VORLESUNG

Automatisierung industrieller Workflows

Teil D: Die Sprache SLX

- ODEMx- Elemente -

Joachim Fischer

Inhalt C.4

- **Teil A** Aspekte dynamis
- **Teil B** Die Moc
- **Teil C** Die ausf SLX
- **Teil D** Modellie

- **C.1** Einführung und Ba
- **C.2** Stochastische Pro
- **C.3** GPSS-Elemente
- **C.4** ODEMx-Elemente
- **C.5** Obektorientierung
- **C.6** DISCO-Elemente

1. Einführung: C++-Bibliothek ODEM, ODEMx
2. Master-Slave-Synchronisation
   - Allgemeines Prinzip der Master-Slave-Synchronisation
   - Umsetzung in SLX
   - Beispiele: Fähre, Ölhafen
   - Diskussion Master-Slave-Implementierung in SLX
   - Unterbrechbarkeit des Wartens auf Master- bzw. Slave-Verfügbarkeit und der Master-Slave-Kooperation
   - Beispiel: Bagger mit Beladung von Fahrzeugen
   - Abschließende Betrachtung
3. Asynchrone Kommunikation, Behandlung spezieller Zustandsereignisse
4. Behandlung allgemeiner Zustandsereignisse

Beispiel: Transportfahrzeuge – Bagger – Beladung

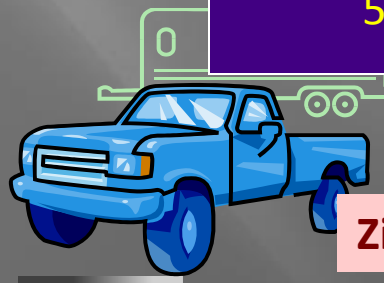
große Lastwagen
20 t

alle 3 min

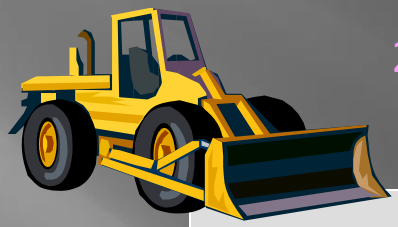
Ankunftsraten

alle 22 min

kleine Lastwagen
5 t



Ziel: 8h -Simulation



240 t/h (4t/min)

60 t/h (1t/min)

Beladungsraten

kleinerBagger

großerBagger-1

großerBagger-2

Beladung

- große Lastwagen bevorzugt von großen Baggern
- kleine Lastwagen **nur** vom kleinen Bagger
- sofortige Unterbrechung der Beladung eines großen Lastwagens durch kleinen Bagger, sobald kleiner Lastwagen eintrifft oder großer Bagger frei wird
- priorisierte Beladung unterbrochener Fahrzeuge

Systemanalyse

Experiment: 2 große und 1 kleiner Bagger

Erweiterte Standard-Report-Ausgabe

max. wartende Fahrzeuge

Queue	Current Contents	Maximum Contents	Contents	Entries	Entries	Zeros	Average Time/Item	Average > 0 Time/Item
SDchain_q	0	1	0.27	639	1	0.16	4.279	4.286

CoopQ Master/Slave	Current Contents	Maximum Contents	Average Contents	Total Entries	Zero Entries	Percent Zeros	Average Time/Item	Average > 0 Time/Item
largeTruck_L	0	2	0.16	2946	2637	89.51	0.537	5.117
largeTruck_L	0	19	3.81	2946	685	23.25	13.028	16.975
smallTruck_L	0	1	0.00	663	663	100.00	0.000	16.975
smallTruck_L	0	4	0.08	663	434	65.46	1.208	3.496

MasterQ

SlaveQ

Digger	Complete SmallTruck Loadings	Complete LargeTruck Loadings	InComplete LargeTruck Loadings	Continued LargeTruck Loadings	Current Work Status
Large_Digger 1	NO	1239	NO	220	LARGE
Large_Digger 2	NO	1238	NO	211	LARGE
Small_Digger 1	662	36	431	NO	SMALL

Summe

Truck Type	Generated Trucks	Loaded Trucks	Exceptional Loaded Trucks	Interrupted Loading of Trucks	Interrupts by L-Digger	Interrupts by S-Truck	Current Waiting Trucks	Current Loading Trucks
smallTruck	663	662	NO	NO	NO	NO	0	1
largeTruck	2515	2513	467	431	169	262	0	2

Execution complete

Objects created: 3,242 passive and 4 active Pucks created: 3,156 Memory: 6 MB Time: 0.21 Seconds

Summe

Anwendungsspezifische Ausgabe

Ausstehende Re-Implementation von CoopQ (Ideensammlung)

- **h7-konforme-Report-Funktionalität** für CoopQ-Objekte
  bei Sicherung, dass lokale Queue-Objekte nur 1-mal im Report erscheinen (nicht nochmal bei Darstellung sämtlicher Queue-Objekte)

  remove &masterQ **from** queue_set;

- **Verzicht** auf **Super-Klasse Actor**, Operation über rufenden Puck,
  Dazu müsste die Klasse Puck um Unterbrechungsflags von Actor erweitert werden

  augment class puck { … } ;

- **coopt**, und **find** könnten zu einer Operation zusammengelegt werden:

  SLX-Erweiterung erlaubt Anweisung mit optionalen Parametern

- ODEMx bietet weitere nützliche Operationen für Master-Slave an:
  - Master kann **Verfügbarkeit** von Slaves bzw. passenden Slaves prüfen, ohne zu blockieren
  - Slaves kann bei der **Reaktivierung schlafender Master**, eine Reihenfolge der Master beachten, die sich durch dynamische veränderbare Gewichte (Prioritätsänderungen) ergeben.

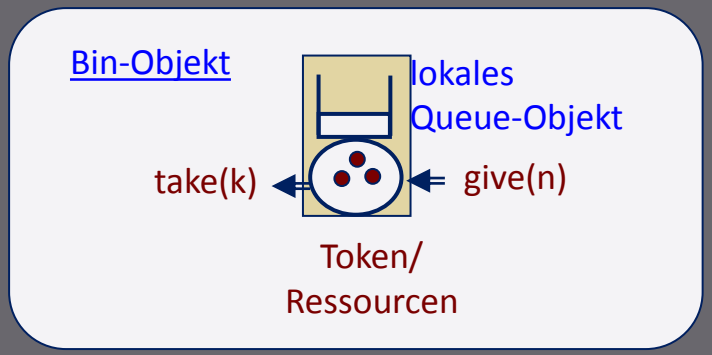
Inhalt C.4

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  ODEMx-Elemente
- **C.5**
  Obektorientierung
- **C.6**
  DISCO-Elemente

1. Einführung: C++-Bibliothek ODEM, ODEMx
2. Master-Slave-Synchronisation
3. Asynchrone Kommunikation, Behandlung spezieller Zustandsereignisse
4. Behandlung allgemeiner Zustandsereignisse

ODEMx-Klasse Bin



Puffer mit Obergrenze
MaxInt (ohne Test)

Nachrichten
(ohne Info,
ohne Identität)
~ **Token**

Token-Pool
~ Tokenzähler

Aufrufer von **take** kann blockieren
mit Erfassung im **queue**-Objekt(h7)
bei Erfolg: Rückgabe der geforderten Tokenanzahl k

Aufrufer von **take** kann während des Wartens per
wait_interrupt
unterbrochen werden

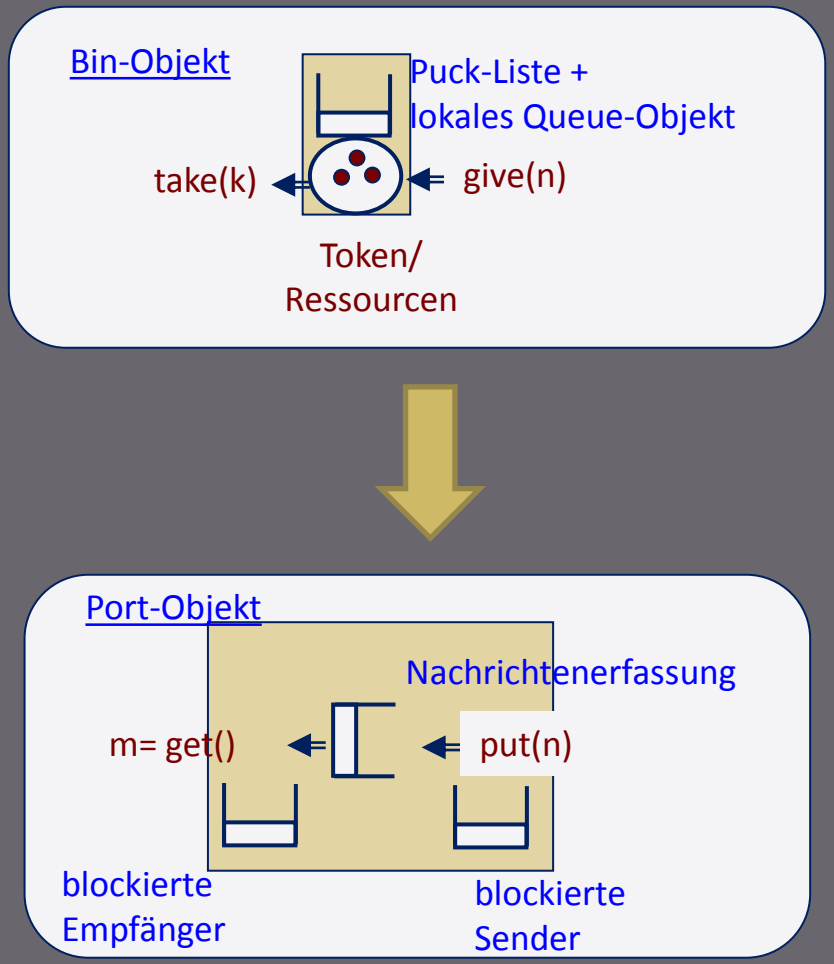
Aufrufer von **give** blockiert nie,
reaktiviert wartende Prozesse

Bin mit Tokenkapazität =1
~ Logic Switch vom h7-Modul

Implementierungsskizze

```
passive class Bin (int token_num) {
        int tokens;
        pointer (puck) waiting_pucks;
        queue statisticQ;

        initial { tokens = token_num; }

        method take (int token_num ) returning int {
                enqueue statisticQ;
                while( token_num > tokens  ) {
                        wait list= waiting_pucks;
                        if( ACTIVE->state == INTERRUPTED {
                                depart statisticQ;
                                return 0;
                        }
                }
                tokens -= token_num;
                if (waiting_pucks !=NULL) reactivate waiting_pucks;
                depart statisticQ;
                return token_num;
        }

        method give (int token_num) {
                tokens += token_num;
                if (waiting_pucks !=NULL) reactivate list= waiting_pucks;
        }
        method avail_tokens returning int {
                return tokens;
        }
} // class Bin
```

ODEMx-Klasse Port



Port als BIN-Erweiterung

(1) bekommt Kapazitätsgrenze
(2) damit kann give ebenfalls blockieren
(3) dafür braucht man weitere Puck-Liste u. ein Queue-Objekt
(4) Token werden zu Nachrichten (abstrakte Klasse)
(5) Token-Pool wird zu einer polymorphen Liste von Objekten von Nachrichten-Ableitungen (Nachrichten-Liste und ein weiteres Queue-Objekt
(6) Operationen take und give können immer nur ein Nachrichten-Objekt bearbeiten werden umbenannt in put und get
(7) Port bekommt zur Abfrage des Zustandes (realisiert als Control-Attribute von Port) zwei Makros FULL, EMPTY
(8) Rückgabewert signalisiert evtl. Unterbrechung

abstract passive class Message {}

Nachrichten ~ Objekte Message-Ableitungen

Für ODEMx-Kenner

Problem: Nachbildung der globalen ODEMx-wait-Funktion

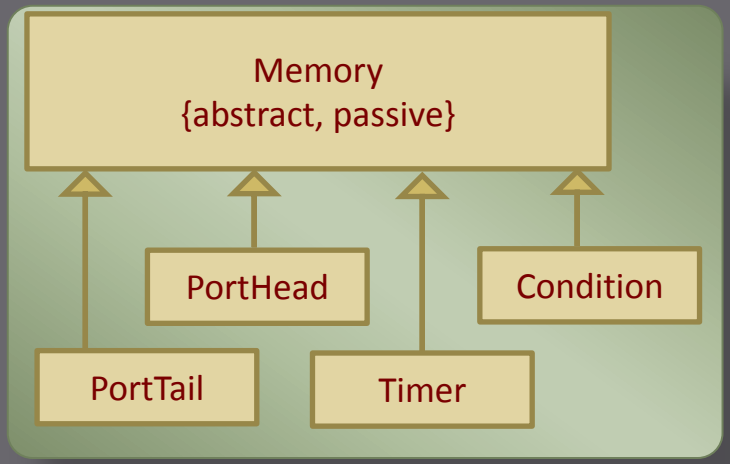
Lösung: Bereitstellung der Anweisung **wait_for_availability** m **atleast** Memory **as** p1, p2, t, c;

Fortsetzung mit Rückgabe von m:

mit m als p1, wenn sich Port p1 befüllen läßt oder
als p2, wenn Port p2 nicht leer ist oder
als t, wenn Timer t nicht abgelaufen ist oder
als c, wenn die Bedingung c erfüllt ist
sonst blockiert der Aufrufer und wird bei Änderung der Bedingungskonstellation reaktiviert

variable Anzahl von Parametern

m ist ein polymorher Zeiger
pointer(Memory) m
p1, p2, t, c sind Zeiger auf Objekte von Memory-Ableitungen



Fortsetzung nach Ruf von wait_for_availabilty: erfolgt i.Abh. des Typs von m

Für ODEMx-Kenner

wait_for_availability m **atleast** Memory **as** port1, port2, timer, cond;

```
inspect m  when PortHead { msg= get(); }
           when PortTail { m->put (new Msg); }
           when Timer {…}
           when Condition {…}
otherwise …;
```

interessanter **dynamischer TypTest** aus Simula
realisiert per SLX-Spracherweiterung
oder
Anwendung des SLX-built-in-Test-Operators *Type*

```
if (type(*m) == type (PortHead)) { pointer (PortHead) ph; msg= ph->get();}
           else
if (type(*m) == type (PortTail))  {…}
           else
if (type(*m) == type (Timer))  {…}
           else
if (type(*m) == type (Condition))  {…}
else { …}
```

Inhalt C.4

- **Teil A** Aspekte dynamis
- **Teil B** Die Moc
- **Teil C** Die ausf SLX
- **Teil D** Modellie

- **C.1** Einführung und Ba
- **C.2** Stochastische Pro
- **C.3** GPSS-Elemente
- **C.4** ODEMx-Elemente
- **C.5** Obektorientierung
- **C.6** DISCO-Elemente

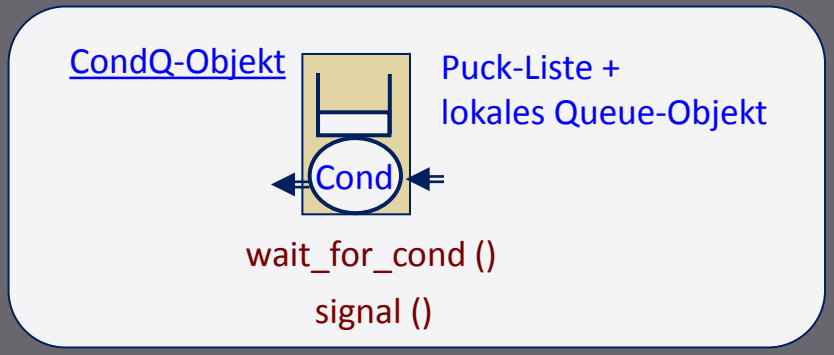
1. C++-Bibliothek ODEM, ODEMx
2. Master-Slave-Synchronisation
3. Asynchrone Kommunikation bei Behandlung spezieller Zustandsereignisse
4. Behandlung allgemeiner Zustandsereignisse

ODEMx ConditionQueue

- Synchronisation mit Zustandsereignissen
  (allg. Lösung in SLX: Control-Variablen als Bestandteile von Bedingungen, die mit wait until ausgewertet werden.

- Vermutung: In speziellen Fällen könnte der Einsatz von CondQ effizienter sein.

Wann ?



Blockierung des Aufrufers
bei Nichterfüllung

Expliziter Aufruf
eines erneuten Tests,
bei evtl. Reaktivierung
der blockierten Prozesse

- Im allg. ist jedoch die SLX-Lösung der ODEMx-Lösung vorzuziehen

Warum ?

```
passive class CondQ {
        set (Actor) ranked (descending priority) waitingQ ;
        string (18) name;
        queue wQ title= "waitforCondQ";

        initial {
                get_slx_name(ME, name);
        }
        method rename (string(*) s) { name= s; }

        overridable method condition returning boolean {return TRUE;};

        method wait_for_cond() returning boolean {
                pointer( Actor) caller;
                pointer ( puck ) caller_puck;
                string (18) caller_name;

                caller_puck= ACTIVE; // caller-Puck
                caller= caller_puck->puck_object;
                place caller into  waitingQ;
                enqueue wQ;
                #ifdef ODEM_DEBUG trace(caller->name, "waits in conditionQ of ", name);#endif
                while (not condition()  ) {
                        wait;  // continue by reactivation of a signaller or a wait_for_cond_interrupt
                }
                remove caller from waitingQ;
                depart wQ;
                if (caller->waitInterrupted) return FALSE;
                else return TRUE;
        }

        method signal() {
                pointer (Actor) act;
                pointer(Actor) caller;
                pointer ( puck ) caller_puck;
                string (18) caller_name;
                caller_puck= ACTIVE; // caller-Puck
                caller= caller_puck->puck_object;
                for (act= each Actor in waitingQ with condition()) {
                        reactivate act->my_puck;
                }
        }
} // CondQ
```

Lösung erfordert noch Klasse **Actor** als Basisklasse

s. Diskussion zur Ideensammlung zu einer verbesserten **CoopQ**-Implementation

Inhalt C.5

- **Teil A**
  Aspekte von Modellierung und Simulation dynamischer Systeme
- **Teil B**
  Die Modellierungssprache UML
- **Teil C**
  Die ausführbare Modellierungssprache SLX
- **Teil D**
  Modellierung von Lieferketten

- **C.1**
  Einführung und Basissprache
- **C.2**
  Stochastische Prozesse in SLX
- **C.3**
  GPSS-Elemente
- **C.4**
  ODEMx-Elemente
- **C.5**
  Obektorientierung
- **C.6**
  DISCO-Elemente

Rückblick - Vorausschau

bisher

- SLX-Kernsprache (Version 1.x)
- SLX-Spracherweiterung
- Standardmodule: Simulation
- Statistics-Modul (bislang nur Zufallszahlen-Generatoren vorgestellt)
- GPSS-Modul h7
- ODEMx Modul (Konzepte und Ideensammlung,
  noch nicht perfekt)

offen:

- Vererbung (OOM/OOP)
- Umgang mit Modellgrößen bei funktioneller Abhängigkeit von Zufallsgrößen (bekannter Verteilungsfunktionen)
  Experimentgestaltung → Experimentauswertung

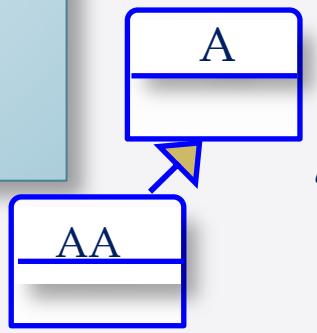
Augment: Vererbungsersatz

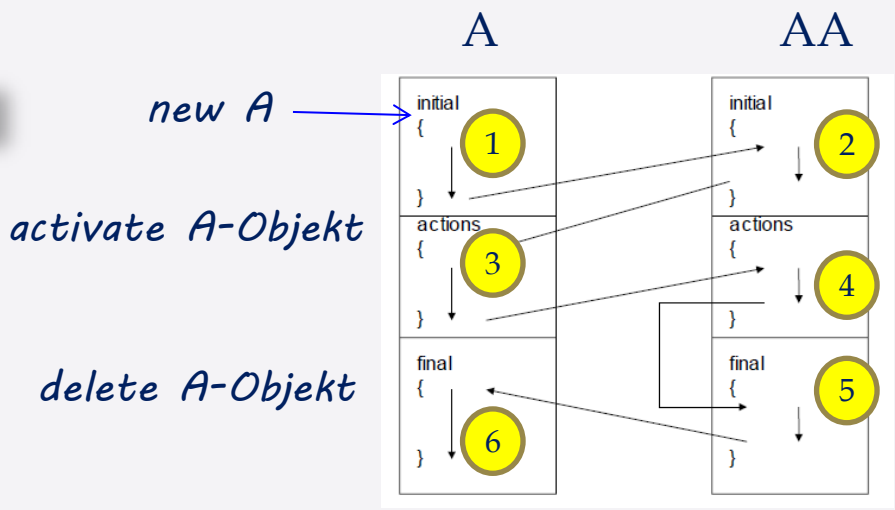
- Version 1.x (Konzept gibt es auch in anderen Sprachen: ObjectiveC)

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

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

A wird um Bestandteile erweitert
besser: A wird durch A + augment(A) ersetzt

…. und dies gegebenenfalls durchaus mehrmals





korrekte Anwendung liegt in der Hand des Nutzers

Aktionsreihenfolge für Augment-Klassen-Objekte

Augment-Anwendung im Modul h7

zwei Puck-Erweiterungen

```
passive class preemption
        {
        pointer(puck)               preemptee;
        };

augment puck
        {
        int                 preemption_count;
        };

passive class facility(string(*) report_title)
        {
        read_only
                {
                string(12)                              title;
                random_variable(time)           usage;
                interval                        total_time,
                                                        avail_time,
                                                        unavail_time;
```

Facility

```
entity_class        queue;     // set=All_queues;

pointer(qcb)        h7_qcb_pool;

class qcb           // Queue control block
        {
        pointer(qcb)                next_qcb;
        pointer(queue)      member_of;
        float                       entry_time;
        };

pointer(qcb)        puck_qq_;                   // keep this pointer outside the puck

augment puck                // extend the Kernel-level puck
        {
        pointer(qcb)        qcb_list;
        clear
                {
                while (qcb_list != NULL)
                        {
                        puck_qq_ = qcb_list;
                        qcb_list = qcb_list -> next_qcb;

                        puck_qq_ -> member_of = NULL;
                        puck_qq_ -> next_qcb = h7_qcb_pool;
                        h7_qcb_pool = puck_qq_;
                        }

                puck_qq_ = NULL;
                }

        final
                {
                if (qcb_list != NULL)
                        {
                        diagnose caller run_time warning (ACTIVE, qcb_list -> member_of)
                                "Warning: _ is a member of one or more queues, including _";

                        while (qcb_list != NULL)
                                {
                                puck_qq_ = qcb_list;
                                qcb_list = qcb_list -> next_qcb;

                                puck_qq_ -> member_of = NULL;
                                puck_qq_ -> next_qcb = h7_qcb_pool;
                                h7_qcb_pool = puck_qq_;
                                }

                        puck_qq_ = NULL;
                        }
                }
        };
```

Queue

Vererbung

- ab Version 2.x Einfachvererbung für Klassen

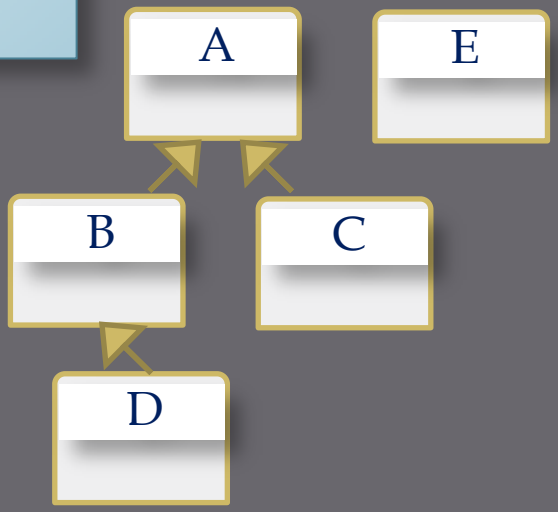
```
class A {
      A-attributes…
};

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



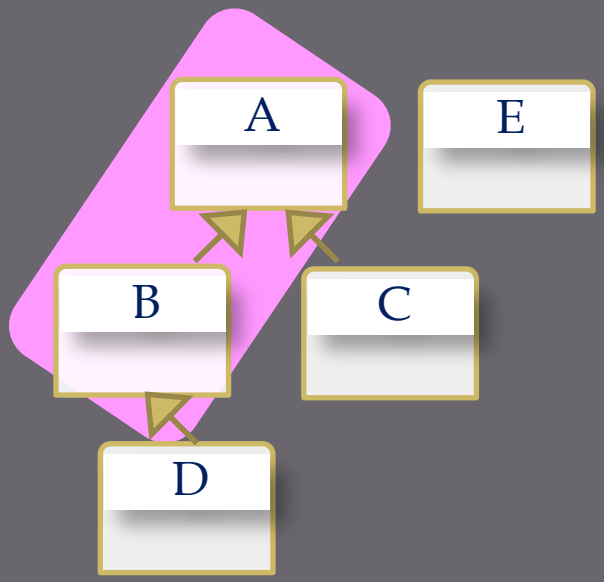
Basis- und abgeleitete Klasse können keine Attribute, Prozeduren oder Methoden gleichen Namens besitzen

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

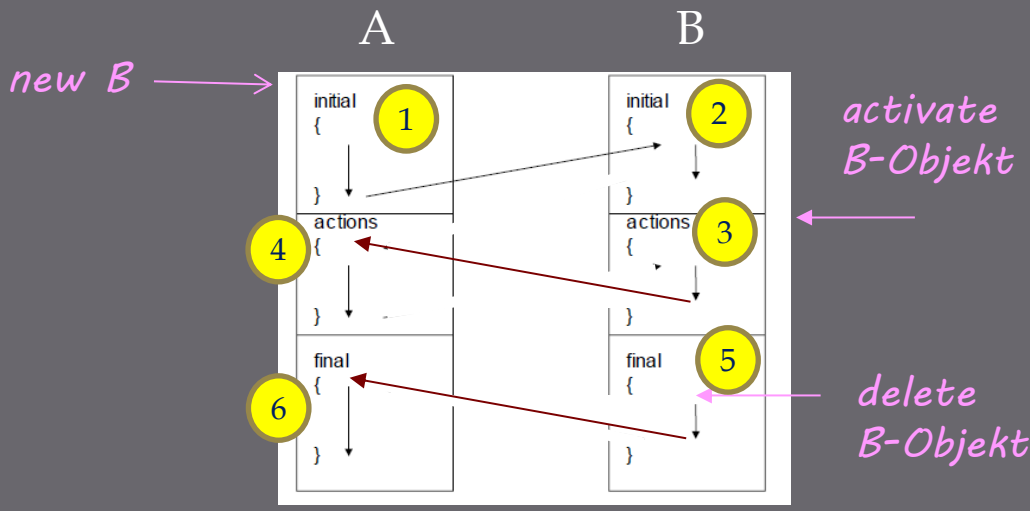
Benutzung der Version 2.0 muss explizit eingeschaltet werden:

#define SLX2 ON

Vererbung, Virtuelle Methoden



Aktionsreihenfolge für Objekte abgeleiteter Klassen



overridable == virtual
override== redefines

Regel: einmal overridable,
danach immer overridable

OFFEN:

1. Wann genau werden die Pucks generiert?
2. Wie kommen sie in die MovingPuck-Liste?

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

}
}
```

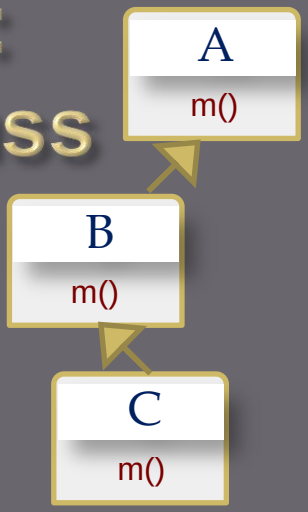
5 6 3 4 7 1 2 8

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*

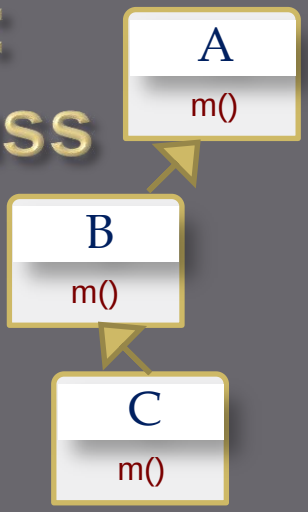
2. **hinzu kommt**
   - bei Ausführung einer Action-Property wird zu Beginn ein Child-Puck für die Action-Property der (falls vorhanden) nächsten Basisklasse generiert

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

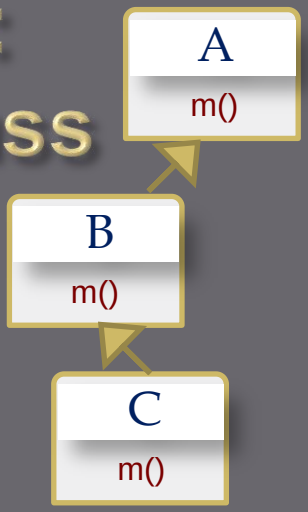
2

```
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";
        }
}
```

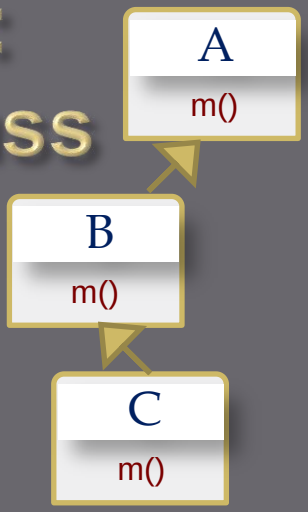
3

```
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";
        }
}
```

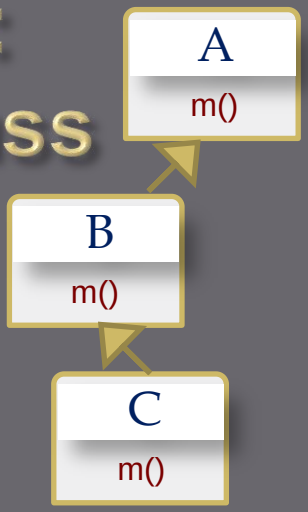
3

```
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";
        }
}
```

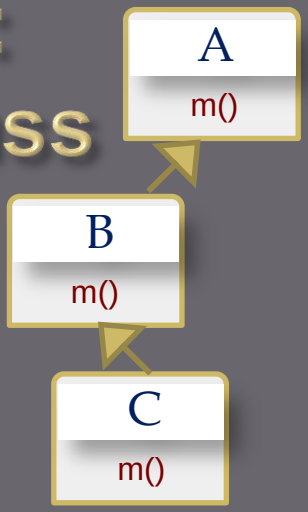
```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print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

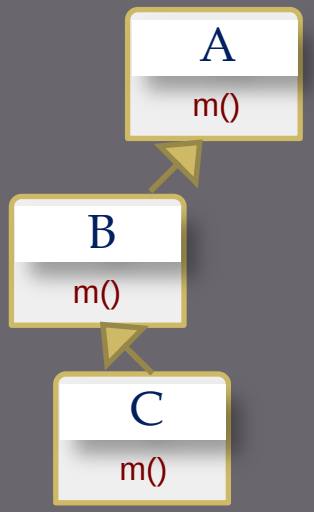
```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print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



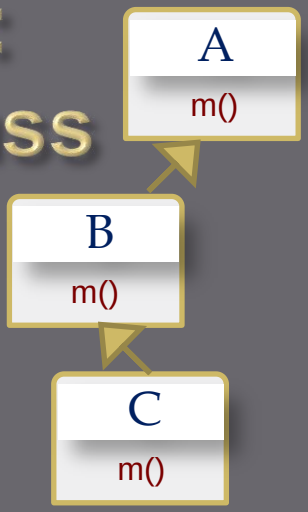
ptrC= (**pointer** (C)) ptraA;

```
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";
        }
}
```

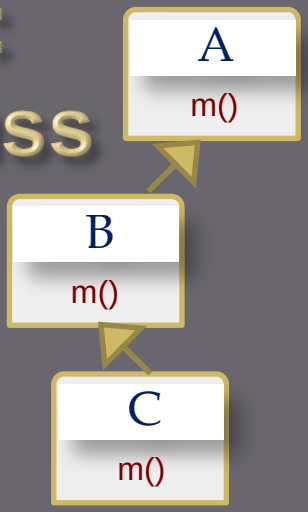
```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print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

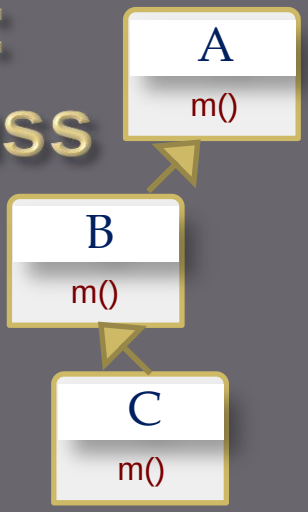
```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print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

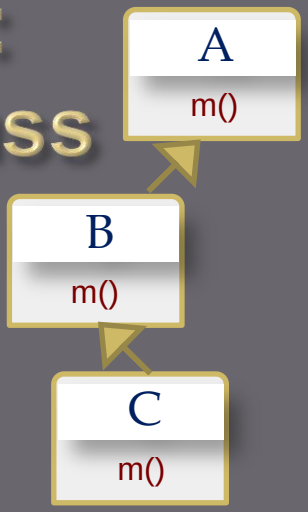
```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print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

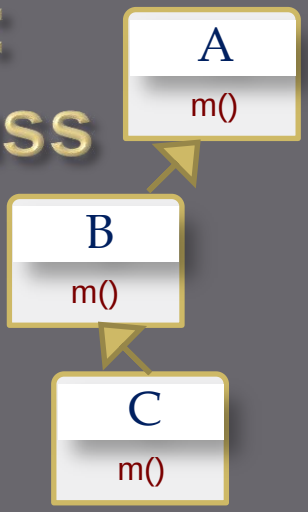
```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print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

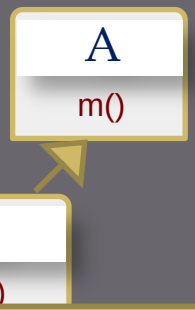
```
class C (in int k) subclass(B(2*k)) {
        int c;
        initial {
                id= FALSE;
                print (id) "C-initial id=_ \n";
        ;}
        override method m() {;
                 print "C-m-body \n";
                A::m();
        };
        actions  {
                print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

```
        final {
                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

Dynamischer Typtest

- Der type-Operator erlaubt die Typidentifikation eines Objektes, auf das ein Pointer verweist.

Beispiel:

```
class A { }

class B subclass (A) { }

procedure main() {
        pointer (A) pa = new B;
        if (type(*pa) == type B) {
                print "pa zeigt auf ein B";
        }
}
```

```
inspect pa when A {...}
           when B {...}
           when C {...}
           ...
           otherwise {... };
```

Solche Simula-Anweisung ließe
sich zusätzlich bei Anwendung von **type** mit SLX-Spracherweiterung bauen

Inhalt C.6

- **Teil A** Aspekte dynamis
- **Teil B** Die Moc
- **Teil C** Die ausf SLX
- **Teil D** Modellie

- **C.1** Einführung und Ba
- **C.2** Stochastische Pro
- **C.3** GPSS-Elemente
- **C.4** ODEMx-Elemente
- **C.5** Obektorientierung
- **C.6** DISCO-Elemente

1. Simula-Bibliothek DISCO
2. SLX-Bibliothek Statistics
   - Modellgrößen, Bewertungsgrößen, Kennwertermittlung
   - Random_Variable, Histogram, Statistics
   - Einfache Simulationsläufe, Histogramm-Auswertung
   - Zeitintervall-basierte Beobachtung und Auswertung
   - Konfidenz-Intervall-Schätzung
   - Sequential Sampling
   - Vergleich zwischen simulierten Systemvarianten

Historie

- Simulationsbibliothek in Simula von Helsgaun (DISCO)
  1980
- Simulationsbibliothek in Simula von Birtwistle (DEMOS)
  1983
- Statistik in C++ von (Ahrens/Fischer)
  1984, 1996
- Statistics in SLX
  2009

Inhalt C.6

- **Teil A** Aspekte dynamis
- **Teil B** Die Moc
- **Teil C** Die ausf SLX
- **Teil D** Modellie

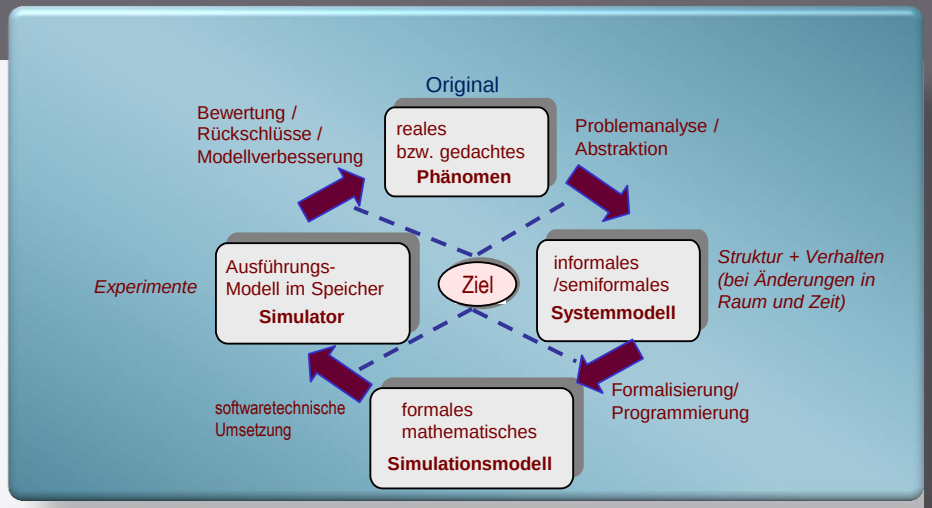
- **C.1** Einführung und Ba
- **C.2** Stochastische Pro
- **C.3** GPSS-Elemente
- **C.4** ODEMx-Elemente
- **C.5** Obektorientierung
- **C.6** DISCO-Elemente

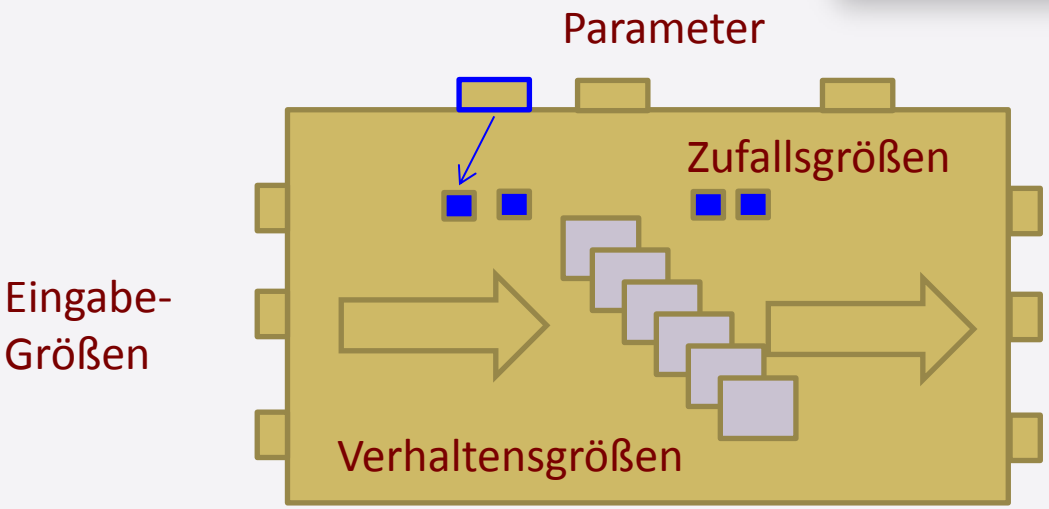
1. Simula-Bibliothek DISCO
2. SLX-Bibliothek Statistics
   - Modellgrößen, Bewertungsgrößen, Kennwertermittlung
   - Random_Variable, Histogram, Statistics
   - Einfache Simulationsläufe, Histogramm-Auswertung
   - Zeitintervall-basierte Beobachtung und Auswertung
   - Konfidenz-Intervall-Schätzung
   - Sequential Sampling
   - Vergleich zwischen simulierten Systemvarianten

Experimente und deren Auswertung

Typischer Ablauf

- Sammlung von Beobachtungsdaten
  je ausgezeichnete Modellvariable
  (Ergebnisgröße)
  in einem Simulationslauf



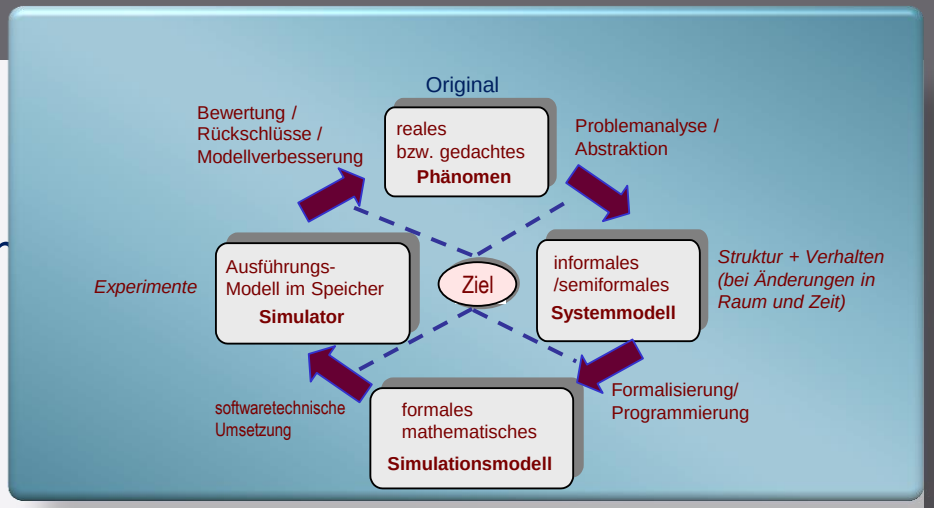


Beispiele
- Länge eines WS
- Antwortzeit

schwankende Werte
~ Beobachtungsdauer
~ Startwerte der ZZG

Experimente und deren Auswertung (2)

Typischer Ablauf

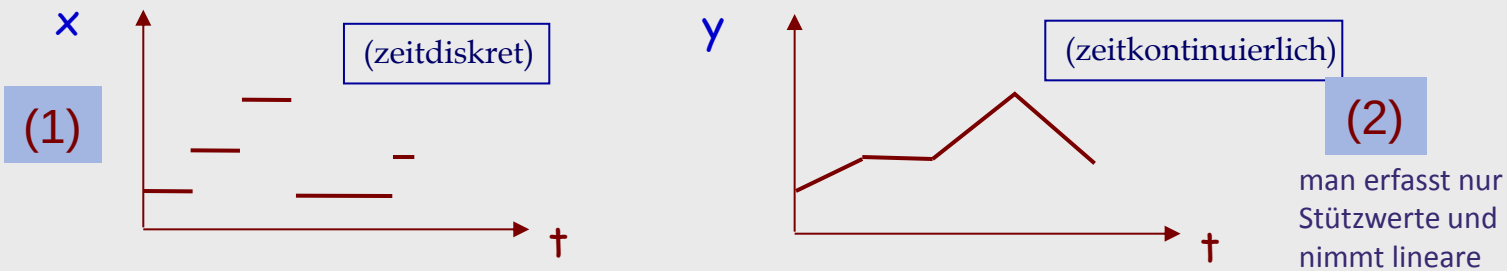


- Sammlung von Beobachtungsdaten je ausgezeichnete Modellvariable (Ergebnisgröße) in einem Simulationslauf
- Verdichtung der gewonnenen Rohdaten zu statistischen Kenngrößen (Mittelwert, Streuung/Standardabweichung) und deren Speicherung
- Durchführung wiederholter Simulationsläufe bei Variation der Startwerte der Zufallszahlen und Ermittlung und Speicherung der Kennwerte
- Berechnung von statistischen Parametern wie Mittelwert und Konfidenzintervall für die Ergebnisgrößen aller realisierten Simulationsläufe

Profile zu beobachtender Modellgrößen

int- oder double- Modellgrößen x, y mögen sich im Laufe der Simulation ändern
(z.B. x Attribut der Klasse X, y Attribut der Klasse Y

Arten von Werterfassungen (Beobachtungen)



man erfasst nur Stützwerte und nimmt lineare Übergänge an

(1) Modellbeobachtungen: x1, x2, x3, …

(2) Modellbeobachtungen: (y1, t1), (y2, t2), (y3, t3)

(3) Modellbeobachtungen: $(x_1, y_1)^{t1}$, $(x_2, y_2)^{t2}$, $(x_3, y_3)^{t1}$, …

Beobachtung ist bei jeder Änderung der Variablen durch das Modell zu garantieren !

Profil enthält Mittelwert, Standardabweichung, Minimum, Maximum

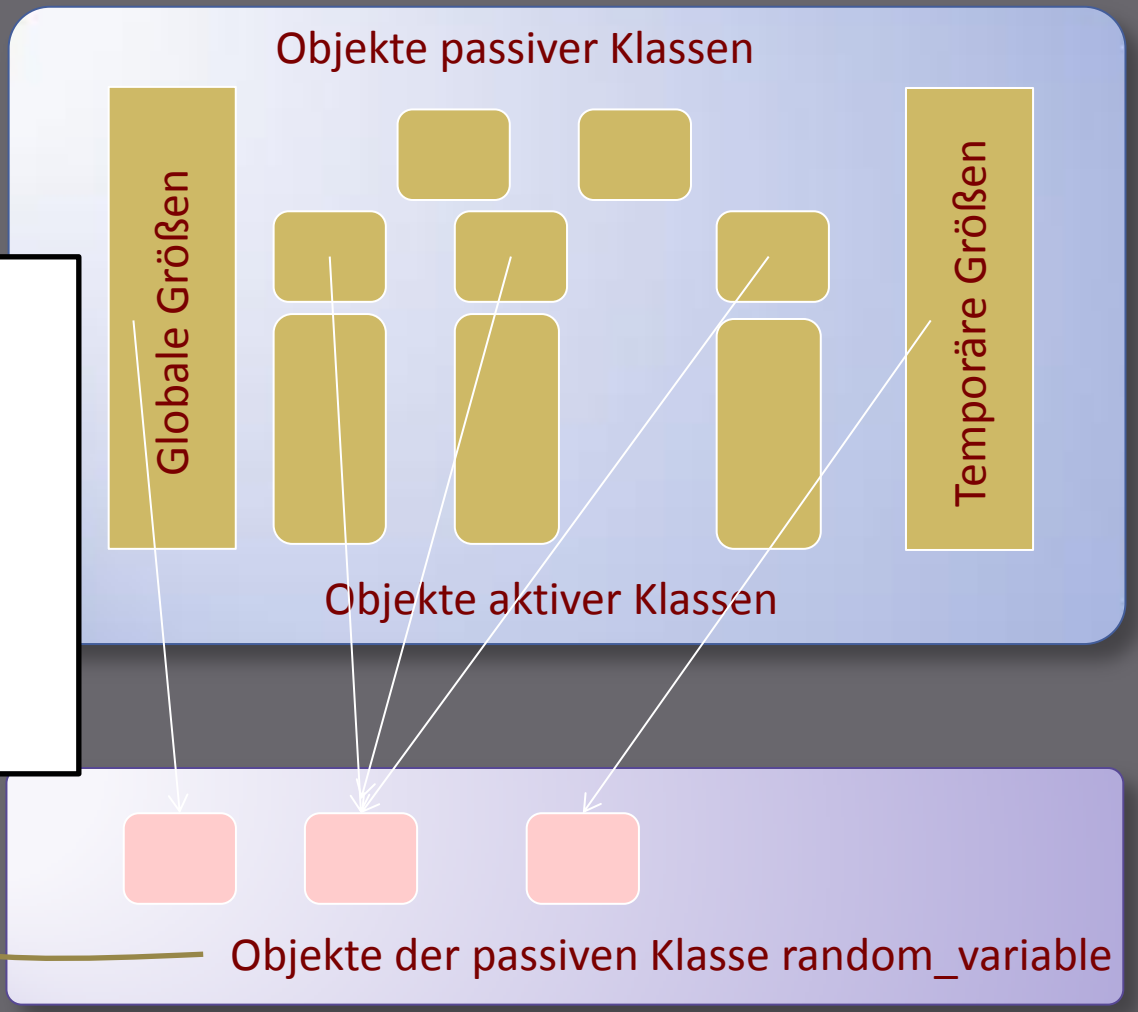
Inhalt C.6

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  ODEMx-Elemente
- **C.5**
  Obektorientierung
- **C.6**
  DISCO-Elemente

1. Simula-Bibliothek DISCO
2. SLX-Bibliothek Statistics
   - Modellgrößen, Bewertungsgrößen, Kennwertermittlung
   - Random_Variable, Histogram, Statistics
   - Einfache Simulationsläufe, Histogramm-Auswertung
   - Zeitintervall-basierte Beobachtung und Auswertung
   - Konfidenz-Intervall-Schätzung
   - Sequential Sampling
   - Vergleich zwischen simulierten Systemvarianten

Erfassung von Beobachtungsdaten (Datenströme)



Kategorien von Ergebnisgrößen

- ungewichtete Beobachtungsdaten
  - Zählungen, Summen
  - Werteverteilung
- zeitgewichtete Daten
- nutzerspezifisch gewichtete Daten

Erzeugung von random_variable-Instanzen

- *Anweisung (Syntax):*

random_variable [**(time** | **weight)]** *random_variable_ident*
histogram start= *start_value* **width=** *width_value* **count=** *count_value*]
[**title=** *report_title*]

- *Beispiele*

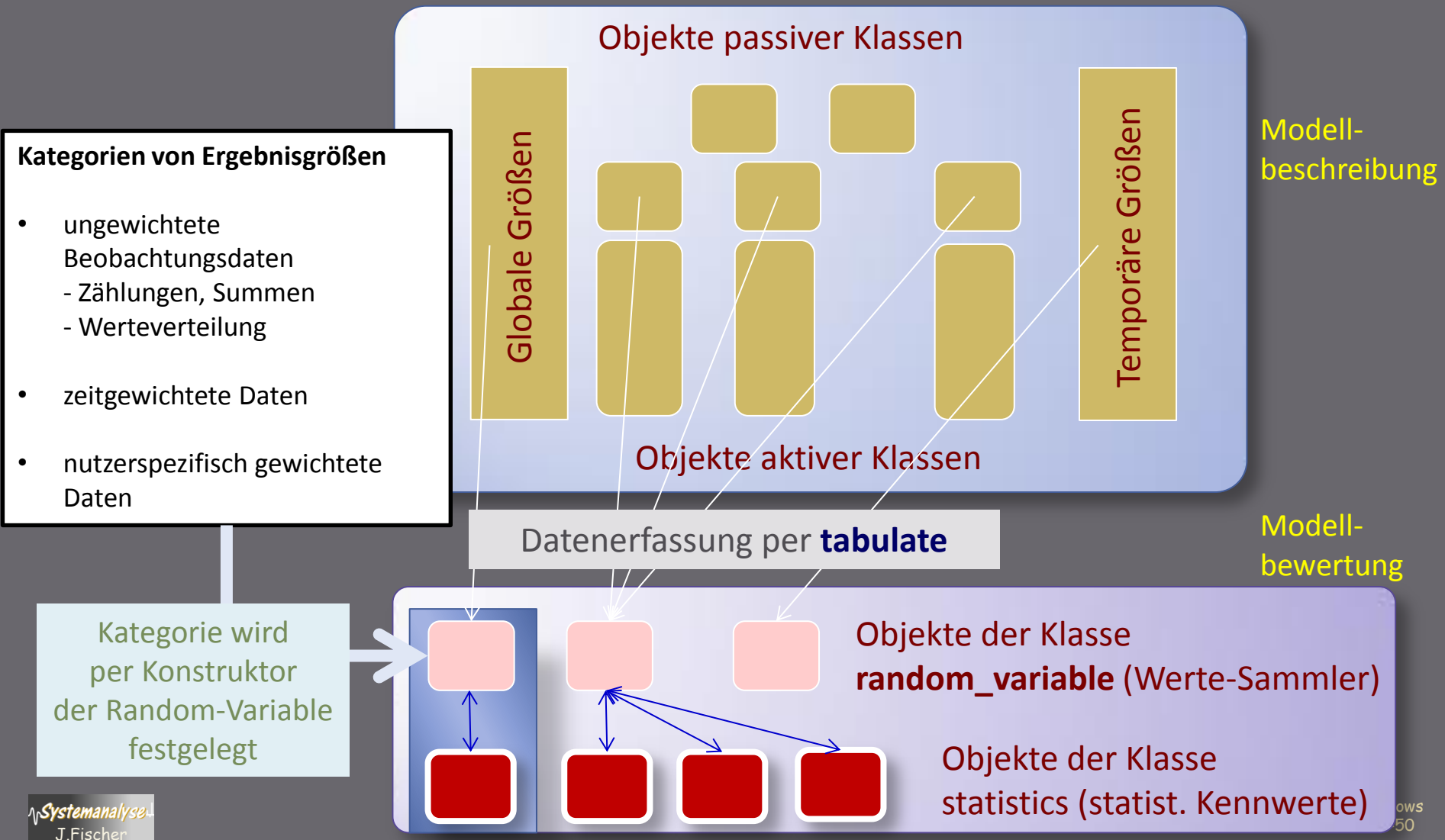
```
random_variable x1;                    //ungewichtet
```

```
random_variable x2
        histogram start= 0.0 width= 0.5 count= 20; // ungewichtet
```

```
random_variable (time) x3;             //zeitgewichtet
```

```
random_variable (weight) x4;           //mit nutzerspezif. Gewicht
```

Erfassung von Beobachtungsdaten (Datenströme)



Random Variable

- per **random_input** werden Größen definiert, die Werte von transformierten (0,1)-gleichverteilten Zufallszahlen darstellen.

- Werden die Wertannahmen beobachtet und (z.B. in Form von Histogrammen) ausgewertet, dann können die Variablen als Instanzen von **Random Variable** verstanden werden.

- Der **Report** weist sie zumindest zusammen mit echten **RandomVariable**n aus. Dies sind Variablen (Instanzen der Klasse random_variable), die die einzelnen Wertrealisierungen (und damit Schwankungen) erfassen und gängige statistische Kennwerte liefern können.

Attribute von random_variable

```
public read_only class random_variable(
                    wtype       rv_weight_type,
                    pointer(histogram)     rv_histo,
                    string(*)   report_title,
                    int         estimated_sample_count)        {

    wtype                   weight_type;        // Gewichtstyp
    pointer(histogram)      histo;              // Verweis auf ein Histogramm
    string(16)              title;
    int                     SOB_batch_size, SOB_half_batch_size;
    private OEM int         nhb;
    control double          current_value;      // aktueller Wert
    double                  tlast,              // Zeit des letzten updates
                            smallest_max,       // kleinstes Max aller Statistiken
                            largest  min;       // größtes Min aller Statistiken
    set(statistics)         by_interval;        // zugeordnete Intervalle
    statistics              master(ME, NULL);   // Statistik über gesamten Zeitbereich
```

Erfassung der Beobachtungen

- Anweisung (Syntax):

tabulate *random_variable_ident* **=** *observed_value*
[{ **weight =** *weight_value* } | { **count=** *count_increment* }]

- Beispiele

```
tabulate x1= time - active->mark_time;//ungewichtet
```

```
tabulate x2 = tanker->load;    // ungewichtet mit Histogramm
```

```
tabulate x3 = tankerQ.length; // zeitgewichtet, bei Eintritt
tabulate x3 = tankerQ.length count= 0; // zeitgewichtet, bei Austritt
```

```
tabulate x4 = t weight= tank->pressure.state; //mit nutzerspezif. Gewicht
```

lokale Zeit

Histogramm ist dann ein Zeitdiagramm

Kernwert-Profil-Erstellung

```
public read_only class random_variable(
                    wtype       rv_weight_type,
                    pointer(histogram)    rv_histo,
                    string(*)   report_title,
                    int         estimated_sample_count)        {

    wtype                   weight_type;          // Gewichtstyp
    pointer(histogram)      histo;                // Verweis auf ein Histogramm
    string(16)              title;
    int                     SOB_batch_size, SOB_half_batch_size;
    private OEM int         nhb;
    control double          current_value;        // aktueller Wert
    double                  tlast,                // Zeit des letzten updates
                            smallest_max,         // kleinstes Max aller Statistiken
                            largest_min;          // größtes Min aller Statistiken
    set(statistics)         by  interval;         // zugeordnete Intervalle
    statistics              master(ME, NULL);     // Statistik über gesamten Zeitbereich
```

durch automatisch generiertes Statistics-Objekt

Attribute von Statistics

```
read_only class statistics(          pointer(random_variable) stat_root_rv,
                                     pointer(interval)        stat_interval)      {
        int         count;                  // Anzahl der Beobachtungen
        double      sum,                    // Summe der Werte
                    sum_of_weights,         // Summe der Gewichtungen
                    sum_of_squares,         // Summe der Quadrat-Werte
                    min_value,              // Minimaler Wert
                    max_value;              // Maximaler Wert
        // SOB variables
        double      SOB_low_sum,            // lower half-batch sum of values
                    SOB_high_sum,           // upper half-batch sum of values
                    SOB_mean,               // current (recursively calculated) mean
                    SOB_svar_sum;           // current (recursively calculated) variance sum
        int         SOB_batch_count,
                    SOB_sample_no;

        pointer(histogram)          histo;      //zugeordnetes Histogramm
        pointer(random_variable)    root_rv;    //zugeordnete random-Variable
        pointer(interval)           my_interval;//zugeordnetes Intervall
```

Die Klasse Statistics

Wieviel Speicherplätze werden zur Profilbestimmung benötigt ?

- Anzahl der Beobachtungen
- Summe der bisher beobachteten Werte
- Summe der Quadrate der bisher beobachteten Werte
- Minimum
- Maximum

Berechnung erfolgt erst zur Abruf-Zeit

Mittelwert m

$$m = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Streuung/Varianz s^2

Standardabweichung s

$$s^2 = \frac{1}{n-1} \left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n} \left(\sum_{i=1}^{n} x_i \right)^2 \right)$$

Dynamische Kennwertermittlung

- SLX-Makros

sample_count (*rv* [**over** *interval*] **)**	**int**	Anzahl der Beobachtungen
sample_max (*rv* [**over** *interval*] **)**	**float**	Maximum
sample_min (*rv* [**over** *interval*] **)**	**float**	Minimum
sample_sum (*rv* [**over** *interval*] **)**	**float**	Zeitintegral über die Werte
sample_mean (*rv* [**over** *interval*] **)**	**float**	Erwartungswert
sample_variance (*rv* [**over** *interval*] **)**	**float**	Varianz
sample_stdev (*rv* [**over** *interval*] **)**	**float**	Standardabweichung
sample_time_per_unit (*rv* [**over** *interval*] **)**	**float**	Zeitintegral dividiert durch die Anzahl der Beobachtungen

Berechnung durch
vorinstallierte Master-Statistik oder
optionale Intervall-Statistik

Varianten der Statistics-Instanzerzeugung

durch Anweisungen

(1) **random_variable** als Master-Statistics-Instanz

(2) **interval**

(3) **observe**

aus: „Simulation needs SLX“ (S. 4-53)

„Es wird nicht empfohlen, direkt mit Objekte der Klasse **statistics** zu arbeiten. Die Anweisungen für die SLX-Klassen **random_variable** und **interval** machen eine Anwendung der …Prozeduren für die Klasse **statistics** nicht notwendig.“

Inhalt C.6

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  ODEMx-Elemente
- **C.5**
  Obektorientierung
- **C.6**
  DISCO-Elemente

1. Simula-Bibliothek DISCO
2. SLX-Bibliothek Statistics
   - Modellgrößen, Bewertungsgrößen, Kennwertermittlung
   - Random_Variable, Histogram, Statistics
   - Einfache Simulationsläufe, Histogramm-Auswertung
   - Zeitintervall-basierte Beobachtung und Auswertung
   - Konfidenz-Intervall-Schätzung
   - Sequential Sampling
   - Vergleich zwischen simulierten Systemvarianten

Beispiel-Anforderungen

Bank mit
- 3 Schaltern
- 1 gemeinsame Warteschlange der Schalter
- Kundenstrom

storage clerk

random_variable

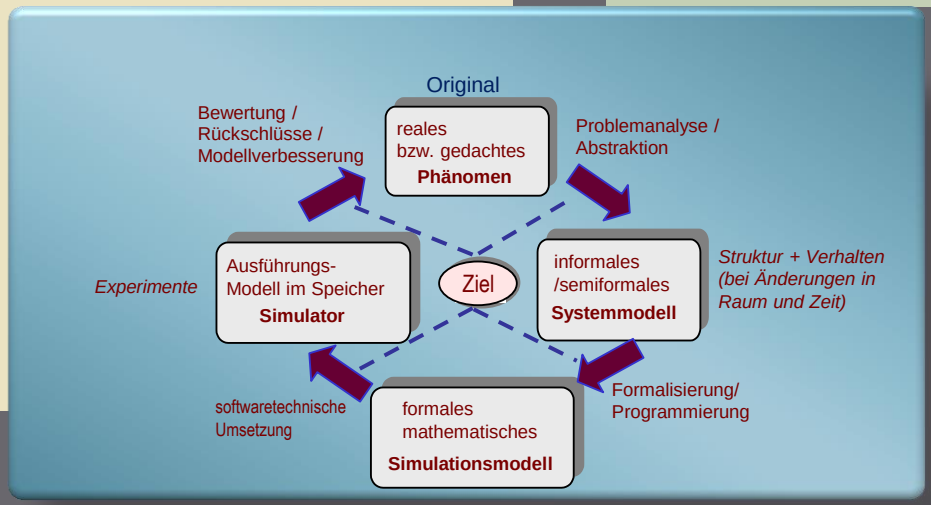
Untersuchungsziel: statistische Kennwerte für
- Wartezeit (zusätzlich mit Histogramm)
- Warteschlangenlänge
- Verweilzeit

rv_queuing_time

rv_queue_length

rv_elapsed_time

Randbedingung:
- Bank schließt nach 8 h
- Simulation ist beendet, wenn letzter Kunde die Bank verlassen hat



Beispiel: SLX-Umsetzung

```
//*****************************************************************************
// Example EX0022
//*****************************************************************************
import <stats>
import <h7>
module basic  {
        rn_stream               arrive,
                                service ;

        random_variable         rv_elapsed_time,
                                rv_queueing_time histogram start=0.0 width=0.5 count = 20;
        random_variable (time)  rv_quelength;

        storage clerk capacity=3;
        control int in_customer, out_customer;
        int que_length;
        constant float close_time=8*60, service_time=1.3 ;
        boolean door_closed;

        class Customer {
        actions {
                in_customer ++; // increment customer counter
                que_length++; // increment queue length
                tabulate rv_quelength=que_length; // tabulate
                enter clerk; // try to catch a clerk
                que_length --; // decrement queue length
                tabulate rv_quelength=que_length count = 0;
                tabulate rv_queueing_time= time - ACTIVE->mark_time;
                advance rv_expo ( service , service_time ); // service time
                leave clerk;
                out_customer ++;
                tabulate rv_elapsed_time = time - ACTIVE->mark_time;
        } //actions
        } // Customer
```

```
        procedure run_model () {
                float intensity=2.0;
                fork  { // arriving customer
                        forever {
                                advance rv_expo ( arrive , 1/intensity );
                                activate new Customer;
                                if ( door_closed ) terminate;
                        }
                }
                fork { // Controlling the bank
                        advance close_time;
                        door_closed = TRUE;
                        terminate;
                }

                wait until ( (time > close_time) && ( in_customer == out_customer ));
                report ( system );
        }

        procedure main() {
                run_model();
        }// main
}
```

EX-0022 (Bank).slx: SLX-64 UL211 Lines: 7,466 Errors: 0 Warnings: 0 Lines/Second: 278,752 Memory: 4 MB

Execution begins

System Status at Time 484.9196

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
arrive	924	200000	200924	OFF	0.43
service	924	400000	400924	OFF	0.63

Random Variable	#Observed	Mean or ~Value	Std Dev or ~Error	Sig. Digits	Minimum	Maximum
rv_elapsed_time	924	2.531	2.038		0.01	12.04
rv_quelength	924	2.362	3.213		0.00	13.00
rv_queueing_time	924	1.240	1.559		0.00	7.65

Lower	Upper	Frequency	Percent	
0.0	0.5	415	44.913	**
0.5	1.0	117	12.662	**********
1.0	1.5	114	12.338	**********
1.5	2.0	73	7.900	********
2.0	2.5	46	4.978	*****
2.5	3.0	30	3.247	***
3.0	3.5	29	3.139	***
3.5	4.0	18	1.948	**
4.0	4.5	34	3.680	****
4.5	5.0	15	1.623	*
5.0	5.5	9	0.974	*
5.5	6.0	10	1.082	*
6.0	6.5	6	0.649	
6.5	7.0	4	0.433	
7.0	7.5	3	0.325	
7.5	8.0	1	0.108	

Storage	Capacity	Total %Util	Avail %Util	Unavl %Util	Entries	Average Time/Puck	Current Status	%Avail	Average Contents	Current Contents	Maximum Contents
clerk	3	82.00			924	1.29	AVAIL	100.00	2.46	0	3

Execution complete

Objects created: 46 passive and 925 active Pucks created: 928 Memory: 4 MB Time: 0.13 seconds