Forgalomtechnikai kód az ACTROS VTC 3000 forgalomirányító berendezésben

Budapest XVI. Csömöri út-János utca csomópont jelzésterveinek működése alapján

Polgár János

MSc hallgató

Tettamanti Tamás

PhD hallgató

Segédlet a Közúti irányító és kommunikációs rendszerek II. c. tárgyhoz

Budapest, 2010. március

Tartalomjegyzék

Hogyan használjuk a segédletet?

Olvasás közben célszerű a megfelelő programrészeket a számítógépen is követni, így könnyebb a megértés. Az összes a gyártó által előre definiált programelem pontos felépítése, szintaktikája, stb. megtalálható az ACTROS Java helpjében [2]. A gyakran használt német kifejezések magyar jelentése, a forgalomfüggő program jelzésterve és a csomópont helyszínrajza a dokumentum végén található.

1. A jelzéstervek keretrendszere

1.1. Osztályok, objektumok

Az ACTROS berendezés Java nyelven programozható. A Java objektumorientált programozási nyelv, melyet a gép programozásakor nagyon jól ki lehet használni. A programnyelv alapja az objektum, melyeket osztályokba lehet sorolni.
Az ACTROS forgalomtechnikában számos osztályt (class) találunk. Osztályt alkotnak a detektorok, a jelzőcsoportok, a fixidejű programok, a forgalomfüggő programok, az IO-kártyák és még sok minden más. Ezekbe az osztályokba soroljuk az objektumokat (object). Fontos, hogy az egy osztályba tartozó objektumok azonos tulajdonságokkal rendelkeznek, ugyanúgy kell őket létrehozni egy ún. konstruktor segítségével. Az ACTROS esetében minden osztálynak számos előre megírt függvénye és eljárása (összefoglaló néven metódusa) van, melyek minden az adott osztályba tartozó objektum tulajdonságainak lekérdezésére vagy megváltoztatására alkalmasak.
Egy osztálynak lehetnek alosztályai is (subclass). Például a detektor osztályból származtatható a DetektorIntern osztály. Az alosztályok megöröklik az osztályok tulajdonságait és metódusait. Az általunk létrehozott .java fájlok is alosztályok, és ezek a kódjuk elején extends szó után álló osztályba tartoznak.

1.2. A VT-Package felépítése

A VT-Package tartalmazza a csomópont jelzésterveit és az ahhoz kapcsolódó kiegészítő programrészeket. Tartalma jobban meghatározható úgy, hogy minden olyan elem, ami nem a hardverek működtetéséhez és ellenőrzéséhez szükséges, ebben foglal helyet.
Elemei a következő osztályok:

- Változó osztály: Var.java
- Inicializáló osztály: Init.java
- Programok, rendszerprogramok osztályai: SVprog.java, FixProg1.java, stb.
- (Fázisok és fázisátmenetek osztályai)
- (Ellenőrző osztályok)

A zárójelbe tett részeket jelen program nem tartalmazza. A csomópont programjának ismertetésekor is követjük ezt a sorrendet. Itt kell megjegyeznünk, hogy az ACTROS berendezés programozásakor két alapvető megközelítést használnak a programozók. Németországban definiálnak fázisokat és fázisátmeneteket (a közbensőidők alatt futnak), Magyarországon viszont nem. Előbbi esetben a forgalomfüggő logikák egy része elhelyezhető a fázisokat tartalmazó programrészekben, míg utóbbinál szinte minden ilyen utasítást a jelzési program .java fájlja tartalmazza.

A következőkben ismertetett osztályok és alosztályok mindegyike importálással kezdődik. Az importálás során felkészítjük programunkat már kész Package-ek (és az azokban található osztályok és metódusaik) felhasználására.

Pl.:
```
import sg.Sg;
import vt.BlinkProg;
import vt.TeilKnoten;
```

Ez azt jelenti, hogy a következő programrészben a már kész sg és vt nevű Package-ek Sg, BlinkProg és TeilKnoten osztályát fogjuk felhasználni. Minden programrészben csak az ott használt Package-eket kell importálni. A továbbiakban az egyes programrészeknél nem térünk ki az importálásra.

Az ACTROS forgalomtechnikához tartozik egy vtbib.jar állomány is, mely a felhasznált osztályok és metódusok hardverközeli programkódjait is tartalmazza.

1.3. A berendezés üzeme

Bekapcsoláskor elsőként a Var.java, majd az Init.java (őket lásd alább) fut le egyszer, azaz megtörténik az inicializálás. Ezután a berendezés sárga villogóba (SvProg) kapcsol, majd következik a bekapcsolási program (BeProg). A bekapcsoló programból jut el az időtervben megadott fixidejű vagy forgalomfüggő program futtatásához. A berendezés leállításakor az éppen futó jelzéstervből a kikapcsoló programra (KiProg), majd sötétre kapcsol. A folyamat az 1. ábrán is követhető.

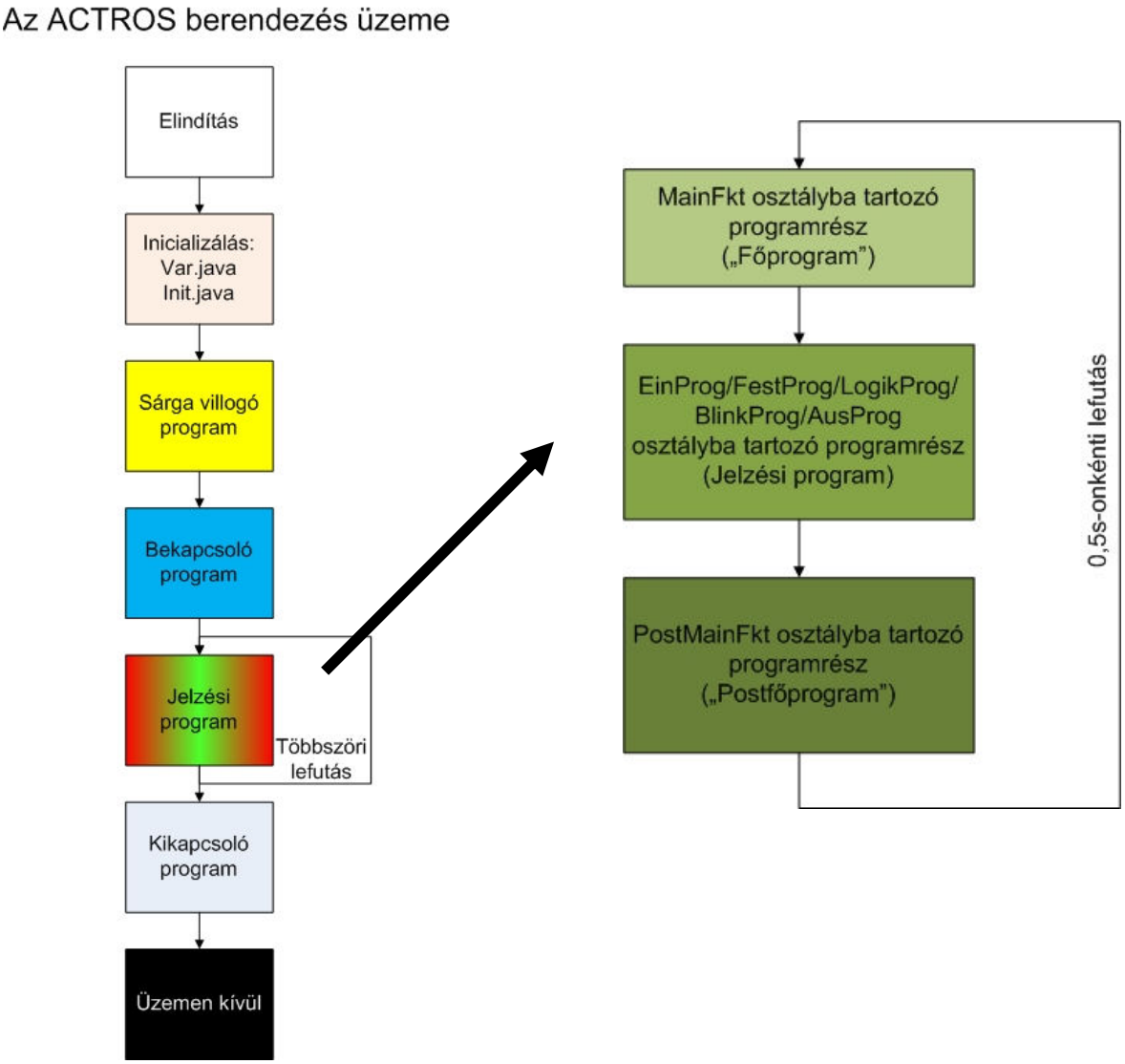
Egy jelzési program futásakor (ez a bekapcsoló, fix, forgalomfüggő, kikapcsoló, sárga villogó programokra egyaránt vonatkozik) meghatározott sorrendben 0,5s-os időközökkel több Java fájl is lefut (2. ábra):

- MainFkt osztályba tartozó programrész („Főprogram”)
- EinProg, vagy FestProg, vagy LogikProg, vagy BlinkProg, vagy AusProg osztályba tartozó programrész (Jelzési program)
- PostMainFkt osztályba tartozó programrész („Postfőprogram”).

E három lépés mindig ugyanebben a sorrendben fut le, de ha ez nem sikerül egy kiválasztott ciklusidőn belül, akkor ciklusidő felülírás következik be, és a berendezés első lekapcsolási fokozatba kerül.

Az „Főprogramban” van a minimális zöldidők megadása, ebben a részben lehet forgalomtechnikai lekérdezéseket végezni a jelzéstervben való későbbi felhasználáshoz. Ha nincs forgalomtechnikai jelentősége, elhagyható, a rendszer nem ragaszkodik a meglétéhez.
A Jelzési programban a jelzésterv adatai, a fázisok kapcsolása, esetleges forgalomfüggő logikák megvalósítása, további lekérdezések lehetnek.
A „Postfőprogramban” található a megfelelő nyomógombok és detektorok alapállapotba helyezése, ha az megtehető. Ha nincs forgalomtechnikai jelentősége, akkor elhagyható.
Jelen esetben a MainFkt osztályba egyetlen programrész sem tartozik, a második lépésben futó részek megvannak hasonló (magyar) névvel, a harmadik lépésben futó rész viseli az AltalanosResz.java nevet.

Bekapcsoláskor a berendezés azzal a programmal indul a sárga villogó program után, mely a kikapcsolásakor futott.

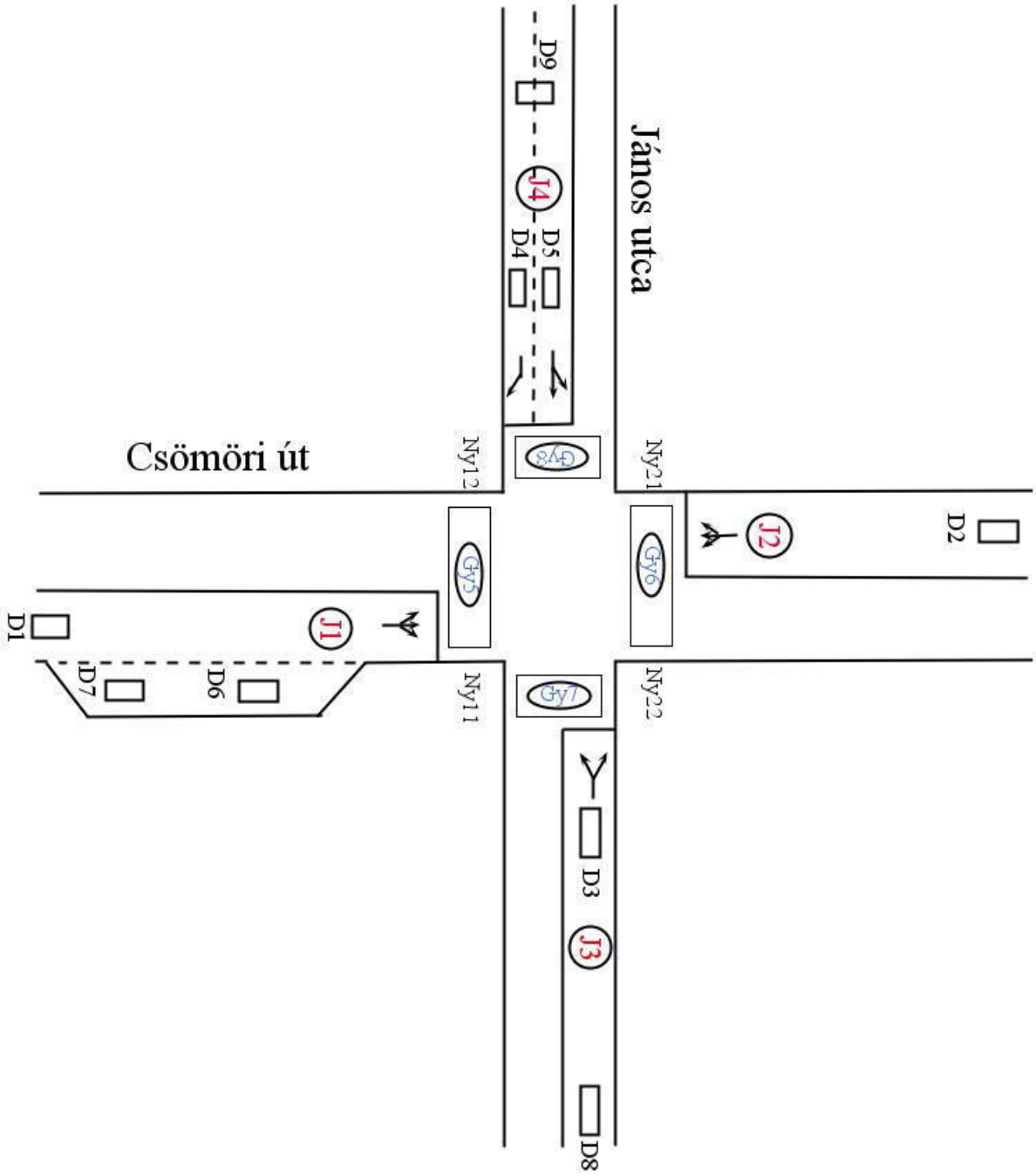


1. ábra A programrészek futási sorrendje

2. ábra Jelzési program futása

2. A jelzési program elemei

A könnyebb érthetőség kedvéért a csomópont egyszerűsített felépítését is megrajzoltuk a 3. ábrán.



3. ábra A csomópont sematikus ábrája

2.1. Var.java

ACTROS berendezés programozásakor szokásos, hogy a globális változókat egy Var vagy V nevű osztályban definiálják. Globális voltuk miatt az összes programrész módosíthatja tartalmukat. Emellett a csomópont irányítására használt objektumok deklarálása is itt történik meg.

Az objektumok deklarálása a `public static osztálynév objektumnév;` sorral, a változóké a `public static változótípus változónév;` sorral történik. A public a globális változóra, a static pedig a változó statikusságára utal. A dinamikus változó mérete a program futása közben változtatható, de ilyenre az ACTROS-nak nincs szüksége. A lokális változók a private szóval deklarálhatók.

Magát a forgalomirányító berendezést is definiálni kell önmagának. (Anlage)

```
public static Anlage anlage;
```

A gép 3 részcsomópontot is tud egyidőben kezelni, ezért definiálni kell ezeket is. (tk1, tk2, tk3) Jelen esetben csak egy részcsomópontot vezérlünk.

```
public static TeilKnoten tk1;
```

A következőkben a jelzőcsoportokat hozzuk létre (Sg). Mindegyik a jelzőcsoport osztályba fog tartozni, de külön-külön objektumok lesznek, így minden egyes jelzőcsoportot (egy irány egy jelzőcsoporttal van irányítva) külön-külön definiálni kell. Jelen esetben a járműjelzőket j11, j21...; a gyalogos jelzőket gy51, …; az egy fénypontú villogó jelzőket sv91, …; a gyalogos nyomógombok visszajelzőjét ViNy1,…; jelöli. Utóbbiak fiktív jelzőcsoportok, csak akkor villognak, ha a hozzájuk tartozó gyalogos nyomógombot megnyomták. A hb1 és hb2 jelű jelzőcsoportok sem valódi jelzőcsoportok, hanem a vakok és gyengénlátók számára a csomópontban elhelyezett hangosbeszélők tápfeszültségének vezérlésére szolgálnak. Az ismétlőjelzők (Wiederholer) hasonló logikával vannak elnevezve.

```
…
public static Sg j41;
public static Sg gy51;
…
public static Sg viNy1;
…
public static Wiederholer j12;
…
```

Detektor objektumokat d1…d9 névvel illet a program. A gyalogos nyomógombok is a detektor osztályba tartoznak, de megkülönböztethetőségük kedvéért ők ny1 és ny2 névre hallgatnak. A nyomógombokra a főirányt keresztező gyalogátkelőhelyeken való átjutáshoz lehet szükség..

```
public static Detektor d1;
…
public static Detektor ny1;

…
```

Az egyes jelzéstervek, valamint a futáshoz szükséges egyéb utasításokat tartalmazó programrészek osztálya már szabadon megválasztható, mely utal az objektum nevére is, ezek

azonban alosztályok lesznek. (az eddig definiált objektumok osztálya kötött volt: Sg, Detektor, stb.) Pl. public static FixProg1 prog1;. Ez a sor egy prog1 nevű FixProg1 alosztályba tartozó objektumot hoz létre, a public static FixProg2 prog2 sor pedig egy prog2 nevű, FixProg2 alosztályba tartozót. Ilyen esetekben az alosztály lehet azonos (pl. public static FixProg prog1; public static FixProg prog2;), ez a programozótól függ, gyakorlati jelentősége nincs. Természetesen a különféle programok nem ugyanabba az osztályba tartoznak. Definiálunk még bekapcsoló programot (beprog), kikapcsoló programot (kiprog), sárga villogó programot (svprog). Az egyénileg létrehozott alosztálynevek még az ACTROS számára nem jelentenek semmit. Az egyes jelzéstervek java fájljában történik meg az alosztályok osztályhoz kötése az extends szóval. Az osztályok az „A berendezés üzeme" című részben tárgyaltak lehetnek. Az általunk létrehozott alosztályok megöröklik az osztályok metódusait, így ezeket nem kell újra megírni.

```
public static BeProg beprog;
public static KiProg kiprog;
public static SvProg svprog;
public static FixProg1 prog1;
public static FixProg2 prog2;
public static FixProg3 prog3;
public static ForgfProg prog4;
```

Az alt nevű Általános rész a nyomógombok és detektorok kezeléséhez szükséges, említettük, hogy ez az alosztály a PostMainFkt osztályba tartozik.

```
public static AltalanosResz alt;
```

Ezután a fénypontok típusainak megadása következik: piros, sarga, jmzold, stb. A zold jelen esetben a hangosbeszélőt takarja.

```
public static LmpTyp piros;
public static LmpTyp sarga;
public static LmpTyp sargavill;
…
```

A jelzőcsoportok típusok szerint csoportosíthatók: járműjelzők, gyalogosjelzők, visszajelzők, stb. csoportja, melyeket szintén definiálni kell. (Szerencsésebb lett volna, ha a kódban mindjárt a jelzőcsoportok definiálása után következének a típusok definíciói.)

```
public static SgTyp jm;
public static SgTyp gy;
public static SgTyp sv;
…
```

Az ACTROS-nak „meg kell mondani" azt is, hogy az általunk létrehozott fénypontok hogyan világítsanak. Pl. a piros folyamatosan ég, a sárga villogó villog.

```
piros = new LmpEinfachUgaUge();
sarga = new LmpBlinkUge();
…
```

Ezután azt adjuk meg, hogy az előzőleg definiált jelzőcsoporttípusok milyen fénypontok felhasználásával tudnak működni. Minden egyes fénypont-kombináció más osztályba tartozik, így különféle nevük van. Pl. a RoGeGr osztályba tartozó jelzőcsoport állapota lehet piros, piros-sárga, zöld, sárga, sárga villogó, sötét. Mindegyik jelzőcsoporttípust a név.init eljárással lehet létrehozni, de a paraméterlista osztályonként változó. Például a RoGeGr.init eljárás a következő bemenő paraméterekkel bír: RoGeGr.init(fénypontok típusai, piros-sárga hossza, sárga hossza, sötét-sárga hossza)

```
jm = RoGeGn.init(piros, sarga, jmzold, 2, 3, 3);
gy = RoGnGnBl.init(piros, gyzold, 5);
…
```

Összegezve megállapítható, hogy a Var.java programrészben a létrehozott objektumokkal még nem történt semmi a deklarálásukon kívül.

A Java kód többi részében a Var-ban létrehozott objektumokra a Var.objektumnév összetétellel lehet hivatkozni. Nem elég csak az objektumnév megadása, hiszen egy másik programrészben nincsenek definiálva ezek az objektumok.

2.2. Init.java

Ebben a részben kell megadni az előzőleg létrehozott objektumok tulajdonságait. Ezt a folyamatot nevezzük inicializálásnak, melyet az Init.java `main(…)` metódusa végez el. Meghívja a jelzőcsoportok, a közbensőidő mátrix, a detektorok, a jelzési programok, az óra és a hardver inicializáló eljárását. [initialisiere..(…)]. A meghívási sorrend fix, amelyet a rendszer le is tesztel induláskor. A sorrendtől való eltérés az inicializálás megszakításához vezet. Hangsúlyoznunk kell, hogy a `main(…)` eljárás az Init.java *végén* található, mivel az általa meghívott eljárásoknak meg kell előzniük a meghívás helyét. A segédletben a könnyebb érthetőség kedvéért vettük előre ezt a fontos metódust.

```
public static void main(String args[])
…
  Init init1 = new Init(); //init objektum létrehozása
  Actros.registerInitialisierung(init1); //init objektum meghívása
  init1.initialisiereSg(); // a megírt metódusok meghívása
  init1.initialisiereZwz();
  init1.initialisiereDet();
  init1.initialisiereProgs();
  VtVar.endInit();
```

```
init1.initialisiereUhr();
init1.initialisiereHW();
```
…

Ezekben az eljárásokban az egyes objektumok konstruktorát kell meghívni. Ennek pontos paraméterlistája objektumonként változik. A konstruktor paraméterei a kódban kommentként vannak részletezve. Pl. a j11 járműjelző adatait a következőképpen adjuk meg: pl. Var.j11 = new Sg(Var.tk1, "11m", Var.jm, 5, 2, 1, 1). Szinte minden esetben ugyanilyen struktúrájú a meghívás: az objektum = new osztálynév (konstruktor paraméterek). Természetesen a fiktív jelzőcsoportokat is inicializálni kell. (A hangosbeszélővel a továbbiakban nem foglalkozunk, mert annak vezérlése hiányzik ebből a programból lévén, hogy a tanszéki géphez nincs ilyen tartozék csatlakoztatva.)

```
Var.j11 = new Sg(Var.tk1, "11m", Var.jm, 5, 2, 1, 1);
//részcsomópont, név, jcs típus, min szabad, min tilos, irány, szám
…
```

Az ismétlőjelzők az inicializálásnál a „normál” jelzőkből származtathatók, nem kell az adatokat külön megadni.

```
Var.j12 = new Wiederholer("12M");
Var.j11.initWh(Var.j12);
…
```

A közbensőidő mátrixot (ZwzMatrix) a Var.java-ban nem szükséges deklarálni, azonban itt kell, a benne lévő értékekkel együtt. A deklaráció a kód végén található. Elegendő private változóként megadni, mivel más programrészek nem kell, hogy hozzáférjenek ennek adataihoz. (A közbensőidők teljesülését a berendezés maga vizsgálja, nem a többi programrész.) Minden egyes a mátrixban lévő számot külön-külön kell megtanítani az ACTROS-nak a következő eljárás segítségével: zwz.setzeZwz(Var.j11, Var.j31, 6, 6) Vigyázat! Az eljárás három bemenő paraméterrel is meghívható, ekkor a két irány között a közbensőidő oda-vissza ugyanannyi.

```
zwz = new ZwzMatrix(); //mátrix inicializálása
zwz.setzeZwz(Var.j11, Var.j31, 6, 6);
zwz.setzeZwz(Var.j11, Var.j41, 6, 6);
//kihaladó irány, behaladó irány, közb. i. ki-be, közb. i. be-ki között
…
```

A detektorok inicializálására részletesen nem térünk ki.

```
Var.d3 = new Detektor(Var.tk1, "D3", 0, 0, 0, 3, Var.j31);
//részcsomópont, név, tartós foglaltság ideje, max. küszöb túllépés
//ciklusonként, nem foglaltság ideje, jelzőcsoport
…
```

A jelzési programoknál a new szó után azt az alosztályt kell megadni, amit a Var.java-ban előzőleg deklaráltunk. Figyelem! A konstruktor meghívható jelzőcsoport-hozzárendeléssel is, ekkor a programszám előtt egy tömbben kell megadni a vezérelt jelzőcsoportokat. A zöldhullámpont A és B tulajdonképpen a stop pontot jelenti. A forgalomfüggő program inicializálása ugyanígy történik.

```
Var.prog1 = new FixProg1(Var.tk1, "P1", 1, 60, 10, 11, 0, 0);
//részcsomópont, név, prgszám, periódusidő, zöldhullámpont A,
//zöldhullámpont B, várakozási idő, átállási idő
…
```

A bekapcsoló, kikapcsoló, sárga villogó program konstruktora egyszerűbb, itt kevesebb adatot kell megadni. (Ebben az esetben is van lehetőség jelzőcsoport-hozzárendelésre.)

```
Var.beprog = new BeProg(Var.tk1, "Be", 30, 19);
//részcsomópont, név, programszám, periódusidő
…
```

Napi terv (TagesPlan) megadása: a konstruktorban a nevet és a 0:00-kor éppen futtatandó program nevét kell megadni. Ezután az initProgWunsch(kezdő óra, perc, programnév) eljárással lehet megadni az adott napon futtatandó programok sorrendjét perc pontossággal. Jelen esetben külön definiálva van munkanapi, valamint szabad- és munkaszüneti napi terv.

```
TagesPlan hp = new TagesPlan("H-P", Var.svprog); //inicializálás
hp.initProgWunsch(6, 0, Var.prog1);
//kezdés órája, kezdés perce, programnév
hp.initProgWunsch(6, 30, Var.prog2);
…
```

A napi tervekből heti tervet kell készíteni (WochenPlan), melynek létrehozásakor a név után a hét hét napjára vonatkozó napi terv nevét kell felsorolni.

```
new WochenPlan("Fix", hp, hp, hp, hp, hp, hv, hv);
//név, napi terv, napi terv…
```

A hardverelemek inicializálása több részből áll össze.

```
Var.anlage = new Anlage(Var.tk1);
```

Ezzel a metódussal inicializálódik, ill. indul el a berendezés. A berendezés inicializálásánál minden rendelkezésre álló részcsomópontot meg kell adni a nevével. Ezek után már nem lehet további részcsomópontot hozzáadni. A részcsomópontok a rendszerből automatikusan indulnak, ez csak annyit jelent, hogy a rendszer ismeri a csomópontot és tulajdonságait. A berendezés vektorok létrehozásával a hardver definiálás is elkezdődik.

Ciklusidő: A ciklusidő beállítása az egész berendezésre vonatkozik és csak a berendezés vektorok létrehozása után történhet. Az alapérték az 1 másodperc, de általában fél másodpercre szokták állítani a következő eljárással:

```
Var.anlage.setZyklZeitHalbeSek();
```

Hardver szerinti fénypont hozzárendelés: A rendszer tudomására kell hozni a lámpateljesítményt, hogy a megfelelő ellenőrzést meg tudja valósítani. Ez a kívánt típus objektumának létrehozásával történik. A hozzátartozó áram és feszültség értékek a rendszerben vannak tárolva. A programunkban két esetet is feltételeznek: 40V ill. 230V tápfeszültségről működtetett fénypontok is lehetnek a jelzőfejekben (természetesen nem egyidejűleg keverve).

```
lmpRotLed40V = new LmpOCITRotLed40V();
…
```

Kapcsolókártyák és az IO-kártya inicializálása:

```
SchalterKarte sk24_1 = new SchalterKarte();
SchalterKarte sk24_2 = new SchalterKarte();
IoKarte io1 = new IoKarte();
```

Kapcsolócsatornák kijelölése: itt történik a jelzőcsoportok egyes lámpáinak a kapcsolókártya csatornáihoz rendelése. A mérőcsatorna értéke 1, ha a kapcsolócsatornát ellenőrizni akarjuk, ellenkező esetben -1.

```
new SchaltKanal(Var.j11, Var.piros, lmpRotLed40V, 1, sk24_1, 1, -1);
//jelzőcsoport, lámpatípus, HW szerinti lámpatípus, fő/mellékhuzal,
kapcsolókártya, csatornaszám, mérőcsatorna
…
```

Virtuális kapcsok inicializálása: egy jelzőcsoportot virtuális kapoccsal kell ellátni, ha egy lámpának nem kell saját kimeneti csatorna. Pl.: a fő- és ismétlő-, zöld és a sárga lámpák egy csatornán keresztül vezéreltek. Az ismétlő piros lámpacsoport nem mehet együtt a piros főcsoporttal.

```
new VirtKlemme(Var.j12, Var.sarga);
//jelzőcsoport, lámpatípus
…
```

I/O csatornák: ezzel az objektummal történik a detektorok és a kimenetek hozzárendelése az egyes csatornákhoz. A berendezésben a csatornák elosztása a következő:

- az első 8 csatornánál (1-8) optikai csatlakozós bemenet van (5-24 V, egyenfeszültség).
- a második 8 csatorna (9-16) szenzoros bemenetű (24-230 V, váltakozó vagy egyenfeszültség).
- a detektorok mind a 16 bemenetet használhatják.

- az utolsó 8 csatorna (17-24) kimenetként szolgál.

```
new IoKanal(Var.d1, io1, 1);
//detektor objektum, IO kártya referenciája, csatornaszám
…
```

Az eddig felsorolt eljárások eljárás voltukból fakadóan nem futottak le, hiszen semmi nem hívta meg őket. Ezt a fejezet elején tárgyalt `main(…)` metódus teszi meg.

Az egyes eljárások végén található system.out.println(„szöveg”); parancsok a soron porton történő ellenőrzéshez szükségesek. Egy esetleges programhiba esetén ezek segítségével tájolható be az a pont, ahol a vezérlés elakadt.

2.3. FixProg1.java

Az első sor jelzi azt, hogy a FixProg1 alosztály a FestProg osztályhoz tartozik.

```
public class FixProg1 extends FestProg
…
```

Ezután következik az Init.java-ban lévő konstruktor bemenő paramétereinek beolvasása. Azért van két beolvasás, mert kétféle konstruktorral is meghívható az objektum. (lásd feljebb)

```
public FixProg1(TeilKnoten kn, String name, int num, int umlZeit, int
gwpa, int gwpb, int warteZeit, int versatzZeit)
// Részcsomópont, programnév, programszám, átfutási idő, gwpa, gwpb,
//várakozási idő, átállási idő
    { super(kn, name, num, umlZeit, gwpa, gwpb, warteZeit,
versatzZeit);
    }
…
```

A programrész egyetlen eljárása a programmFunktion(). Ebben az egyes jelzőcsoportok kapcsolását kell csak megadni, hiszen a nem forgalomfüggő programok esetében a berendezésnek más dolga nincs, mint a jelzőcsoportok megfelelő sorrendben történő kapcsolása. A K.BEKI (jelzőcsoport, kezdet, vég) eljárás segítéségével lehet ezt megtenni. (A K.java rész ismertetését lásd a 19. oldalon) Az argumentumokban a szabad jelzés kezdetét és végét kell megadni úgy, hogy a piros-sárga idejét is bele kell számítani. Pl. a K.BEKI(Var.j11, 1, 28); sor azt jelenti, hogy a j11 jelzőcsoport az 1. másodpercben váltson piros-sárgára, a 3.-ban zöldre, és a 28.-ban sárgára. Emlékeztetőül: a piros-sárga és a sárga időtartamát nem magától találja ki, hanem a jelzőcsoportok inicializálásánál adtuk meg. Minden forgalomtól nem függő program ugyanúgy működik.

```
public void programmFunktion()
 {K.BEKI(Var.j11, 1, 28);
 //jelzőcsoport, kezdet, vég
```

```
K.BEKI(Var.j21, 1, 28);
K.BEKI(Var.j31, 33, 41);
…
```

2.4. Beprog.java, KiProg.java, SVProg.java

Alapvető felépítésük megegyezik a fix programéval, azzal a különbséggel, hogy ők az EinProg, AusProg és BlinkProg osztályhoz tartoznak.

```
public class BeProg extends EinProg
…
```

A konstruktor bemenő paramétereinek beolvasása (két konstruktor van a jelzőcsoport-hozzárendelés opcionális volta miatt):

```
public BeProg(TeilKnoten arg0, String arg1, int arg2, int arg3)
            //részcsomópont, név, szám, periódusidő
    {
        super(arg0, arg1, arg2, arg3);
    }
```

A programmFunktion() eljárásban nem a K.BEKI eljárást, hanem a jelzőcsoport setSg(jelzési állapot, idő) nevű eljárását használjuk. A jelzési állapotot is számmal jelezzük, például a piros kódja a 10, a piros-sárgáé a 30, a zöldé a 4, a sárga villogóé 2, a sötété 1.

```
public void programmFunktion()
    {   Var.j11.setSg(4, 14);
        //jelzési állapot kódja, bekapcsolási idő
        Var.j21.setSg(4, 14);
        Var.j31.setSg(10, 5);
        Var.j41.setSg(10, 5);
        …}
```

Azért nem a K.BEKI eljárást használjuk, mert itt nem csak a szabad jelzéseket kell beállítani, hanem minden jelzőcsoportnak meg kell adni valamilyen (akár sötét) állapotot, és az előbb említett eljárás nem ezt teszi. A hangosbeszélők tápfeszültségét csak a be- és kikapcsoláskor kell változtatni, azért csak ezekben a programrészekben van jelen ez a „jelzőcsoport”. Ha hiba lép fel, és a berendezés átkapcsolódik sárga villogó programba, akkor a hangosbeszélőknek tilos szabad utat jelezni még saját hibájuk esetén is, így kikapcsoljuk őket. A programok kódja nagyon hasonlít a fix programéhoz, ezért ezt itt nem közöljük.

2.5. ForgfProg.java

Ez egy forgalomtól függő program, mely zöld nyújtásokkal és ún. ugrásokkal operál. Ugrásnál a jelzési program egy részét a gép kihagyja. Pl. a 20. másodpercből a 34. másodpercre ugrik, kihagyva a közöttük lévő jelzésképeket. A jelzéstervből megállapítható,

hogy a gyalogátkelőhelyek minden periódusban kapnak szabad jelzést. A telepített gyalogos nyomógombok ennek ellenére azért szükségesek, hogy a főirányt (Csömöri út) keresztezni kívánó gyalogosok megjelenését detektálni tudjuk. Az érzékelés szükségességének kulcsa az „Az első nyújtási lehetőség” című alfejezetben olvasható. Az ACTROS számára az extends LogikProg összetétel mutatja meg, hogy forgalomfüggő program fut benne. Mivel konstruktora a fix programéval azonos, az első néhány sor már ismerős lesz más részekből.

Megjegyzés: sok programnyelvvel ellentétben a Java megengedi azt, hogy a változókat azután deklaráljuk, mint ahogyan használjuk őket. Az ebben a részben használt lokális (private) változók deklarációja a kód végén található, de már a programmFunktion() eljáráson kívül. Ettől függetlenül az áttekinthetőség és a megszokás miatt célszerű a lokális változók deklarációját az alprogram elejére írni.

A programmFunktion() eljárásban először a forgalomfüggő logika feltételeit találjuk, azonban ennek ismertetése előtt tekintsük át a következőkben használt metódusokat.

A detektor osztály egy függvénye a getAnforderung(), mely a detektor foglaltságát adja vissza: foglalt-1, nem foglalt-0. A visszakapott eredmény közvetlenül is összehasonlítható más értékekkel, nem kell egy változóban eltárolni, és úgy összehasonlítani.
Pl. `x=Var.d4.getAnforderung() || Var.d5.getAnforderung()…`

Nem pedig:
```
a1=Var.d4.getAnforderung();
a2=Var.d5.getAnforderung();
x=a1||a2;
```

A detektor osztály egy másik függvénye a dynLuecke(időköz). A függvény segítségével lekérdezhető egy detektor állapota a paraméterként megadott időköz letelte után. Az időközt ms-ban kell megadni. Visszatérési értéke szintén boolean típusú. A függvény visszaadott értéke 1, ha a detektor a megadott időtartamon belül bármikor foglalt volt, ellenkező esetben 0.

A detektor osztály egy harmadik függvénye a getBelSek(), mely a detektor foglaltságának időtartamát adja meg. Pl. az if (Var.d7.getBelSek() > 7) then utasítás1; sor azt jelenti, hogyha a 7-es detektor több, mint 7 másodperce foglalt, akkor hajtsa végre az utasítás1-et.

A részcsomópont, mint osztály függvénye a getProgSek(), mellyel a részcsomópontban futó program aktuális ciklusidejét kérhetjük le.

A setProgSek(idő) eljárás ennek a fordítottja, segítségével a programozó átállíthatja a ciklusidőt. Az argumentumként megadott új ciklusidő értéket másodpercben kell megadni. A ciklusidő változtatásánál ügyelni kell arra, hogy szükség esetén a jelzőcsoportok új állapotait még az átállítás előtt meg kell adni. A zöld nyújtást is ezen eljárással a legkönnyebb megoldani, de ilyenkor a jelzőcsoportok átállítása nem szükséges. Zöld nyújtásnál a

ciklusidőbe való beavatkozással a régi forgalomirányító berendezések stop-pontbeli „lefagyasztását” valósíthatjuk meg. Az ACTROS-t hagyományos értelemben „lefagyasztani” csak kézi vezérléssel lehet, a zöld nyújtáshoz a ciklusidő változtatása a jó megoldás.

A getProgAnf().getProg().getNummer() függvény az aktuálisan futó program kódját kérdezi le. Ellenőrzésre szokás használni bonyolultabb logikák megvalósításakor.

Vizsgáljuk meg, milyen összefüggéseket tartalmaz a programmFunktion eljárás!

2.5.1. Az első nyújtási lehetőség

```
public void programmFunktion()
    {
  x = Var.d4.getAnforderung() || Var.d5.getAnforderung()  ||
      Var.ny1.getAnforderung() || Var.ny2.getAnforderung();
  if(Var.tk1.getProgSek() == 16 &&
      Var.tk1.getProgAnf().getProg().getNummer() == 4)
            if(!x && !Var.d3.getAnforderung())
                Var.tk1.setProgSek(16);
            else
            if(!Var.d1.dynLuecke(4000) || !Var.d2.dynLuecke(4000))
            {   if(++nyujtas12 < 70)
                    Var.tk1.setProgSek(16);
                else
                    nyujtas12 = 0;
            } else
            { nyujtas12 = 0;
            }
…
```

Az x nevű változó a (d4 vagy d5 vagy ny1 vagy ny2) értéket veszi fel. (Emlékeztetőül: a d jelzésűek a járműérzékelő detektorok, az ny jelzésűek a gyalogos nyomógombok.)
Ha (a ciklusidő 16 s és a 4-es számú program fut),
akkor ha (x hamis és d3 hamis),
akkor a ciklusidő legyen 16 s
különben
ha (a d1 vagy a d2 időablaka nem 4 s)
akkor ha (a növelt értékű nyujtas12 változó értéke <70)
akkor a ciklusidő legyen 16 s
különben a nyujtas12 legyen 0
különben nyujtas12 legyen 0

Magyarázat: a kódrészlet célja a következő: ha a 16. másodpercben x hamis és a d3 sem foglalt, akkor várakozunk, mindaddig, amíg x igaz vagy d3 foglalt lesz. Ha az előbbi feltétel teljesül, vizsgáljuk meg a d1 vagy a d2 detektor foglaltságát. Ha ezeken 4s-on belüli a járművek követési ideje, akkor nyújtani kell. A maximális nyújtás hossza 35s.
A nyujtas12 változó azért 12-re végződik, mert az 1-2. jelzőcsoportot érinti a nyújtás (a többi hasonló változó neve is erre a logikára épül), ezzel a változóval vizsgáljuk a nyújtás időtartamát. A ++ jel segítségével minden futáskor az ACTROS megnöveli a nyujas12 változó értékét 1-el. A 0,5s-onkénti futás miatt kell 70-nel összehasonlítani.
Az x változóban szereplő detektorok, valamint a d3 detektor egyaránt mellékirányban (János utca) van elhelyezve, így ha ezek közül egyik sem foglalt, akkor a főirányt jármű nem akarja keresztezni. Ha a gyalogos nyomógombokat sem nyomta meg senki, akkor a főirány forgalmába a gyalogosok sem avatkoznak bele. Ilyen feltételek mellett felesleges kiadni a zöldet a teljesen üres mellékirányú sávoknak, a főirány kap csak zöldet (korlátlan nyújtás). Ha a mellékirányban elhelyezett detektorok valamelyikén jármű áll (vagy gyalogos jelentkezett), akkor a már maximalizálva van a nyújtás 35s-ra.
Ha a 16. másodpercben van a program, a mellékirányban vannak járművek vagy gyalogosok, de a főiránybeli követési idő kisebb, mint 4 másodperc, akkor a gép 0,5 s-onként nyújtja a zöldet az 1-2. iránynak. Ha a nyújtás időtartama elérte a maximumot, nullázza a nyujtas12 változót. Ha a 16. s-ban van a program, a mellékirányban vannak járművek, és a követési időkre vonatkozó feltétel nem teljesül, akkor többet nem lehet nyújtani (nem állítjuk a ciklusidőt 16-ra), ezért a nyujtas12-t kinullázzuk a következő periódusra való előkészítésként.
A 16,5-dik másodpercben is teljesül, hogy x vagy d3 igaz (hiszen nem tűntek el onnan a járművek), és a követési időre vonatkozó feltétel sem teljesül (mivel 0,5s-mal ezelőtt is túl nagy volt a követési idő, ha most jönne egy jármű még nagyobb lenne), így megint nem állítjuk vissza a ciklusidőt. A következő futás a 17. másodpercben van, amelyben már a legkülső if utasítás belépési feltétele sem teljesül, így teljesen elhagyja a vezérlés ezt a programrészt.

2.5.2. A második nyújtási lehetőség

```
if(Var.d7.getBelSek() > 8 && Var.d6.getBelSek() > 8)
            busz1 = true;
        if(Var.tk1.getProgSek() > 18)
        {   busz1 = false;
            busznyujtas = 0;}
        if(Var.tk1.getProgSek() == 18 && busz1 && ++busznyujtas < 12)
            Var.tk1.setProgSek(18);
```

Ha a (d6 és d7 több, mint 8s óta foglalt)
akkor busz1 legyen igaz

Ha a (ciklusidő >18)

akkor busz1 legyen hamis, busznyujtas=0

Ha (ciklusidő 18 s és busz1 igaz és növelt értékű busznyujtas<12)

akkor ciklusidő legyen 18.

Magyarázat: a d6 és d7 detektorok a Csömöri úton Csömör felé tartó autóbuszok megállójában vannak elhelyezve. Ha ezek több, mint 8s óta foglaltak, akkor a megállóban autóbusz tartózkodik. (Azért van két detektor, hogy csak autóbusz megállóba állása esetén teljesüljön a feltétel, mert csak ő olyan hosszú, hogy mindkettőt egyszerre hangolja el.) Ezt a busz1 változó igazzá tételével jelezzük. A megállóban tartózkodó autóbusz számára a 18. másodpercben van lehetőség nyújtásra, melynek hossza 6s. Azt feltételezzük, hogy ennyi idő alatt a jármű kiáll a megállóból és elhalad a stopvonal felett. Az előzőekben leírt módszerrel 6s-ig nyújtjuk az 1-2. jelzőcsoport zöldjét a ciklusidő visszaállításával. Ha a busznyujtas változó elérte maximális értékét, akkor a ciklusidő nem áll vissza 18s-ra. 18,5-nél a busznyujtas változó ismét nagyobb, mint a maximum, így ismét nem állítjuk vissza a ciklusidőt. A 19. másodpercben hamissá tesszük a busz1-et, és nullázzuk a busznyujtas-t. A következő if utasítás első feltétele (18s) sem teljesül, így ezt a nyújtási lehetőséget is elhagyjuk. Mivel kikötöttük, hogy csak a 18. másodpercben kaphat nyújtást az ott tartózkodó busz, a periódusban ezután érkező autóbuszok már hiába állnak a megállóban megfelelő ideig, nem kaphatnak nyújtást, hiszen nincs is szükségük rá. A 18. másodperc után más irányok kapnak szabad jelzést, így ebben az esetben fázisbeillesztéssel lenne biztosítható a dedikált jármű csomóponton való átjuttatása, azonban ilyen intézkedés jelen programban nincs. Az utascserét elvégző autóbusz számára a következő periódusban sem kell zöldet nyújtani, mivel a megálló nagyon közel van a stopvonalhoz, így onnan kiállva hamar átjut a csomóponton.

2.5.3. Az első ugrási lehetőség

```
if(Var.tk1.getProgSek() == 19 && !Var.d3.getAnforderung())
            Var.tk1.setProgSek(30);
```

Ha a (ciklusidő 19 s és a d3 hamis)

akkor a ciklusidő legyen 30s.

Magyarázat: ha a 19. másodpercben a János utca déli részéből senki sem érkezik, akkor teljesen felesleges kiadni ennek az iránynak a zöldet. A pontos okfejtéshez a kód végére kell ugranunk a K.BEKI eljárások közé némi mellékes magyarázattal.

Megfigyelhetjük, hogy a j11, j21, gy71, gy51, gy61 jelzőcsoportokhoz kétféle szabad jelzés kapcsolás tartozik. Az alapesetben, ha nem történik a 19. másodpercben ugrás, akkor a j11, j21 a 21-dik, a gy71 pedig a 20. másodpercben vált át az átmeneti jelzésképre; emellett a gy51 és gy61 jelzőcsoport pedig a 28-dik másodpercben vált zöldre. Ha viszont a 19. másodpercben ugrás történt a 30. másodpercre, ezeket az átmenetire (és zöldre) kapcsolásokat kihagytuk. A balesetek elkerülésének érdekében be kell szúrni egy újabb átmeneti képre kapcsolást ezeknél

a jelzőcsoportoknál, ez pedig a 31. másodpercben történik meg. (Ha nem volt ugrás, és a 21. másodpercben már átmenetire kapcsolódtak a jelzőcsoportok, elérve a 31. másodpercet az újabb parancs már hatástalan lesz rájuk, nem csinálnak piros-sárga-piros átmenetet.) A zöldre sem kapcsolt j31 jelző sem fog ilyen állapotokon végigmenni. A gy51 és gy61 jelzőcsoportok zöldre kapcsolásával a közbensőidő mátrix előírásai miatt kell a 38. másodpercig várni. (ki 1—be 6: 7s)

2.5.4. A harmadik nyújtási lehetőség

```
if(Var.tk1.getProgSek() == 32)
            if(!Var.d8.dynLuecke(3000))
            {   if(++nyujtas3 < 6)
                    Var.tk1.setProgSek(32);
                else
                    nyujtas3 = 0;
            } else
            { nyujtas3 = 0;}
            }
```

Ha a (ciklusidő 32 s)
akkor ha (a d8 időablaka nem 3 s)
akkor ha (növelt nyujtas3<6)
akkor a ciklusidő legyen 32 másodperc
különben nyujtas3 legyen 0
különben nyujtas3 legyen 0.

Magyarázat: az előző két nyújtás működésének analógiájára működik ez is, de itt csak a ciklusidő 32. másodperce és a d8 detektoron mért követési idő szabja meg a feltételeket. A többi irányban lévő detektorokhoz nem kötjük a nyújtást vagy annak elhagyását. A maximális nyújtás 3 másodperc, a detektor időablaka szintén. Megjegyzés: amíg az előző intézkedés a fázis kihagyására irányult, ez a nyújtására.

2.5.5. A második ugrási lehetőség

```
if(Var.tk1.getProgSek() == 36 && !x)
            Var.tk1.setProgSek(44);
```

Ha a (ciklusidő 36 s és x hamis)
akkor a ciklusidő legyen 44s.

Magyarázat: a 4-es és 5-ös detektorok a 4-es jelzőcsoport által irányított sávokban vannak. Ha ezeken nem áll jármű, akkor ez a fázis kihagyható, így j41-et nem kell zöldre kapcsolni. A gy51, gy61 jelzők kapcsolásával nem kell törődni, hiszen azok csak a 45. másodpercben váltanak át az átmeneti jelzésképre; a többi jelző pedig nem vált jelzésképet az átugrott idő

alatt. (kivéve persze a j41, de ezt akartuk kihagyni) A 44. másodperc valószínűleg azért nem 45, mert így nem az ugrás pillanatában kell jelzőket átkapcsolni, nem lesznek időbeliségi problémák.

2.5.6. A negyedik nyújtási lehetőség

```
if(Var.tk1.getProgSek() == 44)
            if(!Var.d9.dynLuecke(3000))
            {   if(++nyujtas4 < 12)
                    Var.tk1.setProgSek(44);
                else
                    nyujtas4 = 0;
            } else
            { nyujtas4 = 0;}
            }
```

Ha a (ciklusidő 44s)
akkor ha (a d9 időablaka nem 3s)
akkor ha (növelt nyujtas4 <12)
akkor ciklusidő legyen 44 s
különben nyujtas4=0
különben nyujtas4=0

Magyarázat: a számadatokat kivéve működése teljesen megegyezik a harmadik nyújtási lehetőséggel. Az természetesen nem lehetséges, hogy a második ugrási lehetőséget kihasználva 44s-ba jutva a negyedik nyújtási lehetőséget is igénybe vegyük. Ez fizikailag nem lehetséges, mert a j41 jelzőcsoportot nem kapcsolta semmi zöldre, így ezt nem lehet nyújtani. (A gyalogost lehetne, de nem szokás egymagában, nincs is gyalogosfázist nyújtó detektor.)

Ha a forgalomfüggő program változtatási lehetőségeiből egyet sem kell igénybe venni, akkor fix programként működik. A változtatások ellenére a jelzőcsoportokat itt is a K.BEKI eljárások váltják szabadra ill. tilosra.

```
K.BEKI(Var.j11, 2, 21);
K.BEKI(Var.j21, 2, 21);
K.BEKI(Var.j11, 2, 31);
K.BEKI(Var.j21, 2, 31);
K.BEKI(Var.j31, 26, 33);
```

2.6. AltalanosResz.java

Ez a rész a PostMainFkt osztályhoz tartozik, így az aktuális program után fut le. Az elején itt is a konstruktor paramétereinek beolvasása található. Eljárása az `inJedemCyklus()` (=minden

ciklusban) tartalmazza a gyalogos nyomógombok „nullázását” a nekik megfelelő gyalogos jelzők bekapcsolásakor. Ezt a K. GYALOGOS_VISSZAJELZŐ eljárással (lásd 19. oldal) lehet elvégezni. Másrészt definiáltak egy „hiba detektort”, amely a valódi detektorok hibája esetén válik foglalttá. Ebben az esetben fix időtervű programra tér át a berendezés. A valódi csomópontban más rendszerű detektorkártyák vannak a gépben, melyek képesek saját hibájuk felismerésére. A tanszéki berendezésben lévők nem alkalmasak erre, így nekik másként kell „megmondani”, hogy detektorhiba lépett fel. Az else kifejezés utáni rész a forgalomfüggő program (4) futtatását kéri a vezérléstől függetlenül attól, hogy mi van a napi/heti tervekben megadva.

```
protected void inJedemZyklus()
    {   if(Var.tk1.getAktProg().getNummer() > 0 &&
Var.tk1.getAktProg().getNummer() < 5)
        {   K.GYALOGOS_VISSZAJELZŐ(Var.gy51, Var.viNy1, Var.ny1);
             //jelzőcsoport, visszajelző, detektor
            K.GYALOGOS_VISSZAJELZŐ(Var.gy61, Var.viNy2, Var.ny2);
        }
         if (Var.derr.belegt())
             Var.tk1.setProgWunsch(Vt.KEIN_PROGRAMMWUNSCH,
StgEbene.STG_VT_ANWENDER);
        else
             Var.tk1.setProgWunsch(Var.prog4,
StgEbene.STG_VT_ANWENDER);     }
```

2.7. LmpHb.java, LmpHb230v.java

A fénypontok hardverszintű azonosításáért felelnek.

2.8. K.java

Ez az osztály az eddigiektől eltérően a vtbib.jar-ban található meg, célja a magyar programozók munkájának megkönnyítése. A benne lévő metódusok logikája nagyon hasonló az ACTROS elődgépének (VTC2000) programozási módszereihez. Az osztály a jelzőcsoportok kapcsolására alkalmas eljárásokat, valamint igényfázis beszúrását tartalmazza. Számunkra a K.BEKI(…) és a K.GYALOGOS_VISSZAJELZŐ(…) eljárások fontosak.
A BEKI a jelzőcsoport szabadra ill. tilosra kapcsolását végzi el. (A szabad időbe az előkészítő időt is bele kell számolni!) Az első eljárás feltételhez köti a jelzőcsoport bekapcsolását. A bemenő paraméterek között meg kell adni a jelzőcsoportot, a szabad jelzés kezdetét és végét. A freiSperr() eljárás végzi el a tényleges kapcsolást.

```
public static void BEKI(Sg jcs, int kezdet, int veg, boolean felt)
```

```
{ if (felt)
  {jcs.freiSperr(kezdet, veg);
  }
}
```

A második esetben nem kötünk ki feltételt a jelzőcsoport szabadra állításához. A programban ehhez nem készül külön eljárás, hanem a feltételes eljárást hívjuk meg IGAZ-ra állított feltétellel. (Így nem akadályozza semmi a szabadra állítást.

```
public static void BEKI(Sg jcs, int kezdet, int veg)
{ BEKI(jcs, kezdet, veg, true);
}
```

Megjegyzés: a jelzőcsoportok kapcsolása pusztán a freiSperr(kezdet, vég) eljárás meghívásával is végrehajtható, - Németországban ezt használják - azonban nálunk elterjedt a „körmönfontabb” megoldás is.

A gyalogos visszajelző a valóságban tulajdonképpen a megnyomott gomb felett villogó „Várjon!” feliratot jelenti. Ebben az esetben a nyomógombot detektornak tekintjük, mivel két állapota lehetséges (megnyomták, nem nyomták meg) Az első if feltétel akkor válik igazzá, ha a nyomógombhoz tartozó jelzőcsoport pirosat mutat, és a gombot megnyomták. Ekkor a visszajelző villogni kezd,melyet a schalteSofort(villogó állapot) eljárás indít el. A második feltétel a villogás megszűntetésére szolgál akkor, ha a jelzőcsoport zöldet kapott.

A járműdetektoroknál nem volt szükség ilyen szétválasztásra, mint ebben az esetben. Ennek oka az, hogy a járműdetektoroknál nem kellett tárolni azt, hogy a detektoron volt foglaltság, mivel azt minden 0,5 s-ban a megfelelő programrész futásakor lekérdeztük. A gyalogos nyomógomboknál tárolni kell azt, hogy megnyomták, mert csak ennek hatására fog zöld jelzést kapni a gyalogátkelőhely. A tárolás megvalósítása a villogás: nem a nyomógomb marad benyomódva, hanem a LED-ek jelzik ennek megtörténtét.

```
public static void GYALOGOS_VISSZAJELZO(Sg jcs, Sg visszajelző,
Detektor nyg)
   {if (jcs.getZustand() == Zustand.ROT &&
                  nyg.getAnforderung())
       {visszajelző.schalteSofort(Zustand.GELBBLINKEN);
       }
       if (jcs.getZustand() == Zustand.GRUEN)
       {visszajelző.schalteSofort(Zustand.DUNKEL);
       }
}
```

3. Kiegészítő információk

3.1. A berendezés paraméterek

A VT-Bib könyvtár „exp” (experimentell) Package-ében található osztályok azon paraméterek alkalmazását engedélyezik, amelyek változtathatók, és külső fájlban lettek létrehozva.

A vt Package azon ősosztályai, amelyekből a program, a fázis és a fázisátmenet osztályok származnak, megfelelően elő vannak készítve a paraméterek felhasználására. Az is fontos, hogy a forgalomtechnikát tartalmazó osztályok paramétereit ismertté tegyük a rendszer számára. Ez az inicializáló osztályban történik a programinicializálás végén.

Egy objektum definiálásakor, az objektumban deklarált változók a paraméterfájlba kerülnek.

3.2. Ellenőrző program (Pruefung)

Az ellenőrzés lehetővé tételéhez szükséges, hogy a Pruefung osztályt az aktuális VT package-be beépítsük, ill. a Pruefung osztály main metódusában az aktuális részcsomópontot bejelöljük.
Amennyiben a berendezés egynél több részcsomóponttal rendelkezik, a Pruefung osztályt másodszor is (ill. harmadszor, ha 3 részcsomópont van) be kell építeni a Package-be (mindig egy új névvel!).

3.3. Műveletek a Java-ban

Az egyes műveleteket a következő szimbólumok jelölik:

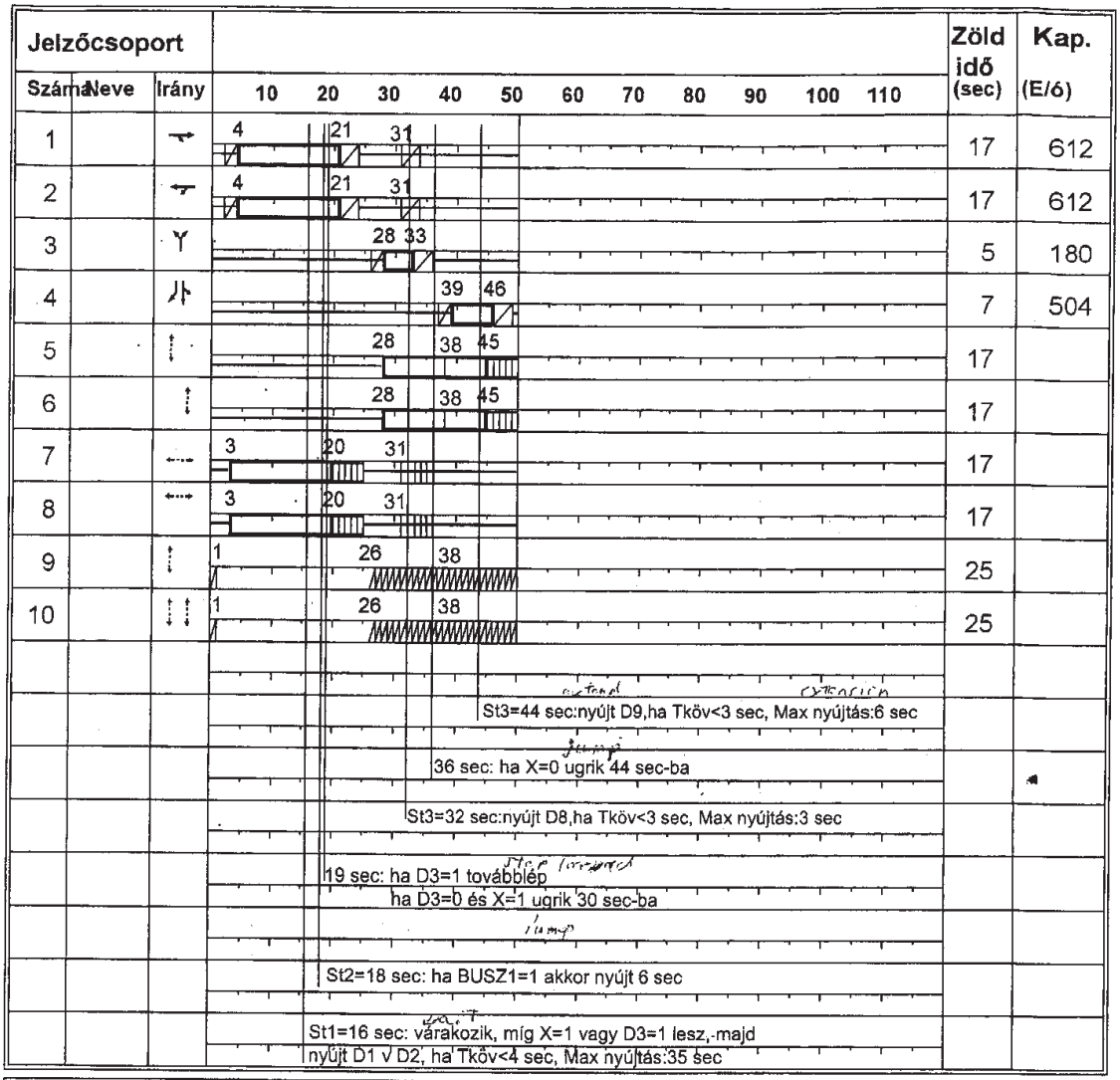
Művelet	Szimbólum
Értékadás	=
Összehasonlítás	== (két egyenlő jel)
ÉS kapcsolat	&&
VAGY kapcsolat	||
negálás	!

Felhasznált irodalom

[1] Uwe Wilbrand: Java forgalomtechnika az ACTROS VTC 3000 berendezésben (Ford.: Tettamanti Tamás) 2008; pp. 18
[2] ACTROS Java Help by Signalbau Huber GmbH

Szómagyarázatok

die Anlage: berendezés
das Ausprogramm: kikapcsoló program
das Blinkenprogramm: sárga villogó program
das Einprogramm: bekapcsoló program
(das) FestProg (Festenzeitprogramm): fixidejű, nem forgalomfüggő program
gelb: sárga
grün (gruen): zöld
IoKanal: kimeneti/bemeneti csatorna
(der) LmpTyp (Lampentyp): fénypont típusa
die Prüfung: ellenőrzés
rot: piros
schalten: kapcsolni
der Schaltkanal: kapcsolócsatorna (a kártya kapcsoló kimenete)
(die) Sg (Signalgruppe): jelzőcsoport
(der) SgTyp (Signalgruppentyp): jelzőcsoport típusa
sofort: azonnal
der Tagesplan: napi terv
die Uhr: óra
(die) VT (Verkehrstechnik): forgalomtechnika
Wiederholer: ismétlőjelző
der Wochenplan: heti terv
der Zustand: állapot
die ZwzMatrix (Zwschenzeitmatrix): közbensőidő mátrix



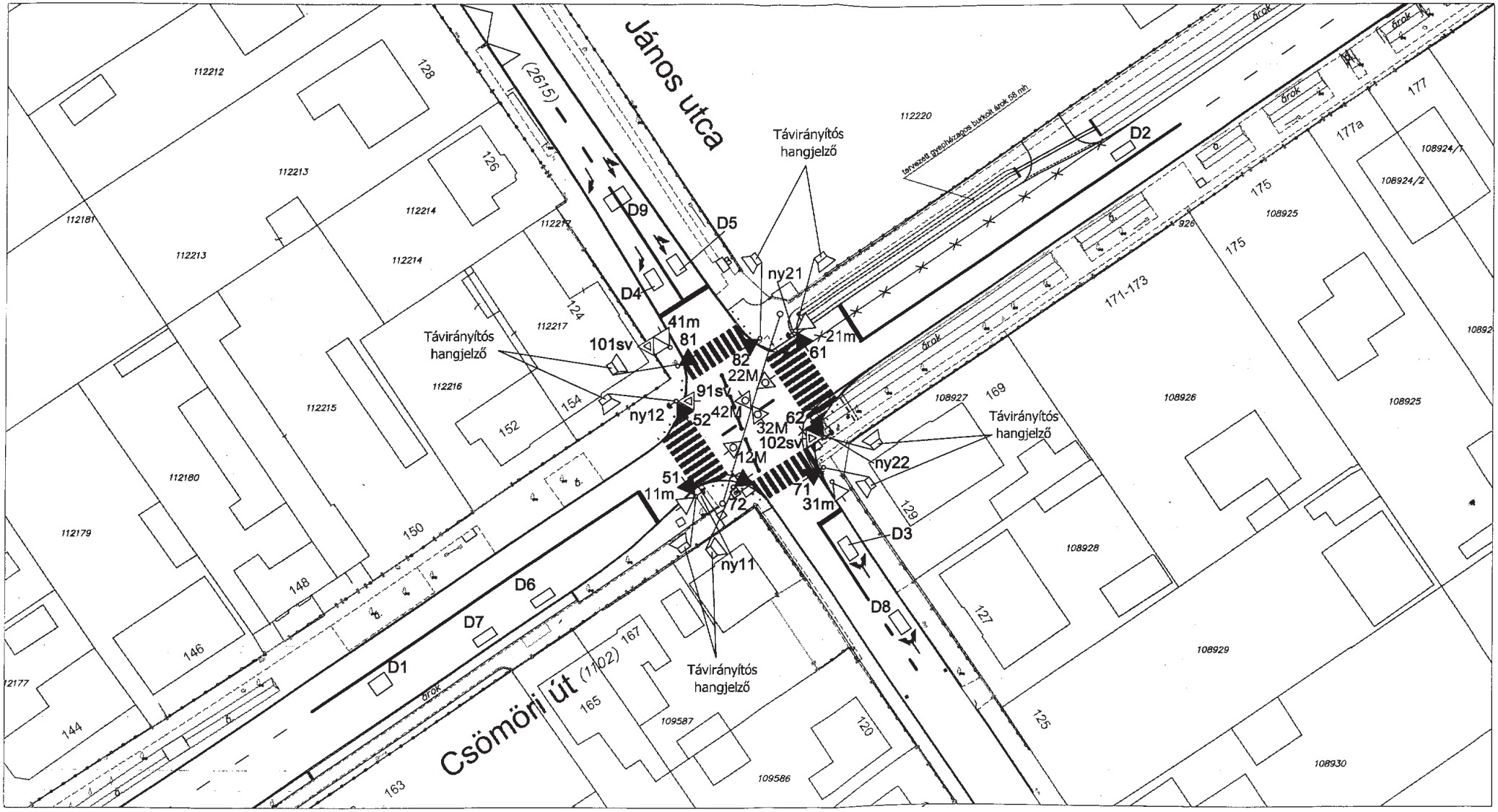
Megjegyzés:

X=D4vD5vny1vny2

BUSZ1= ha D7-ra rálép a jármű és folyamatos foglaltság mellett D6 is foglalt lesz és 8 sec óta foglalt mind a két detektor.

Tervező: Wendl Árpádné	Ellenőr: Makay János	Dátum: 2007.11.27
Tárgy: **BUDAPEST XVI. ker.** **Csömöri út - János utca csp.**		**VILATI-SBH** 1103.Budapest, Gyömrői út 120.
Fázisterv Forgalomtól függő program		Tervszám: 7509.410

Prog.száma:	Prog.neve:	P = sec	Be:	Ki:	Prog.váltás:	Rajzszám:	
4		50			10	K-2	8



Tervező: Wendl Árpádné	Ellenőr, felelős tervező: Makay János (K2-2/01-5526)	Dátum: 2008. június
Tárgy: Budapest, XVI. ker. Csömöri út - János utca csp. Közúti jelzőberendezés kiviteli terve		**VILATI - SBH**
		Témaszám: 7509.410
Részművelet: Jelzőlámpa számozási terv		Rajzszám: K - 2 / M
		Méretarány: 1:500