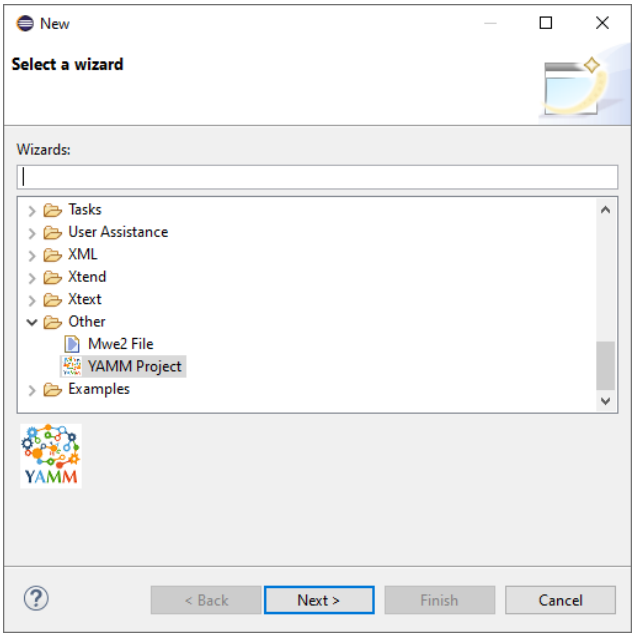
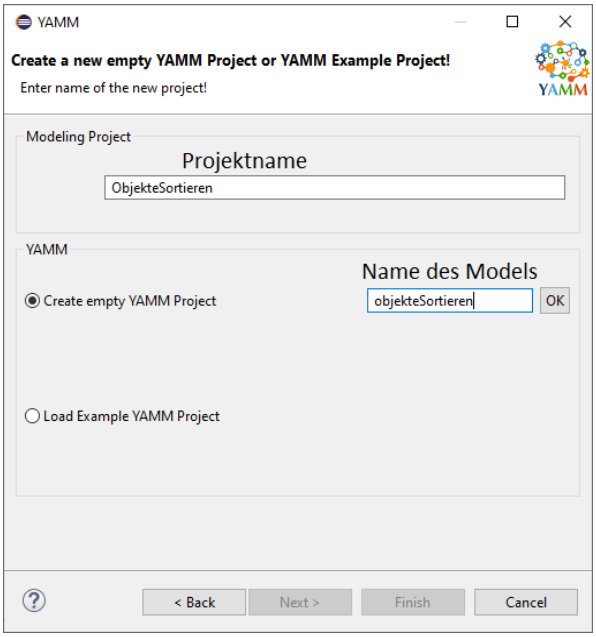
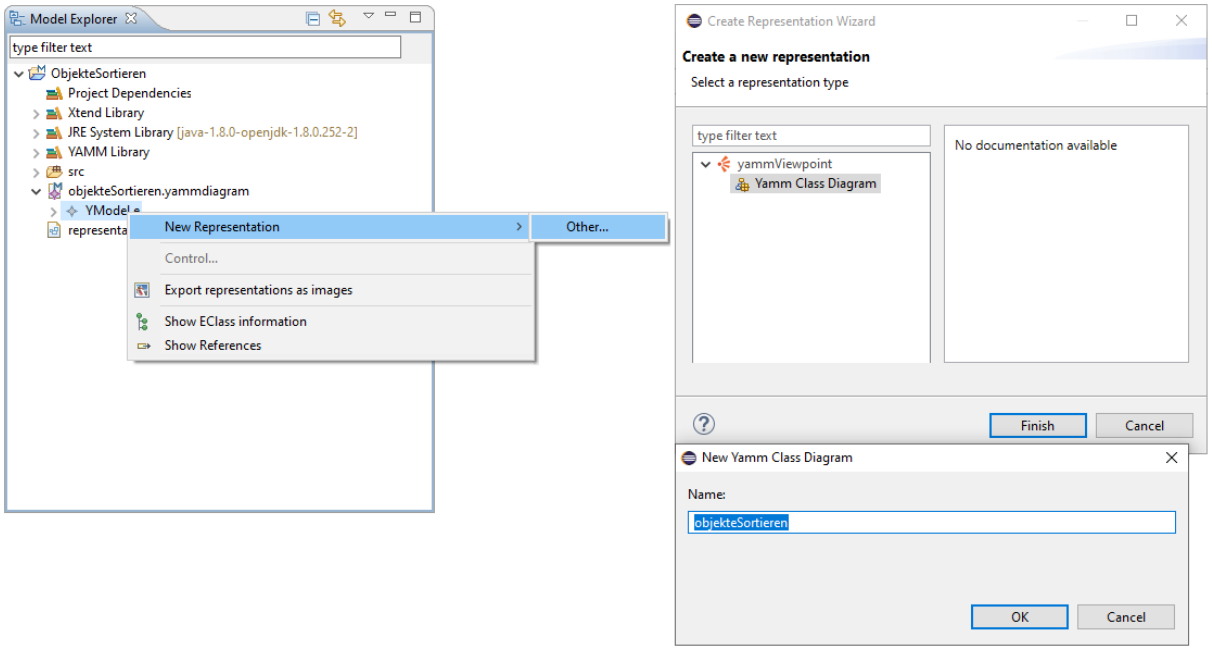
Projekt erstellen

1. File – new – Other.. – Other – YAMM Project
   -- Next – Projektname eingeben – Create empty YAMM Projekt auswählen – name des Models eingeben – OK – Finish

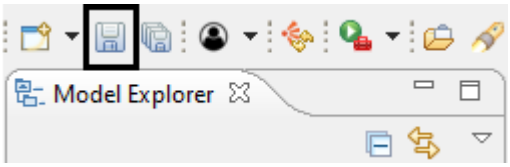




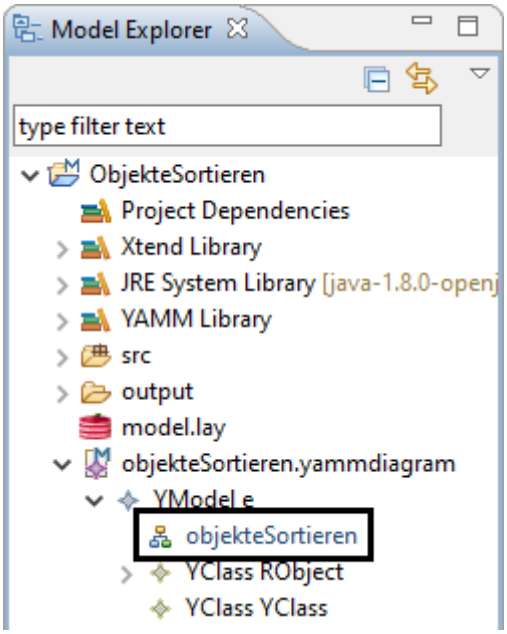
2. Klassendiagramm erzeugen:
   Wenn kein Model Explorer vorhanden: Window – Perspective – Open Perspective – Other…
   -- Modeling – Open
   Wenn immer noch nicht vorhanden: Window – Show View – Other… -- Sirius – Model Explorer – Open



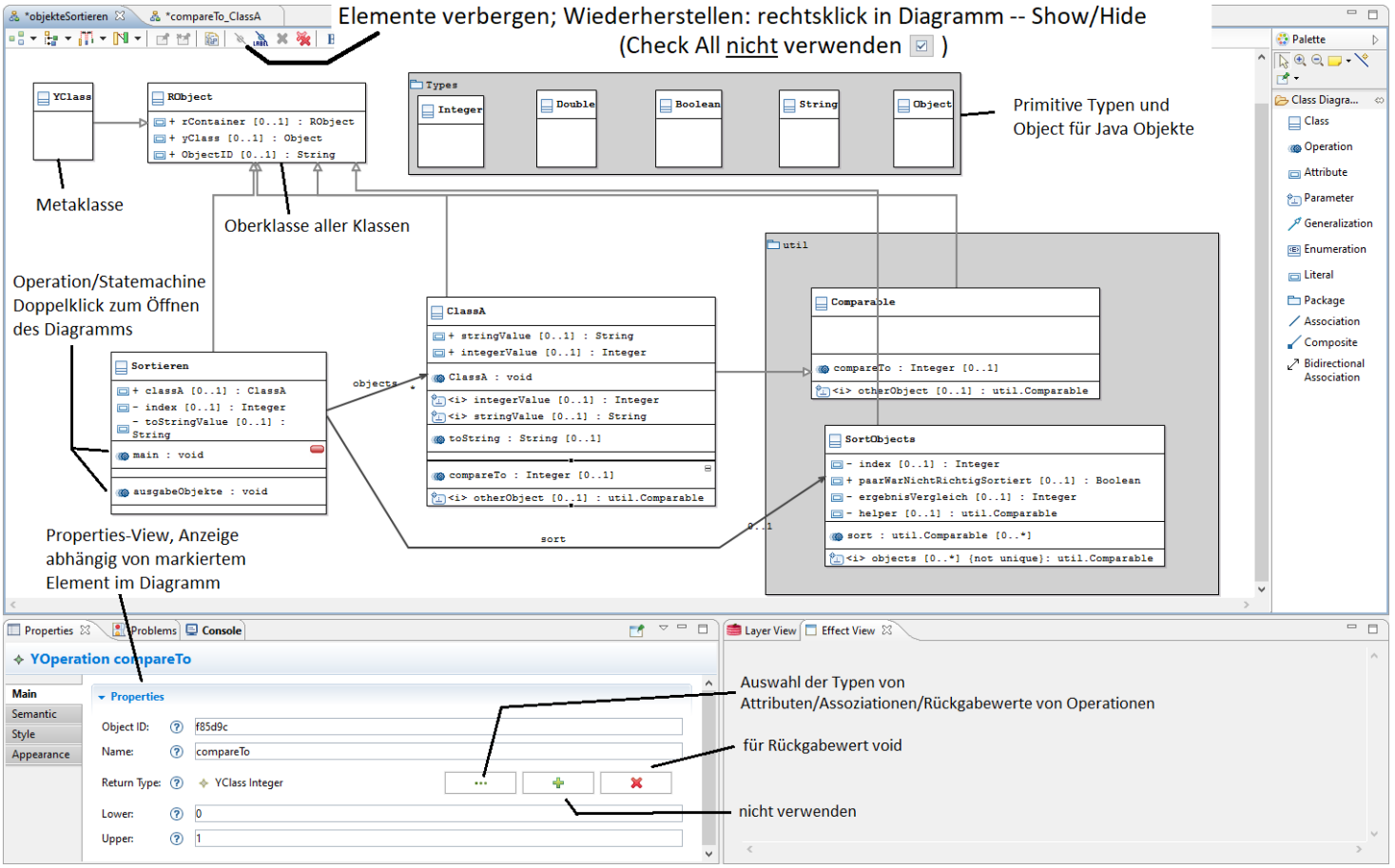
Nach dem Erzeugen eines Klassendiagramms immer speichern:



3. Diagramme öffnen: aufklappen des Models im Modelexplorer und Doppelklick auf Diagrammelement.

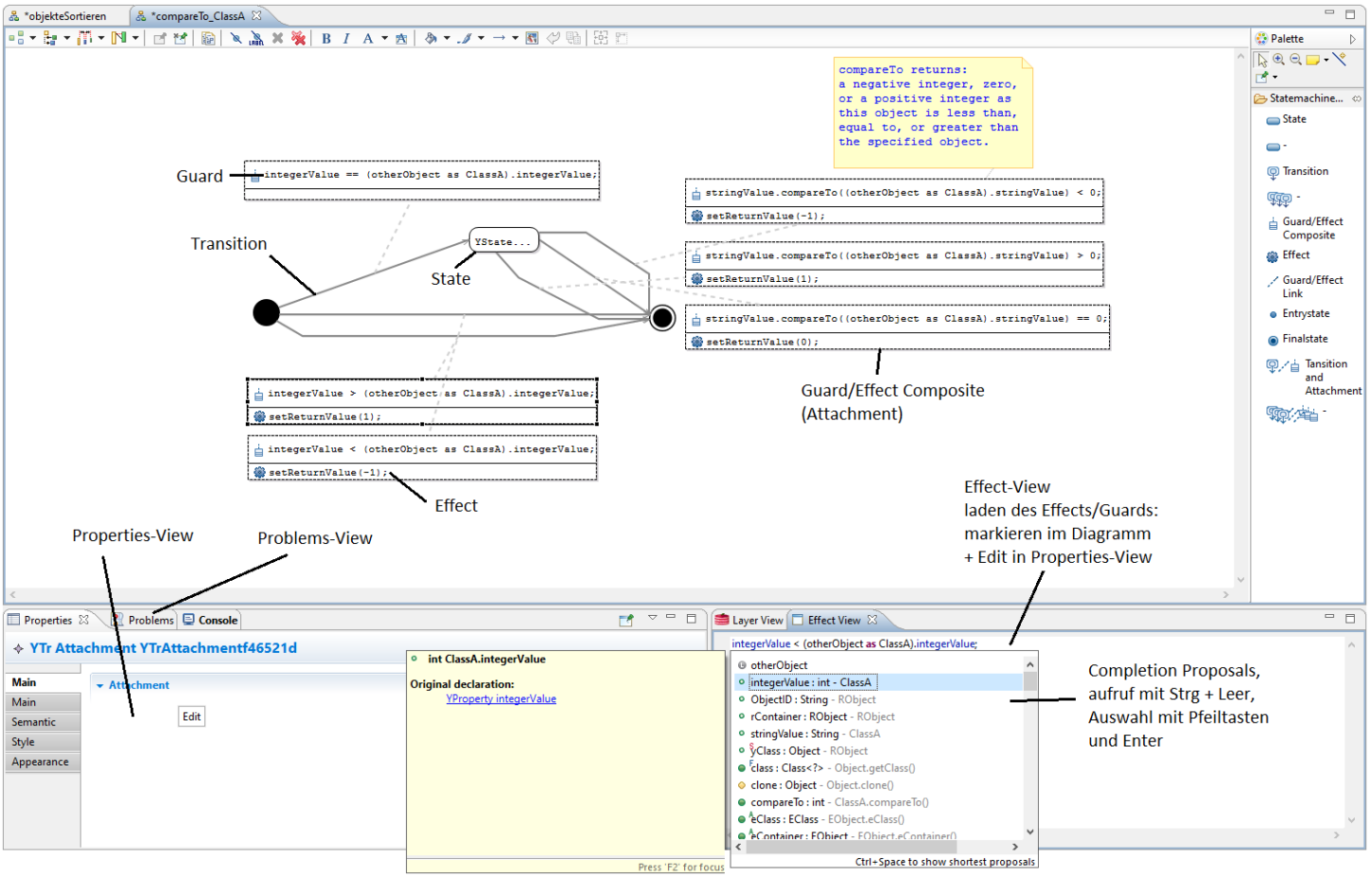


Klassendiagramm:



- Wenn Properties View nicht vorhanden: Window – Show View – Other… -- General – Properties – Open
- Editieren von Namen/Label der Elemente: markieren + F2, oder markieren und schreiben
- Öffnen/Erzeugen eines Statemachinediagramms mit Doppelklick auf Operation
- Assoziationen und Attribute sind äquivalent. Änderung der grafischen Darstellung über Kontextmenü – AssociationToAttribute/AttributeToAssociation
- Per Drag & Drop lässt sich die Reihenfolge von Parametern und Attributen verändern
- Attribute/Assoziationen/Operation: Upper: -1 → *
- Undo/Redo : Edit – Undo/Redo oder Shortcut: Windows: Strg + z, Strg + y
  Linux: Strg + z, Strg + Umschalt + z
  bezieht sich auf das gesamte Model, nicht nur das geöffnete Diagramm
- Keine Umlaute oder Eszett verwenden
- Speichern eines Diagramms speichert das gesamte Model (alle Diagramme)
- Copy/Paste möglich
- Autolayout: Hintergrund anwählen – Symbol ganz links in der Menüleiste
- Weiter Layoutmöglichkeiten in der Menüleiste und über Kontextmenü. Z.b.: Rechtsklick auf Connection – Remove Bend-Point oder Format – Line Style – Rectlinear Style Routing; hide RObject und DES Package

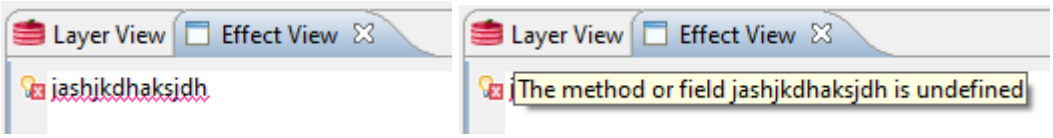
Statemachine-/Operationdiagramm:



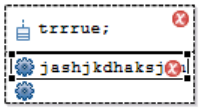
- Per Drag & Drop lässt sich die Reihenfolge von Effekten verändern
- Keine Umlaute oder Eszett verwenden
- Jeweils genau ein Entry- und Finalstate
- In der Effect-View werden die Guards und Effekte eingeben. Dazu ein Effekt oder Attachment markieren und in der Properties-View auf Edit klicken. Die Eingabe wird in das Diagramm übernommen wenn ein Element ausserhalb der Effect-View angeklickt wird.
- Strg + Leertaste drücken während der Eingabe öffnet die Liste der Vervollständigungvorschläge. Mit den Pfeiltasten navigieren und Enter zum Auswählen drücken.
- Doppelklick auf eine Transition markiert das zugehörige Attachment
- Speichern eines Diagramms speichert das gesamte Model (alle Diagramme)
- Undo/Redo bezieht sich auf das gesamte Model, nicht nur das geöffnete Diagramm
- Die Namen der States müssen eindeutig sein und dürfen keine Leerzeichen enthalten
- Copy/Paste möglich

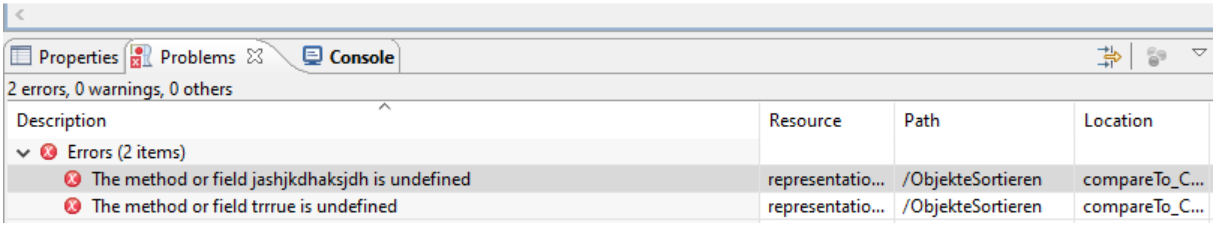
Anzeige von Fehlern:

- In der Effect-View werden Fehler angezeigt; Mauszeiger über linke Fehleranzeige blendet die Meldung ein:



- Alle Fehler in einem Diagramm anzeigen: Rechtsklick in das Diagramm – Validate diagram
- Die Fehler werden grafisch und in der Problems-View (windows – Show View – Other.. -- General -- Problems) angezeigt. Doppelklick auf Eintrag in Problems-View markiert Grafisches Element in Diagramm
- Nach Korrektur des Fehlers muss die Validierung neu gestartet werden
- Die Validierung bezieht sich nur auf ein Diagramm, nicht auf das gesamte Model
- Validierung des Klassendiagramms: Doppelte Namen





- Fehler werden auch beim Simulationsstart in der Console-View angezeigt

Effekte/Guards:

- Ausdrücke müssen mit einem Semikolon abschließen
- es sind mehrer Ausdrücke pro Effekt/Guard möglich
- der letzte Ausdruck eines Guards muss boolean zurückgeben
- die Guards aller ausgehenden Transitionen eines States sollten sich gegenseitg ausschließen
- die Guards aller ausgehenden Transitionen eines States werden in zufälliger Reihenfolge ausgeführt

Parametertypen:

- in:
  - Integer, Double und Boolean werden als call-by-value übergeben
  - andere Objekte ausser Listen als call-by-reference
  - Listen: nur die Elemente werden übergeben und eine neue Liste erzeugt
- inout:
  - alles ausser Listen wird als call-by-reference übergeben
  - Listen werden wie bei dem in-Parameter übergeben (neu Liste erzeugt), Listenoperationen werden auch auf der übergebenene Liste ausgeführt
- inDeep: wie in aber als tiefe Kopie (keine kopie wenn es Java-objekte sind)

Assoziationen:

- Bidirektionale Assoziation:
  - setzen in eine Richtung setzt auch die gegenüberliegende
  - kann direkt erstellt werden: Element in der Palette, oder über angabe der Opposite-Assoziation in der Properties-View
- Composite:
  Ein Objekt kann sich nur in einer Composite Assoziation befinden. Wenn sie einer anderen Composite Assoziation hinzugefügt wird, wird sie automatisch aus der vorherigen entfernt. Das zu der Assoziation geöhrige Objekt wird Container genannt. Zugriff auf den Container über obj.rContainer():RObject.

Pseudoparalle Ausführung, Initial aktive Operationen und Observer:

- die Simulation beginnt mit einer als initial aktiv markierten Operation
- Auswahl über Kontextmenü auf Operation, oder Klick auf Hintergrund im Statemachinediagramm – Properties-View-- initialActive
- wenn mehrere markiert sind werden sie Pseudoparallel ausgefürt
- der Kontextwechsel ist determinitisch und abhängig vom Run-To-Completion (rtc) Wert der Statemachine (Klick auf den Hintergrund im Statemachinediagramm – Properties-View - rtc)
- es gibt drei möglichkeiten:
  - effect: Wechsel nach jedem ausgeführten Effekt
  - transition: Wechsel nach Änderung des States
  - statemachine: die gesammte Statemachine wir ausgeführt
  - observer: spezielle Operation, s.u.
- Wenn alle Guards der ausgehenden Transitionen eines aktiven States false sind wird immer gewechselt
- Zugriff auf in anderen Threads erzeugten Objekte: (getInstances(1, „KlassenName“) as java.util.List<KlassenName>).isEmpty/get(0)
- Observer:
  - als Observer markierte Operationen laufen immer Parallel zu allen anderen

- sie werden während jedem Kontextwechsel der normalen Operationen aktiviert
- sie laufen wie rtc: statemachine

Mehrfachvererbung:

- Klassen können, anders als in Java, beliebig viele Oberklassen haben
- Auswahl einer speziellen Operation mit:
  obj.operation(…)[Oberklasse]
- dies wird auch anstelle von super verwendet
- Aufruf der Super-Konstruktors ist so nicht möglich (ist vorerst generell nicht möglich)

Operationsaufrufe:

- Pro Effekt darf es nur einen Operationsaufruf geben. Er muss im letzen Ausdruck des Effekts stehen. Sein Rückgabewert darf nur in einer Zuweisung verwendet werden. Auf der linken Seite der Zuweisung darf kein Java-Methodenaufruf stehen. Zu beachten: Property Access (s.u.) ist auch ein Operationsaufruf.
- in Guards können keine Operationsaufrufe verwendet werden

Arrays:

- Arrays werden nicht unterstützt.

Property Access:

- Anstelle des Aufrufs von Gettern kann der Aufruf erfolgen wie ein Zugriff auf public Attribute: a.propertyA <-> a.getPropertyA(). Anstelle des Setters kann eine Zuweisung verwendet werden: a. propertyA = b <-> a.setPropertyA (b). Hover über Ausdruck zeigt die verwendeten Operationen an.

Extension Methods:

- Möglich mit als Extension registrierten Klassen. Statische Methoden, bei denen der erste Parameter vor den Methodenaufruf gesetzt wird, der Operations-/Methodenname wird rot dargestellt. Bsp.: einStringProperty.toFirstUpper anstelle von StringExtension.toFirstUpper(einStringProperty). Innerhalb einer Klasse können die Operationen auch auf diese Weise aufgerufen werden.
  Verwendbare Extensions:
  https://javadoc.io/doc/org.eclipse.xtext/org.eclipse.xtext.xbase.lib/2.14.0/index.html
  dann Link: org.eclipse.xtext.xbase.lib

Returnwert einer Operation setzen:

- an beliebiger Stelle und beliebig oft kann ein Rückgabewert gesetzt werden mit dem Befehl „setReturnValue(…)“
- Der Befehl kann bei einem Durchlauf beleibig oft ausgeführt werden. Beim Erreichen des Finalstates wird der zuletzt gesetzte Wert zurückgegeben.

Input während Ausführung:

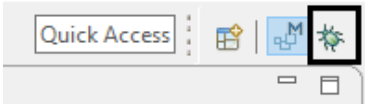
- Mit dem Befehl YammInput.integer/double/boolean/stringDialog(message:String): Integer/Double/Boolean/String wird während einer Simulation ein Eingabefenster geöffnet.

Weiter Hinweise:

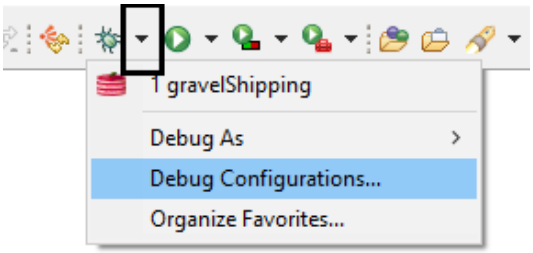
- Klammern bei Operations-/Methodenaufrufen ohne Parameter sind nicht notwendig
- Ausgabe in der Console-View mit dem Befehl „Print(:String)“
- Casten mit „as“ hinter dem Objekte
- Es gibt keine lokalen Variablen
- Java-Objekte können Attributen vom Typ Types.Object zugewiesen werden. Aufruf von Methode/attributen mit cast.
- Die Sprache ist sehr ähnlich zu Java. Es lassen sich alle Klassen der Java-Standartbibliothek verwenden. Weitere Bibliothen lassen sich über ein speziell erzeugtes Eclipse Wrapper Plugin anbinden.
  Vorgehen:
  - Wrapperplugin: new – Other.. -- Plug-in from Existing JAR Achive
  - In die Manifestdatei Abhängigkeit zu online.yamm.coreModelDsl, online.yamm hinzufügen
  - In die Manifestdatei den Eintrag „Eclipse-RegisterBuddy: online.yamm.coreModelDsl, online.yamm“ in die zweite Zeile hinzufügen
  - eine Updatesite erstellen und das Plugin installieren, oder eine zweite Eclipse-Application starten (Run Configurations)
  - das jar dem Build Path des YAMM-Projekt als Externens JAR hinzufügen

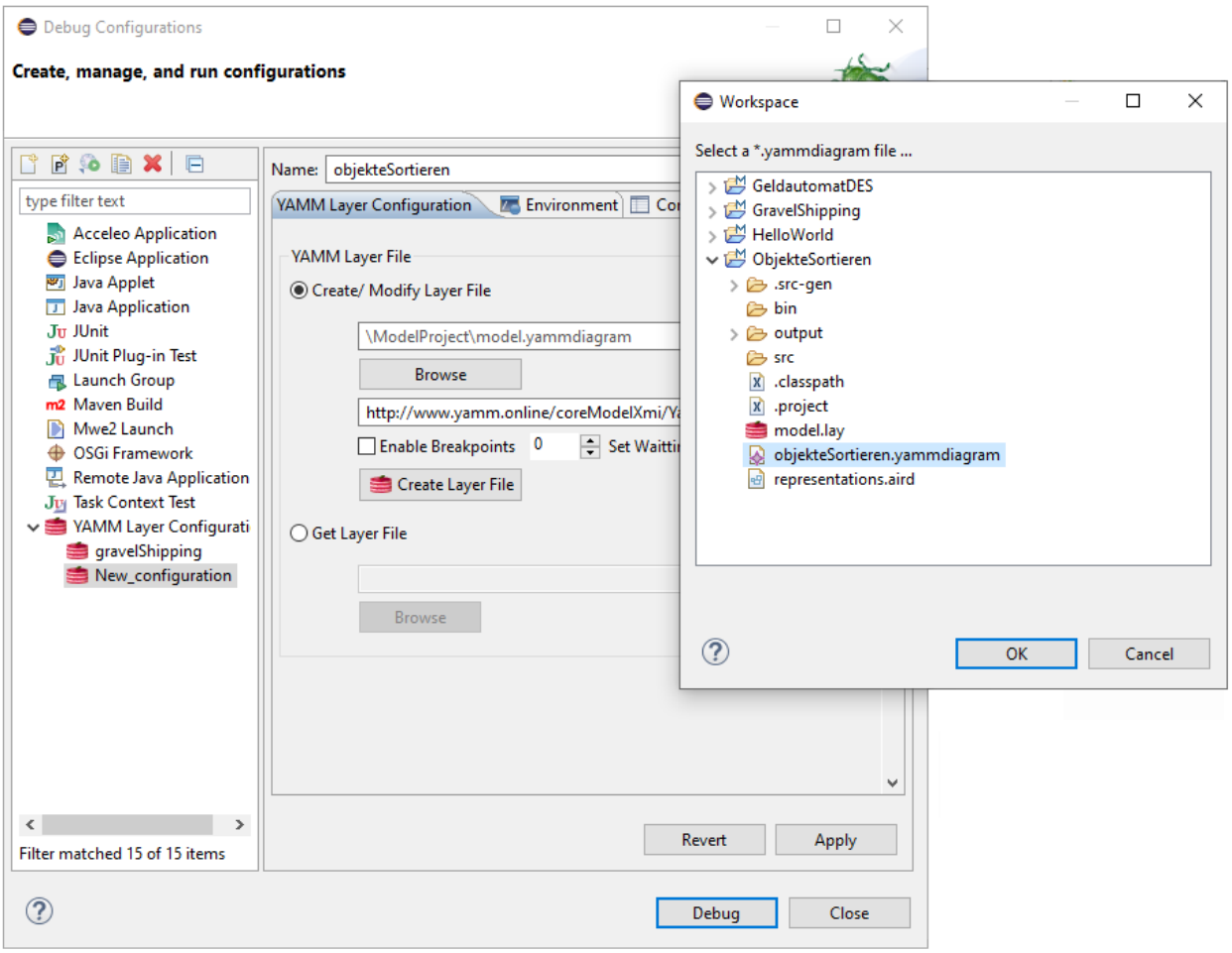
Simulation

1. In die Debug Perspective wechseln, oben rechts oder Window – Perspective – Open Perspective – Other… -- Debug – Open

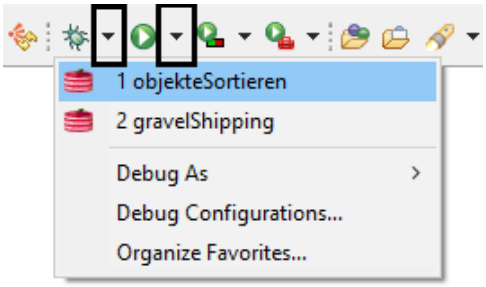


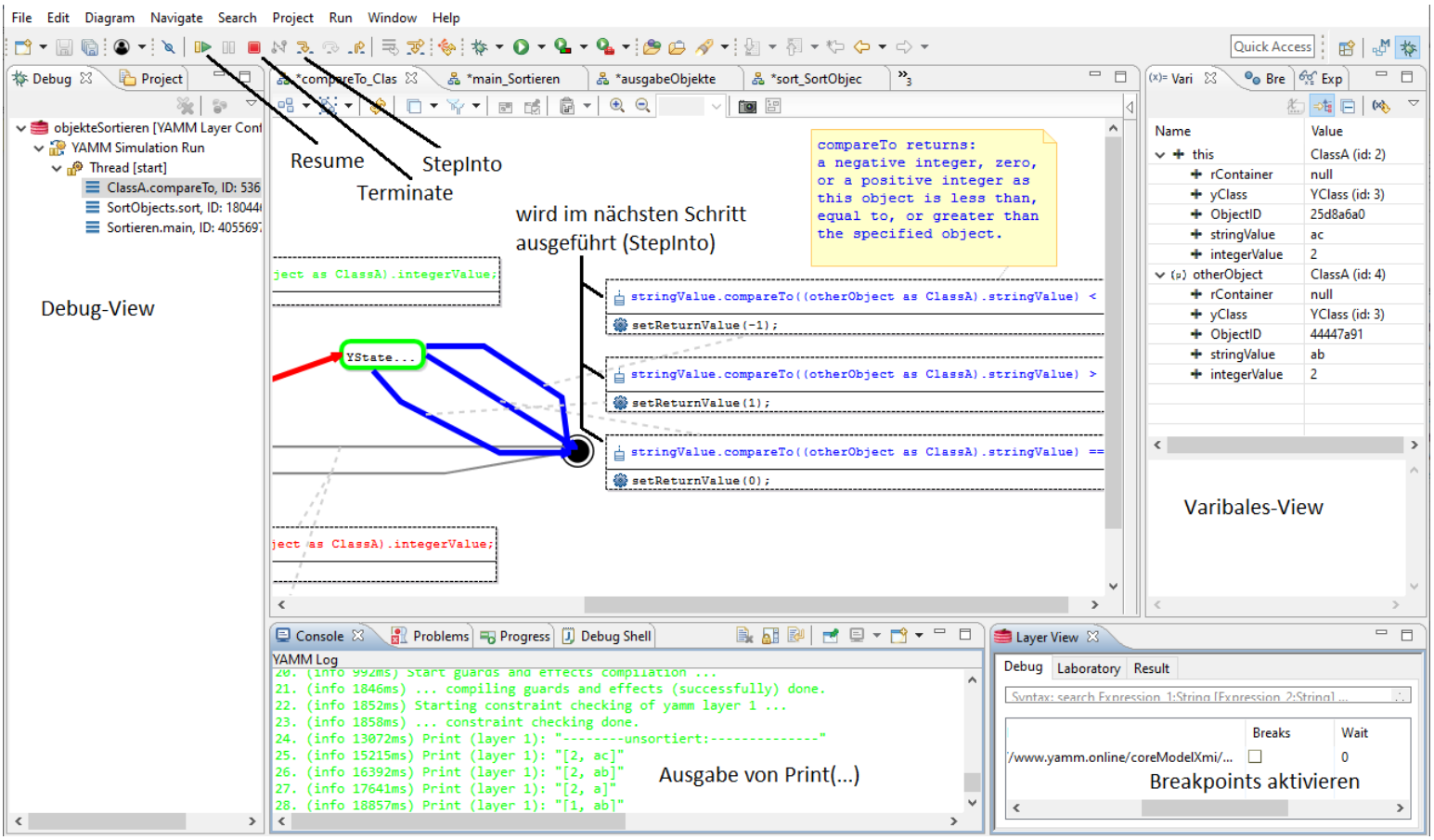
2. Lauch Konfiguration erstellen:
Pfeil neben dem Debug Symbol in der Menüleiste – Debug Configurations… -- Doppelklick auf YAMM Layer Configuration – Name angeben – Browse -- *.yammdiagram auswählen – OK – Apply – Debug oder Close





3. Simulation starten:
Über das Run Symbol (keine grafische Simulation) oder das Debug Symbol in der Menüleiste
(wenn im vorherigen Schritt auf Close geklickt wurde ist die Konfiguration noch nicht in der Auswahl
→Debug/Run Configurations… -- Debug/Run)





Debug-Modus:

- Mit StepInto wird die Simulation schrittweise durchlaufen. Der als nächstes auszuführende Effekt oder die Guards an allen ausgehenden Transitionen ist blau eingefärbt.
- Mit Resume läuft die Simulation bis zum nächsten Breakpoint, oder bis zum Ende durch. Nach dem Simulationsstart muss dazu der Hacken Breaks in der Layer-View gesetzt werden.
- Die Werte der Properties des aktiven Objekts werden in der Variables-View angezeugt.
- In der Debug-View wird der Stackframe angezeigt und Parallele Threads, falls mehrer Operationen als initial aktiv markiert sind.
- Die Debug-View muss immer geöffnet sein und ihr Elemntenbaum komplett aufgeklappt.
- Breakpoints müssen vor dem Start der Simulation angelegt werden. In Der Modeling Perspective ein Effekt markieren und in der Properties-View den hacken setzen, dannach das Model speichern.