Dokumentace k programu AstroMIKČ

Michal Čonos

28. září 2002

1 Úvod

V roce 1996 jsem začal psát knihovnu obsahující základní astronomické funkce, potřebné pro výpočty poloh nebeských těles, jejich západů, konjunkcí, apod. Nad touto knihovnou psanou v jazyce C jsem tehdy vystavěl program, který si vzal z klávesnice údaje o pozorovateli (jeho polohu, natočení, ...), aktuální datum a čas. Program pak na oplátku pak velmi pomalu vykreslil na obrazovku momentální uspořádání několika hvězd, Slunce, Měsíce a několika terestrických planet. To bylo na počítači Amiga 1200, s frekvencí 14,7MHz.

Dnes, tedy o 6 let později, mě napadlo vytvořit něco podobného pod platformu Java. Zpočátku jsem byl velmi skeptický, co se rychlosti vykreslování týče, ale po napsaní zkušební verze apletu mne obavy opustily. Java již dlouho není interpret ale runtime-compiler a frekvence dneších procesorů se pohybují o řád výše než před šesti lety. Dnešní AstroMIKC tedy umí realtime zobrazování poloh hvězd. Co ale neumí zobrazit, jsou planety, Slunce a Měsíc. Výpočty jejich poloh jsou mnohem složitější než výpočty hvězd, neboť jejich nebeské souřadnice se vzhledem k Zemi neustále mění[1]. Tento program je spíše určen pro předváděcí účely — demonstruje, jak se otáčí nebeská sféra — a také pro rychlou orientaci mezi aktuálně viditelnými hvězdami. Není určen pro přesné stanovení poloh jednotlivých hvězd — to se v programu nikde nedočteme.

Hvězdné mapy jsem si opět dělal sám, neboť abych mohl nebeskou sféru „roztočit“, tak jsem nemohl vzít seznamy hvězd obsahující desetitisíce sou-

---

[1] Pochopitelně, že rovníkové souřadnice hvězd na nebeské sféře se také mění s časem, ale v tomto programu to a) neuvažujeme, b) tyto změny jsou pro laické účely velmi malé, až zanedbatelné.

řadnic. Oproti Amiga verzi je ale tento seznam opět rozšířenější, ale zdaleka ne ještě v podobě, kterou si představuji.

2 Popis balíku

Balík AstroMIKČ byl vyvíjen GNU nástroji na systému Linux a jeho distribuce je tedy pro tyto systémy typická. Na WWW stránkách *http://vellum.cz/~mikc/* se nachází domovská stránka, na které je možné buď AstroMIKČe přímo provozovat ve webovém prohlížeči jako Java aplet (je třeba, aby prohlížeč podporoval Javu a mít ji zapnutou) nebo si stáhnout kompletní distribuci s predkompilovaným bytekódem a obsahující zdrojové soubory. Pokud si vyberete druhou možnost, můžete si AstroMIKČe vykompilovat. K tomu ale potřebujete Java SDK, které je ke stažení na *http://www.javasoft.com.*

2.1 Obsah balíku a jeho rozbalení

Po stažení do svého adresáře musíme nejdříve distribuci rozbalit, to provedeme následujícím způsobem:

```
gzip -dc astromikc.tar.gz | tar -xvf -
```

Tím se nám distribuce rozpakuje. Balík má následující adresářovou strukturu:

```
api/
doc/
html/
src/
Makefile
```

Adresář `api/` obsahuje dokumentace vytvořené programem `javadoc`. V `doc/` je umístěna tato dokumentace a pokyny pro instalaci. Po vykompilování obsahuje adresář `html/` zkompilované soubory a hlavní HTML stránku nazvanou `main.html`. Konečně samotné zdrojové soubory jsou v `src/` a soubor `Makefile` obsahuje pravidla pro GNU make.

2.2 Kompilace balíku

Balík se vykompiluje snadno zadáním příkazu make:

```
make
```

Teď by se měly v adresáři `html/` nacházet všechny třídy potřebné pro spuštění.

2.3 Vytvoření dokumentace ke třídám a jejich metodám

Programátorská dokumentace(API reference) se vytvoří zadáním:

```
make api
```

Nyní je v adresáři `api/` dokumentace popisující jednotlivé třídy a jejich metody.

2.4 Puštění programu

Program se dá pustit dvěma způsoby: buď použitím utility `make`, nebo přímo z příkazové řádky.

2.4.1 Puštění programu pomocí utility make

Na příkazový řádek napíšeme:

```
make run
```

Program by se měl rozeběhnout a nahodit okno. Je potřeba mít nastavenou cestu k programu `appletviewer`.

2.4.2 Puštění programu z příkazové řádky

Na příkazovou řádku napíšeme:

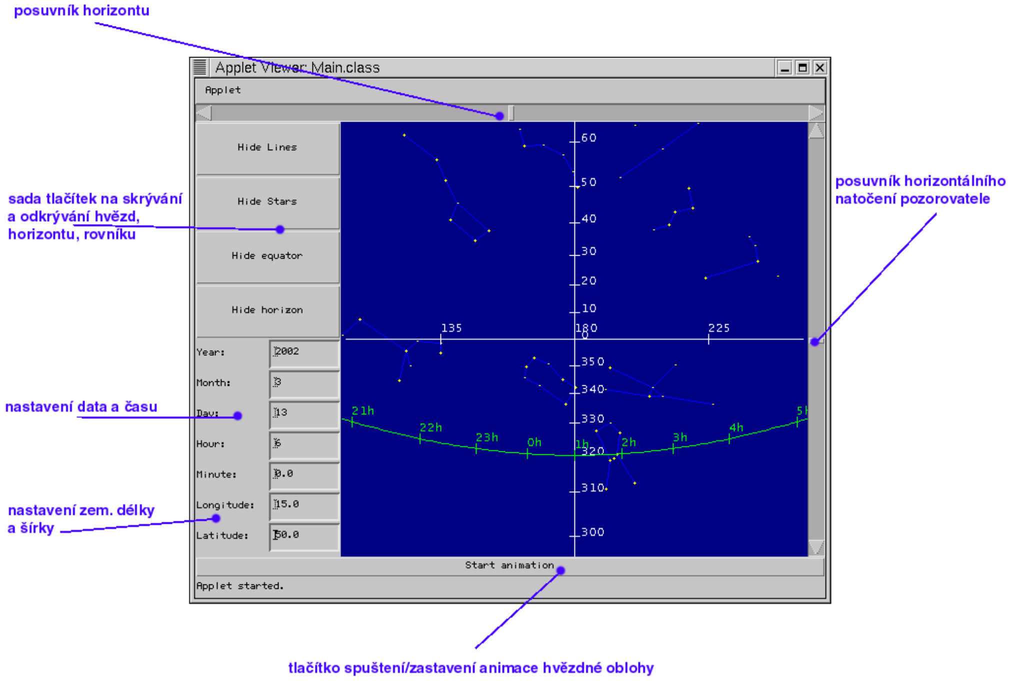
```
appletviewer html/main.html
```

Program se musí spustit stejně jako v předchozím případě.

3 Uživatelské rozhraní

Program využívá grafické rozhraní Javy AWT. Měl by se tedy rozeběhnout i na prohlížečích, které podporují pouze starší verze Javy. V případě obtíží, je možné stáhnout Java Runtime Environment z *http://www.javasoft.com*, verze 3.0, na které je program testován.

Po spuštění uvidíme následující okno:



posuvník horizontu

sada tlačítek na skrývání a odkrývání hvězd, horizontu, rovníku

nastavení data a času

nastavení zem. délky a šírky

posuvník horizontálního natočení pozorovatele

tlačítko spuštení/zastavení animace hvězdné oblohy

Year: 2002
Month: 3
Day: 13
Hour: 6
Minute: 0.0
Longitude: 15.0
Latitude: 50.0

Applet started.

Obr.1 — Uživatelské rozhraní programu AstroMIKČ

Horizontálním a vertikálním posuvníkem můžeme měnit natočení pozorovatele. Aktuální astronomický azimut[2] se dá skrýt/zobrazit tlačítkem *Hide/Show Horizon*. Výchozí pozice pozorovatele je na sever, proto je číslo na horizontální stupnici rovno 180°. Obdobně vertikálním posuvníkem měníme vertikální polohu natočení pozorovatele. Číslo protínající obzorníkovou

---

[2] Astronomický jih (180°) začíná na zeměpisném severu, směr otáčení je stejný — po směru hodinových ručiček.

kružnici zobrazuje aktuální náklon ve stupních. Na začatku je roven 0°. Můžeme si všimnout, že polárka se zobrazuje v blízkosti 50°, což je pro naši zeměpisnou šířku 50° v pořádku.

Žlutou čarou je zobrazen světový rovník, místo stupňů raději používáme hodinovou míru, s přepočtem $1h = 15°$. Světový rovník se dá skrýt/zobrazit tlačítkem *Hide/Show Equator*.

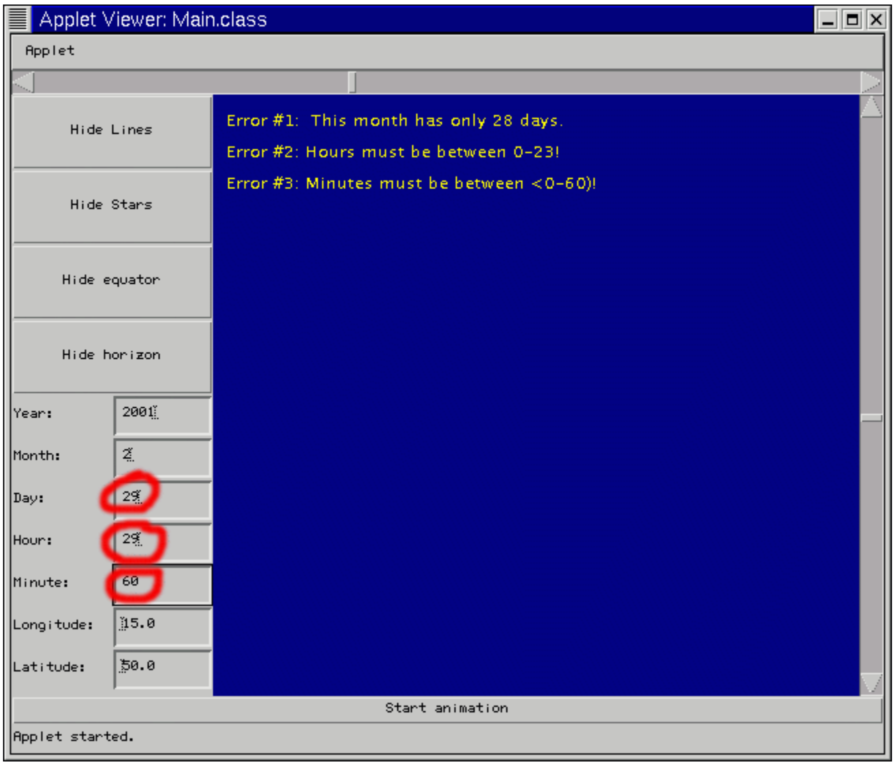
Tlačítky *Hide/Show Stars*, resp. *Hide/Show Lines* se dají skrýt/zobrazit samotné hvězdy, resp. spojnice mezi nimi.

Talčítkem *Start/Stop Animation* se pouští/zastavuje animace nebeské sféry.

Vstupní textová pole jsou pro zadávání číselných časových a datumových hodnot, měsíc je povolen v intervalu (1–12), 1—Leden, 2—Únor, ... Den v závislosti na roce a měsíci v intervalu (1–31), hodina v intervalu (0–23), minuta v intervalu (0–60). Položky zeměpisné šířky a délky nejsou omezeny, neboť funkce nad nimi pracující jsou periodické. Jsou zadávány ve stupních. Rok rovněž není omezen, ale je třeba si uvědomit fakt, že i rovníkové souřadnice hvězd nejsou neměnné a v programu jsou vztažené k roku 1975, tedy chyba narůstá se zvětšující se časovou vzdáleností od tohoto roku. V případě zadání nesprávné hodnoty se na ploše standardně zobrazující hvězdnou oblohu objeví všechny chybové hlášky způsobené nepovolenými vstupními hodnotami.

4 Chybová hlášení

Program AstroMIKČ při každém vložení hodnoty do polí obsahujících datum, čas nebo zeměpisné souřadnice kontroluje platnost zadaných dat. Na následujícím obrázku vidíme výpis nekolika chybových hlášek způsobených zadaním neplatných dat (na obrázku zakroužkovány červeně).



```
Error #1:  This month has only 28 days.
Error #2: Hours must be between 0-23!
Error #3: Minutes must be between <0-60)!
```

Year: 2001
Month: 2
Day: 29
Hour: 29
Minute: 60
Longitude: 15.0
Latitude: 50.0

```
Applet started.
```

Obr.2 — Ukázka možných chybových hlášení programu AstroMIKČ

Běží-li animace a uživatel zadá nesprávnou hodnotu, animace se zastaví a na obrazovce je vypsána chyba. Dokud uživatel chybu neopraví, vykreslování oblohy nebude probíhat.

4.1 Přehled chybových hlášení

- *Chyba:* `Month must be between (1-12)!`
  Význam: Měsíc není v povoleném rozsahu, 1—Leden, 2—Únor, . . .
- *Chyba:* `Day number must be greater than 0!`
  Význam: Číslo dne musí být větší než nula, v závislosti na měsíci a roce v intervalu (1–31).
- *Chyba:* `This month has only ... days!`
  Význam: Číslo dne je větší než povolené pro daný měsíc.
- *Chyba:* `Hours must be between (0-23)!`
  Význam: Hodiny mohou nabývat hodnot pouze v rozsahu (0–23).

- *Chyba:* `Minutes must be between (0-60)!`
  Význam: Minuty mohou nabývat hodnot pouze v rozsahu (0–60).

5 Dodatek - Výpis zdrojového kódu

Modul **BasicAstroTrans.java**. Tato třída se stará o základní astronomické transformace a výpočty.

```
/**
        This class includes some basic astronomical transformations:

        <ul>
        <li>convertGregorian2Julian()
        <li>convertJulian2Gregorian()
        <li>localStarTime()
        <li>transformDLe2dl()
        </ul>

        @author Michal Conos
        @version 0.0.1
*/

class BasicAstroTrans {
        /**
        Converts <b>GregorianDate</b> into Julian date (which is here represented
        as a <b>double</b> value)

        @param gdate <b>GregorianDate</b> object

        @return Julian date

        @see <b>GregorianDate</b> object
        */
        public static double convertGregorian2Julian ( GregorianDate gdate ) {
                double m, y, a, b, jd1, jd2, M, Y, D;

                M = m = gdate.getM();
                Y = y = gdate.getY();
                D = gdate.getD();

                if ( m < 2 ) {
                        y--;
                        m += 12;
        }

        a = Math.floor (y/100);
        b = 2 - a + Math.floor(a/4);

        jd1 = Math.floor (365.25*y) + Math.floor(30.6001*(m+1)) + D +
              1720994.5;
        jd2 = jd1 + b;

        if ( ( Y == 1582 && M >= 10 && D >= 15 ) || ( Y >= 1582 ) ) {
                return jd2;
        } else {
                return jd1;
        }
}


/**
Sorry, this function is <b>NOT</b> implemented yet.
*/
public static GregorianDate convertJulian2Gregorian (double JD) {
        GregorianDate gd = new GregorianDate (0.0, 0.0, 0.0);
        return gd;

}


/**
This function returns <b>StarTime</b> object, it is calculated from
Julian date, longitude of observer ( for whom StarTime is computed ),
local time of observer.

@param JD Julian date
@param longitude observer's longitude
@param t observer's local time

@return <b>StarTime</b> object

@see <b>StarTime</b> object
*/
```

```
public static StarTime localStarTime ( double JD, double longitude,
                                       double t ) {
        double T3, S0, S, s;
        StarTime st = new StarTime ( 0.0, 0.0, 0.0);

        T3 = (JD - 2451545.0)/36525;
        S0 = 6.697374558 + 2400.05133691*T3 + 0.0000258622*T3*T3 -
             0.0000000017*T3*T3*T3  ;

        S0 -= Math.floor (S0 / 24)*24;

        if (S0 < 0) S0+=24;

        S = S0 + 1.0027379093*t;
        if (S > 24) S-=24;
        s = S + longitude;


        st.sets (s);
        st.setS (S);
        st.setS0 (S0);

        return st;
}


/**
This function in major does a transformation from one spherical system
into another one.
<p>
We've an object which has coordinates <i>D</i>, <i>L</i> in one
sphericalsystem and we want to know its coordinates in another
one as <i>d</i>, <i>l</i>, we also know the angle between theese
two systems which is <i>e</i>.

@param DLe <b>SystemDLe</b> object.

@return <b>Systemdl</b> object.
```

```
        */
        public static Systemdl transformDLe2dl (SystemDLe DLe) {
                double D, L, e, d, l;
                double sin_d, tan_l;
                double x,y;
                Systemdl sdl = new Systemdl (0, 0);

                double cos_e, sin_e, sin_D, cos_D, cos_L, cos_D_sin_L;

                D = DLe.getD();
                L = DLe.getL();
                e = DLe.gete();

                cos_e = Math.cos(e);
                sin_e = Math.sin(e);
                sin_D = Math.sin(D);
                cos_D = Math.cos(D);
                cos_L = Math.cos(L);
                cos_D_sin_L = cos_D * Math.sin(L);

                y  = cos_D * cos_L;
                x  =  sin_e * sin_D + cos_e * cos_D_sin_L;
                sin_d = cos_e * sin_D - sin_e * cos_D_sin_L;
                tan_l = y / x;

                d = Math.asin (sin_d);
                l = Math.atan ( tan_l );
                if ( x < 0 ) l += Math.PI;

                sdl.setd (d);
                sdl.setl (l);

                return sdl;
        }
}
```

Modul **GregorianDate.java**. Provádí výpočty a převody mezi Gregoriánským a Juliánským datem.

```
public class GregorianDate {
        private double Y, M, D;


        /**
        This is a constructor for a <b>GregorianDate</b> object.
        It will take three parameters : year, month and date.

        @param Y represents year
        @param M represents month
        @param D represents day

        @return  object <b>GregorianDate</b>
        */
        public GregorianDate (double Y, double M, double D) {
                this.Y = Y;
                this.M = M;
                this.D = D;
        }

        /**
        Gets year field from <b>GregorianDate</b> object

        @return year of <b>GregorianDate</b> object
        */
        public double getY () {
                return this.Y;
        }

        /**
        Gets month field from <b>GregorianDate</b> object

        @return month of <b>GregorianDate</b> object
        */
        public double getM () {
                return this.M;
        }
```

```
/**
Gets day field from <b>GregorianDate</b> object

@return day of <b>GregorianDate</b> object
*/
public double getD () {
        return this.D;
}

/**
Sets year as a new value of year field in <b>GregorianDate</b> object.

@param Y stands for a new value of year
*/

public void setY ( double Y ) {
         this.Y = Y;
}


/**
Sets month as a new value of month field in <b>GregorianDate</b>
object.

@param M stands for a new value of month
*/
public void setM ( double M ) {
         this.M = M;
}

/**
Sets day as a new value of day field in <b>GregorianDate</b> object.

@param D stands for a new value of day
*/

public void setD ( double D ) {
         this.D = D;
}
```

```
}
```

Modul **Point2D.java**. Implementuje základní operace nad bodem ve 2-D soustavě.

```
public class Point2D {
        private double x, y;

        /**
        Contructor of point in 2-dimensional space.

        @param x x-axis value
        @param y y-axis value

        @return new object <b>Point2D</b>

        @see <b>Point3D</b>
        */
        public Point2D (double x, double y) {
                this.x = x;
                this.y = y;
        }

        /**
        Gets x-axis value of <b>Point2D</b> object.

        @return x-axis value
        */
        public double getx () {
                return this.x;
        }

        /**
        Gets y-axis value of <b>Point2D</b> object.

        @return y-axis value
        */
        public double gety () {
                return this.y;
        }
        /**
        Sets x-axis value of <b>Point2D</b> object.

        @param x-axis new value
        */
        public void setx ( double x ) {
                 this.x = x;
        }

        /**
        Sets y-axis value of <b>Point2D</b> object.

        @param y-axis new value
        */
        public void sety ( double y ) {
                 this.y = y;
        }

        /**
        Returns the angle of <b>Point2D</b> defined as atan(y/x).
        NOTE: atan has a period of PI

        @return value described above
        @see <b>angle2()</b>
        */
        public double angle() {
                return Math.atan(y/x);
        }

        /**
        Returns the distance of <b>Point2D</b> from the origin
        defined as (0, 0)

        @return value described above.
        */
        public double distance() {
                return Math.sqrt(x*x+y*y);
        }

}
```

Modul **Point3D.java**. Implementuje základní operace nad bodem ve 3-D soustavě.

```
public class Point3D {
        private double x, y, z;

        /**
        Contructor of point in 3-dimensional space.

        @param x x-axis value
        @param y y-axis value
        @param z z-axis value

        @return new object <b>Point3D</b>

        @see <b>Point2D</b>
        */
        public Point3D (double x, double y, double z) {
                this.x = x;
                this.y = y;
                this.z = z;
        }

        /**
        Gets x-axis value of <b>Point3D</b> object.

        @return x-axis value
        */
        public double getx () {
                return this.x;
        }

        /**
        Gets y-axis value of <b>Point3D</b> object.

        @return y-axis value
        */
        public double gety () {
                return this.y;
}

/**
Gets z-axis value of <b>Point3D</b> object.

@return z-axis value
*/
public double getz () {
        return this.z;
}

/**
Sets x-axis value of <b>Point3D</b> object.

@param x-axis new value
*/
public void setx ( double x ) {
         this.x = x;
}

/**
Sets y-axis value of <b>Point3D</b> object.

@param y-axis new value
*/
public void sety ( double y ) {
         this.y = y;
}

/**
Sets z-axis value of <b>Point3D</b> object.

@param z-axis new value
*/
public void setz ( double z ) {
         this.z = z;
}

/**
Returns the angle of <b>Point3D</b> defined as atan(y/x).
```

```
        NOTE: atan has a period of PI

        @return value described above
        */
        public double angleXY() {
                return Math.atan(y/x);
        }

        /**
        Returns the angle of <b>Point3D</b> defined as atan(x/z).
        NOTE: atan has a period of PI

        @return value described above
        */
        public double angleXZ() {
                return Math.atan(z/x);
        }

        /**
        Returns the angle of <b>Point3D</b> defined as atan(z/y).
        NOTE: atan has a period of PI

        @return value described above
        */
        public double angleZY() {
                return Math.atan(z/y);
        }

        /**
        Returns the distance of <b>Point3D</b> from the origin defined
        as (0, 0)

        @return value described above.
        */
        public double distance() {
                return Math.sqrt(x*x+y*y+z*z);
        }

}
```

Modul **Star.java**. Základní implementace objektu hvězda spolu s její reprezentací v rovníkových souřadnicích.

```
public class Star {
        private double a, d;

        /**
        Constructor for a <b>Star</b> object.
        <b>Star</b> object has two coordinates on space sphere,
        rectascension and declination.

        @param a rectascension of <b>Star</b>
        @param d declination of <b>Star</b>

        @return new <b>Star</b> object
        */
        public Star (double a, double d) {
                this.a = a;
                this.d = d;
        }

        /**
        Gets a rectascension value of <b>Star</b>

        @return rectascension value
        */
        public double geta () {
                return this.a;
        }

        /**
        Gets a declination value of <b>Star</b>

        @return declination value
        */
        public double getd () {
                return this.d;
        }
        /**
        Sets a rectascension value of <b>Star</b>

        @param a rectascension value
        */

        public void seta (double a) {
                this.a = a;
        }

        /**
        Sets a declination value of <b>Star</b>

        @param d declination value
        */
        public void setd (double d) {
                this.d = d;
        }
}
```

Modul **StarMap.java**. Data hvězdné mapy, spolu s metodami kreslení objektů mapy.

```
/**
        This class is responsible for computing and drawing stars,
        horizon, equator, etc.

        In major it consists of:

        <ul>
        <li>drawHorizon()
        <li>drawLine()
        <li>drawEquator()
        <li>drawStars()
        </ul>

        @author Michal Conos
        @version 0.1.1
*/

import java.awt.*;

public class StarMap {
        private int width  = 640;
        private int height = 480;

        private final int RADIUS = 450;


        private double starmap[] = {

 /* Ursa Minor, Ursae Minoris, UMi */
         2.07, 89.09, 2.02,    /* 1 ,    0       */
        17.40, 86.36, 4.36,    /* 23,    1       */
        16.48, 82.05, 4.23,    /* 22,    2       */
        15.44, 77.52, 4.32,    /* 16,    3       */
        14.50, 74.15, 2.08,    /* 7      4       */
        15.20, 71.55, 3.05,    /* 13     5       */
 /* Virgo, Virginis , Vir */
```

```
       13.00, 11.06, 2.84,    /* 47     6       */
       12.54,  3.32, 3.38,    /* 43     7       */
       13.33, -0.28, 3.38,    /* 79     8       */
       12.00,  8.40, 4.04,    /* ??     9       */
       12.40, -1.19, 2.74,    /* 29    10       */
       12.18, -0.32, 3.90,    /* 15    11       */
       11.49,  1.54, 3.60,    /* 5     12       */
       13.08, -5.24, 4.38,    /* 51    13       */
       13.23,-11.02, 0.97,    /* 67    14       */
/* Ursa Maior, Ursae Maioris, UMa */
       13.46, 49.26, 1.86,    /* 85    15       */
       13.22, 55.03, 2.06,    /* 79AB  16       */
       12.52, 56.06, 1.77,    /* 77    17       */
       12.14, 57.10, 3.31,    /* 69    18       */
       11.02, 61.53, 1.79,    /* 50    19       */
       11.00, 56.31, 2.37,    /* 48    20       */
       11.52, 53.50, 2.44,    /* 64    21       */
/* Orion, Orionis, Ori */
        5.46, -9.41, 2.05,    /* 53    22       */
        5.39, -1.57, 1.77,    /* 50    23       */
        5.53,  7.24, 0.42,    /* 58    24       */
        5.34,  9.55, 3.39,    /* 39    25       */
        5.23,  6.20, 1.46,    /* 24    26       */
        5.30, -0.19, 2.24,    /* 34    27       */
        5.13, -8.14, 0.13,    /* 19    28       */
        5.34, -1.13, 1.69,    /* 46    29       */
/* Taurus, Tauri, Tau */
        4.34, 16.28, 0.86,    /* 87    30       */
        4.27, 19.08, 3.54,    /* 74    31       */
        4.18, 15.34, 3.65,    /* 54    32       */
        5.24, 28.35, 1.65,    /* 112   33       */
        5.36, 21.08, 3.03,    /* 123   34       */
        3.25,  9.35, 3.75,    /* 2     35       */
        3.47, 24.04, 5.0,     /* BU    36       */
/* Lyra, Lyrae, Lyr */
       18.36, 38.46, 0.03,    /* 3     37       */
       18.43, 37.35, 4.36,    /* 6     38       */
       18.49, 33.20, 3.42,    /* 10    39       */
       18.58, 32.39, 3.24,    /* 14    40       */
       18.54, 36.52, 6.2,     /* d2    41       */
```

```
/* Cygnus, Cygni, Cyg */
       20.40, 45.11, 1.25,    /* 50    42        */
       20.21, 40.11, 2.23,    /* 37    43        */
       19.44, 45.04, 2.87,    /* 18    44        */
       19.16, 53.19, 3.76,    /* 1     45        */
       20.45, 33.53, 2.46,    /* 53    46        */
       21.11, 30.07, 3.20,    /* 64    47        */
       19.29, 27.54, 3.08,    /* 6     48        */
/* Aquila, Aquilae, Aql */
       19.49,  8.48, 0.76,    /* 53    49        */
       19.24,  3.04, 3.36,    /* 30    50        */
       19.04, -4.55, 3.43,    /* 16    51        */
       19.04, 13.50, 2.99,    /* 17    52        */
       20.10,  0.54, 3.24,    /* 65    53        */
/* Cassiopeia, Cassiopeiae, Cas */
        1.52, 63.33, 3.38,    /* 45    54        */
        1.24, 60.06, 2.68,    /* 37    55        */
        0.55, 60.35, 2.39,    /* 27    56        */
        0.39, 56.24, 2.23,    /* 18    57        */
        0.07, 59.01, 2.37,    /* 11    58        */
/* Andromeda, Andromedae, And */
        2.02, 42.13, 2.10,    /* 57    59        */
        1.08, 35.29, 2.05,    /* 43    60        */
        0.07, 28.57, 2.06,    /* 21    61        */
        0.55, 38.22, 3.87,    /* 37    62        */
        0.48, 40.57, 4.40,    /* 35    63        */
/* Hercules, Herculis, Her */
       17.14, 36.50, 3.16,    /* 67    64        */
       16.42, 38.58, 3.50,    /* 44    65        */
       16.40, 31.39, 2.81,    /* 40    66        */
       16.59, 30.58, 3.92,    /* 58    67        */
       17.55, 37.15, 3.87,    /* 91    68        */
       17.38, 46.01, 3.80,    /* 85    69        */
       16.19, 46.22, 3.90,    /* 22    70        */
       16.08, 45.00, 4.27,    /* 11    71        */
       16.29, 21.33, 2.74,    /* 27    72        */
       17.13, 14.25, 3.06,    /* 64    73        */
       16.20, 19.13, 3.76,    /* 20    74        */
       17.14, 24.52, 3.13,    /* 65    75        */
       17.29, 26.08, 4.41,    /* 76    76        */
       17.45, 27.44, 3.42,    /* 86    77        */
       17.57, 30.12, 4.41,    /* 94    78        */
/* Leo, Leonis, Leo */
        9.44, 23.53, 2.98,    /* 17    79        */
        9.51, 26.08, 3.88,    /* 24    80        */
       10.15, 23.33, 3.44,    /* 36    81        */
       10.18, 19.58, 1.98,    /* 41    82        */
       10.07, 12.05, 1.35,    /* 32    83        */
       10.06, 16.53, 3.53,    /* 30    84        */
       11.12, 20.40, 2.56,    /* 68    85        */
       11.47, 14.43, 2.14,    /* 94    86        */
       11.12, 15.34, 3.35,    /* 70    87        */
/* Cepheus, Cephei, Cep */
       23.38, 77.30, 3.21,    /* 35    88        */
       22.48, 66.04, 3.53,    /* 32    89        */
       22.28, 58.17, 4.34,    /* 27    90        */
       21.18, 62.29, 2.45,    /* 5     91        */
       21.28, 70.27, 3.23,    /* 8     92        */
       20.44, 61.44, 3.43,    /* 3     93        */
       20.29, 62.55, 4.28,    /* 2     94        */
       22.10, 58.05, 3.35,    /* 21    95        */
/* Pegasus, Pegasi, Peg */
       21.30,  9.46, 2.39,    /* 8     96        */
       22.08,  6.04, 3.52,    /* 26    97        */
       22.40, 10.42, 3.40,    /* 42    98        */
       23.03, 15.04, 2.48,    /* 54    99        */
        0.11, 15.03, 2.84,    /* 88   100        */
       23.02, 27.56, 2.50,    /*     101       */
       22.41, 30.05, 2.95,    /* 44   102        */
       22.48, 24.28, 3.48,    /* 48   103        */
/* Bootes, Bootis, Boo */
       14.14, 19.19,-0.05,    /* 16   104        */
       14.30, 30.29, 3.59,    /* 25   105        */
       14.31, 38.25, 3.02,    /* 27   106        */
       15.01, 40.29, 3.50,    /* 42   107        */
       15.14, 33.24, 3.49,    /* 49   108        */
       14.44, 27.11, 2.37,    /* 36   109        */
       14.40, 13.49, 3.78,    /* 30   110        */
/* Corona Borealis, Coronae Borealis, CrB */
       15.56, 26.57, 4.15,    /* 13   111        */
```

```
      15.42, 26.22, 3.85,    /* 8    112       */
      15.33, 26.48, 2.24,    /* 5    113       */
      15.26, 29.11, 3.68,    /* 3    114       */
      15.31, 31.27, 4.13,    /* 4    115       */
/* Crater, Crateri, Crt */
      11.35, -9.8,  5.00,    /* ??   116       */
      11.25,-10.05, 4.50,    /* ??   117       */
      11.20,-14.39, 3.56,    /* 12   118       */
      11.23,-17.33, 4.08,    /* 15   119       */
      11.38,-18.00, 4.50,    /* ??   120       */
      11.55,-17.30, 4.50,    /* ??   121       */
      10.58,-18.10, 4.07,    /* 7    122       */
      11.10,-22.41, 4.48,    /* 11   123       */
/* Libra, Librae, Lib */
      15.02,-25.11, 3.27,    /* 20   124       */
      14.49,-15.56, 2.75,    /* 9    125       */
      15.15, -9.18, 2.61,    /* 27   126       */
      15.34,-14.42, 3.91,    /* 38   127       */
/* Pisces, Piscium, Psc */
       1.10, 29.57, 4.51,    /* 83   128       */
       1.30, 15.13, 3.62,    /* 99   129       */
       1.44,  9.02, 4.26,    /* 110  130       */
       2.01,  2.38, 3.82,    /* 113  131       */
       1.40,  5.22, 4.44,    /* 106  132       */
       1.01,  7.45, 4.28,    /* 71   133       */
       0.47,  7.27, 4.44,    /* 63   134       */
      23.58,  6.44, 4.01,    /* 28   135       */
      23.38,  5.29, 4.13,    /* 17   136       */
      23.26,  6.14, 4.28,    /* 10   137       */
      23.15,  3.09, 3.69,    /* 6    138       */
      23.26,  0.57, 5.90,    /* ??   139       */
      23.02,  3.41, 4.52,    /* 4    140       */
/* Aquarius, Aquarii, Aqr */
      23.13, -6.11, 4.22,    /* 90   141       */
      23.14, -9.13, 4.25,    /* 91   142       */
      23.41,-14.41, 4.51,    /* 105  143       */
      23.21,-20.14, 3.98,    /* 98   144       */
      23.08,-21.19, 3.64,    /* 88   145       */
      22.53,-15.57, 3.28,    /* 76   146       */
      22.48,-13.44, 3.98,    /* 71   147       */
```

```
        22.51, -7.43, 3.79,    /* 73   148       */
        22.15, -7.55, 4.15,    /* 43   149       */
        22.05,-14.00, 4.25,    /* 33   150       */
        22.20, -1.31, 3.84,    /* 48   151       */
        22.04, -0.27, 2.93,    /* 34   152       */
        21.30, -5.41, 2.87,    /* 22   153       */
        21.08,-11.28, 4.52,    /* 13   154       */
        20.46, -9.35, 3.77,    /* 2    155       */
 /* Gemini, Geminorum, Gem */
         6.36, 16.25, 1.92,    /* 24   156       */
         6.21, 22.32, 2.87,    /* 13   157       */
         6.42, 25.09, 2.98,    /* 27   158       */
         7.09, 30.18, 4.40,    /* 46   159       */
         7.33, 31.57, 1.58,    /* 66   160       */
         7.43, 28.05, 1.14,    /* 78   161       */
         7.42, 24.28, 3.57,    /* 77   162       */
         7.18, 22.02, 3.53,    /* 55   163       */
 /* Scorpius, Scorpii, Sco */
        16.50,-38.00, 3.03,    /* u1   164       */
        16.48,-34.15, 2.29,    /* 26   165       */
        16.34,-28.10, 2.81,    /* 23   166       */
        16.27,-26.23, 0.91,    /* 12   167       */
        16.19,-25.32, 2.88,    /* 20   168       */
        16.06,-20.36, 3.97,    /* 9    169       */
        15.58,-22.30, 2.32,    /* 7    170       */
        15.57,-26.03, 2.91    /* 6    171       */
 };

        private int coords[] = new int[ 2*starmap.length ];

        private int counter;
        private int cnum = 0;

        public int LINES  = 157;

        private int lines[]=
{
   0,  1,  1,  2,  2,  3,  3,  4,  4,  5,  6,  7,  7,  8, 30, 32,
   7,  9,  7, 10, 10, 11, 11, 12, 10, 13, 13, 14, 15, 16, 16, 17, 17, 18,
  18, 19, 19, 20, 20, 21, 21, 18, 22, 23, 23, 24, 24, 25, 25, 26, 26, 27,
  27, 28, 30, 31, 31, 32, 31, 33, 30, 34, 32, 35, 31, 36, 37, 38, 38, 39,
  39, 40, 40, 41, 42, 43, 43, 44, 44, 45, 43, 46, 46, 47, 43, 48, 49, 50,
  50, 51, 50, 52, 50, 53, 54, 55, 55, 56, 56, 57, 57, 58, 59, 60, 60, 61,
  60, 62, 62, 63, 64, 65, 65, 66, 66, 67, 67, 64, 64, 68, 68, 69, 65, 70,
  70, 71, 66, 72, 72, 73, 72, 74, 67, 75, 75, 76, 76, 77, 77, 78, 79, 80,
  80, 81, 81, 82, 82, 83, 82, 84, 82, 85, 85, 86, 85, 87, 88, 89, 89, 90,
  90, 91, 91, 92, 91, 93, 93, 94, 90, 95, 96, 97, 97, 98, 98, 99, 99,100,
 100, 61, 61,101,101, 99,101,102,101,103,104,105,105,106,106,107,107,108,
 108,109,109,110,111,112,112,113,113,114,114,115,116,117,117,118,118,119,
 119,120,120,121,118,122,119,123,124,125,125,126,126,127,128,129,129,130,
 130,131,131,132,132,133,133,134,134,135,135,136,136,137,137,138,138,139,
 139,136,138,140,141,142,142,143,143,144,144,145,145,146,146,147,147,148,
 148,149,149,150,151,152,152,153,153,154,154,155,149,152,156,157,157,158,
 158,159,159,160,160,161,161,162,162,163,163,156,164,165,165,166,166,167,
 167,168,168,169,168,170,168,171,0,1
};

        /**
        Constructs a <b>StarMap</b> object. Here we’ve coordinates of stars
        we draw.

        @return new <b>StarMap</b> object
        */
        public StarMap() {
                counter = 0;
                cnum    = 0;
        }

        /**
        Constructs a <b>StarMap</b> with specified width and height.

        @param w width
        @param h height

        @return new <b>StarMap</b> object
        */
        public StarMap(int w, int h) {
                counter = 0;
                cnum    = 0;
                width   = w;
        height  = h;
}

/**
Sets new width for a <b>StarMap</b>

@param width new width
*/
public void setWidth( int width ) {
        this.width = width;
}

/**
Sets new height for a <b>StarMap</b>

@param height new height
*/
public void setHeight( int height ) {
        this.height = height;
}

/**
Computes coordinates of all stars and stores them. To do this
we need to know observer's local time, longitude, latitude, azimut
and eleveation. Also <b>GregorianDate</b> is needed.

@param g <b>Graphics</b> object, where we've drawn (currently we
do not any drawing here).
@param gd <b>GregorianDate</b>
@param t observer's local time
@param longitude observer's longitude
@param latitude observer's latitude
@param azimut observer's azimut
@param elevation observer's elevation
*/
public void computeMap (Graphics g, GregorianDate gd, double t,
                        double longitude, double latitude, int azimut,
                        int elevation ) {
        Star star;
```

```
StarTime st;

Systemdl dl;
SystemDLe dle = new SystemDLe (0,0,0);

double a, d;
double JD;

Point3D p3d;
Point2D p2d;


double local_time;

int x,y;
int w2 = width >> 1;
int h2 = height >> 1;

JD = BasicAstroTrans.convertGregorian2Julian ( gd );



while ( ( star = getNextStar() ) != null )  {
        a = star.geta();
        d = star.getd();

        a *= Math.PI/180;
        d *= Math.PI/180;

        dle.setD (d);
        st = BasicAstroTrans.localStarTime ( JD, longitude, t );
        local_time = -a + st.gets() * 15 * Math.PI / 180;
        dle.setL (Math.PI/2 - local_time);
        dle.sete ( (latitude - 90) * Math.PI / 180);
        dl = BasicAstroTrans.transformDLe2dl (dle);

        p3d = Trans3D.transformPolartoCartesian (dl, RADIUS);
        Trans3D.rotate3DXZ( p3d, azimut );
        Trans3D.rotate3DYZ( p3d,
             ( elevation < 0 ) ? 360 + elevation : elevation );
```

```
                p2d = Trans3D.transformPoint3Dto2D (p3d, width >> 1);

                x = (int)Math.round(p2d.getx()+w2);
                y = (int)Math.round(h2 - p2d.gety());

                coords[ cnum++ ] = x;
                coords[ cnum++ ] = y;

        }


        clearCounter();
}

/**
On demand draws horizon as viewed from observer's position.

@param g where to draw
@param azimut observer's azimut
@param elevation observer's elevation
*/
public void drawHorizon( Graphics g, int azimut, int elevation ) {

        Point3D p3d = new Point3D( 0.0, 0.0, 0.0 );
        Point3D p3d2 = new Point3D( 0.0, 0.0, 0.0 );
        Point2D p2d;
        Point2D p2d2;
        double a = 0;
        double PI2 = 2*Math.PI;
        double rad_elevation = elevation * Math.PI / 180;
        int x, y, ox = 0, oy = 0, x2, y2;
        int w2 = width >> 1;
        int h2 = height >> 1;
        double da = Math.PI / 180;
        int az = azimut;
        int el = elevation;

        g.setColor( Color.white );

        g.drawLine(w2, 0, w2, height);
```

```
while ( a < PI2 ) {
        p3d.setx( RADIUS * Math.sin(a) );
        p3d.sety( 0.0 );
        p3d.setz( RADIUS * Math.cos(a) );


        p3d2.setx( 0.0 );
        p3d2.sety( p3d.getx() );
        p3d2.setz( p3d.getz() );

        Trans3D.rotate3DYZ( p3d, elevation * Math.PI/180 );
        Trans3D.rotate3DYZ( p3d2, elevation * Math.PI/180 );

        p2d = Trans3D.transformPoint3Dto2D (p3d, width >> 1);
        p2d2 = Trans3D.transformPoint3Dto2D (p3d2, width >> 1);


        x = (int)Math.round(p2d.getx()+w2);
        y = (int)Math.round(h2 - p2d.gety());

        x2 = (int)Math.round(p2d2.getx()+w2);
        y2 = (int)Math.round(h2 - p2d2.gety());

        if ( a > 0 &&
             x >= 0 && x < width && y >= 0 && y < height &&
             ox >= 0 && ox < width && oy >=0 && oy < height) {
                g.drawLine( ox, oy, x, y );
        }

        if ( ( az % 45 )  == 0 ) {
                g.drawString(Integer.toString( az ), x, y - 7 );
                g.drawLine(x,y-5,x,y+5);
        }

        if ( ( el % 10 ) == 0 ) {
                g.drawString(Integer.toString( el ), x2+7, y2 );
                g.drawLine(x2-5,y2,x2+5,y2);

        }
```

```
                ox = x;
                oy = y;
                a += da;
                az++;
                el++;
                az %= 360;
                el %= 360;
        }

}

/**
On demand draws equator as viewed from observer's position.

@param g where to draw
@param azimut observer's azimut
@param elevation observer's elevation
@param latitude observer's latitude
@param longitude observer's longitude
*/
public void drawEquator( Graphics g, int azimut, int elevation,
                         double latitude, double longitude ) {
        Point3D p3d = new Point3D( 0.0, 0.0, 0.0 );
        Point2D p2d;

        double a = 0;
        double da = Math.PI / 180;
        double PI2 = 2 * Math.PI;

        int x, y, ox = 0, oy = 0;
        int h2 = height >> 1;
        int w2 = width  >> 1;
        int eq = (int) ( longitude * 15 );

        g.setColor( Color.green );

        while ( a < PI2 ) {
                p3d.setx( -RADIUS * Math.sin(a) );
                p3d.sety( 0.0 );
                p3d.setz( -RADIUS * Math.cos(a) );

                Trans3D.rotate3DYZ( p3d,
                                  ( latitude - 90 ) * Math.PI / 180 );
                Trans3D.rotate3DXZ( p3d,  azimut * Math.PI / 180 );
                Trans3D.rotate3DYZ( p3d,  elevation * Math.PI / 180 );

                p2d = Trans3D.transformPoint3Dto2D (p3d, width >> 1);

                x = (int)Math.round(p2d.getx()+w2);
                y = (int)Math.round(h2 - p2d.gety());

                if ( a > 0 &&
                     x >= 0 && x < width && y >= 0 && y < height &&
                     ox >= 0 && ox < width && oy >=0 && oy < height) {
                        g.drawLine( ox, oy, x, y );
                }


                if ( ( eq % 15 )  == 0 ) {
                        g.drawString(Integer.toString( eq/15 )+"h",
                                     x, y - 7 );
                        g.drawLine(x,y-5,x,y+5);
                }

                ox = x;
                oy = y;

                a += da;
                eq++;
                eq %= 360;
        }

}

/**
On demand draws lines between stars as viewed from observer's position.

@param g where to draw
*/
```

```
public void drawLines( Graphics g ) {
        int i = 0;
        int x1, x2, y1, y2;
        Star star;

        clearCounter();

        g.setColor( Color.blue );
        for ( i = 0; i < lines.length; i+=2 ) {
        x1 = coords[ 2*lines[i]       ];
                y1 = coords[ 2*lines[i]   + 1 ];
                x2 = coords[ 2*lines[i+1]     ];
                y2 = coords[ 2*lines[i+1] + 1 ];

                if ( x1 >= 0 && x1 < width && y1 >= 0 && y1 < width &&
                     x2 >= 0 && x2 < width && y2 >= 0 && y2 < width ) {

                        g.drawLine( x1, y1, x2, y2 );
                }
        }

        clearCounter();
}

/**
On demand draws stars as viewed from observer's position.

@param g where to draw
*/
public void drawStars( Graphics g ) {
        int i = 0;
        int x, y;
        Star star;

        clearCounter();

        g.setColor( Color.yellow );
        while ( ( star = getNextStar() ) != null )  {

                x = coords[i];
                y = coords[i+1];

                if ( x >= 0 && x < width && y >=0 && y < height ) {
                        drawStar (g, x, y);
                }

                i += 2;
        }

        clearCounter();
}

/**
Draws one star. How it is drawn depends on its magnitude.
This function is not finished yet, some improvements should occur.

@param g where to draw
@param x stars' x-position on screen
@param y stars' y-position on screen
*/
public void drawStar(Graphics g, int x, int y) {
        /* star magnitude */
        double m = (6 - starmap [counter - 1])/3;
        int r;
        m++;
        r = (int) Math.floor ( m );
        if ( r < 0 ) r=3;

        switch (r) {
                case 0:
                case 1:
                        g.drawLine (x, y, x+1,y);
                        break;
                case 2:
                        g.drawLine (x, y+1, x, y-1);
                        g.drawLine (x-1, y, x+1, y);
                        break;
                default:
                        g.fillOval (x,y, r, r);
        }
```

```
	}

	/**
	This method is currently used private. It returns next <b>Star</b>
	in <b>StarMap</b>.

	@return next <b>Star</b> in map.
	*/
	public Star getNextStar () {
		double RA, RA_frac;
		int RA_int;

		if ( counter <  starmap.length) {
			counter += 3;
			RA = starmap[counter-3];
			RA_int = (int) Math.floor(RA);
			RA_frac = RA - RA_int;
			RA = RA_int + RA_frac/0.6;
			RA *= 15;

			return new Star (RA, starmap[counter-2]);
		} else {
			return null;
		}
	}

	/**
	Sometimes we need to clear the next star counter.
	*/
	public void clearCounter() {
		counter = 0;
		cnum    = 0;
	}

}
```

Modul **StarTime.java**. Vlastní převody hvězdného času na lokální a opačně dělá modul **BasicAstroTrans.java**.

```
public class StarTime {
        private double s, S, S0;

        public StarTime (double s, double S, double S0) {
                this.s = s;
                this.S = S;
                this.S0 = S0;
        }

        public double gets () {
                return this.s;
        }

        public double getS () {
                return this.S;
        }

        public double getS0 () {
                return this.S0;
        }

        public void sets ( double s ) {
                 this.s = s;
        }

        public void setS ( double S ) {
                 this.S = S;
        }

        public void setS0 ( double S0 ) {
                 this.S0 = S0;
        }
}
```

Modul **SystemDLe.java**. Obsahuje definice pro převody mezi astronomickými souřadnými systémy.

```
public class SystemDLe {
        private double D, L, e;

        /**
        <b>SystemDLe</b> constructor. Please see astronomical documentation for
        further details.
        */
        public SystemDLe (double D, double L, double e) {
                this.D = D;
                this.L = L;
                this.e = e;
        }

        public double getD () {
                return this.D;
        }

        public double getL () {
                return this.L;
        }

        public double gete () {
                return this.e;
        }

        public void setD ( double D ) {
                 this.D = D;
        }

        public void setL ( double L ) {
                 this.L = L;
        }

        public void sete ( double e ) {
                 this.e = e;
        }
}
```

Modul **Systemdl.java**. Obsahuje definice pro převody mezi astronomickými souřadnými systémy.

```
public class Systemdl {
        private double d, l;

        /**
        <b>Systemdl</b> constructor. Please see astronomical documentation for
        further details.
        */
        public Systemdl (double d, double l) {
                this.d = d;
                this.l = l;
        }

        public double getd () {
                return this.d;
        }

        public double getl () {
                return this.l;
        }

        public void setd ( double d ) {
                 this.d = d;
        }

        public void setl ( double l ) {
                 this.l = l;
        }

}
```

Modul **Trans3D.java**. Obsahuje výpočty a transformace pro 3D, 2D soustavy.

```
/**
        This class implements some oprations on points in 3-dimensional space.

        <ul>
        <li>transformPoint3Dto2D()
        <li>initTables()
        <li>rotateXY()
        <li>rotate3DXY()
        <li>rotate3DXZ()
        <li>rotate3DYZ()
        <li>transformPolartoCartesian()
        </ul>

        @author Michal Conos
        @version 0.0.5
*/

public class Trans3D {

        static double sinus_a[] = new double[360];
        static double cosinus_a[] = new double[360];

        /**
        Transforms <b>Point3D</b> into 2-dimensional <b>Point2D</b>.

        @param p <b>Point3D</b>
        @param fov field of view

        @return transformed <b>Point2D</b>
        */
        public static Point2D transformPoint3Dto2D ( Point3D p, double fov ) {
                double x3d, y3d, z3d;
                double x2d, y2d;

                y2d = x2d = -10000;
                double k;

                x3d = p.getx();
                y3d = p.gety();
                z3d = p.getz();


                if ( z3d > 0 ) {
                        k = z3d + fov;
                        x2d = fov * x3d / k;
                        y2d = fov * y3d / k;
                }

                return new Point2D (x2d, y2d);
        }

        /**
        When we use a fast version of sinus and cosinus, we need to do some
        initial inits. The user is responsible to call this method before
        calling any from the fast-computing functions.
        */
        public static void initTables() {
                int i;

                for ( i = 0; i < 360; i++ ) {
                        sinus_a[ i ] = Math.sin( i*Math.PI / 180 );
                        cosinus_a[ i ] = Math.cos( i*Math.PI / 180 );
                }
        }

        /**
        Rotation of <b>Point2D</b> about angle <i>a</i>.

        @param p <b>Point2D</b> to rotate
        @param a angle of rotation
        */
        public static void rotateXY ( Point2D p, double a ) {
                double x, y, cos_a, sin_a;

                x = p.getx();
        y = p.gety();

        cos_a = Math.cos(a);
        sin_a = Math.sin(a);

        p.setx ( x * cos_a - y * sin_a );
        p.sety ( x * sin_a + y * cos_a );
}

/**
Rotation of <b>Point3D</b> in XY-space about angle <i>a</i>
Fast version.

@param p <b>Point3D</b>
@param a angle of rotation
*/
public static void rotate3DXY( Point3D p, int a ) {
        double x, y, cos_a, sin_a;

        x = p.getx();
        y = p.gety();

        cos_a = cosinus_a[ a ];
        sin_a = sinus_a[ a ];

        p.setx( x * cos_a - y * sin_a );
        p.sety( x * sin_a + y * cos_a );
}

/**
Rotation of <b>Point3D</b> in XZ-space about angle <i>a</i>
Fast version.

@param p <b>Point3D</b>
@param a angle of rotation
*/
public static void rotate3DXZ( Point3D p, int a ) {
        double x, z, cos_a, sin_a;

        x = p.getx();
        z = p.getz();

        cos_a = cosinus_a[ a ];
        sin_a = sinus_a[ a ];

        p.setx( x * cos_a - z * sin_a );
        p.setz( x * sin_a + z * cos_a );
}

/**
Rotation of <b>Point3D</b> in YZ-space about angle <i>a</i>
Fast version.

@param p <b>Point3D</b>
@param a angle of rotation
*/
public static void rotate3DYZ( Point3D p, int a ) {
        double y, z, cos_a, sin_a;

        y = p.gety();
        z = p.getz();

        cos_a = cosinus_a[ a ];
        sin_a = sinus_a[ a ];

        p.sety( y * cos_a - z * sin_a );
        p.setz( y * sin_a + z * cos_a );
}

/**
Rotation of <b>Point3D</b> in XY-space about angle <i>a</i>

@param p <b>Point3D</b>
@param a angle of rotation
*/
public static void rotate3DXY( Point3D p, double a ) {
        double x, y, cos_a, sin_a;

        x = p.getx();
        y = p.gety();
        cos_a = Math.cos(a);
        sin_a = Math.sin(a);

        p.setx( x * cos_a - y * sin_a );
        p.sety( x * sin_a + y * cos_a );
}

/**
Rotation of <b>Point3D</b> in XZ-space about angle <i>a</i>

@param p <b>Point3D</b>
@param a angle of rotation
*/
public static void rotate3DXZ( Point3D p, double a ) {
        double x, z, cos_a, sin_a;

        x = p.getx();
        z = p.getz();

        cos_a = Math.cos(a);
        sin_a = Math.sin(a);

        p.setx( x * cos_a - z * sin_a );
        p.setz( x * sin_a + z * cos_a );
}

/**
Rotation of <b>Point3D</b> in YZ-space about angle <i>a</i>

@param p <b>Point3D</b>
@param a angle of rotation
*/
public static void rotate3DYZ( Point3D p, double a ) {
        double y, z, cos_a, sin_a;

        y = p.gety();
        z = p.getz();

        cos_a = Math.cos(a);
```

```
                sin_a = Math.sin(a);

                p.sety( y * cos_a - z * sin_a );
                p.setz( y * sin_a + z * cos_a );
        }

        /**
        Transforms polar system with radius (described as <b>Systemdl</b>
        and <i>R</i>) into cartesian.

        @param sdl <b>Systemdl</b> object
        @param R radius

        @return <b>Point3D</b> - point in cartesian system.
        */
        public static Point3D transformPolartoCartesian ( Systemdl sdl,
                                                          double R ) {
                double d, l;
                double cos_d;

                d = sdl.getd();
                l = sdl.getl();


                cos_d = Math.cos(d);

                return new Point3D (R*Math.sin(l)*cos_d,
                                    R*Math.sin(d),
                                    R*Math.cos(l)*cos_d);
        }
}
```

Modul **Main.java**. Obsahuje vlastní interface a spouštění jednotlivých akcí.

```
import java.applet.*;
import java.awt.*;
import java.util.*;

class Sky extends Canvas {
        Image offImage;
        Graphics offGraphics = null;
        StarMap map = new StarMap();
        public int draw_lines = 1;
        public int draw_stars = 1;
        public int draw_horizon = 0;
        public int draw_equator = 0;
        public int draw_animation = 0;

        public boolean write_text_on_screen = false;
        public String errmsg;


        int Width = 400;
        int Height = 300;

        Dimension d;

        double t, longitude, latitude;
        GregorianDate gd;
        int azimut, elevation;

        /**
        Sky constructor. Here is drawn all what is shown on the applet screen.

        @param gd <b>GregorianDate</b>
        @param longitude observer's longitude
        @param latitude observer's latitude
        @param time observer's local time
        @param azimut observer's azimut
        @param elevation observer's elevation
```

```
@return new <b>Sky</b> object
*/
public Sky( GregorianDate gd, double longitude, double latitude,
            double time, int azimut, int elevation ) {
        this.gd = gd;
        this.longitude = longitude;
        this.latitude = latitude;
        this.t = time;
        this.elevation = elevation;
        this.azimut  = azimut;
        setBackground( new Color ( 0.0f, 0.0f, 0.5f)  );
}


/**
Sets local time.

@param t local time
*/
public void setTime( double t ) {
        this.t = t;
}

/**
Sets observer's azimut.

@param azimut observer's azimut
*/
public void setAzimut( int azimut ) {
        this.azimut = azimut;
}


/**
Sets observer's elevation.

@param elevation observer's elevation.
*/
public void setElevation( int elevation ) {
        this.elevation = elevation;
```

```
}

/**
Sets observer's latitude.

@param latitude observer's latitude
*/
public void setLatitude( double latitude ) {
        this.latitude = latitude;
}

/**
Sets observer's longitude.

@param longitude observer's longitude
*/
public void setLongitude( double longitude ) {
        this.longitude = longitude;
}

/**
Sets <b>GregorianDate</b>.

@param gd <b>GregorianDate</b>
*/
public void setGregorianDate( GregorianDate gd ) {
        this.gd = gd;
}

public void update (Graphics g) {
        paint(g);
}


/**
        Draws error messages on screen...
*/
private void drawError(Graphics g, String err, int width, int height) {
                int ypos = 0;
                int i=1;
                StringTokenizer st = new StringTokenizer( errmsg,
                                                    "\n", false );
                Font f = getFont() ;
                int LINE_SPACE = 10;
                Color c = g.getColor();
                String str;
                int font_height = getFontMetrics(f).getHeight();

                g.clearRect( 0, 0, Width, Height );


                while ( st.hasMoreTokens() ) {
                        str = st.nextToken();
                        g.setColor( Color.yellow );
                        g.drawString("Error #"+i++ +": "+str , 10,
                           ypos += ( font_height + LINE_SPACE ) );
                }

                g.setColor( c );
}

/**
        The main paint() function.
        It has two states: it draws stars and constellations or
        it writes error in console-style screen.
*/
public void paint( Graphics g ) {
        d = size();
        Width = d.width;
        Height = d.height;


        map.setWidth( Width );

        /* An error has occured, so draw flush errors on screen
          and return immidiately */
        if ( write_text_on_screen ) {
                drawError( g, errmsg, Width, Height );
                return;
        }
```

```
                /* Normal drawing process, no errors */
                map.setHeight( Height );

                if ( offGraphics == null ) {
                        offImage    = createImage( Width, Height );
                        offGraphics = offImage.getGraphics();
                }

                offGraphics.clearRect( 0, 0, Width, Height );
                map.computeMap( offGraphics, gd, t, longitude, latitude,
                                             azimut, elevation );
                if ( draw_lines == 1) {
                        map.drawLines( offGraphics );
                }
                if ( draw_stars == 1) {
                        map.drawStars( offGraphics );
                }
                if ( draw_horizon == 1 ) {
                        map.drawHorizon( offGraphics, azimut, elevation );
                }
                if ( draw_equator == 1 ) {
                        map.drawEquator( offGraphics, azimut, elevation,
                                                      latitude, longitude );
                }

                g.drawImage( offImage, 0, 0, Width, Height, null );

        }


}

/* NOTE:
 * local time ‘t’ is measured in hours, whenever it is needed it is converted into
 * longitude is also measured in hours from GMT, also automatically converted into
 * latitude is measured in degrees.
 */

public class Main extends Applet implements Runnable {
```

```
double Y = 2002;
double M = 3;
double D = 13;
double t = 6;
double longitude = 1;
double latitude = 50;

final int START_AZIMUT = 180;
final int START_ELEVATION = 0;
int azimut = START_AZIMUT;
int elevation = START_ELEVATION;

int appletWidth = 640;
int appletHeight = 480;

Thread anime = null;

GregorianDate gd = new GregorianDate (Y, M, D);

Sky sky;

Scrollbar horiz_slide = new Scrollbar( Scrollbar.HORIZONTAL,
                                    START_AZIMUT, 1, 0, 360);
Scrollbar vert_slide  = new Scrollbar( Scrollbar.VERTICAL,
                                    START_ELEVATION, 1, -90, 90);
Button    anime_control = new Button( "Start animation" );
TextField year_field = new TextField( Integer.toString((int)Y), 4);
TextField month_field = new TextField( Integer.toString((int)M), 4);
TextField day_field = new TextField( Integer.toString((int)D), 4);
TextField hour_field = new TextField( Integer.toString((int)t), 4);
TextField minute_field = new TextField( Double.toString(
                                              (t-(int)t) * 60 ), 4);
TextField longitude_field = new TextField( Double.toString(
                                              15 * longitude ), 4);
TextField latitude_field = new TextField( Double.toString( latitude ),
                                              4);
Button    draw_lines = new Button( "Hide Lines" );
Button    draw_stars = new Button( "Hide Stars" );
Button    draw_equator = new Button( "Draw Equator" );
Button    draw_horizon = new Button( "Draw Horizon" );

public void init() {

        Panel south_panel = new Panel();

        Panel west_panel = new Panel();


        /* Layout */
        setLayout( new BorderLayout() );
        add( "North", horiz_slide );
        add( "East",  vert_slide );
        add( "South", anime_control );

        south_panel.setLayout( new GridLayout( 7, 2 ) );
        south_panel.add( new Label("Year:") );
        south_panel.add( year_field );
        south_panel.add( new Label("Month:") );
        south_panel.add( month_field );
        south_panel.add( new Label("Day:") );
        south_panel.add( day_field );
        south_panel.add( new Label("Hour:") );
        south_panel.add( hour_field );
        south_panel.add( new Label("Minute:") );
        south_panel.add( minute_field );
        south_panel.add( new Label("Longitude:") );
        south_panel.add( longitude_field );
        south_panel.add( new Label("Latitude:") );
        south_panel.add( latitude_field );

        north_panel.setLayout( new GridLayout( 4, 1 ) );
        north_panel.add( draw_lines );
        north_panel.add( draw_stars );
        north_panel.add( draw_equator );
        north_panel.add( draw_horizon );

        west_panel.setLayout( new GridLayout ( 2, 1 ) );
        west_panel.add( north_panel );
        west_panel.add( south_panel );
        add( "West", west_panel );

        add( "Center" , sky = new Sky(gd, longitude, latitude,
                                      t, azimut, elevation ) );

        Trans3D.initTables();

}

private boolean ___convert_boolean( int val ) {
        if ( val == 0 ) return false; else return true;
}

/**
        Checks for the leap year.
        @return true when it is a leap year, otherwise false
*/
private boolean is_leap_year( int Y ) {
        if  ( ( ( (int)Y & 3 ) == 0 ) &&
            (  ___convert_boolean((int)Y % 100) ==
               ___convert_boolean((int)Y % 400) ) )  {
                /* leap year */
                return true;
        } else {
                return false;
        }
}

/**
        Returns days in given month with care about year.
        @return -1 if month is out of 1-12. Otherwise it
        returns 28-31 depending on the given year and month.
*/
private int days_per_month( int M, int Y ) {
        int days_in_feb;

        if ( M < 1 || M > 12 ) return -1;
```

```
                /* February at first ... */
                if ( M==2 ) {
                        /* leap year */
                        if ( is_leap_year( Y ) )
                                return 29;
                        else
                                return 28;
                }

                if ( M<8 ) {
                        return 30 + (M&1);
                } else {
                        return 31 - (M&1);
                }
        }

        /**
                Checks whether the input values are correct.

                @return  true if input is correct, false otherwise.
        */
        public boolean checkInput() {
                String err = new String(" ");
                double hh,mm;
                int days;
                boolean rc = true;



                hh = Double.valueOf( hour_field.getText() ).doubleValue() ;
                mm = Double.valueOf( minute_field.getText() ).doubleValue();


                /* month check */
                if ( M<1 || M>12 ) {
                        err += new String("Month could be between 1-12!\n");
                }

                if ( D<1 ) {
                        err += new String("Day number must be positive!\n");
        }

        days = days_per_month( (int) M, (int) Y );

        if ( days == -1 ) {
                /* some error informing about bad month could
                   be here in general, but in this code it is above.
                  */
        }
        else
        if ( D > days ) {
                err += new String("This month has only "+days+
                                  " days.\n");
        }

        if ( hh > 23 || hh < 0 ) {
                err += new String("Hours must be between 0-23!\n");
        }


        if ( mm >= 60 || mm < 0 ) {
                err += new String("Minutes must be between <0-60)!\n");
        }

        if ( err.length() > 1 ) {
                writeError( err );
                rc = false;
        }

        return rc;
}


public void start() {
        /* Restart the animation only if it was played before */
        if ( sky.draw_animation == 1 ) {
                start_anime();
        }
}
```

```
/**
This is start of the <b>Sky</b> rotating animation.
*/
public void start_anime() {
        if ( anime == null ) {
                anime = new Thread(this);
                anime.start();
        }
}

/**
Sometimes we need to stop the animation.
*/
public void stop_anime() {
        anime = null;
}

public void stop() {
        stop_anime();
}

public void run() {

        boolean change_day, change_month;

        while ( anime != null ) {
                change_month = change_day = false;

                t += 0.2;
                if ( t > 24 ) {
                        D++;
                        change_day = true;

                        if ( D > days_per_month( (int)M, (int) Y ) ) {
                                change_month = true;
                                D = 1;
                                M++;

                                if ( M > 12 ) {
                                        M = 1;
                                        Y++;
                                        year_field.setText(
                                             Double.toString( Y ) );
                                }
                        }

                        if ( change_day )
                          day_field.setText(
                            Integer.toString( (int) D ) );
                        if ( change_month )
                          month_field.setText(
                            Integer.toString( ( int ) M ) );

                        gd.setY( Y );
                        gd.setM( M );
                        gd.setD( D );
                        sky.setGregorianDate( gd );
                        t = 0;
                }

                sky.setTime( t );

                hour_field.setText( Integer.toString(
                                     (int) Math.floor( t ) ) );
                minute_field.setText( Double.toString(
                                      ( t - Math.floor( t ) ) * 60 ) );

                sky.repaint();

                try {
                        Thread.sleep (20);
                }

                catch ( InterruptedException e ) {};
        }
}

public void writeError( String errmsg ) {
        stop_anime();
        anime_control.setLabel( "Start animation" );
        if (sky.draw_animation == 1 ) sky.draw_animation ^= 1;
        sky.write_text_on_screen = true;
        sky.errmsg = errmsg;
        sky.repaint();
}


/**
Button and Textfiled are handled here.
*/
public boolean action ( Event e, Object arg ) {
        int month;

        /* Draw Stars button */
        if ( e.target == draw_stars ) {
                sky.draw_stars ^= 1;
                if ( sky.draw_stars == 1 ) {
                        draw_stars.setLabel( "Hide stars" );
                } else {
                        draw_stars.setLabel( "Draw stars" );
                }
                /* If animation is not running, we must repaint
                   the window manually */
                if ( anime == null ) {
                        sky.repaint();
                }

                return true;
        }

        /* Draw Lines button */
        if ( e.target == draw_lines ) {
                sky.draw_lines ^= 1;
                if ( sky.draw_lines == 1 ) {
                        draw_lines.setLabel( "Hide lines" );
                } else {
                        draw_lines.setLabel( "Draw lines" );
                }

                /* If animation is not running, we must repaint
```

```
           the window manually */
        if ( anime == null ) {
                sky.repaint();
        }

        return true;
}

/* Draw Horizon button */
if ( e.target == draw_horizon ) {
        sky.draw_horizon ^= 1;
        if ( sky.draw_horizon == 1 ) {
                draw_horizon.setLabel( "Hide horizon" );
        } else {
                draw_horizon.setLabel( "Draw horizon" );
        }

        /* If animation is not running, we must repaint
           the window manually */
        if ( anime == null ) {
                sky.repaint();
        }

        return true;
}

/* Draw Equator button */
if ( e.target == draw_equator ) {
        sky.draw_equator ^= 1;
        if ( sky.draw_equator == 1 ) {
                draw_equator.setLabel( "Hide equator" );
        } else {
                draw_equator.setLabel( "Draw equator" );
        }

        /* If animation is not running, we must repaint
           the window manually */
        if ( anime == null ) {
                sky.repaint();
        }
        return true;
}

/* Anime control button */
if ( e.target == anime_control &&
     sky.write_text_on_screen == false) {
        sky.draw_animation ^= 1;
        if ( sky.draw_animation == 1 ) {
                anime_control.setLabel( "Stop animation" );
                start_anime();
        } else {
                anime_control.setLabel( "Start animation" );
                stop_anime();
        }
        return true;
}


/* Year textfield */
if ( e.target == year_field ) {
        Y = Double.valueOf(
            year_field.getText() ).doubleValue();

        if ( ! checkInput() ) {
                return true;
        }

        sky.write_text_on_screen = false;
        gd.setY( Y );
        sky.setGregorianDate( gd );
        sky.repaint();
        return true;
}

/* Month textfield */
if ( e.target == month_field ) {
        M = Double.valueOf(
            month_field.getText() ).doubleValue();
        if ( ! checkInput() ) {
                return true;
        }

        sky.write_text_on_screen = false;
        gd.setM( M );
        sky.setGregorianDate( gd );
        sky.repaint();
        return true;
}

/* Day textfield */
if ( e.target == day_field ) {
        D = Double.valueOf( day_field.getText() ).doubleValue();

        if ( ! checkInput() ) {
                return true;
        }

        sky.write_text_on_screen = false;
        gd.setD( D );
        sky.setGregorianDate( gd );
        sky.repaint();
        return true;
}


/* Hour or Minute textfield */
if ( e.target == hour_field || e.target == minute_field ) {
        t = Double.valueOf(
               hour_field.getText() ).doubleValue() +
            Double.valueOf(
               minute_field.getText() ).doubleValue() / 60;

        if ( ! checkInput() ) {
                return true;
        }

        sky.write_text_on_screen = false;
        sky.setTime( t );
```

```
                        sky.repaint();
                        return true;
                }

                /* Longitude textfield */
                if ( e.target == longitude_field ) {
                        longitude = Double.valueOf(
                                longitude_field.getText() ).doubleValue() / 15;

                        if ( ! checkInput() ) {
                                return true;
                        }

                        sky.write_text_on_screen = false;
                        sky.setLongitude( longitude );
                        sky.repaint();
                        return true;
                }

                /* Latitude textfield */
                if ( e.target == latitude_field ) {
                        latitude = Double.valueOf(
                                   latitude_field.getText() ).doubleValue();

                        if ( ! checkInput() ) {
                                return true;
                        }

                        sky.write_text_on_screen = false;
                        sky.setLatitude( latitude );
                        sky.repaint();
                        return true;
                }

                return false;
        }


        /**
        Here we handle scrollbars events.
        */
        public boolean handleEvent ( Event e ) {
                /* Horizontal slide control */
                if ( e.target == horiz_slide ) {
                        azimut = horiz_slide.getValue();
                        sky.setAzimut( azimut );
                        sky.repaint();
                }

                if ( e.target == vert_slide ) {
                        elevation = vert_slide.getValue();
                        sky.setElevation( elevation );
                        sky.repaint();
                }

                return super.handleEvent(e);
        }

}
```