AnyLogic and Java

Nathaniel Osgood

Advantages of AnyLogic
(as compared to other Agent-Based Modeling Software)

- Primarily declarative specification
- Less code
- Great flexibility
- Access to Java libraries
- Support for multiple modeling types
- Support for mixture of modeling types

Painful Sides of AnyLogic Education/Advanced

- Export of model results: Lack of trajectory files
- Lack of a built-in debugger
- Need for bits of Java code
- Many pieces of system

Internals of AnyLogic files: XML

C:\Usask\Classes\ABMCMCC2009\Models\HybridABMNetworkModeling1\HybridABMNetworkModeling1 Anylogic 6_2_2.alp - Notepad++

File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ?

AnalyzeTBCaseContacts.R | EraseFileInDirectory.pl | FindMissingBrowseFiles.pl | CINFilesToCSV2.pl | CreateDataDictionaryFromSpreadsheetIThinkEquations

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
***********************************************
                AnyLogic Project File
***********************************************
-->
<AnyLogicWorkspace WorkspaceVersion="1.9" AnyLogicVersion="6.2.2.200806031102" AlpVersion="6.2.2">
<Model>
    <Id>1257613518087</Id>
    <Name><![CDATA[HybridABMNetworkModeling1 Anylogic 6_2_2]]></Name>
    <ExcludeFromBuild>false</ExcludeFromBuild>
    <EngineVersion>6</EngineVersion>
    <JavaPackageName><![CDATA[hybridabmnetworkmodeling]]></JavaPackageName>
    <ActiveObjectClasses>
        <!--   =========   Active Object Class   ========  -->
        <ActiveObjectClass>
            <Id>1257613518149</Id>
            <Name><![CDATA[Main]]></Name>
            <ExcludeFromBuild>false</ExcludeFromBuild>
            <ClientAreaTopLeft><X>0</X><Y>0</Y></ClientAreaTopLeft>
            <PresentationTopGroupPersistent>true</PresentationTopGroupPersistent>
            <IconTopGroupPersistent>true</IconTopGroupPersistent>
            <Generic>false</Generic>
            <GenericParameters><![CDATA[T]]></GenericParameters>
            <AgentProperties>
                <SpaceType>CONTINUOUS</SpaceType>
                <EnvironmentDefinesInitialLocation>true</EnvironmentDefinesInitialLocation>


            </AgentProperties>

            <DatasetsCreationProperties>
                <AutoCreate>true</AutoCreate>
                <SamplesToKeep>100</SamplesToKeep>
                <FirstUpdateAtTime>true</FirstUpdateAtTime>
                <FirstUpdateTime>0.0</FirstUpdateTime>
                <FirstUpdateDate>1263556975211</FirstUpdateDate>
```

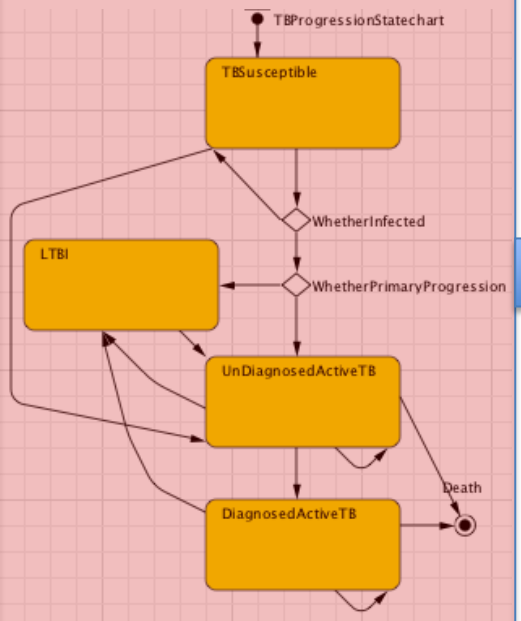
Normal text file | 54746 chars 58036 bytes 1646 lines | Ln:1 Col:1

Java Code: When & How Much?

- “Java” is a popular cross-platform “object oriented” programming language introduced by Sun Microsystems
- Anylogic is written in Java and turns models into Java
- AnyLogic offers lots of ways to insert snippets (“hooks”) of Java code
- You will need these if you want to e.g.
  - Push AnyLogic outside the envelop of its typical support
    - e.g. Enabling a network with diverse Agent types
  - Exchange messages between Agents
  - Put into place particular initialization mechanisms
  - Collect custom statistics over the population

Stages of the Anylogic Build

Modification Possible



Modification Not Possible

Java Code

Main | Main | Person | Main.java

```
double  initialPrevalenceOfInfection ) {
    if (initialPrevalenceOfInfection == this.initialPrevalenceOfIn
      return;
    }
    this.initialPrevalenceOfInfection = initialPrevalenceOfInfecti
    onChange_initialPrevalenceOfInfection();
    onChange();
  }

  void onChange_initialPrevalenceOfInfection() {
    int index;
    index = 0;
    for ( Person object : Population ) {
      object.set_isInitiallyInfected((uniform() < initialPrevalenc
      index++;
    }
```

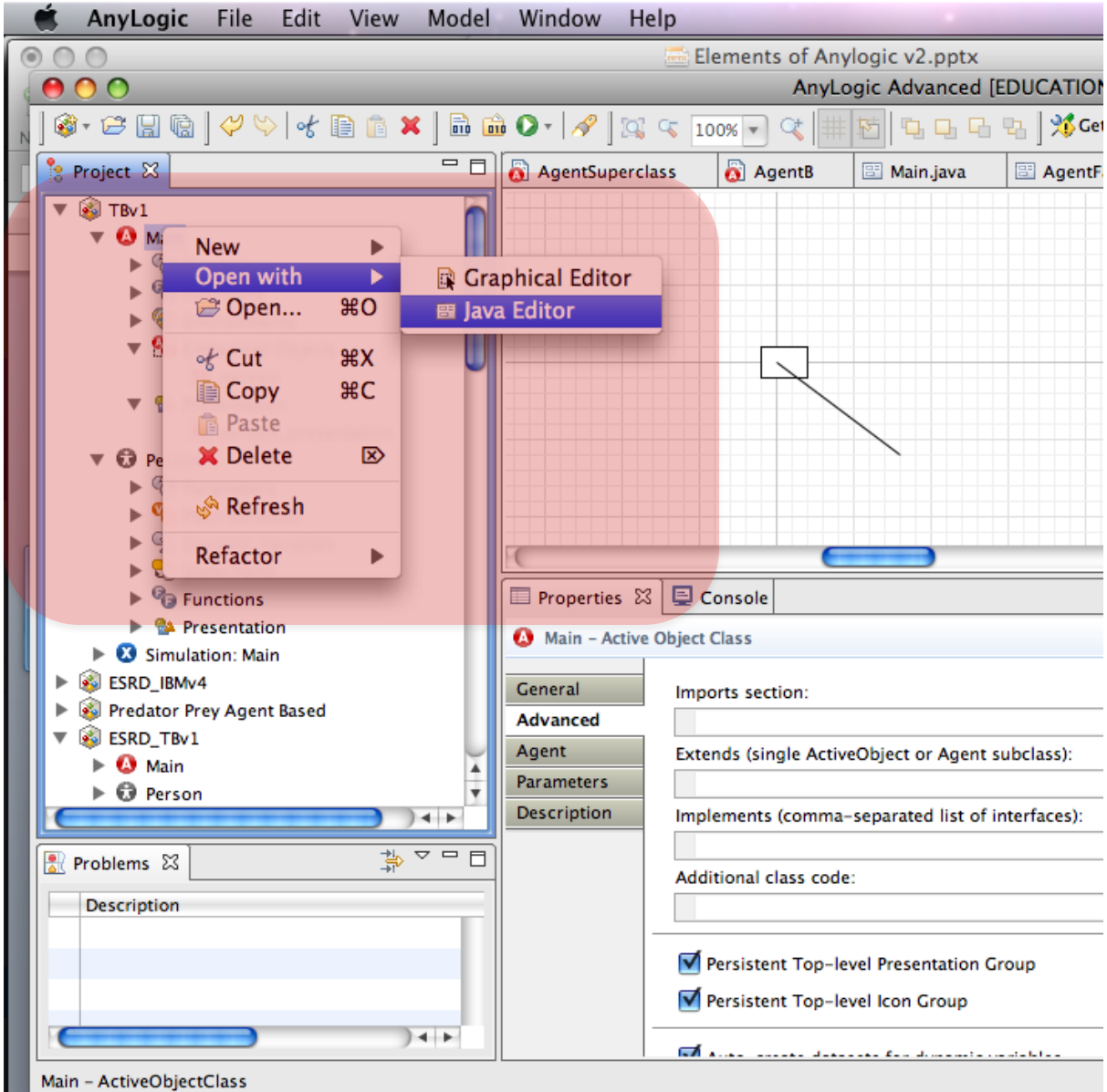
JVM Byte Code

Person.class

Inspecting the Java code

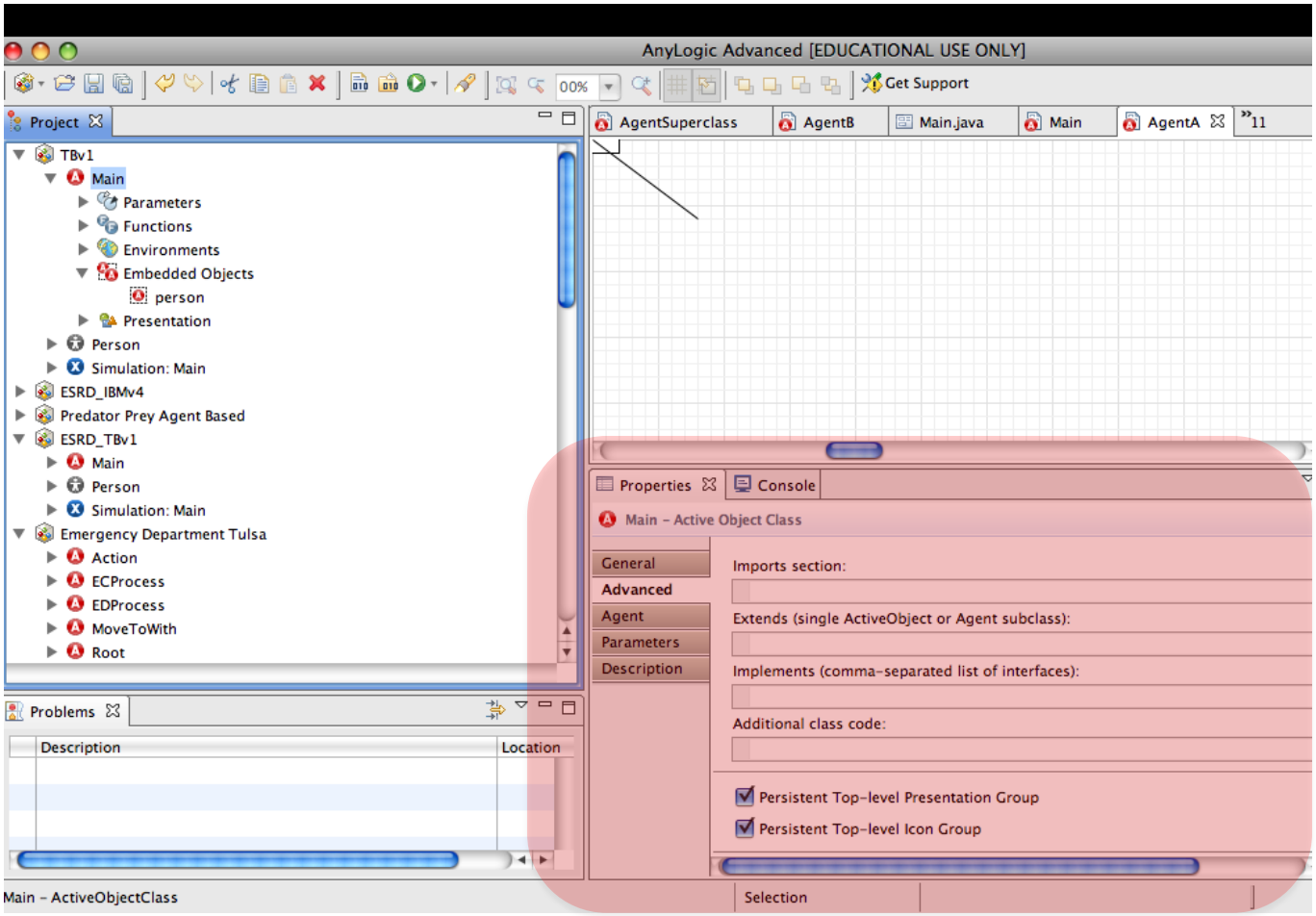
- As a step towards creating an executable representation of the code, AnyLogic creates a Java representation
  - If you want to see the Java code for a model, you will need to do a “build”
- Sometimes it can be helpful to look at this Java code
  - To find errors about which AnyLogic may be complaining
  - Advanced: To see how things are being accomplished or “work”

Requesting Viewing of Java Code



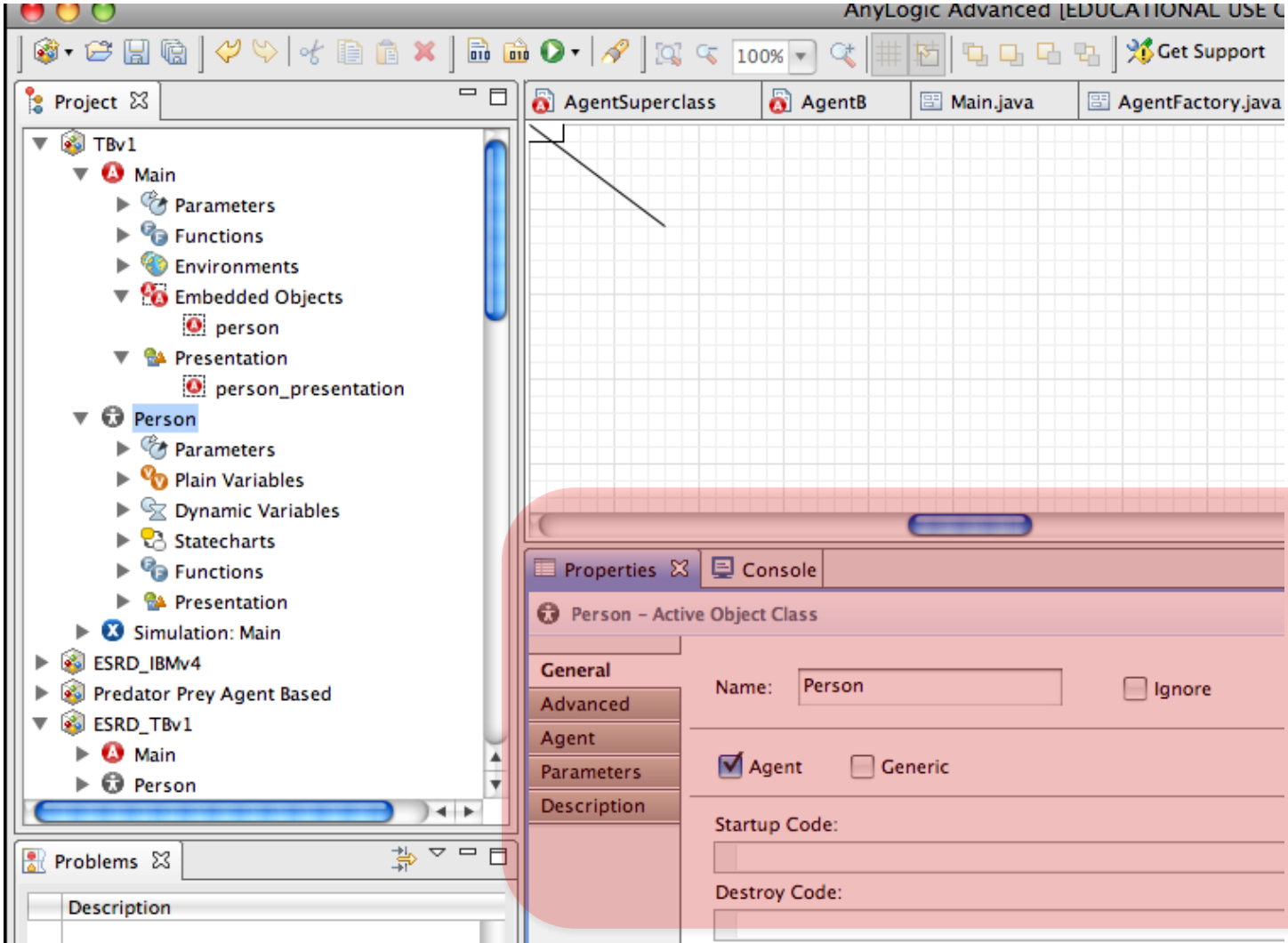
Examples of Where to Insert Code Object Properties

- “Advanced”



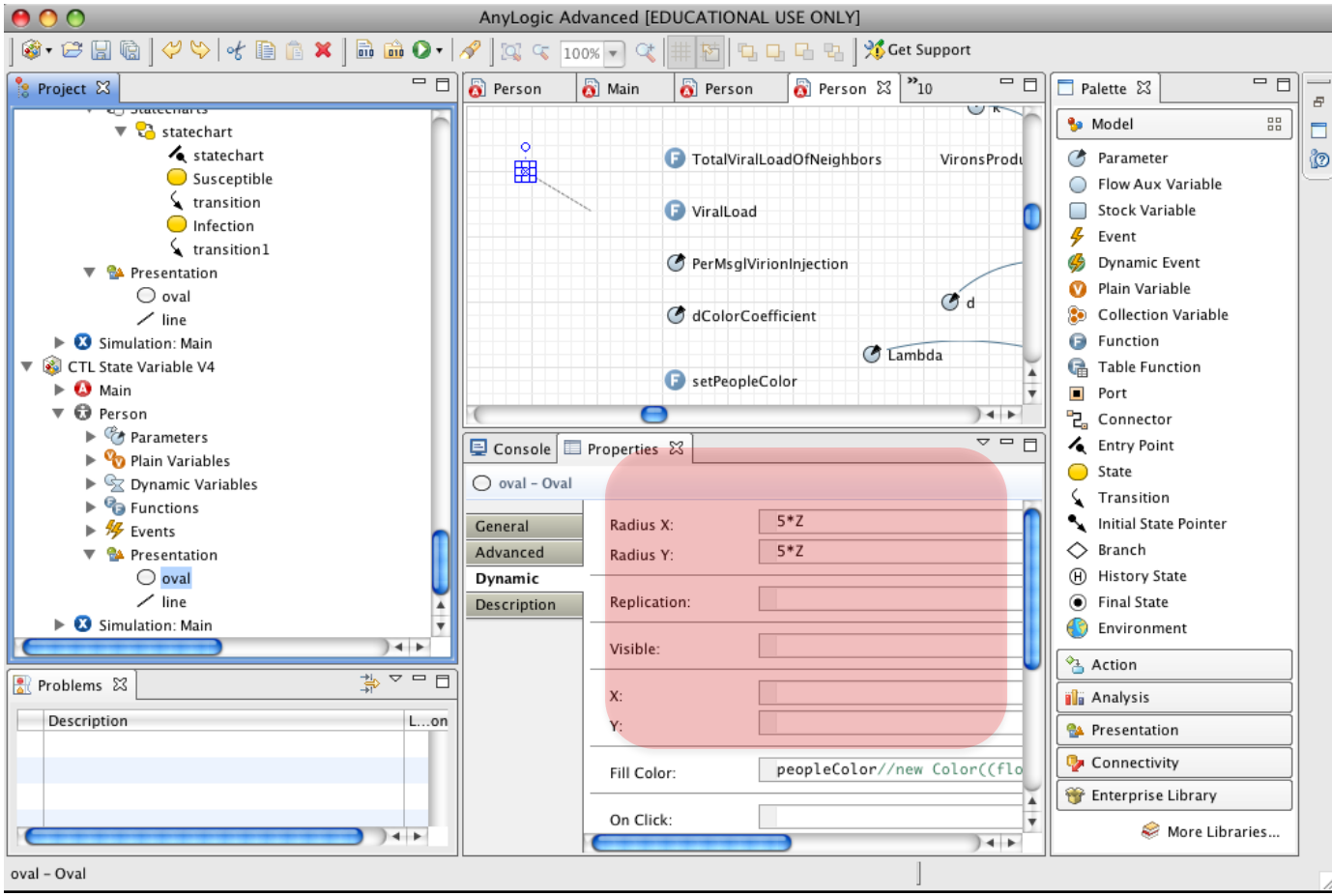
Examples of Where to Insert Code Object Properties

- "General"



Example of Where to Insert Code Presentations Properties

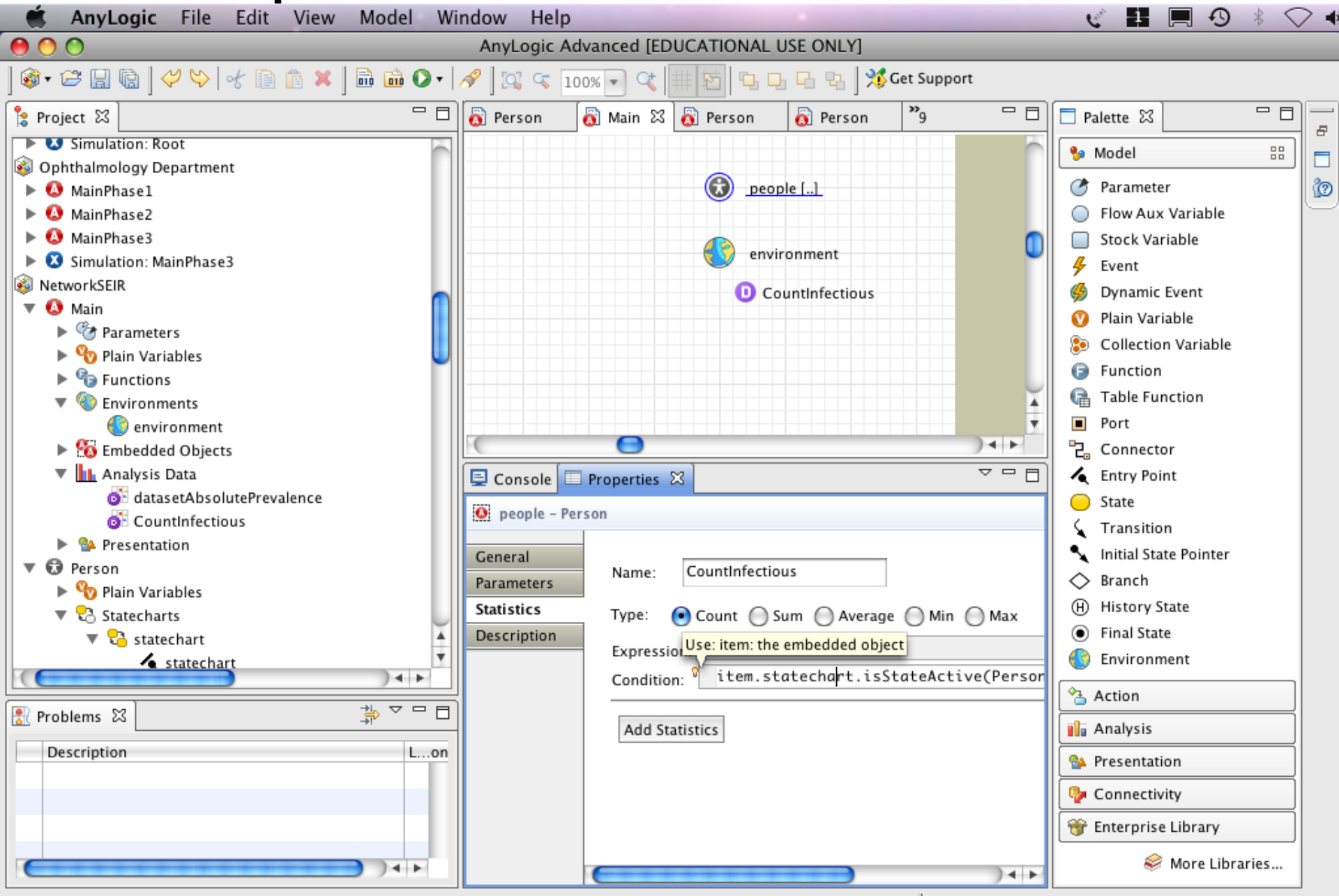
- "Dynamic" properties of presentation elements (especially of Agents)



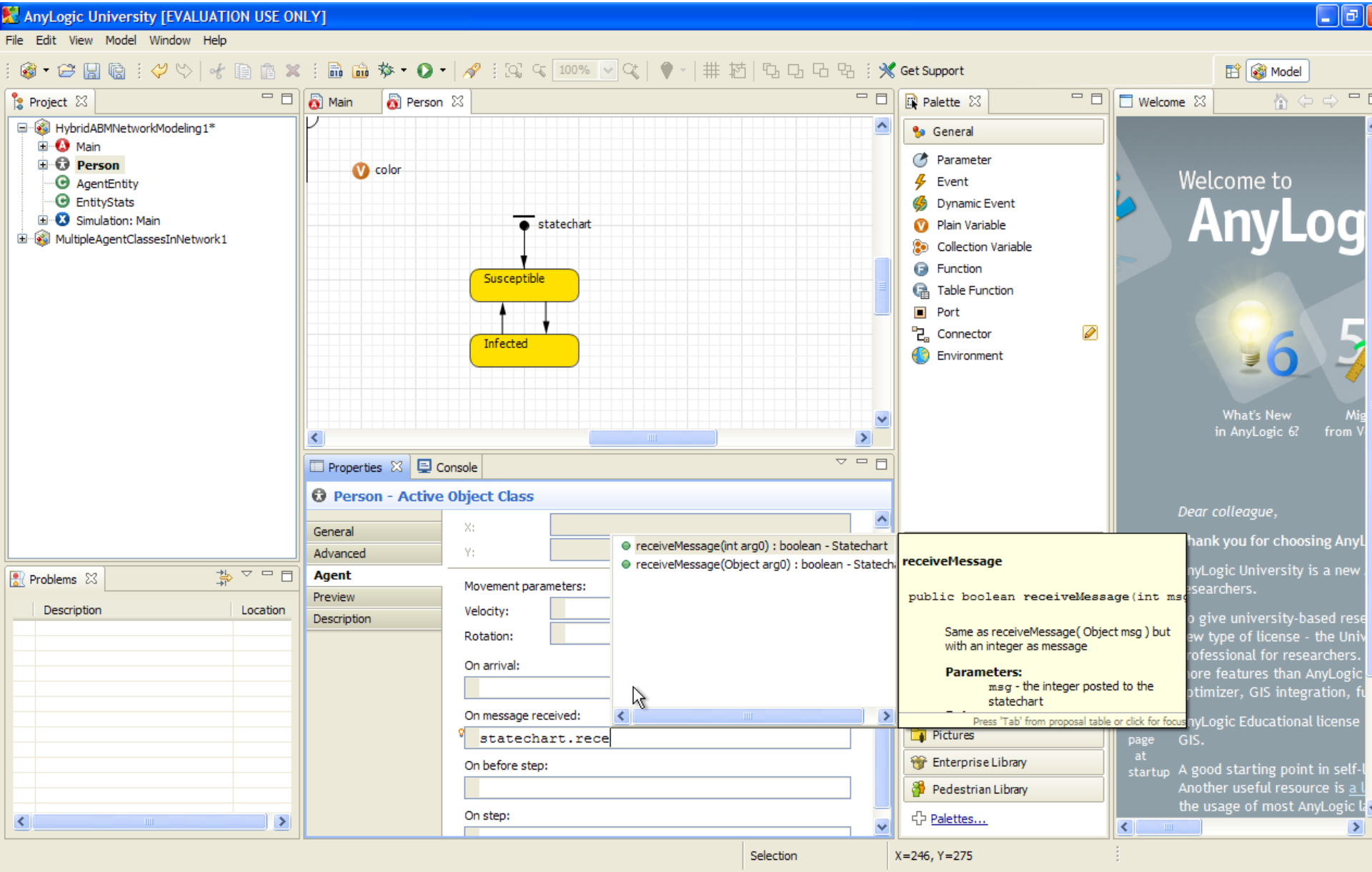
Tips to Bear in Mind While Writing Code

- Click on the “light bulb” next to fields to get contextual advice (e.g. on the variables that are available from context
- While typing code, can hold down the Control key and press the “Space” key to request autocompletion
  - This can help know what parameters are required for a method, etc.
- Java is case sensitive!
- Can press “Control-J” to go to the point in Java code associated with the current code snippet
- Can press “build” button after writing snippet to increase confidence that code is understood

Example of Contextual Information



Autocompletion Info (via Control-Space)



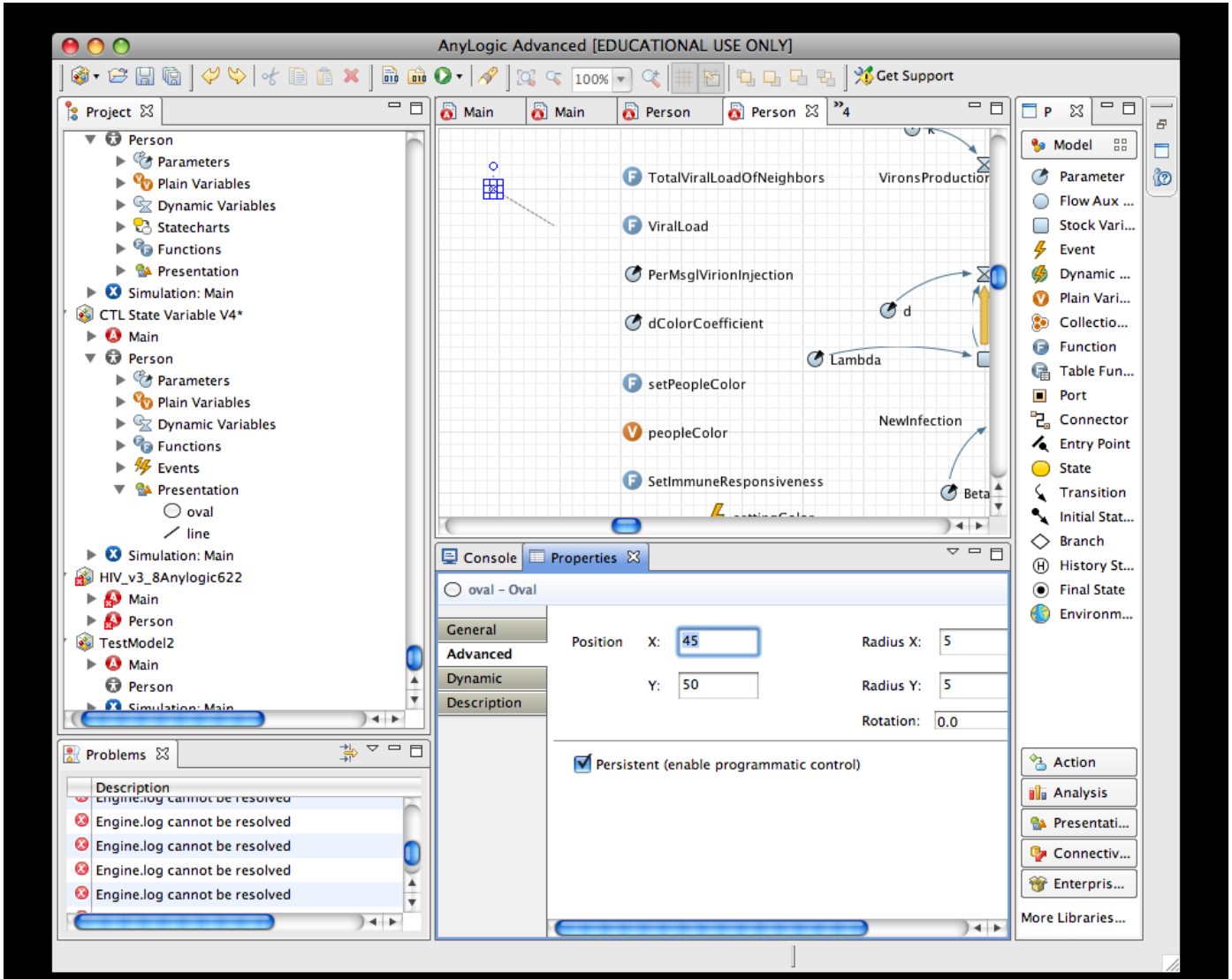
Finding the Enclosing “Main” class from an Embedded Agent

- From within an embedded Agent, one can find the enclosing “Main” class by calling get_Main()
  - This will give a reference to the single instance (object) of the Main class in which the agent is embedded
  - An alternative approach is to call ((Main) getOwner)

Presentation Properties

- Both key customizable classes (“Main”, various Agent classes) can be associated with “Presentation” elements
- These elements are assembled during execution into animations & presentations of the agents
- Many of these presentation elements have properties that can be set to Java expressions

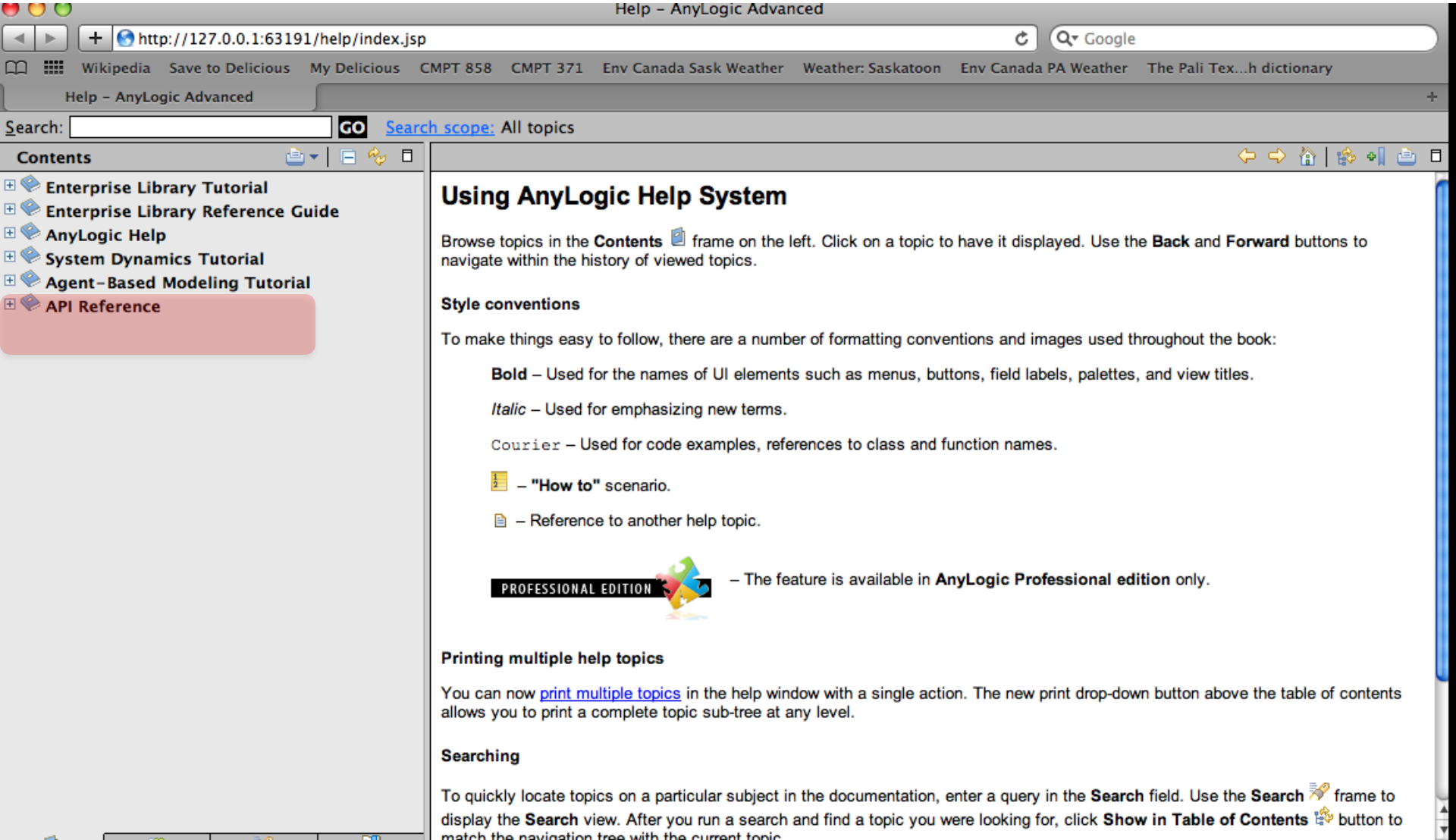
Enabling Programmatic Control



Getting to the AnyLogic Help

- Choose “Help”/”Help Contents”
- AnyLogic help includes many components
  - Tutorials
  - User references
  - AnyLogic “library” information

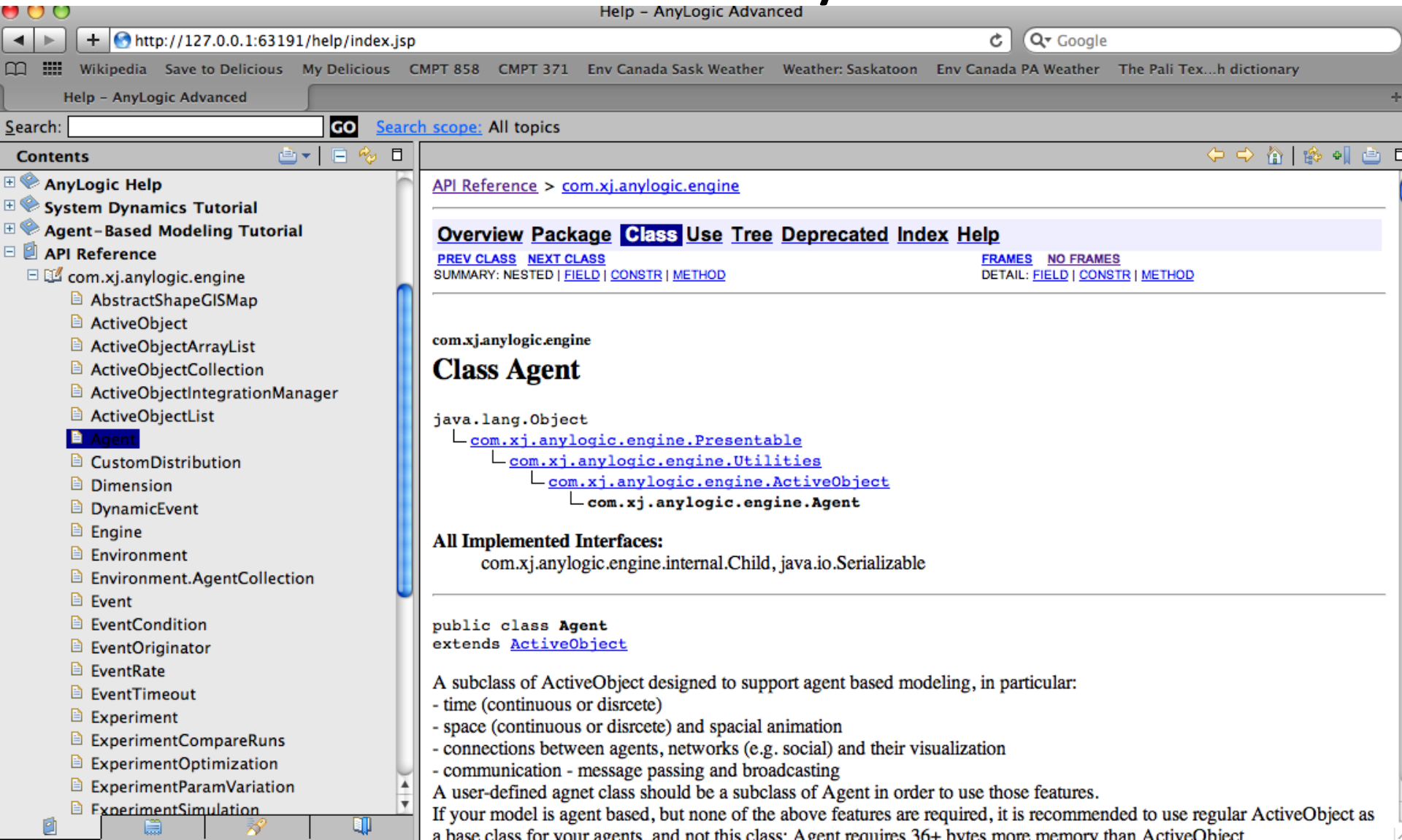
Getting Information on the Anylogic (Java) Libraries



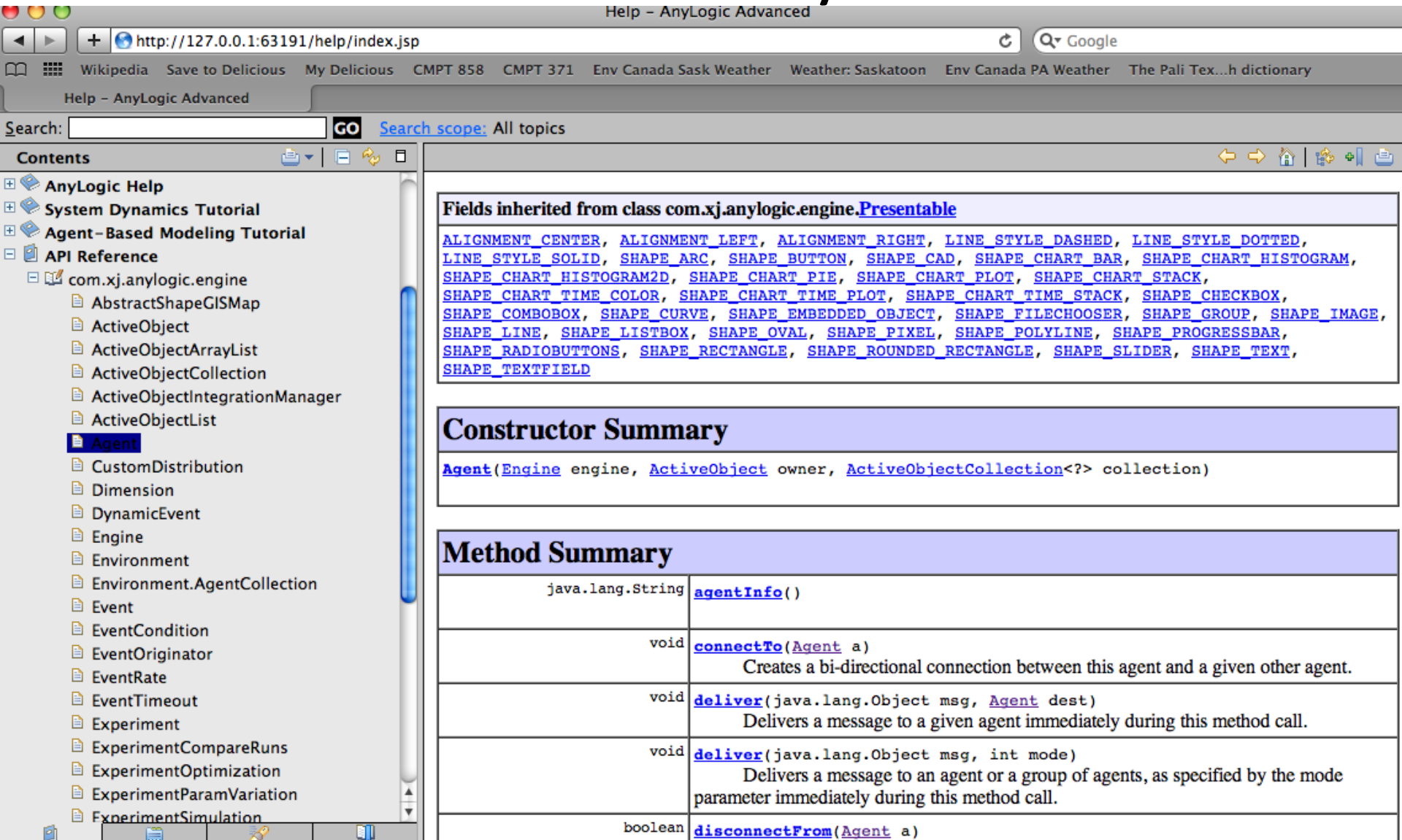
The Notion of a Code “Library”

- A “library” lets third parties (e.g. xjtek) share compiled code they have developed with others
- The classes built into our AnyLogic projects (e.g. Agent, ActiveObject, NetworkResourcePool, etc.) are contained in the library
- The available libraries that come with AnyLogic & Java have many additional components that can offer tremendous additional functionality
  - By tapping into this functionality, we can avoid having to write code ourselves
- To use a library, you need to know what is in it!

Finding out Information Interfaces for Library Elements 1



Finding out Information
Interfaces for Library Elements 2



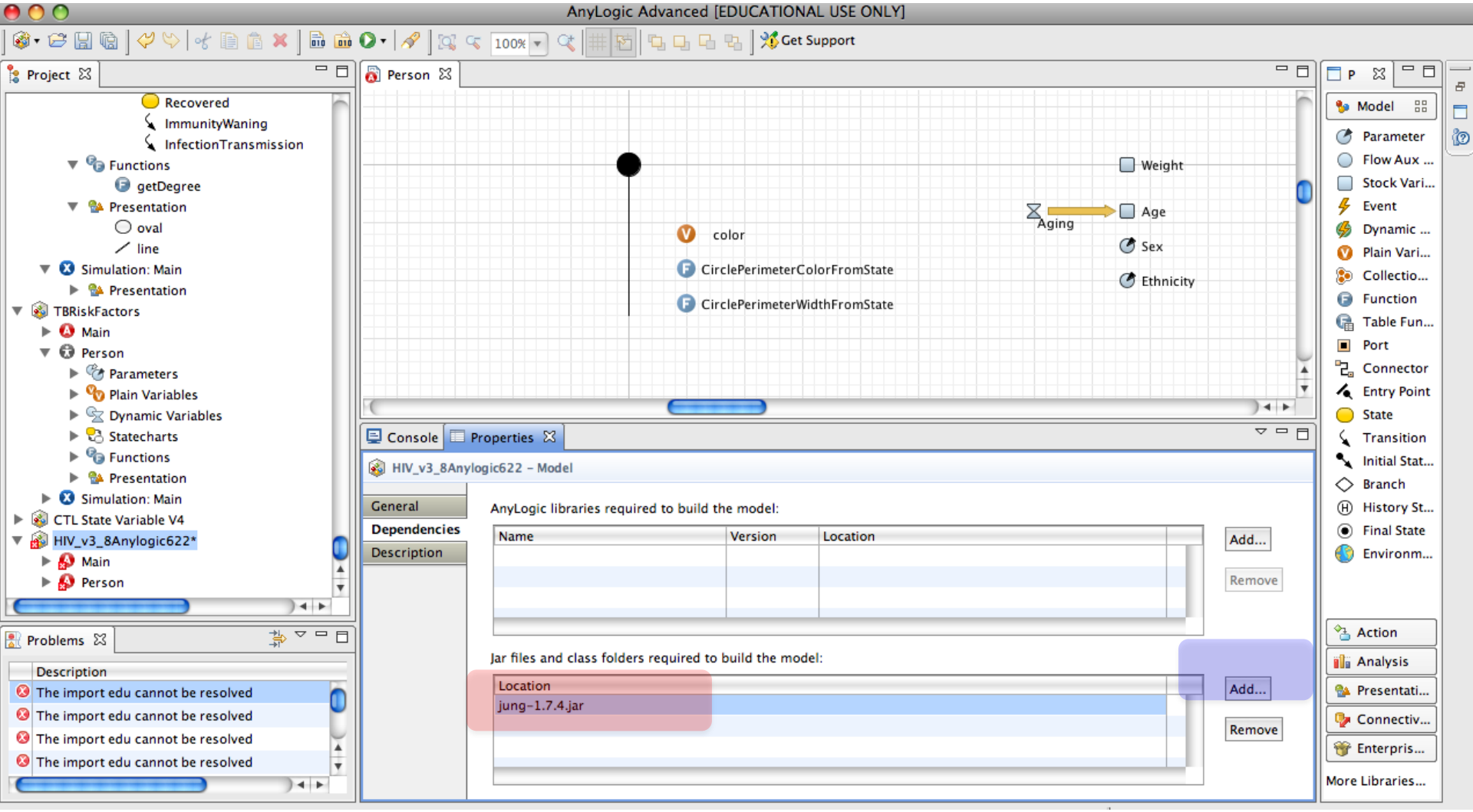
Using Libraries

- There are two major libraries that are “built in” and can be used without additional reference: Java libraries & AnyLogic libraries
- To use an object in the Java libraries, you will use an “import” statement

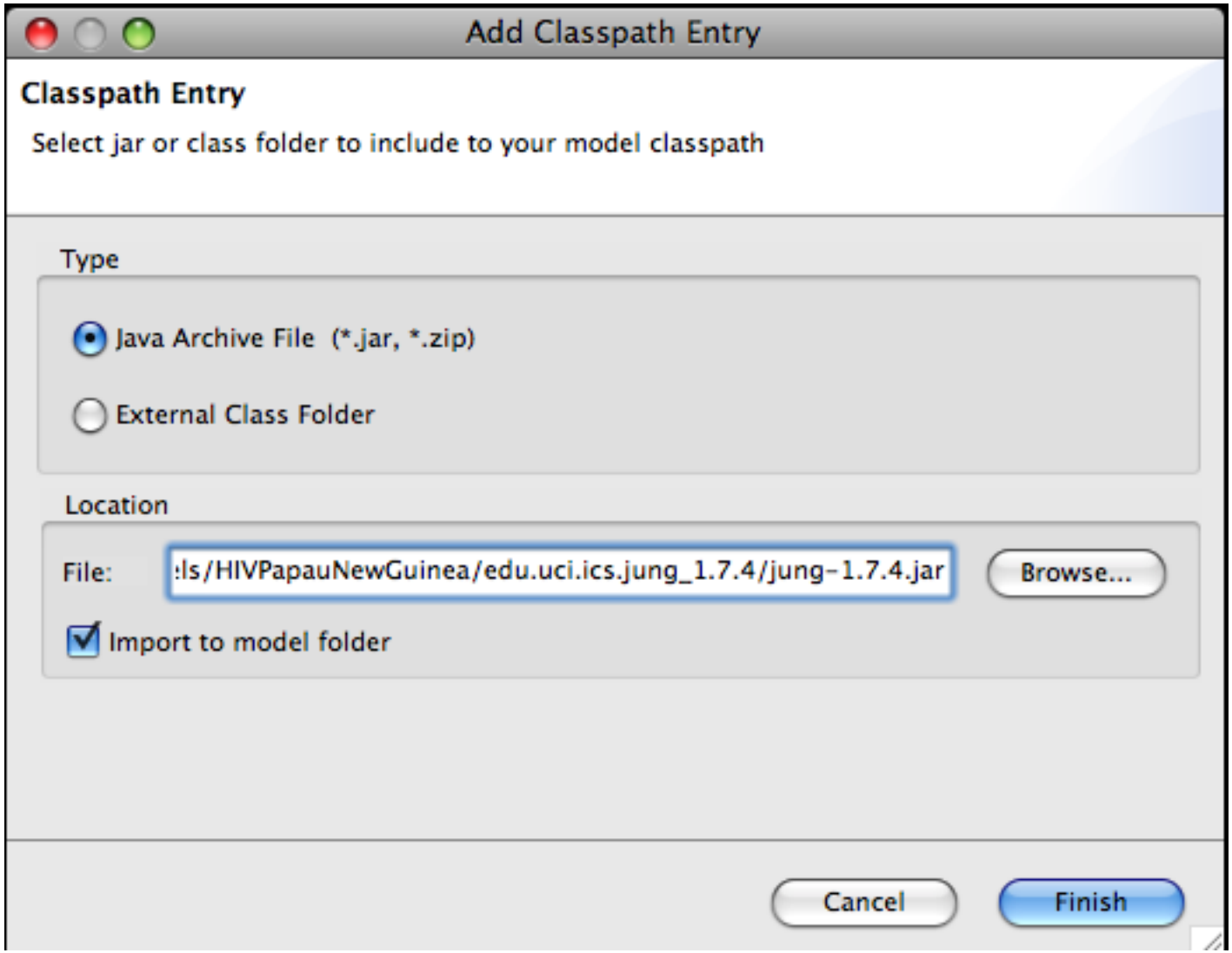
Using External Libraries

- There are tremendous numbers of 3rd party libraries available for Java
- The functionality associated with these libraries is incredibly diverse
- Many of these libraries are available for free; others are sold
- It is very easy to make use of the functionality of 3rd party libraries from AnyLogic
  - In order to do this, AnyLogic needs to “know about” the external library.

Adding External Libraries 1



Adding External Libraries 2



Common Contextual Variables that are Used by Code Snippets

- In statistics: "item" indicates current agent
- In "On Message Received" handler for agent: "msg" indicates received message
- In Dynamic properties of an Agent's replicated line property: "index" indicates current person's index
- In "Parameters" properties of Agent populations (used to set properties of agents within population): "index" indicates the index of the current agent in the population

Example code to Export Dataset

FileOutputStream fos = **new FileOutputStream(“Filename”);**

PrintStream p = **new PrintStream(fos);**

p.println(datasetName.toString()); // outputs comma delimited values

Useful Bits of Java Code

- get_Main() gets reference to Main object
- ActiveObject.trace(str) outputs string to log
- Engine.getTime() gets the current time
- agents.size() gets number of objects in collection agents
- agents.item(i) gets item i from agent collection
- uniform() generates a random number from 0..1

Useful Bits of Java Code : General Expressions

- ActiveObject.traceln(Stringstr) outputs string to log
- time() gets the current internal model time (different from the time in the external world)
- Members of **com.xj.anylogic.engine.Utilities**
  - uniform() generates a random number from 0..1
  - uniform(x) gen. a random number in range 0 to x
  - lognormal(double meanNormal, double sigmaStdDevNormal, double minNormal) draws from a lognormal distribution
  - normal(double meanNormal, double sigmaStdDevNormal) draws from a normal distribution
  - Many other probability distributions

Methods on Populations of Agents (in Main class)

- population.size() gets number of objects in collection *population*
- *population.statName()* retrieves the current value of the population statistic statName, as computed for population *population*.
- population.item(int i) gets item i from *population* collection
- add_*populationname()* *Adds agent to that population*
- remove_*populationname()* *Removes agent from that population*

Useful Java Code: Methods to Call on (or from within, using "this") an Agent

- a.getConnectionsNumber() returns number of connections between this agent and others
- get_Main() gets reference to Main object
- toString() gets string rendition of agent
- a.getConnections() gets a collection (linked) list of agents to which this agent is connected (& over which we can iterate)
- a.connectTo(Agent b) connects a to b
- a.disconnectFrom(Agent b) disconnects b from a
- a.disconnectFromAll() disconnects all agents from a
- a.getConnectedAgent(int i) gets the ith agent connected to a
- a.isConnectedTo(Agent b) indicates if a is connected to b

Methods on Statecharts (Called from within Agent code)

- isStateActive(int*statename*) indicates whether agent is in a given state (composite or simple)
- getActiveSimpleState() Get number of simple state. Can then compare to different state names, e.g. in switch statement.

Methods on Process Flow Diagrams

- *source*.inject(int count) injects a count of entities into the *source* object (i.e. into an object of type Source)

Gotchas

- Changing rates for leaving a state do not get updated until leave & reenter state (including by a self-transition)

Example Use of getActiveSimpleState

```
switch (TBProgressionStatechart.getActiveSimpleState())
{
    case LTBI:
     return Color.YELLOW;
    case UnDiagnosedActiveTB:
     return Color.RED;
    case DiagnosedActiveTB:
     return Color.ORANGE;
    case TBSusceptible:
    default:
     return Color.BLACK;
}
```

Useful Snippets: Handling Messages

- Sending
  - sender.deliver(msg, receiver) immediately deliver a message from sender to receiver
  - sender.send(msg, receiver) deliver a message from sender to receiver
  - environment.deliverToRandom(msg) [within Main] immediately deliver a message to a random agent in the environment
  - send("Infection", RANDOM_CONNECTED) [within an agent] send a message to a randomly selected agent connected to this one (where those agents are selected w/uniform prob)
- Receive message
  - TBProgressionStatechart.receiveMessage(msg) to forward message received by agent to statechart

Useful Snippets

- Fields of dynamic properties of line object for Agent Presentation (Under "*Dynamic*" tab of line's properties)
  - Replication: getConnectionsNumber()
  - dX: getConnectedAgent(index).getX() - getX()
  - dY: getConnectedAgent(index).getY() - getY()
  - These basically allow for appropriate initiation of visual properties of the inter-agent connections
- In Agent's "On Message Received" Handler (Under "Agent" tab of Person)
  - *statechartname*.receiveMessage(msg)
  - This forwards a message received by this agent to statechart; note that if there are different messages, destined for different statecharts, they can be dispatched here to different targets