VORLESUNG

Automatisierung industrieller Workflows

Teil C: Die Sprache SLX

- Vertiefung der Basissprache -

Joachim Fischer

Position

- **Teil A**
  Aspekte von Modellierung und Simulation dynamischer Systeme
- **Teil B**
  Die Modellierungssprache UML
- **Teil C**
  Die ausführbare Modellierungssprache SLX
- **Teil D**
  Weitere Modellierungssprachen
- **Teil E**
  Modellierung von Stahlwerksprozessen

- **C.1**
  Einführung und Basissprache
- **C.2**
  Stochastische Prozesse in SLX
- **C.3**
  Vertiefung der SLX-Basissprache
- **C.4**
  GPSS-Elemente
- **C.5**
  DISCO-Elemente
- **C.6**
  Basissprache (Ergänzung)

Vertiefung der Basissprache - Inhalt

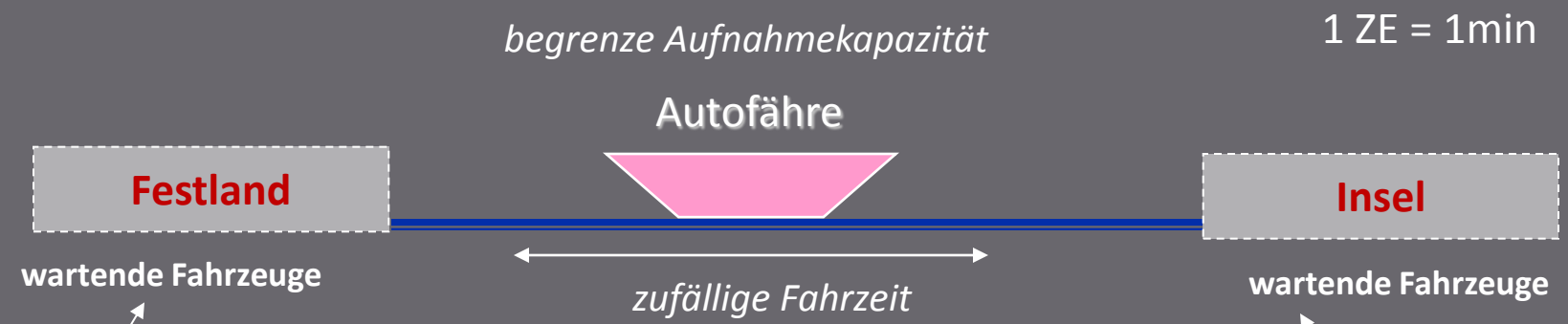
1. Beispiel: Autofähre (Überblick)
2. Erinnerung: Set-Operationen
3. Beispiel: Autofähre (Ablauf)
4. Unterbrechung von per Interrupt
5. Beispiel: Drucker mit zeitweiligem Ausfall
6. Prozessparallelität
7. Prioritäten
8. Klassenvererbung

Inhalt der letzten Vorlesung

Einige Präzisierungen

Beispiel: Autofähre

- Wortmodell und informale Darstellung
- Erste Diskussion einer SLX-Implementation



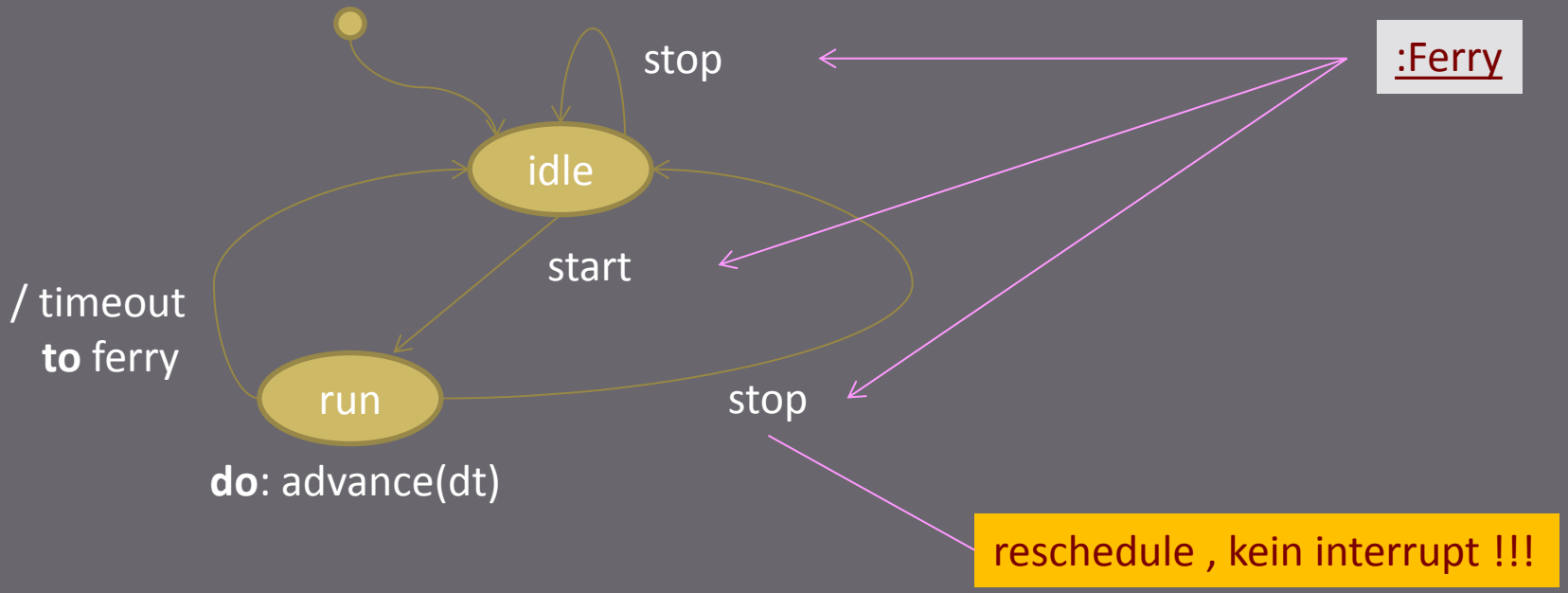
Vorschrift
- bei Ankunft fahren zunächst alle Autos von der Fähre
- dann fahren wartende Autos auf die Fähre
- mindestens 25 min verweilt die Fähre an einer Anlegestelle

zufällige Ankunftszeiten

zufällige Ankunftszeiten

Simulationsziel
- Simulation eines Tages (zunächst 8h)
- separate Zählung von Fahrten
  (später: und Leerfahrten
  sowie Auslastung der Fähre)

Clock als UML-Zustandsautomat



letzte Vorlesung: Diskussion eines alternativen Interrupts

Idee

puck-Pointer->move_time

zu ändern,
scheitert (geschütztes Attribut im Gegensatz zu priority)

```
method timer_stop () {
        #ifdef FERRY_DEBUG
                print (time, myFerry->name, name) "___._ ++++ _: stops _ \n";
        #endif
        if (is_running) {
                #ifdef FERRY_DEBUG
                        print (time, myFerry->name, name) "___._ ++++ _ RESCHEDULES _ \n";
                #endif
                reschedule myPuck at time;
                yield to myPuck;
        }
        is_interrupted= TRUE;
}
```

```
actions {
        myPuck= ACTIVE;

        forever {
                is_running= FALSE;
                is_interrupted= FALSE;
                wait;
                has_timeout= FALSE;
                is_running= TRUE;
                #ifdef FERRY_DEBUG
                        print (time, name) "___._ .... _: Internal--- Start \n";
                #endif
                advance(alarmTime);
                is_running= FALSE;
                has_timeout= TRUE;
                #ifdef FERRY_DEBUG
                        print options= red (time, name, myFerry->name) "___._ .... _: Internal ---TimeOut for: _ \n";
                #endif
                if (myFerry->myPuck->state == WAITING ) reactivate myFerry->myPuck;
                if ( is_interrupted) {
                        #ifdef FERRY_DEBUG
                                print options= bold (time, name, myFerry->name) "___._ .... _: is interrupted by _\n";
                        #endif
                }
        } //forever
} //actions
```

start

timeout

stop

```
method timer_start () {
        #ifdef FERRY_DEBUG
                print (time, myFerry->name, name) "___._ ++++ _: restarts _ \n";
        #endif
        if (not is_running) {
                reactivate myPuck;
        }
}
```

Offenes Problem: set (1)

set ist eigentlich ein generischer Listentyp

```
set(X) xs;
set(X) ranked LIFO xsl;

class X {
        int a;
        int b;
}

set(X) ranked
    (ascending a, descending b) s;
```

```
set(...) ranked FIFO <Name>
set(...) ranked LIFO <Name>
set(...) ranked (ascending a1, ...) <Name>
set(...) ranked (descending d1, ...) <Name>
```

- einziger Mengentyp:
  geordnete Menge von Objektzeigern
- Definition (homogenes Set):
  set(<Klasse>) <Name>
- Definition (universelles Set):
  set(*) <Name>
- Sortierung einstellbar:
  - FIFO (Standard),
  - LIFO,
  - aufsteigend/absteigend nach Attributen
    (nur für homogene Sets)

Offenes Problem: set (2)

```
class X  (int pa, pb) {
        int a;
        int b;
        initial {
                a= pa;
                b= pb;
        }
}

set(X) ranked
    (ascending a, descending b) container;

procedure main() {
        int i;
        int j;
        pointer(X) x, first_x;
        for ( i=1 ; i<=10 ; i++ ) {
                place new X(i, 0) into container;
        }
        for (x= each X in container) {
                j++;
                print (j, x->a, x->b) "_. Element ( a= _, b= _) \n";
        }
        first_x= last X in container;

        place new X(11,0) into container after first_x;
 Semantic error: "container" is a ranked set; its ranking cannot be overridden
}
```

System – und Nutzer-Report

System Status at Time 488.0308

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
Arrival 1 arrival	80	400000	400080	OFF	0.19
Arrival 2 arrival	75	600000	600075	OFF	N/A
trip	42	200000	200042	OFF	N/A

Versuchsanzahl ~ bislang zu gering

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
trip_time	12	4.138	2.488		0.27	9.13

Lower	Upper	Frequency	Percent	
0.0	0.5	1	8.333	**********
1.0	1.5	2	16.667	**********
3.5	4.0	3	25.000	**********
4.0	4.5	1	8.333	**********
5.0	5.5	3	25.000	**********
6.0	6.5	1	8.333	**********
9.0	9.5	1	8.333	**********

Vehicles: 155
FERRY: Ferry 1

current vehicles	trips	transports
0	12	136

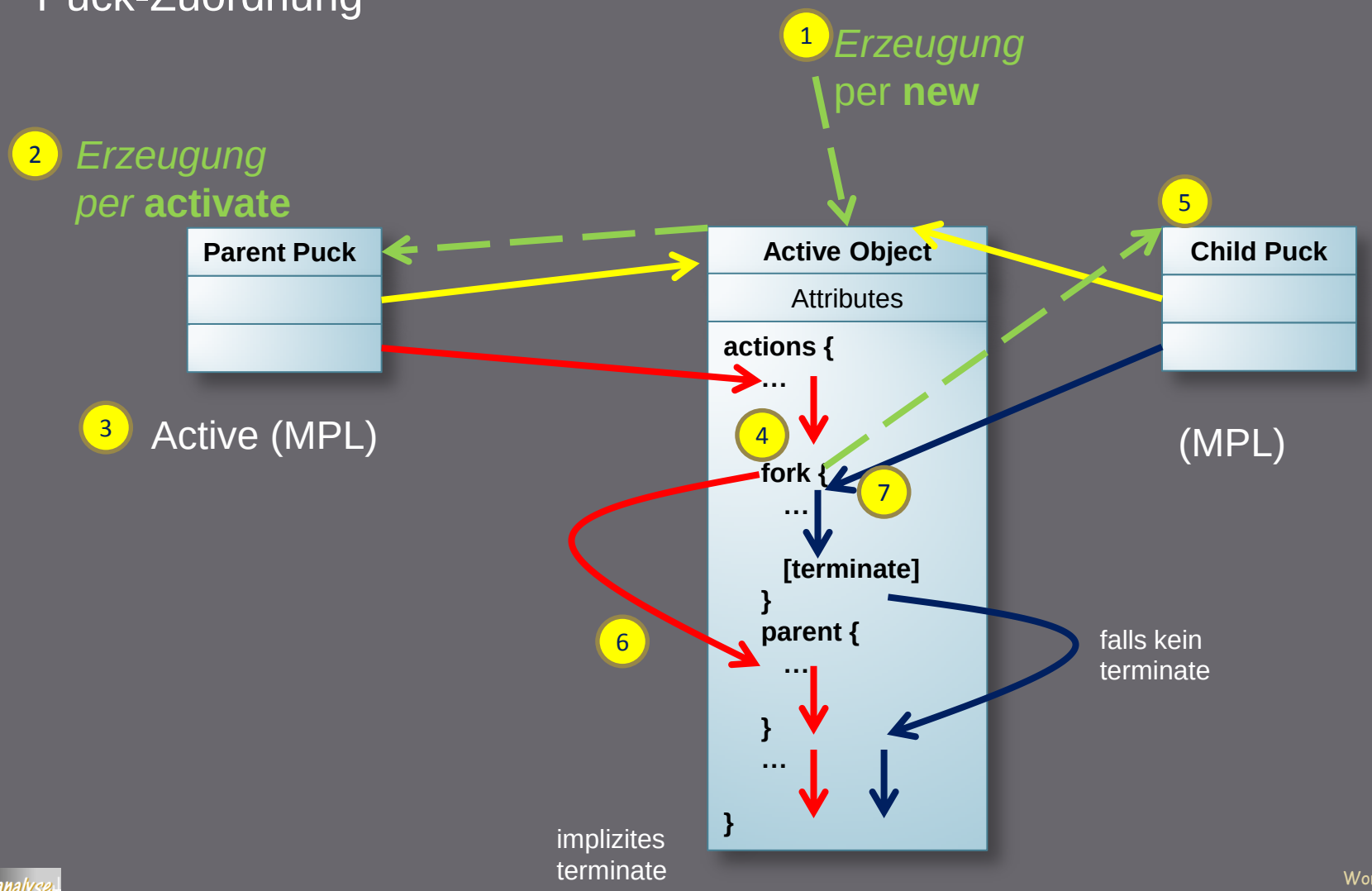
Station: ISLAND current vehicles= 12
Station: MAINLAND current vehicles= 7
Execution complete

Vertiefung der Basissprache - Inhalt

1. Beispiel: Autofähre (Überblick)
2. Erinnerung: Set-Operationen
3. Beispiel: Autofähre (Ablauf)
4. Unterbrechung von per Interrupt
5. Beispiel: Drucker mit zeitweiligem Ausfall
6. Prozessparallelität
7. Prioritäten
8. Klassenvererbung

Interne Prozessparallelität

- Puck-Zuordnung



```
//*****************************************************************************
// Module Basic EX0018
//*****************************************************************************
module basic {
        rn_stream arrivals, service ;
        int shutdown_time = 5*8*60,  jobs_in,    jobs_cancelled ;
        control int jobs_printed ;
        float total_queueing_time;
        pointer( cl_printer ) printer;
        set ( cl_printer_job ) waiting_line; // Queue for arriving jobs

class cl_printer_job ( in int in_job_num ) {
        int job_number;
        pointer ( puck ) my_puck; // Puck for Parent printer_job process
        initial {
                job_number = in_job_num;
        }
        actions {
                my_puck = ACTIVE; // Store the pointer to the parent puck
                place ME into waiting_line; // Place in Job Queue
                if ( not printer->printer_busy ) // Printer process sleeping?
                        reactivate printer->my_puck;
                fork {// local parallelism for Timeout
                        pointer ( cl_printer_job ) temp_job;
                        advance 1.5; // max. waiting time
                        // Is my Job printing in the waiting line ?
                        for ( temp_job = each cl_printer_job in waiting_line ) {
                            if ( temp_job == ME ) {
                                remove ME from waiting_line ;
                                reactivate my_puck; // Child wakes the Parent
                                total_queueing_time += (time - temp_job->my_puck->mark_time); // Statistics
                                jobs_cancelled ++;
                                terminate;
                            }
                        }
                        terminate ;
                } // fork
                wait; // Sleep until wake up
                my_puck = NULL ;
                terminate ;
        }
} // cl_printer_job
```

Parent Puck ~ PrinterJob

Child Puck ~ Wecker
zum Warte-Abbruch

Abbruch des Wartens
ohne Ausdruck

Frage: an welcher Stelle setzt
die Steuerung des Parent-Pucks
fortgesetzt ?

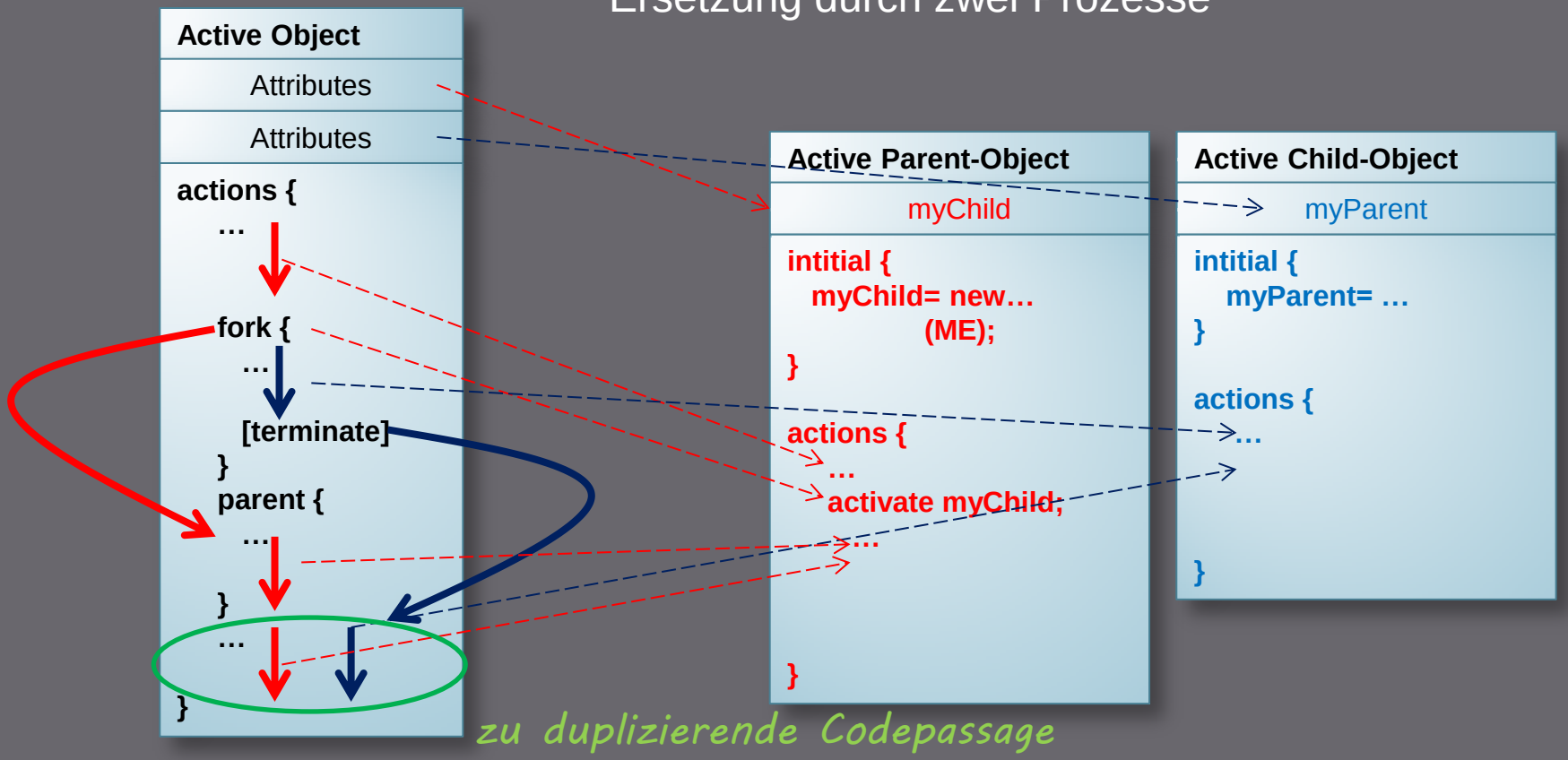
```
class cl_printer {
        control boolean printer_busy;
        pointer ( puck ) my_puck; // Puck for Printer Process
        pointer ( cl_printer_job ) owner; // Current Print Job

        actions {
                my_puck = ACTIVE; // Store the Pointer to the Puck
                forever {
                        wait; // Wait for Print Jobs
                        // While Contents of Job Queue != 0
                        while ( (first cl_printer_job in waiting_line) != NULL ) {
                                owner = first cl_printer_job in waiting_line; // Take First Job
                                remove owner from waiting_line ;
                                printer_busy = TRUE;
                                total_queueing_time +=
                                        (time - owner->my_puck->mark_time); // Statistics
                                advance rv_uniform ( service , 0.5, 15.0 ) ; // printing time
                                printer_busy = FALSE;
                                jobs_printed ++;
                                // Wake up the Sleeping Print Job Process
                                reactivate owner->my_puck; //parent puck
                                owner = NULL;
                        } // while
                } // forever
        } // actions
}

procedure main() {
        printer = new cl_printer ;
        activate printer;
        while (time < shutdown_time) {
                jobs_in++;
                activate new cl_printer_job ( jobs_in ); // Create new jobs
                advance rv_uniform( arrivals , 10.0,20.0 ) ; // interarrivel time
        };
        wait until ( (jobs_in - jobs_cancelled) == jobs_printed);

        print ( jobs_in, jobs_cancelled, jobs_printed, total_queueing_time / jobs_printed*60 )
                "Incoming Jobs :_"
                „Cancelled Jobs :_"
                "Printed Jobs : _"
                "Mean Queueing Time/Job : _.__ seconds";
}
}
```

Beginn der Bedienung

Beendigung der Bedienung

Auflösung innerer Parallelität

Ersetzung durch zwei Prozesse



```
//*************************************************************************
// Module Basic EX0018 modified
//*************************************************************************
module basic {
        rn_stream arrivals, service ;
        int shutdown_time = 5*8*60,
            jobs_in,
            jobs_cancelled ;
        control int jobs_printed ;
        float total_queueing_time;
        pointer( cl_printer ) printer;
        set ( cl_printer_job ) waiting_line; // Queue for arriving jobs


class cl_timer (in pointer (cl_printer_job) job) {      // urspruenglich Child
        pointer (cl_printer_job) owner;                 // Parent-Verweis
        initial {
                owner= job;
        }
        actions {
                advance 1.5; // max. waiting time
                // Is my job blocked in the waiting line ?
                if (waiting_line contains owner) {
                        remove owner from waiting_line ;
                        reactivate owner->my_puck; // wakes the blocked cl_printer_job
                        total_queueing_time +=
                                (time -owner->my_puck->mark_time); // Statistics
                        jobs_cancelled ++;
                }
                owner= NULL;
                terminate ;
        }
}
```

Abbruch des Wartens auf Bedienung durch Reaktivierung

```
class cl_printer_job ( in int in_job_num ) {
        int job_number;
        // Puck for Parent printer_job process
        pointer ( puck ) my_puck;
        pointer(cl_timer) my_timer;

        initial {
                job_number = in_job_num;
                my_timer= new cl_timer(ME);
        }
        actions {
                my_puck = ACTIVE; // Store the pointer to the puck
                place ME into waiting_line; // Place in Job Queue
                activate my_timer;
                if ( not printer->printer_busy ) // Printer process sleeping?
                        reactivate printer->my_puck;
                wait; // Sleep until wake up
                  // by timer without print or by printer after print
                my_puck = NULL ;
                terminate ;
        }
} // cl_printer_job
```

Abbruch des Wartens
oder Bedienungsende

```
class cl_printer {
        control boolean printer_busy;
        pointer ( puck ) my_puck; // Puck for Printer Process
        pointer ( cl_printer_job ) owner; // Current Print Job

        actions {
                my_puck = ACTIVE; // Store the Pointer to the Puck
                forever {
                        wait; // Wait for Print Jobs
                        // While Contents of Job Queue != 0
                        while ( (first cl_printer_job in waiting_line) != NULL ) {
                                owner = first cl_printer_job in waiting_line; // Take First Job
                                remove owner from waiting_line ;
                                printer_busy = TRUE;
                                total_queueing_time +=
                                        (time - owner->my_puck->mark_time); // Statistics
                                advance rv_uniform ( service , 0.5, 15.0 ) ; // printing time
                                printer_busy = FALSE;
                                jobs_printed ++;
                                // Wake up the Sleeping Print Job Process
                                reactivate owner->my_puck;
                                owner = NULL;
                        } // while
                } // forever
        }// actions
}
```

Beendigung der Bedienung

```
procedure main() {
        printer = new cl_printer ;
        activate printer;
        while (time < shutdown_time) {
                jobs_in++;
                activate new cl_printer_job ( jobs_in ); // Create new jobs
                advance rv_uniform( arrivals , 10.0,20.0 ); // interarrivel time
        };
        wait until ( (jobs_in - jobs_cancelled) == jobs_printed);

        print ( jobs_in,
                jobs_cancelled,
                jobs_printed ,
                total_queueing_time / jobs_printed*60 )
        "Incoming Jobs :_"
        „Cancelled Jobs :_"
        "Printed Jobs : _"
        "Mean Queueing Time/Job : _.__ seconds"
        ;
}
}
```

Ausgabe UNI [10, 20]

- Incoming Jobs : 163
- Cancelled Jobs : 6
- Printed Jobs : 157

Ausgabe UNI [5, 20]

- Incoming Jobs : 194
- Cancelled Jobs : 35
- Printed Jobs : 159

Ausgabe EXPO [5]

- Incoming Jobs : 470
- Cancelled Jobs : 271
- Printed Jobs : 199

Vertiefung der Basissprache - Inhalt

1. Beispiel: Autofähre (Überblick)
2. Erinnerung: Set-Operationen
3. Beispiel: Autofähre (Ablauf)
4. Unterbrechung von per Interrupt
5. Beispiel: Drucker mit zeitweiligem Ausfall
6. Prozessparallelität
7. Prioritäten
8. Klassenvererbung

Prioritäten

Änderung der Priorität

- direkte Veränderung des Attributwertes

  ACTIVE->priority = *prio_value*

- Aktivierung eines Prozesses

  activate *ptr_class_ident* **priority** *prio_value*

- Reaktivierung eines Prozesses

  reactivate *ptr_class_ident* **priority** *prio_value*

- Generierung von lokal parallelen Prozessen

  fork priority *prio_value*

Vertiefung der Basissprache - Inhalt

1. Beispiel: Autofähre (Überblick)
2. Erinnerung: Set-Operationen
3. Beispiel: Autofähre (Ablauf)
4. Unterbrechung von per Interrupt
5. Beispiel: Drucker mit zeitweiligem Ausfall
6. Prozessparallelität
7. Prioritäten
8. Klassenvererbung

Vererbungsersatz

- Version 1.x Vererbungsersatz für Klassen

```
class A {
      A1-attributes…
      A1-Methoden…
      A1-Properties…
};

augment A  {
      A2-attributes…
      A2-Methoden…
      A2-Properties…
};
```

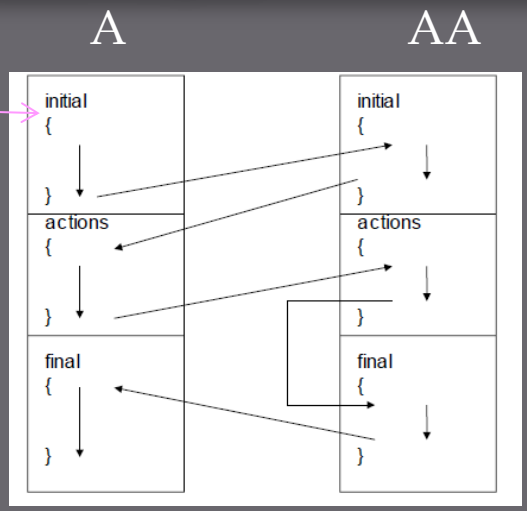
A wird durch neues A (AA) ersetzt
(AA übernimmt altes A und fügt Attribute hinzu)

```
set (A)  liste_von_As; // Polymorphie,
                       // d.h. Liste enthält AA-Objekte
                       // aber keine A-Objekte
```

new A

activate A-Objekt

delete A-Objekt



korrekte Anwendung liegt
in der Hand des Nutzers

Aktionsreihenfolge für Objekte von Augment-Klassen

Vererbung

- ab Version 2.x Einfachvererbung für Klassen

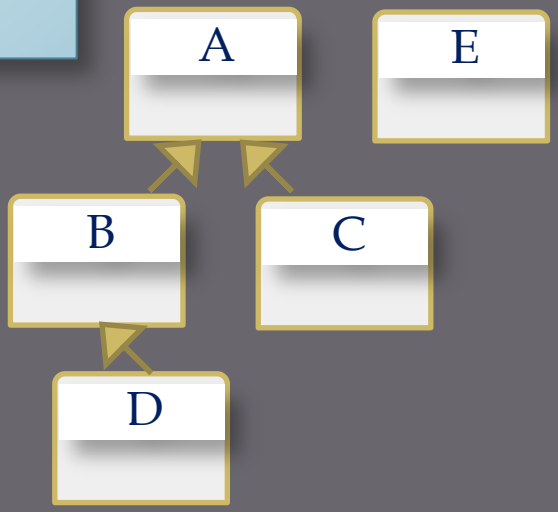
```
class A {
      A-attributes…
};

class B subclass (A) {
      B-attributes…
};

class C subclass (B)
       C-attributes…
};
….
```

```
set (A)  liste_von_As; // Polymorphie,
                       // d.h. Liste kann auch B-Objekte
                       // enthalten
```

Liste_von_As kann keine E-Objekte enthalten

SLX2 erlaubt dynamischen Cast

ptr_B= ptr_A; //mit dynamischen Typtest
//evtl. Laufzeitfehler



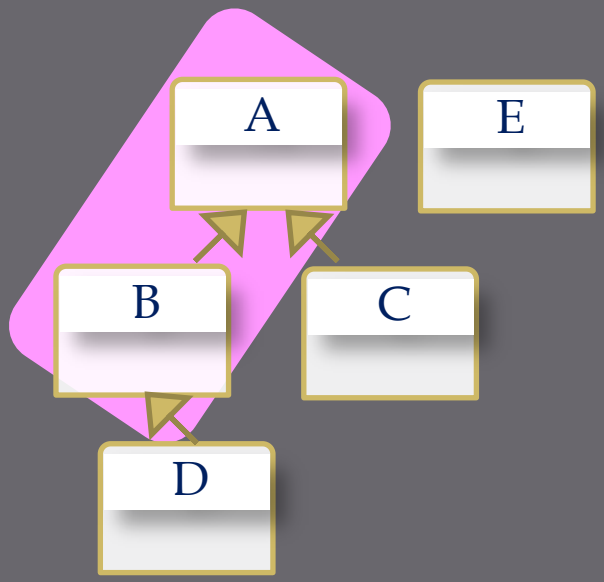
Basis- und abgeleitete Klasse können keine Attribute, Prozeduren oder Methoden gleichen Namens besitzen

Es sei denn, es handelt sich um redefinierbare Attribute, Prozeduren, Methoden

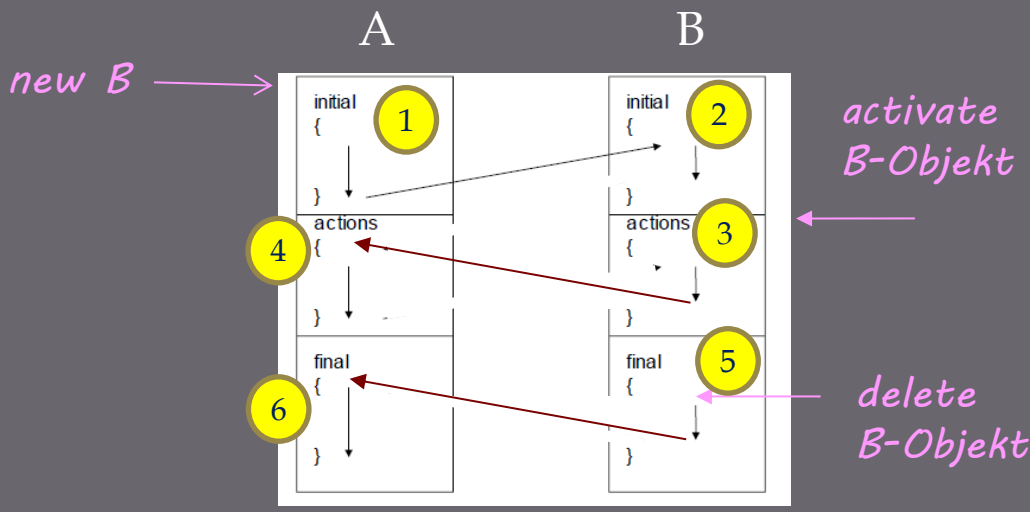
Benutzung der Version 2.0 muss explizit eingeschaltet werden:

#define SLX2 ON

Vererbung



Aktionsreihenfolge für Objekte abgeleiteter Klassen



Regel: einmal overridable,
danach immer overridable

Beispiel: Kontrollfluss

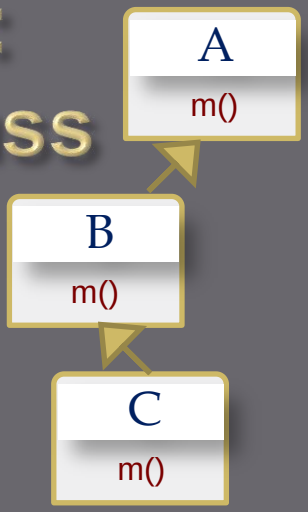

```
//*****************************************************************************
//      Module Subclass
//*****************************************************************************
module basic {
#define SI[illegible]ON

class A (in int i){
        int a;
        overridable int id;
        initial {
                id= i;
                print (id)  "A-initial, id= _ \n";
        }
        overridable method m() {
                 print "A-m-body \n";
        }
        actions  {
                print "A-actions \n";
        }
        final {
                print  "A-final \n";
        }
        static procedure p() {
                id= 100;
        }
};

class B (in int j) subclass(A(2*j)) {
        int b;
        override boolean id;
        initial {
                id= TRUE;
                print (id) "B-initial id=_ \n";
        }
        override method m() {;
                 print "B-m-body \n";
                A::m();
        };
        actions  {
                print "B-actions \n";
                //terminate;
        }

        final {
                print "B-final \n";
        }
}

class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print "C-actions \n";
                //terminate;
        }

        final {
                print "C-final \n";
        }
}

procedure main() {
        pointer (C) ptrC;
        pointer (A) ptrA;
        ptrA= new C(1);

        ptrA->p();
        activate ptrA;
        advance 1.0;
        ptrA->m();

        ptrA=new A(2);
        activate ptrA;
        advance 1.0;

        ptrA= new C(2);
        ptrA->a= 1;
        //ptrA->b= 2;
        //** Semantic error: "b" is not a member of class "A"
       (pointer (C)) ptrA->c= 3;

        ptrC= ptrA;
        ptrC->a= 100;
        ptrC->c= 300;

        destroy ptrC;
        ptrA= new A(3);
        //ptrC= ptrA;
        //** Exceution error at time 2: "ptrA is a pointer(A). A pointer(C) isrequired here

}
}
```

5 6 3 4 7 1 2 8

Execution begins
A-initial id= 4
B-initial id= TRUE
C-initial id= FALSE

Puck main 1/1

Beispiel: Kontrollfluss




```
//*****************************************************************************
//      Module Subclass
//*****************************************************************************
module basic {
#define SLX2 ON

class A (in int i){
        int a;
        overridable int id;
        initial {
                id= i;
                print (id)  "A-initial, id= _ \n";
        }
        overridable method m() {
                 print "A-m-body \n";
        }
        actions  {
                print "A-actions \n";
        }
        final {
                print  "A-final \n";
        }
        static procedure p() {
                id= 100;
        }
};
```

3

```
class B (in int j) subclass(A(2*j)) {
        int b;
        override boolean id;
        initial {
                id= TRUE;
                print (id) "B-initial id=_ \n";
        }
        override method m() {;
                 print "B-m-body \n";
                A::m();
        };
        actions  {
                print "B-actions \n";
                //terminate;
        }

        final {
                print "B-final \n";
        }
}
```

2

```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print "C-actions \n";
                //terminate;
        }

                print "C-final \n";
        }
}
```

1

```
procedure main() {
        pointer (C) ptrC;
        pointer (A) ptrA;
        ptrA= new C(1);

        ptrA->p();
        activate ptrA;
        advance 1.0;
        ptrA->m();

        ptrA=new A(2);
        activate ptrA;
        advance 1.0;

        ptrA= new C(2);
        ptrA->a= 1;
        //ptrA->b= 2;
        //** Semantic error: "b" is not a member of class "A"
       (pointer (C)) ptrA->c= 3;

        ptrC= ptrA;
        ptrC->a= 100;
        ptrC->c= 300;

        destroy ptrC;
        ptrA= new A(3);
        //ptrC= ptrA;
        //** Exceution error at time 2: "ptrA is a pointer(A). A pointer(C) isrequired here

}
}
```

Puck C 1/1

…
C-actions
B-actions
A-actions
…

Puck C 1/1

Puck C 1/2

Puck C 1/3

ACHTUNG

weitere Situation, die zur impliziten Generierung von Pucks führt

1. **bislang bekannt**
   - main-Start
   - **activate** *classId-objectNr-1*
   - **fork** *classId-objectNr-n*

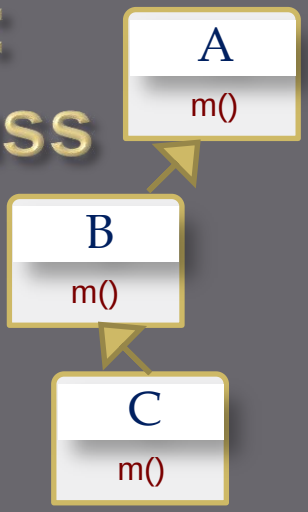
2. **hinzu kommt**
   - bei Ausführung einer Action-Property wird zu Beginn ein Child-Puck für die Action-Property der (falls vorhanden) nächsten Basisklasse generiert

Bem.-1: u.U. muss bei Bedarf der nutzerdefinierte Puck-Pointer „my_pucK“ zu Beginn jeder Action-Property einer Vererbungskette neu gesetzt werden:

my_puck= **ACTIVE**;

Bem.-2: Puckfreigabe erfolgt bei Beendigung der zu steuernden Action-Property (Bedingung: kein nutzerdefinierter Zeiger zeigt noch auf den Puck)

Beispiel: Kontrollfluss




```
//*****************************************************************************
//      Module Subclass
//*****************************************************************************
module basic {
#define SLX2 ON

class A (in int i){
        int a;
        overridable int id;
        initial {
                id= i;
                print (id)  "A-initial, id= _ \n";
        }
        overridable method m() {
                print "A-m-body \n";
        }
        actions  {
                print "A-actions \n";
        }
        final {
                print  "A-final \n";
        }
        static procedure p() {
                id= 100;
        }
```

3

```
class B (in int j) subclass(A(2*j)) {
        int b;
        override boolean id;
        initial {
                id= TRUE;
                print (id) "B-initial id=_ \n";
        }
        override method m() {;
                print "B-m-body \n";
                A::m();
        };
        actions  {
                print "B-actions \n";
                //terminate;
        }

        final {
                print "B-final \n";
        }
}
```

```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                print "C-m-body \n";
                A::m();
        };
        actions  {
                print "C-actions \n";
                //terminate;
        }

        final {
                print "C-final \n";
        }
}
```

1

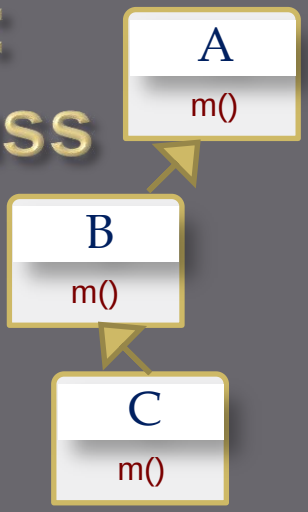
2

```
procedure main() {
        pointer (C) ptrC;
        pointer (A) ptrA;
        ptrA= new C(1);

        ptrA->p();
        activate ptrA;
        advance 1.0;
        ptrA->m();

        ptrA=new A(2);
        activate ptrA;
        advance 1.0;

        ptrA= new C(2);
        ptrA->a= 1;
        //ptrA->b= 2;
        //** Semantic error: "b" is not a member of class "A"
       (pointer (C)) ptrA->c= 3;

        ptrC= ptrA;
        ptrC->a= 100;
        ptrC->c= 300;

        destroy ptrC;
        ptrA= new A(3);
        //ptrC= ptrA;
        //** Exceution error at time 2: "ptrA is a pointer(A). A pointer(C) isrequired here

}
}
```

Puck C 1/1

```
...
C-m-body
A-m-body
...
```

Beispiel: Kontrollfluss




```
//*****************************************************************************
//      Module Subclass
//*****************************************************************************
module basic {
#define SLX2 ON

class A (in int i){
        int a;
        overridable int id;
        initial {
                id= i;
                print (id)  "A-initial, id= _ \n";
        }
        overridable method m() {
                 print "A-m-body \n";
        }
        actions  {
                print "A-actions \n";
        }
        final {
                print  "A-final \n";
        }
        static procedure p() {
                id= 100;
        }
};
```

1 2 5

```
class B (in int j) subclass(A(2*j)) {
        int b;
        override boolean id;
        initial {
                id= TRUE;
                print (id) "B-initial id=_ \n";
        }
        override method m() {;
                 print "B-m-body \n";
                A::m();
        };
        actions  {
                print "B-actions \n";
                //terminate;
        }

        final {
                print "B-final \n";
        }
}
```

4

```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print "C-actions \n";
                //terminate;
        }

        final {
                print "C-final \n";
        }
}
```

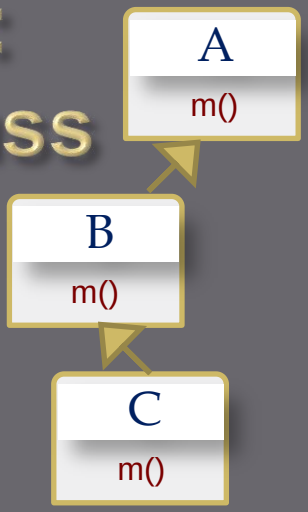
3

```
procedure main() {
        pointer (C) ptrC;
        pointer (A) ptrA;
        ptrA= new C(1);

        ptrA->p();
        activate ptrA;
        advance 1.0;
        ptrA->m();

        ptrA=new A(2);
        activate ptrA;
        advance 1.0;

        ptrA= new C(2);
        ptrA->a= 1;
        //ptrA->b= 2;
        //** Semantic error: "b" is not a member of class "A"
       (pointer (C)) ptrA->c= 3;

        ptrC= ptrA;
        ptrC->a= 100;
        ptrC->c= 300;

        destroy ptrC;
        ptrA= new A(3);
        //ptrC= ptrA;
        //** Exceution error at time 2: "ptrA is a po
}
```

Puck main 1/1

```
…
A-initial, id=     2
C-final
B-final
A-final

…
```

Beispiel: Kontrollfluss




```
//*****************************************************************************
//      Module Subclass
//*****************************************************************************
module basic {
#define SLX2 ON

class A (in int i){
        int a;
        overridable int id;
        initial {
                id= i;
                print (id)  "A-initial, id= _ \n";
        }
        overridable method m() {
                 print "A-m-body \n";
        }
        actions  {
                print "A-actions \n";
        }
        final {
                print  "A-final \n";
        }
        static procedure p() {
                id= 100;
        }
};
```

1 2 5

```
class B (in int j) subclass(A(2*j)) {
        int b;
        override boolean id;
        initial {
                id= TRUE;
                print (id) "B-initial id=_ \n";
        }
        override method m() {;
                 print "B-m-body \n";
                A::m();
        };
        actions  {
                print "B-actions \n";
                //terminate;
        }

        final {
                print "B-final \n";
        }
}
```

4

```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print "C-actions \n";
                //terminate;
        }

        final {
                print "C-final \n";
        }
}
```

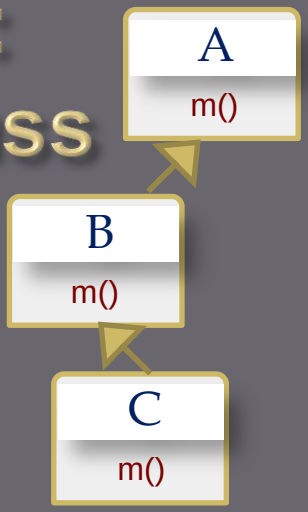
3

```
procedure main() {
        pointer (C) ptrC;
        pointer (A) ptrA;
        ptrA= new C(1);

        ptrA->p();
        activate ptrA;
        advance 1.0;
        ptrA->m();

        ptrA=new A(2);
        activate ptrA;
        advance 1.0;

        ptrA= new C(2);
        ptrA->a= 1;
        //ptrA->b= 2;
        //** Semantic error: "b" is not a member of class "A"
       (pointer (C)) ptrA->c= 3;

        ptrC= ptrA;
        ptrC->a= 100;
        ptrC->c= 300;

        destroy ptrC;
        ptrA= new A(3);
        //ptrC= ptrA;
        //** Exceution error at time 2: "ptrA is a pointer(A). A pointer(C) isrequired here

}
}
```

Puck A 1/1

…
A-actions
…

Beispiel: Kontrollfluss




```
//*****************************************************************************
//      Module Subclass
//*****************************************************************************
module basic {
#define SLX2 ON

class A (in int i){
        int a;
        overridable int id;
        initial {
                id= i;
                print (id)  "A-initial, id= _ \n";
        }
        overridable method m() {
                 print "A-m-body \n";
        }
        actions  {
                print "A-actions \n";
        }
        final {
                print  "A-final \n";
        }
        static procedure p() {
                id= 100;
        }
};
```

```
class B (in int j) subclass(A(2*j)) {
        int b;
        override boolean id;
        initial {
                id= TRUE;
                print (id) "B-initial id=_ \n";
        }
        override method m() {;
                 print "B-m-body \n";
                A::m();
        };
        actions  {
                print "B-actions \n";
                //terminate;
        }

        final {
                print "B-final \n";
        }
}
```

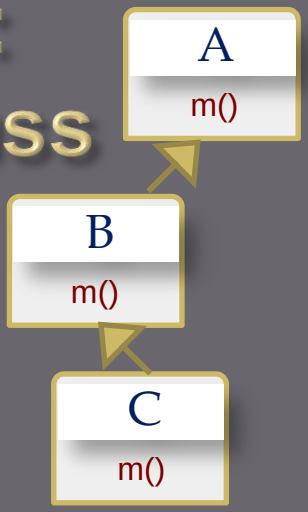
```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print "C-actions \n";
                //terminate;
        }

        final {
                print "C-final \n";
        }
}
```

```
procedure main() {
        pointer (C) ptrC;
        pointer (A) ptrA;
        ptrA= new C(1);

        ptrA->p();
        activate ptrA;
        advance 1.0;
        ptrA->m();

        ptrA=new A(2);
        activate ptrA;
        advance 1.0;

        ptrA= new C(2);
        ptrA->a= 1;
        //ptrA->b= 2;
        //** Semantic error: "b" is not a member of class "A"
       (pointer (C)) ptrA->c= 3;

        ptrC= ptrA;
        ptrC->a= 100;
        ptrC->c= 300;

        destroy ptrC;
        ptrA= new A(3);
        //ptrC= ptrA;
        //** Exceution error at time 2: "ptrA is a pointer(A). A pointer(C) isrequired here
}
```

Puck main 1/1

```
…
A-initial id=     8
B-initial id=     8
C-initial id=     8
 …
```

Beispiel: Kontrollfluss




```
//*****************************************************************************
//      Module Subclass
//*****************************************************************************
module basic {
#define SLX2 ON

class A (in int i){
        int a;
        overridable int id;
        initial {
                id= i;
                print (id)  "A-initial, id= _ \n";
        }
        overridable method m() {
                 print "A-m-body \n";
        }
        actions  {
                print "A-actions \n";
        }
        final {
                print  "A-final \n";
        }
        static procedure p() {
                id= 100;
        }
};
```

```
class B (in int j) subclass(A(2*j)) {
        int b;
        override boolean id;
        initial {
                id= TRUE;
                print (id) "B-initial id=_ \n";
        }
        override method m() {;
                 print "B-m-body \n";
                A::m();
        };
        actions  {
                print "B-actions \n";
                //terminate;
        }

        final {
                print "B-final \n";
        }
}
```

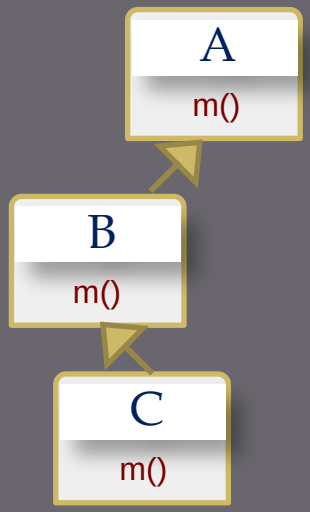
```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print "C-actions \n";
                //terminate;
        }

        final {
                print "C-final \n";
        }
}
```

```
procedure main() {
        pointer (C) ptrC;
        pointer (A) ptrA;
        ptrA= new C(1);

        ptrA->p();
        activate ptrA;
        advance 1.0;
        ptrA->m();

        ptrA=new A(2);
        activate ptrA;
        advance 1.0;

        ptrA= new C(2);
        ptrA->a= 1;
        //ptrA->b= 2;
        //** Semantic error: "b" is not a member of class "A"
       (pointer (C)) ptrA->c= 3;

        ptrC= ptrA;
        ptrC->a= 100;
        ptrC->c= 300;

        destroy ptrC;
        ptrA= new A(3);
        //ptrC= ptrA;
        //** Exceution error at time 2: "ptrA is a pointer(A). A pointer(C) isrequired here
}
```

Puck main 1/1

…
Freigabe vom A-Objekt
…Korrekte und fehlerhafte Attribut-Zugriffe

Cast-Operator



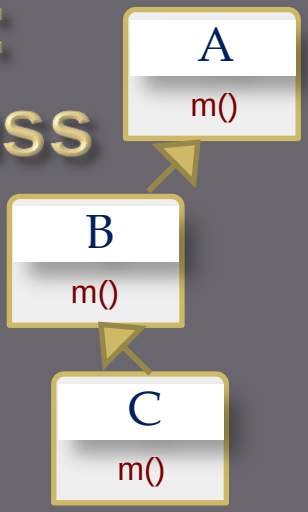
ptrC= (**pointer** (C)) ptraA;

```
procedure main() {
        pointer (C) ptrC;
        pointer (A) ptrA;
        ptrA= new C(1);

        ptrA->p();
        activate ptrA;
        advance 1.0;
        ptrA->m();

        ptrA=new A(2);
        activate ptrA;
        advance 1.0;

        ptrA= new C(2);
        ptrA->a= 1;
        //ptrA->b= 2;
        //** Semantic error: "b" is not a member of class "A"
        (pointer (C)) ptrA->c= 3;

        ptrC= ptrA;
        ptrC->a= 100;
        ptrC->c= 300;

        destroy ptrC;
        ptrA= new A(3);
        //ptrC= ptrA;
        //** Exceution error at time 2: "ptrA is a pointer(A). A pointer(C) isrequired here

}
}
```

Cast-Operator

- erforderlich, falls per „ptrA“ auf echte Attribute/Methoden von B und C zugegriffen werden soll

Beispiel: Kontrollfluss




```
//*****************************************************************************
//      Module Subclass
//*****************************************************************************
module basic {
#define SLX2 ON

class A (in int i){
        int a;
        overridable int id;
        initial {
                id= i;
                print (id)  "A-initial, id= _ \n";
        }
        overridable method m() {
                 print "A-m-body \n";
        }
        actions  {
                print "A-actions \n";
        }
        final {
                print  "A-final \n";
        }
        static procedure p() {
                id= 100;
        }
};
```

```
class B (in int j) subclass(A(2*j)) {
        int b;
        override boolean id;
        initial {
                id= TRUE;
                print (id) "B-initial id=_ \n";
        }
        override method m() {;
                 print "B-m-body \n";
                A::m();
        };
        actions  {
                print "B-actions \n";
                //terminate;
        }

        final {
                print "B-final \n";
        }
}
```

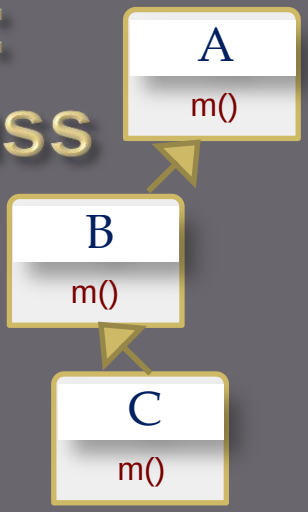
```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print "C-actions \n";
                //terminate;
        }

        final {
                print "C-final \n";
        }
}
```

```
procedure main() {
        pointer (C) ptrC;
        pointer (A) ptrA;
        ptrA= new C(1);

        ptrA->p();
        activate ptrA;
        advance 1.0;
        ptrA->m();

        ptrA=new A(2);
        activate ptrA;
        advance 1.0;

        ptrA= new C(2);
        ptrA->a= 1;
        //ptrA->b= 2;
        //** Semantic error: "b" is not a member of class "A"
       (pointer (C)) ptrA->c= 3;

        ptrC= ptrA;
        ptrC->a= 100;
        ptrC->c= 300;

        destroy ptrC;
        ptrA= new A(3);
        //ptrC= ptrA;
        //** Exceution error at time 2: "ptrA is a pointer(A). A pointer(C) isrequired here

}
}
```

Puck main 1/1

Kopie des Zeigerwertes
für einen Zeiger, der statisch mit der Klasse C qualifiziert ist

Beispiel: Kontrollfluss




```
//*****************************************************************************
//      Module Subclass
//*****************************************************************************
module basic {
#define SLX2 ON

class A (in int i){
        int a;
        overridable int id;
        initial {
                id= i;
                print (id)  "A-initial, id= _ \n";
        }
        overridable method m() {
                 print "A-m-body \n";
        }
        actions  {
                print "A-actions \n";
        }
        final {
                print  "A-final \n";
        }
        static procedure p() {
                id= 100;
        }
};
```

```
class B (in int j) subclass(A(2*j)) {
        int b;
        override boolean id;
        initial {
                id= TRUE;
                print (id) "B-initial id=_ \n";
        }
        override method m() {;
                 print "B-m-body \n";
                A::m();
        };
        actions  {
                print "B-actions \n";
                //terminate;
        }

        final {
                print "B-final \n";
        }
}
```

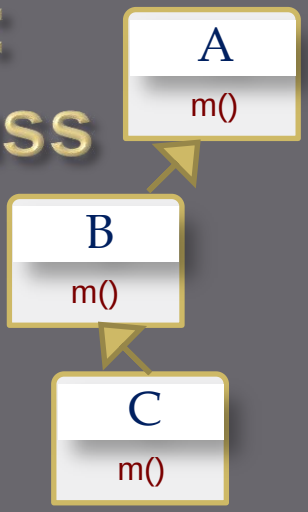
```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print "C-actions \n";
                //terminate;
        }

        final {
                print "C-final \n";
        }
}
```

```
procedure main() {
        pointer (C) ptrC;
        pointer (A) ptrA;
        ptrA= new C(1);

        ptrA->p();
        activate ptrA;
        advance 1.0;
        ptrA->m();

        ptrA=new A(2);
        activate ptrA;
        advance 1.0;

        ptrA= new C(2);
        ptrA->a= 1;
        //ptrA->b= 2;
        //** Semantic error: "b" is not a member of class "A"
       (pointer (C)) ptrA->c= 3;

        ptrC= ptrA;
        ptrC->a= 100;
        ptrC->c= 300;

        destroy ptrC;
        ptrA= new A(3);
        //ptrC= ptrA;
        //** Exceution error at time 2: "ptrA is a pointer(A). A pointer(C) isrequired here

}
}
```

Puck main 1/1

Unproblematische Attribut-Zugriffe

Beispiel: Kontrollfluss




```
//*****************************************************************************
//      Module Subclass
//*****************************************************************************
module basic {
#define SLX2 ON

class A (in int i){
        int a;
        overridable int id;
        initial {
                id= i;
                print (id)  "A-initial, id= _ \n";
        }
        overridable method m() {
                 print "A-m-body \n";
        }
        actions  {
                print "A-actions \n";
        }
        final {
                print  "A-final \n";
        }
        static procedure p() {
                id= 100;
        }
};
```

```
class B (in int j) subclass(A(2*j)) {
        int b;
        override boolean id;
        initial {
                id= TRUE;
                print (id) "B-initial id=_ \n";
        }
        override method m() {;
                 print "B-m-body \n";
                A::m();
        };
        actions  {
                print "B-actions \n";
                //terminate;
        }

        final {
                print "B-final \n";
        }
}
```

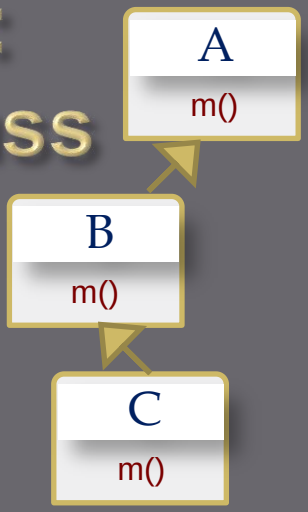
```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print "C-actions \n";
                //terminate;
        }

        final {
                print "C-final \n";
        }
}
```

```
procedure main() {
        pointer (C) ptrC;
        pointer (A) ptrA;
        ptrA= new C(1);

        ptrA->p();
        activate ptrA;
        advance 1.0;
        ptrA->m();

        ptrA=new A(2);
        activate ptrA;
        advance 1.0;

        ptrA= new C(2);
        ptrA->a= 1;
        //ptrA->b= 2;
        //** Semantic error: "b" is not a member of class "A"
       (pointer (C)) ptrA->c= 3;

        ptrC= ptrA;
        ptrC->a= 100;
        ptrC->c= 300;

        destroy ptrC;
        ptrA= new A(3);
        //ptrC= ptrA;
        //** Exceution error at time 2: "ptrA is a po                    ) isrequired here
}
}
```

Puck main 1/1

…
C-final
B-final
A-final
…

Beispiel: Kontrollfluss




```
//*****************************************************************************
//      Module Subclass
//*****************************************************************************
module basic {
#define SLX2 ON

class A (in int i){
        int a;
        overridable int id;
        initial {
                id= i;
                print (id)  "A-initial, id= _ \n";
        }
        overridable method m() {
                 print "A-m-body \n";
        }
        actions  {
                print "A-actions \n";
        }
        final {
                print  "A-final \n";
        }
        static procedure p() {
                id= 100;
        }
};
```

```
class B (in int j) subclass(A(2*j)) {
        int b;
        override boolean id;
        initial {
                id= TRUE;
                print (id) "B-initial id=_ \n";
        }
        override method m() {;
                 print "B-m-body \n";
                A::m();
        };
        actions  {
                print "B-actions \n";
                //terminate;
        }

        final {
                print "B-final \n";
        }
}
```

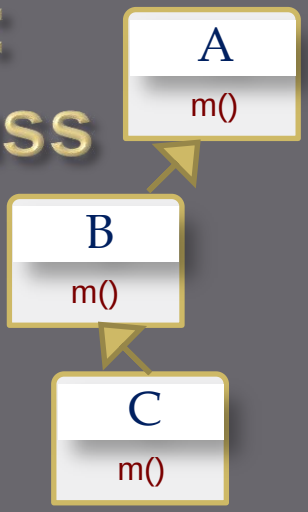
```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print "C-actions \n";
                //terminate;
        }

        final {
                print "C-final \n";
        }
}
```

```
procedure main() {
        pointer (C) ptrC;
        pointer (A) ptrA;
        ptrA= new C(1);

        ptrA->p();
        activate ptrA;
        advance 1.0;
        ptrA->m();

        ptrA=new A(2);
        activate ptrA;
        advance 1.0;

        ptrA= new C(2);
        ptrA->a= 1;
        //ptrA->b= 2;
        //** Semantic error: "b" is not a member of class "A"
       (pointer (C)) ptrA->c= 3;

        ptrC= ptrA;
        ptrC->a= 100;
        ptrC->c= 300;

        destroy ptrC;
        ptrA= new A(3);
        //ptrC= ptrA;
        //** Exceution error at time 2: "ptrA is a pointer(A). A pointer(C) isrequired here
}
}
```

Puck main 1/1

…
A-initial id= 3

Beispiel: Kontrollfluss




```
//*****************************************************************************
//      Module Subclass
//*****************************************************************************
module basic {
#define SLX2 ON

class A (in int i){
        int a;
        overridable int id;
        initial {
                id= i;
                print (id)  "A-initial, id= _ \n";
        }
        overridable method m() {
                 print "A-m-body \n";
        }
        actions  {
                print "A-actions \n";
        }
        final {
                print  "A-final \n";
        }
        static procedure p() {
                id= 100;
        }
};
```

```
class B (in int j) subclass(A(2*j)) {
        int b;
        override boolean id;
        initial {
                id= TRUE;
                print (id) "B-initial id=_ \n";
        }
        override method m() {;
                 print "B-m-body \n";
                A::m();
        };
        actions  {
                print "B-actions \n";
                //terminate;
        }

        final {
                print "B-final \n";
        }
}
```

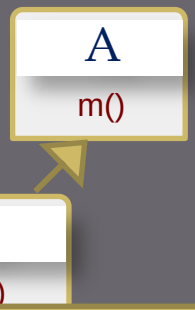
```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print "C-actions \n";
                //terminate;
        }

        final {
                print "C-final \n";
        }
}
```

```
procedure main() {
        pointer (C) ptrC;
        pointer (A) ptrA;
        ptrA= new C(1);

        ptrA->p();
        activate ptrA;
        advance 1.0;
        ptrA->m();

        ptrA=new A(2);
        activate ptrA;
        advance 1.0;

        ptrA= new C(2);
        ptrA->a= 1;
        //ptrA->b= 2;
        //** Semantic error: "b" is not a member of class "A"
       (pointer (C)) ptrA->c= 3;

        ptrC= ptrA;
        ptrC->a= 100;
        ptrC->c= 300;

        destroy ptrC;
        ptrA= new A(3);
        //ptrC= ptrA;
        //** Exceution error at time 2: "ptrA is a pointer(A). A pointer(C) isrequired here
}
```

Puck main 1/1

Laufzeitfehler

Beispiel: Kontrollfluss




```
//*****************************************************************************
//      Module Subclass
//*****************************************************************************
module basic {
#define SLX2 ON

class A (in int i){
        int a;
        overridable int id;
        initial {
                id= i;
                print (id)  "A-initial, id= _ \n";
        }
        overridable method r
                print "A-m-bod
        }
        actions  {
                print "A-actions
        }
        final {
                print  "A-final \n
        }
        static procedure p() {
                id= 100;
        }
};
```

```
class B (in int j) subclass(A(2*j)) {
```

```
procedure main() {
        pointer (C) ptrC;
        pointer (A) ptrA;
        ptrA= new C(1);

        ptrA->p();
        activate ptrA;
        advance 1.0;
        ptrA->m();

        ptrA=new A(2);
        activate ptrA;
        advance 1.0;

        ptrA= new C(2);
        ptrA->a= 1;
        //ptrA->b= 2;
        //** Semantic error: "b" is not a member of
       (pointer (C)) ptrA->c= 3;

        ptrC= ptrA;
        ptrC->a= 100;
        ptrC->c= 300;

        destroy ptrC;
        ptrA= new A(3);
        //ptrC= ptrA;
        //** Exceution error at time 2: "ptrA is a pointer(A). A pointer(C) isrequired here
}
}
```

Puck main 1/1

```
Subclass.rtf: SLX-64 UL211  Lines: 1,841  Errors: 0  Warnings: 0  Lines/Second: 306,133  Memory: 2 MB
Execution begins
A-initial, id= 4
B-initial id=TRUE
C-initial id=FALSE
C-actions
B-actions
A-actions
C-m-body
A-m-body
A-initial, id= 2
C-final
B-final
A-final
A-actions
A-initial, id= 8
B-initial id=TRUE
C-initial id=FALSE
A-final
A-initial, id= 3
C-final
B-final
A-final
Execution complete
Objects created: 7 passive and 5 active  Pucks created: 6  Memory: 2 MB  Time: 0.06 seconds
```

```
                print "C-final \n";
        }
}
```

Statischer und dynamischer Typtest

```
class A {
        int i;
}

class B subclass(A) {
        int j;
}

procedure main() {
        pointer(A) a = new B();

        set(A) as;
        place new B() into as;
}
```

polymorphe Liste
- kann sowohl A-Objekte
- als auch Objekte von direkten oder indirekten Ableitungen enthalten

```
class A {
}

class B (int l) subclass(A) {
        int k = l;
}
procedure main() {
        pointer(A) a = new B(1);
        pointer(B) b = a; // OK: checked at run-time
        pointer(B) bb = new A();  // ERROR
}
```

Cast-Operator
- erforderlich, falls per „a“ auf echte Attribute von B zugegriffen werden soll
- nicht notwendig, falls per „b“ auf Attribute von A und B zugegriffen werden soll

Prozedur = Methode

```
class C {
    method m() {
        …
    }
}
```

Klassenspezifische (nutzereigene) Prozeduren,
die über Objekte der Klasse operieren
(impliziter Vermittlung des **ME**-Operators)

Vererbung und (virtuelle) Methoden

- Klassen in SLX 1.x besitzen keine Methoden,
  nur Properties (als vordefinierte parameterlose Methoden)

- Klassen in SLX 2.x besitzen darüber hinaus nutzerdefinierte Methoden,

```
class C {
    method m() {
        …
    }
}
```

Klassenspezifische (nutzereigene) Prozeduren,
die über Objekte der Klasse operieren
(impliziter Vermittlung des ME-Operators)

- diese können virtuell sein und in Ableitungen redefiniert werden

```
class vehicle {
        overridable method GetVehiclePosition(out double x, out double y) {…}
};

class tractor subclass(vehicle) {
        override method GetVehiclePosition(out double x, out double y) {…};
}
```

identische Signatur
ist zwingend

- Regel: einmal virtuell immer virtuell