection* obsažené v balíku *javax.microedition.io*, Dalším důležitým prvkem potřebným pro připojení k síti je abstraktní třída *InputStream* pro zpracování vstupního bytového proudu. Po spuštění vlákna obsluhujícího připojení k síti je použito rozhraní *Connection*, které představuje nejzákladnější typ připojení, a jeho metoda

open(String name),

jehož parametrem je v aplikaci použita cílová adresa souboru se vstupními daty. Dále je nutné nastavit parametr přístupové metody

setRequestProperty().

První z možností je parametr GET, kde jsou vstupní hodnoty posílány jako součást URL. Druhou možností je metoda POST, která posílá data jako výstupní proud. V tuto chvíli je navázáno spojení a aplikace může načítat data.
Data jsou načítána pomocí metody *read()* třídy *inputStream* po jednotlivých znacích a přiřazována metodou *append()* do proměnné typu *StringBuffer*, dokud není dosaženo konce souboru. Poté je textový řetězec typu *StringBuffer* převeden na řetězec typu *String* metodou *toString()*. Tento řetězec je dále zpracován v cyklu a jednotlivé hodnoty souřadnic a určení typu zobrazovaného prvku jsou uloženy do jednorozměrného pole. Z tohoto pole je potřeba v cyklu odstranit prázdné znaky jako mezery a tabulátory a to metodou *trim()*. V tuto chvíli má aplikace k dispozici pole, které obsahuje identifikátory prvků a jejich příslušných hodnot. Z tohoto pole aplikace načte hodnoty prvků a uloží je do strukturovaného datového typu *data*. Jelikož jsou v poli data typu *String*, musí být příslušné číselné hodnoty převedeny na typ *int* metodou *parseInt()* třídy *Integer*. Aplikace má ošetřené výjimky blokem *try-catch*

Zpracování výjimek – výjimku chápeme jako výjimečné, neočekávané nebo chybové události při běhu programu. Výjimka může být vyvolána při chybném otevření souboru, při překročení mezí pole, apod. Mechanismus výjimek umožňuje tyto chybové stavy zachytit a ošetřit. K základním příkazům patří příkazy *try*, *catch*, a *finally*.

```
try {
/*část zdrojového kódu, ve které se může vyskytnout výjimka*/
 } catch (Exception ex) {
/*tato část se provede, pokud se v bloku try vyskytne výjimka*/
} finally{
/*část, která se provede bez ohledu na vznik výjimky*/
}
```

7.2.2. Třída obsluhující načítání dat z RMS

Třída obsluhující načítání z databáze, využívá technologie RMS (Record Management System) česky systém správy záznamů. Je určen pro ukládání dat pro dlouhodobé použití. Každá aplikace má možnost vytvořit teoreticky neomezený počet tzv. RecordStore, ale každý z nich musí mít jiný název. Záznamy může používat pouze MIDlet, který je vytvořil. Každý RecordStore může obsahovat libovolný počet záznamů, ke kterým se přistupuje pomocí unikátních ID, které začínají od 1. Záznamem se rozumí pole bytů. Třída je tvořena metodami pro generování a zápis a následné načtení dat z RMS do datového typu *data*.

Při zvolení zdroje dat se nejprve vymaže *RecordStore*, kterou aplikace vytvořila v minulosti metodou

deleteRecordStore(String recordStoreName),

kde *recordStoreName* je název používané *RecordStore*. V zápětí vytvoří *RecordStore* nový, který naplní náhodně generovanými hodnotami. Poté se z *RecordStore* data načtou do datového typu *data*.

Pro náhodné generování hodnot je potřeba vytvořit instanci třídy *Random* z balíčku *java.util.Random*. Tento objekt využívá metodu

nextInt(int n),

kde *n* je nejvyšší hodnota náhodného čísla, které metoda vrací.

Pokud chceme pracovat s *RecordStore* musíme importovat balíček *javax.microedition.rms*. Pro otevření *RecordStore* se používá metoda

openRecordStore(String recordStoreName, boolean createIfNecessary),

kde parametr *recordStoreName* je dané *RecordStore* a parametr *createIfNecessary* indikuje, zda má být *RecordStore* vytvořen, pokud neexistuje. Po otevření *RecordStore* je možné pracovat se záznamy uloženými v *RecordStore*. *RecordStore* je nutno zavřít metodou

closeRecordStore(),

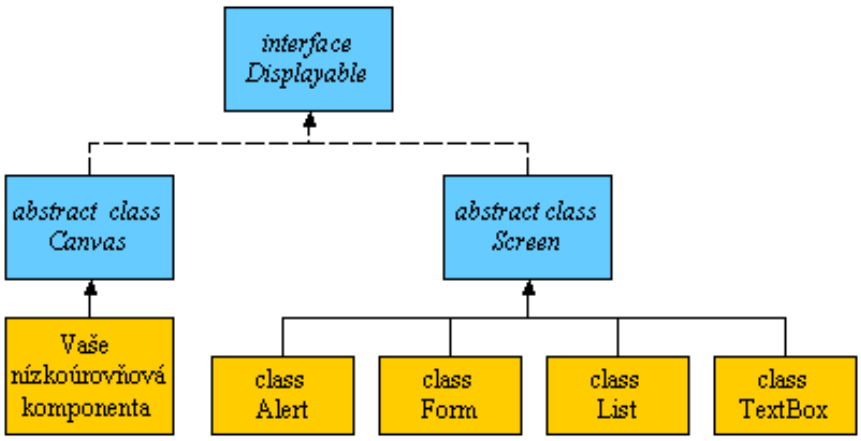
přičemž kolikrát byla otevřena, tolikrát musí být zavřena, aby byla opravdu uzavřena.

V této třídě je dále použit balíček *java.io*. Z daného balíčku jsou použity třídy *DataInputStream, ByteArrayInputStream, DataOutputStream* a *ByteArrayOutputStream.*

7.3. Třída obsluhující vykreslování dat

Základní třída, která obsluhuje vykreslování nízko-úrovňové grafiky na displej je třída Display. V rámci jednoho MIDletu může existovat pouze jedna instance této třídy, kterou můžeme získat použitím metody *Display.getDisplay().* Pro používání třídy Display je potřeba importovat balíček *javax.microedition.lcdui*.
Objekty, které se zobrazují na displeji, musí být potomkem třídy *Displayable*. Třída Display používá metodu *setCurrent()* pro nastavení aktuálního *Displayable* objektu a pro jeho získání se používá metoda *getCurrent()*.

Další třídou, která umožňuje kreslení na displej v tomto případě vysokoúrovňových je třída *Screen*. Hierarchie všech komponent, které lze vykreslit na displej je zobrazena na obrázku 7-3.



Obr. 7-3: Hierarchie všech komponent odvozených od třídy *Displayable*

Pro vytváření dvojrozměrného výstupu na displej se používá třída *Canvas.* Tato třída je potomkem třídy *Displayable* a umožňuje vykreslování čar, textů a různých tvarů na displej. O samotné vykreslování se potom stará metoda *paint()*, která využívá vlastnosti, které má pro kreslení třída *Graphics* z balíčku *javax.microedition.lcdui*. Tato třída obsahuje funkce pro vytváření primitivních geometrických tvarů, vkládání obrázků, práci s barvami a textem.

Před zobrazením dat se máme možnost vybrat prvky GIS, které chceme vykreslit. Vybrané prvky se po potvrzení výběru uloží pomocí metody *getSelectedFlags()* do pole hodnot typu *boolean*, tento typ může nabývat hodnoty pravda (true nebo 1) nebo hodnoty nepravda (false nebo 0). Toto pole se předává jako parametr spolu s načtenými hodnotami metodě kresli třídy obsluhující vykreslování jednotlivých geografických prvků na displej. Podle hodnot v poli se poté vykreslují jednotlivé prvky. Před vykreslováním na *Canvas* je třeba smazat z displeje vše, co na něm je zobrazeno. To provedeme metodou

fillRect(int i,int i1,int i2,int i3),

která vyplní obdélník zvolenou barvou. Parametry *i* a *i1* jsou souřadnice levého horního rohu, *i2* šířka obdélníku a *i3* jeho výška. Pro nastavení barvy se používá metoda

setColor(int red, int green, int blue),

kde jsou parametry *red, green, blue* hodnoty jednotlivých barevných složek.
K vykreslení bodu byla použita již popsaná metoda *fillRect*, která vykreslila obdélník o šířce a výšce 3 body, jak z toho důvodu, že třída *Graphics* neposkytuje metodu pro kreslení bodu, tak kvůli dobré viditelnosti kresleného bodu. Na některých telefonech s vyšší rozlišovací schopností by byl bod špatně rozpoznatelný. Ke kreslení čar má třída *Graphics* metodu

drawLine(int i, int i1, int i2, int i3),

parametry *i* a *i1* jsou souřadnice počátečního bodu čáry a parametry *i2* a *i3* souřadnice jejího koncového bodu. Plochy, v našem případě nepravidelné čtyřúhelníky, byly vykreslovány za pomoci čtyřech čar metodou drawLine a barvy těchto čtyřúhelníků byly vyplněny pomocí metody pro vyplnění trojúhelníku

fillTriangle(int i, int i1, int i2, int i3, int i4, int i5),

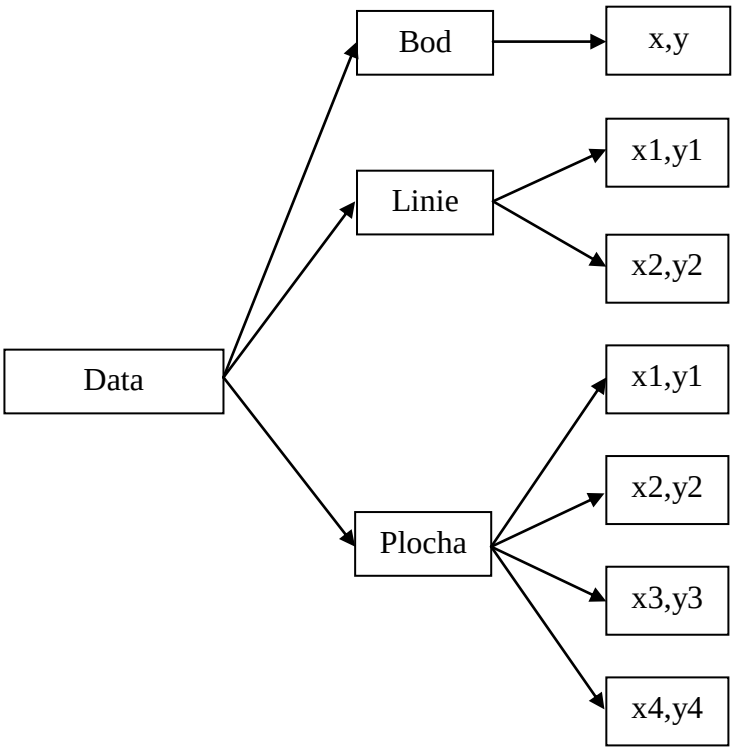
kde parametry *i, i1, i2, i3, i4* a *i5* určují souřadnice jednotlivé souřadnice vrcholů trojúhelníku.
Načítání dat a následné vykreslování probíhá v cyklu od začátku pole do hodnoty jeho délky za pomoci řídícího příkazu

while(výraz) příkaz,

u něhož dochází k vykonávání příkazu do té doby, než výraz nabude hodnoty false.

7.4. Struktura dat

Struktura dat je reprezentována datovým typem *data*, který obsahuje tři datové typy podle typu zobrazovaného objektu (bod, linie, plocha) a každý z nich obsahuje příslušný počet proměnných reprezentujících souřadnice daného typu. Do tohoto typu jsou podle zvoleného způsobu načítání dat ukládány hodnoty souřadnic buď náhodně vygenerovaných a uložených v RMS nebo hodnoty získané ze souboru uloženého na internetu. Struktura datového typu je znázorněna na obrázku 7-4 a ukázka dat uložených pomocí tabulkového editoru je vidět na obrázku 7-5.

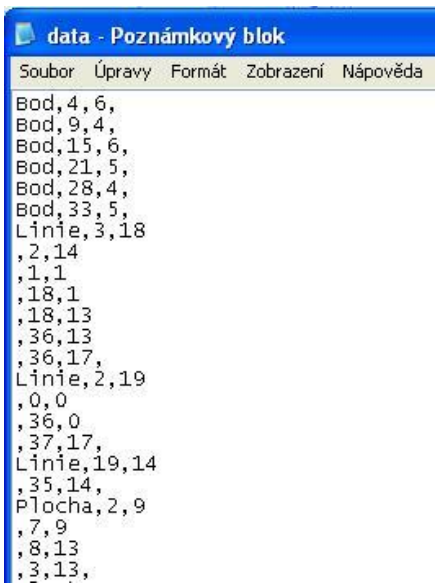


Obr. 7-4: Struktura dat v datovém typu *data*

	A	B	C
1	Bod	4	6
2	Bod	9	4
3	Bod	15	6
4	Bod	21	5
5	Bod	28	4
6	Bod	33	5
7	Linie	3	18
8		2	14
9		1	1
10		18	1
11		18	13
12		36	13
13		36	17
14	Linie	2	19
15		0	0
16		36	0
17		37	17
18	Linie	19	14
19		35	14
20	Plocha	2	9
21		7	9
22		8	13
23		3	13

Obr. 7-5: Ukázka dat uložených pomocí tabulkového editoru

Data jsou v souboru uložena tak, že v prvním sloupci je napsán identifikátor geografického prvku. V druhém a třetím sloupci jsou uloženy příslušné souřadnice daného prvku. Data byla uložena do textového souboru, kde jsou jednotlivé identifikátory a jejich příslušné hodnoty oddělené čárkou, jak je to ukázáno na obrázku 7-6. Z tohoto souboru probíhá čtení tak, jak to bylo popsáno výše.



Obr. 7-6: Data uložená v textovém formátu

7.5. Obsluhování událostí

7.5.1. Obsluhování vysokoúrovňových událostí

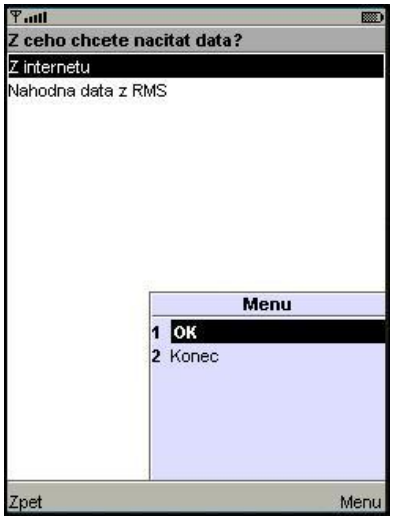
Při interakci uživatele s aplikací, například tak, že vybere položku ze seznamu, se generují události. Aplikace je upozorněna, že má s událostí pracovat, a to pomocí zpětných volání (callback), což jsou vlastně volání metod definovaných programátorem. Aplikace je spouští jako odpověď na akce uživatele za běhu programu. Zpětná volání se používají v mnoha vývojových prostředích, především v těch, která jsou určena pro vývoj aplikací s grafickým uživatelským rozhraním.
Jazyk J2ME proto implementuje rozhraní *CommandListener*, které je určeno pro MIDlety, které potřebují přijímat vysokoúrovňové události. V aplikaci může existovat pouze jeden objekt *CommandListener* pro každou třídu *Displayable*, což je třída objektu, kterou lze zobrazit na displej. Toto rozhraní má jednu metodu, kterou musí příjemce příkazu definovat. Jde o metodu *commandAction*.

public void commandAction(Command c, Displayable d);

První parametr je objekt typu *Command*, což je základní navigační prvek aplikace. S jeho pomocí se identifikuje, který byl do třídy *Displayable* vložen a poté spuštěn. Druhý parametr je objekt té třídy *Displayable*, v níž byla událost vyvolána.
Command je speciální třída, která zapouzdřuje funkce pro správu příkazů daného zařízení. Je schopna příkaz pouze vytvořit a nastavit, o zpracování požadavku se stará rozhraní *CommandListener*. Příkaz se definuje zavoláním kontraktoru

Command(String label, int commandType, int priority),

kde *label* je titulek příkazu, *commandType* je typ příkazu (OK, CANCEL, BACK, EXIT a další) a priorita mění dostupnost a umístění příkazu. Zobrazení příkazů na displeji je předvedeno na obrázku 7-6.



Obr. 7-6: Zobrazení příkazů na displeji

V každé třídě, ve které se mají obsluhovat vysokoúrovňové události, se musí nastavit příjemce těchto událostí za pomoci metody *setCommandListener()*, která je součástí třídy *Displayable* a dědí ji jak třída *Screen*, tak třída *Canvas*, což jsou třídy, které mohou mít příjemce příkazů vkládaných při vzájemné interakci uživatele a aplikace.

public void setCommandListener(CommandListener c);

V naprogramované aplikaci bylo obsluhování vysokoúrovňových událostí použito při přechodech mezi jednotlivými obrazovkami nebo při ukončení aplikace

```
public void commandAction(Command command, Displayable displayable) {
        /*metoda přijímá jako parametry objekt typu Command a objekt typu
        Displayable obrazovky, ve které je CommandListener aktivní*/
    if (command == Exit) destroyApp(true);
        /*při požadavku na ukončení aplikace se přiřadí parametru unconditional
        přiřadí hodnota true*/
    if (command == Ok) {
        /*při požadavku na potvrzení se obslouží událost podle toho, ze které
        obrazovky se událost volá*/
      if (displayable == seznam) {
```

```
    }

    if (displayable == menu) {

        switch (menu.getSelectedIndex()) {
            /*zde se získává index zvolené položky v menu*/
            case 0: {
            } break;

            case 1: {
            } break;
        }
    }
}
```

7.5.2. Obsluhování nízko-úrovňových událostí

Pro přímé kreslení na displej a pro obsluhování událostí, vyvolaných stiskem tlačítka zařízení, na kterém je aplikace spuštěna, se používá třída *Canvas* Třída *Canvas* se často používá při tvorbě her, protože poskytuje metody pro práci s herními akcemi a klávesovými událostmi. O klávesových událostech se hovoří v souvislosti s klávesovými kódy navázanými přímo na konkrétní klávesy daného zařízení. Ve třídě *Canvas* existují následující metody pro obsluhu nízko-úrovňových událostí:

```
protected void keyPressed(int keyCode);
protected void keyReleased(int keyCode);
protected void keyRepeated(int keyCode);
protected void pointerPressed(int x, int y);
protected void pointerDragged(int x, int y);
protected void pointerReleased(int x, int y);
```

Tyto metody představují metody zpětného volání, které by měly být předdefinovány v potomcích třídy *Canvas*.

V naprogramované aplikaci bylo nízko-úrovňové obsluhování událostí použito při posouvání obrazu na displeji, konkrétně byla použita metoda *keyPressed*.

```
protected  void keyPressed(int key) {

    if (key == -2) {
                    /*posunutí směrem dolů*/
                    /*zde dochází k přepočítání souřadnic jednotlivých prvků*/
       repaint();   /*překreslení obrazovky po přepočítání souřadnic*/
    }
}
```

Analogicky dochází k přepočtu a překreslení souřadnic při posunu do ostatních směrů.

8. Závěr

Úkolem bakalářské práce bylo naprogramovat aplikaci, která bude zobrazovat geografická data na mobilních zařízeních. K vytvoření aplikace byl použit programovací jazyk pro tvorbu aplikací pro mobilní zařízení J2ME.

Aplikace byla naprogramována ve vývojovém prostředí NetBeans IDE a odzkoušena na emulátorech tohoto prostředí.

Pro vytvoření bakalářské práce bylo zapotřebí nabýt teoretických znalostí ohledně objektově orientovaného programování a programovacího jazyka J2ME. Dále bylo nutné seznámit se se základními prvky geografických informačních systémů a osvojit si vývojové prostředí NetBeans IDE.

Celá aplikace zahrnuje celkem osm tříd. Jednotlivé třídy obsahují jak základní stavební prvky pro uložení dat, tak třídy řídící běh programu. Výsledný projekt umožňuje načítat ze dvou zdrojů dat, data následně zpracovat a vykreslovat je na displeji zařízení, které je zobrazuje.

Práce je názornou ukázkou programování aplikací pro přenosná zařízení a zařízení s omezenými zdroji.

Seznam použitých zkratek

API – Application programming interface

CDC – Connected Device Configuration

CLDC – Connected Limited Device Configuration

CVM – C Virtual Machine

GIS – Geografický Informační Systém

IDE – Integrated Development Environment

J2EE – Java 2 Enterprise Edition

J2ME – Java 2 Micro Edition

J2SE – Java 2 Standard Edition

JCP – Java Community Process

JDBC – Java Database Connectivity

JNI – Java Native Interface

JVM – Java Virtual Machine

KVM – K Virtual Machine

MIDP – Mobile Information Device Profile

Např. – například

OOP – Objektově orientované programování

PDA – Personal Digital Assistant

RAM – Random Access Memory

RMI – Remote Method Invocation

RMS – Record Management System

ROM – Read Only Memory

Seznam použité literatury a odborných textů:

[1] Quasay H. Mahmoud: Naučte se Java 2 Micro Edition, Grada Publishing, 2002

[2] Jim Keogh, Mario Giannini: Objektově orientované programování bez předchozích znalostí, Computer Press, 2006

[3] http://www.cdsw.cz/gis.htm

[4] www.interval.cz

[5] www.kraj-lbc.cz

[6] www.unesco.org

[7] www.netbeans.org