Classes, Objects & References

Nathaniel Osgood
MIT 15.879

March 14, 2012

What is a Class?

- A class is like a mould in which we can cast particular objects
  - From a single mould, we can create many “objects”
  - These objects may have some variation, but all share certain characteristics – such as their behaviour
    - This is similar to how objects cast by a mold can differ in many regards, but share the shape imposed by the mould
- In object oriented programming, we define a class at “development time”, and then often create multiple objects from it at “runtime”
  - These objects will differ in lots of (parameterized) details, but will share their fundamental behaviors
  - Only the class exists at development time
- Classes define an interface, but also provide an *implementation* of that interface (code and data fields that allow them to realized the required behaviour)

Recall: A Critical Distinction: Design (Specification) vs. Execution (Run) times

- The computational elements of Anylogic support both design & execution time presence & behaviour
  - Design time: Specifying the model
  - Execution time ("Runtime"): Simulating the model
- It is important to be clear on what behavior & information is associated with which times
- Generally speaking, design-time elements (e.g. in the palettes) are created to support certain runtime behaviors

Recall: A Familiar Analogy

- The distinction between model design time & model execution time is like the distinction between
  - Time of Recipe Design: Here, we're
    - Deciding what exact set of steps we'll be following
    - Picking our ingredients
    - Deciding our preparation techniques
    - Choosing/making our cooking utensils (e.g. a cookie cutter)
  - Time of Cooking: When we actually are following the recipe
    - A given element of the recipe may be enacted many times
      - One step may be repeated many times
      - One cookie cutter may make many particular cookies

Cooking Analogy to an Agent Class: A Cookie Cutter

- We only need one cookie cutter to bake many cookies
- By carefully designing the cookie cutter, we can shape the character of many particular cookies
- By describing an Agent class at model design time, we are defining the cookie cutter we want to use

Familiar Classes in AnyLogic

- Main class
- Person class
- Simulation class

Work Frequently Done with Objects

- Reading “fields” (variables within the object)
- Setting fields
- Calling methods
  - To compute something (a “query”)
  - To perform some task (a “command”)
- Creating the objects

"Methods" to Call on (or from within, using "this") an Agent

- a.getConnectionsNumber() returns number of connections between this agent and others
- a.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

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())

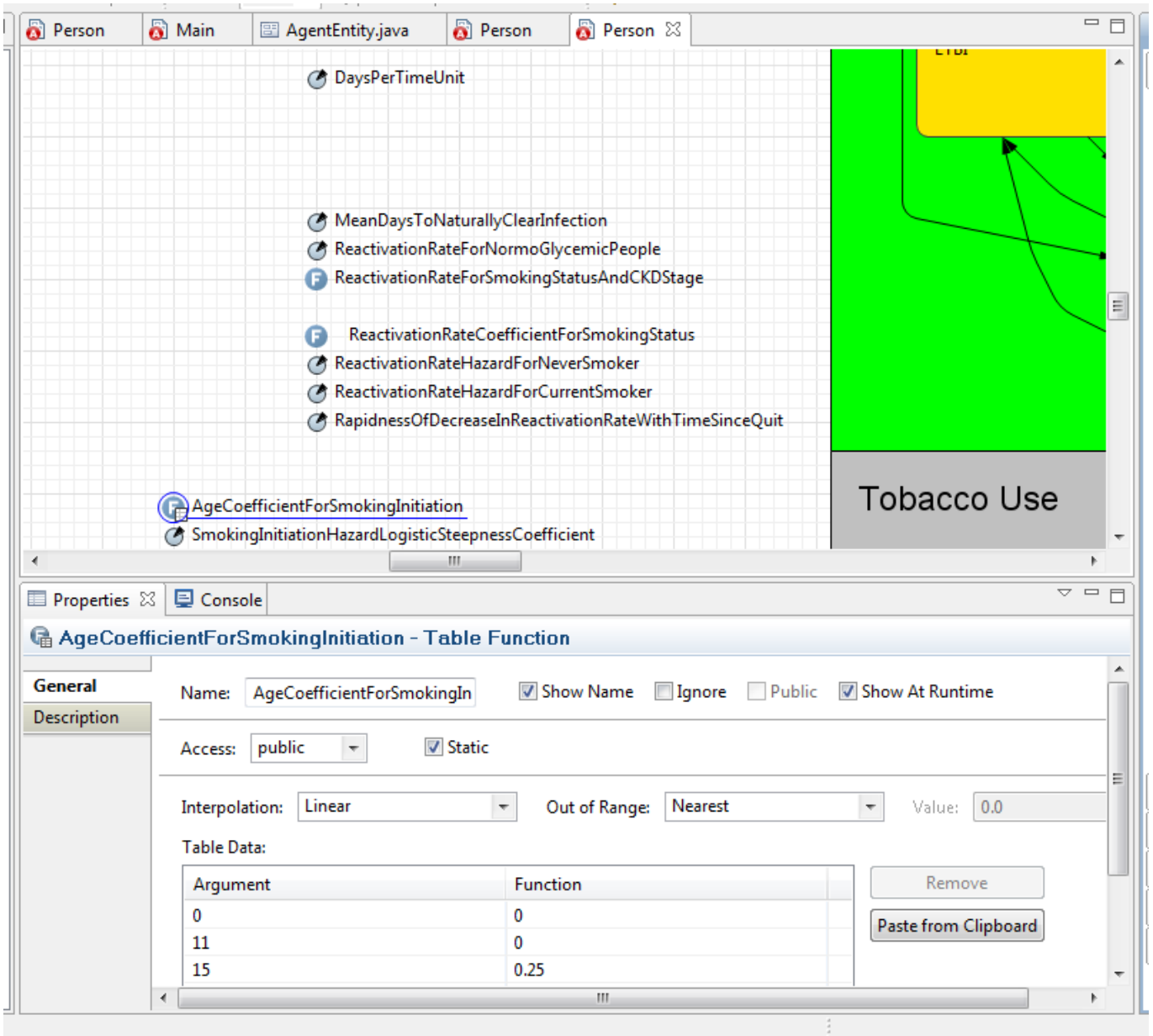
Composition of Methods

- Suppose we have an agent called a
- a.getConnectedAgent(2).toString()
  - This will print out the “name” of the 3rd agent to which a is connected
- a.getConnectedAgent(0).getConnectionsNumber()
  - This will print out the number of connections possessed by the 1st agent to which a is connected

Distinction between Class and Object

- Sometimes we want information or actions that only relates to the class, rather than to the objects in the class
  - Conceptually, these things relate to the mould, rather than to the objects produced by the mould
  - For example, this information may specify general information that is true regardless of the state of an individual object (e.g. agent)
  - We will generally declare such information or actions to be “static” (e.g. static methods, static variables)

Example “Static” (Non-Object-Specific) Method



Values & References

- In Java, variables hold values
  - It is the contents of these variables that is of interest – variables themselves just store values
- There are many types of variables could be
  - Parameters to a function
  - “Local” (temporary) variables within a function
  - Variables within a class (to be found in every object that is “instantiated” from that class
  - “Static” variables associated with a class (only one variable associated with the class – no how many objects of the class are circulating)

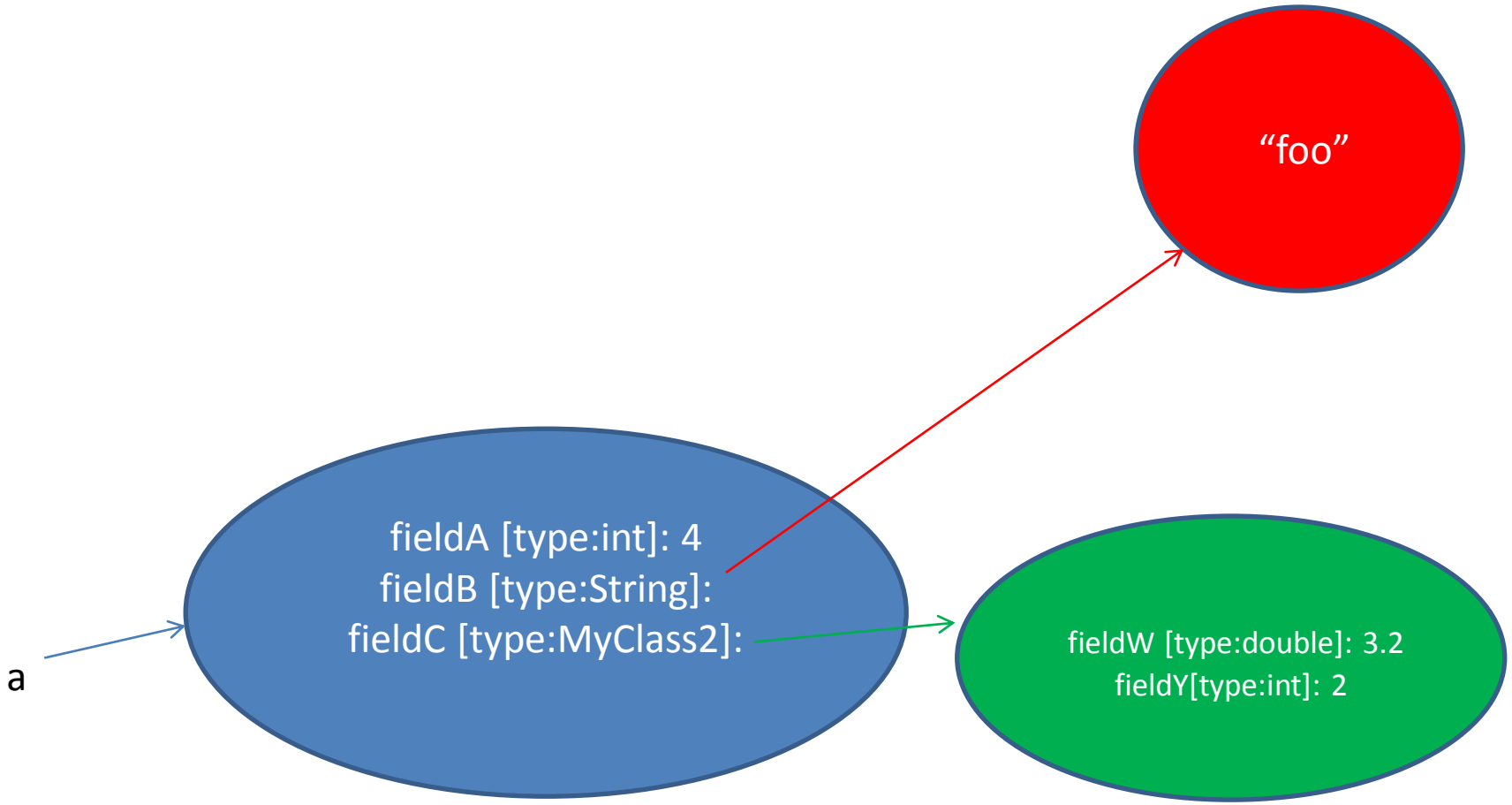
Broad Types of Java Values

- “Primitive” values
  - Here, the value is directly stored in the variable
    - int, double, float, etc.
- References
  - Here, the value within the variable actually points to either
    - An object (could have many other references to it as well!)
    - A distinguished value “null” (means “doesn’t refer to any object”)

Objects in Java

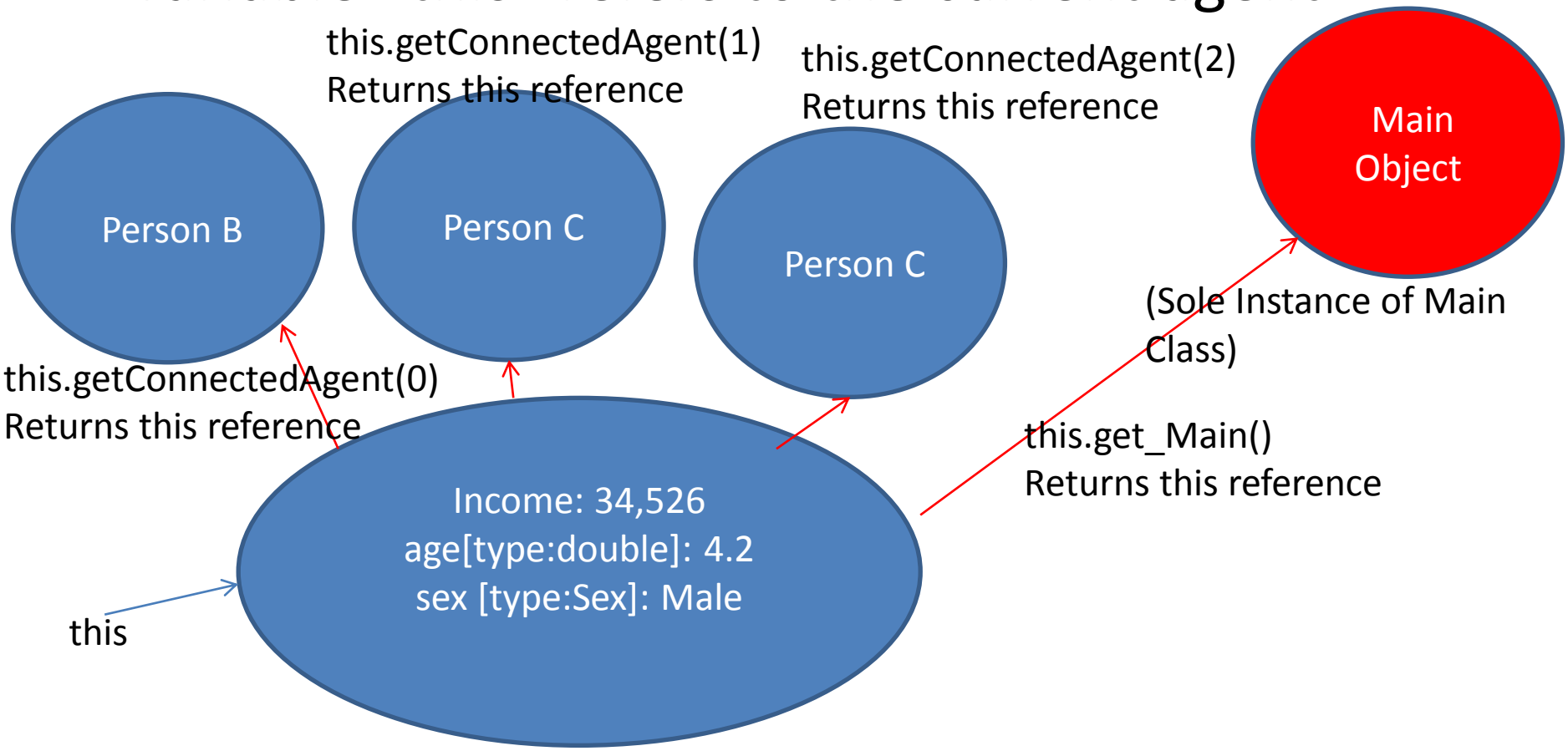
- Contain
  - Data: “Fields”, “Property”
    - These store information
  - Behavior: “Methods”/”Functions”
    - These allow the object to undertake certain tasks

Object can contain References to Other Objects



“this” Variable

- Within an agent’s method execution, the variable “this” refers to the current agent

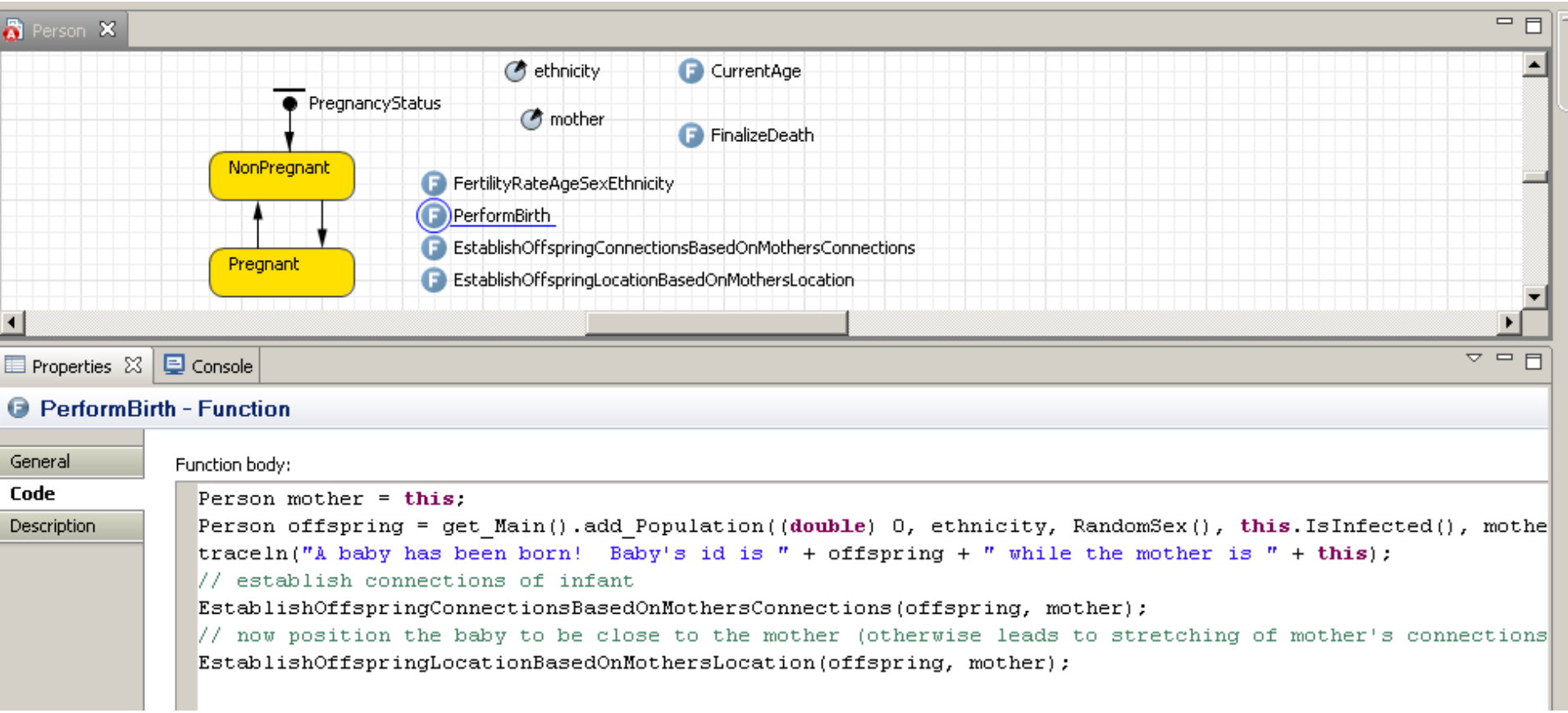


Hands on Model Use Ahead

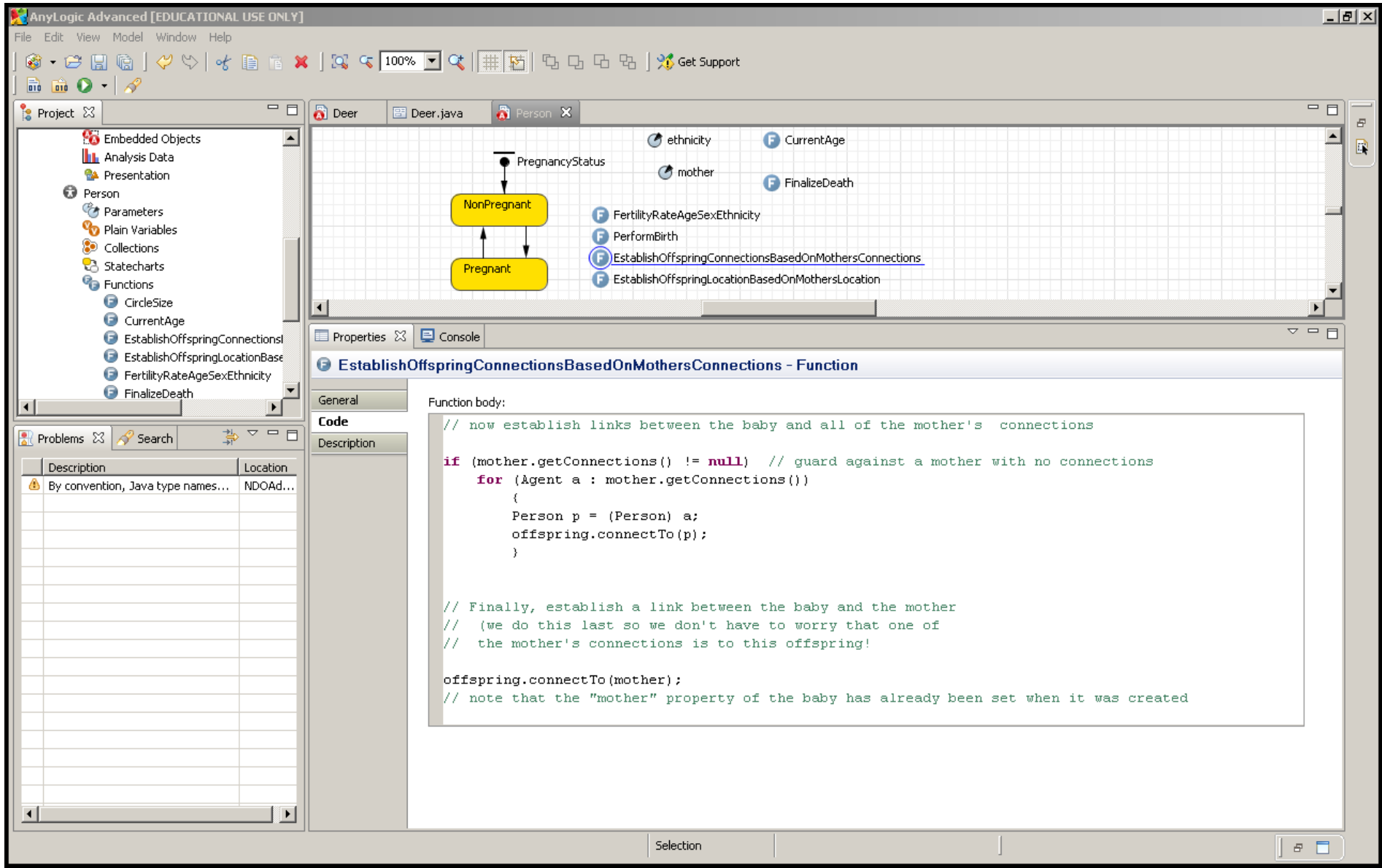
Load provided Model:

ABMModelWithBirthDeath.alp

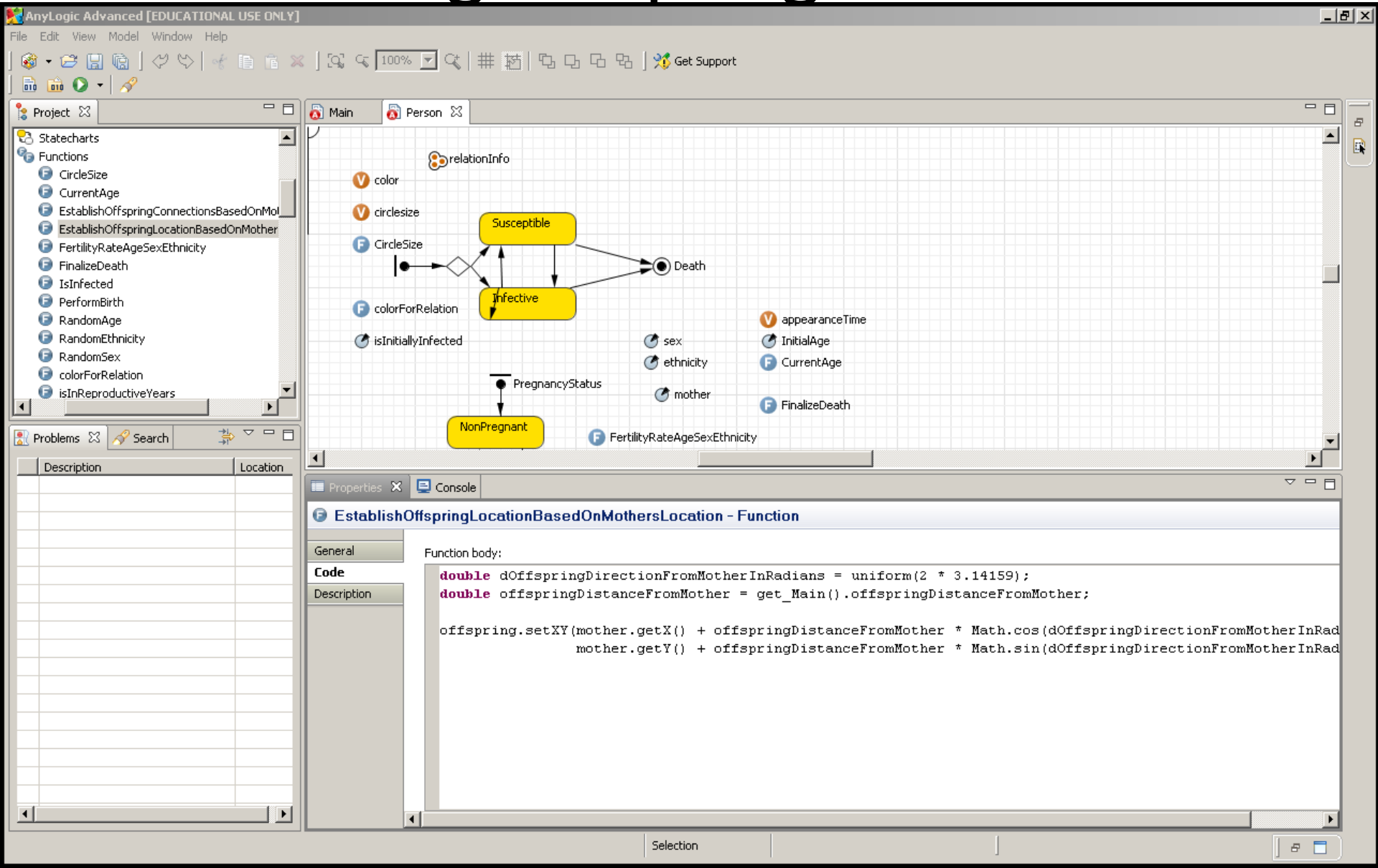
Code to Perform Birth



Establishing Baby's Connection
Looping over Connections



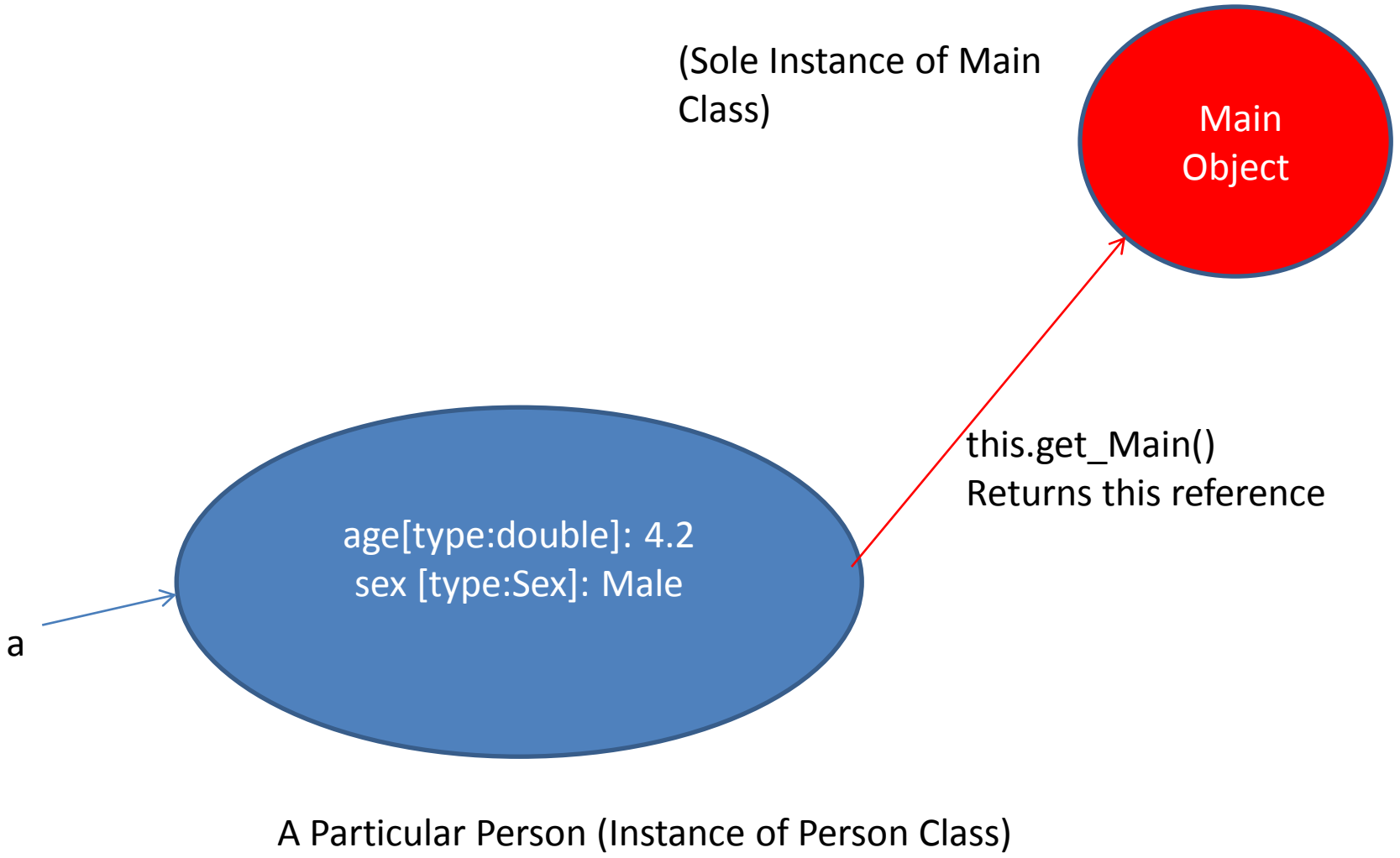
Setting Offspring Location



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)

Reference from Agent Class to Main Object

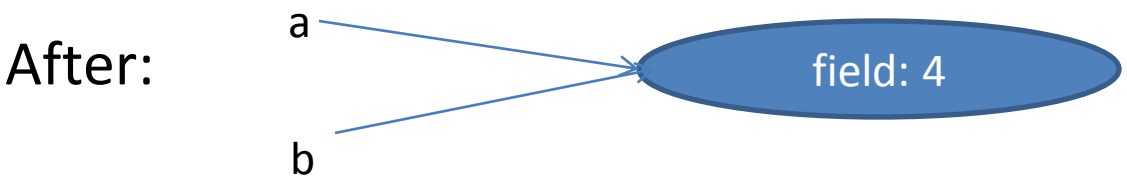


Assignment

- Consider two variables *a* and *b* that hold values
- Consider further the statement *a*=*b*
- How this is interpreted depends on the "type" of *b*
  - If *b* is a "primitive" (e.g. int, double): Here, the assignment will make a copy of that value

    Before: a: 2, b: 4

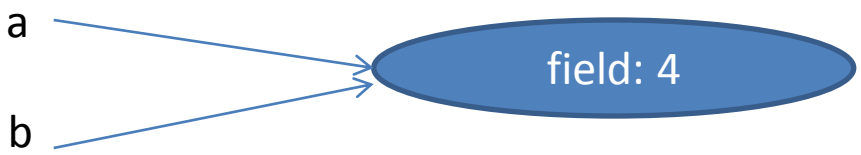
    After: a:4, b:4
  - If *b* holds a reference to an object, a will now hold a reference to that same object



Assignment

– If the programmer later modifies that object through a that same change will be visible through b as well

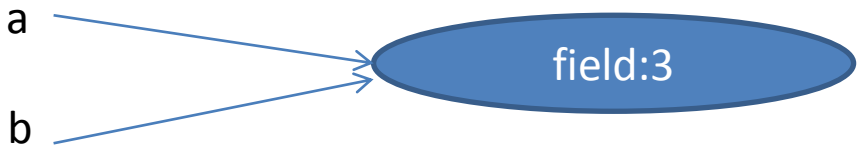
- Before



- Assignment to a “field” (property”) of the object through variable a

  a.field=3

- After



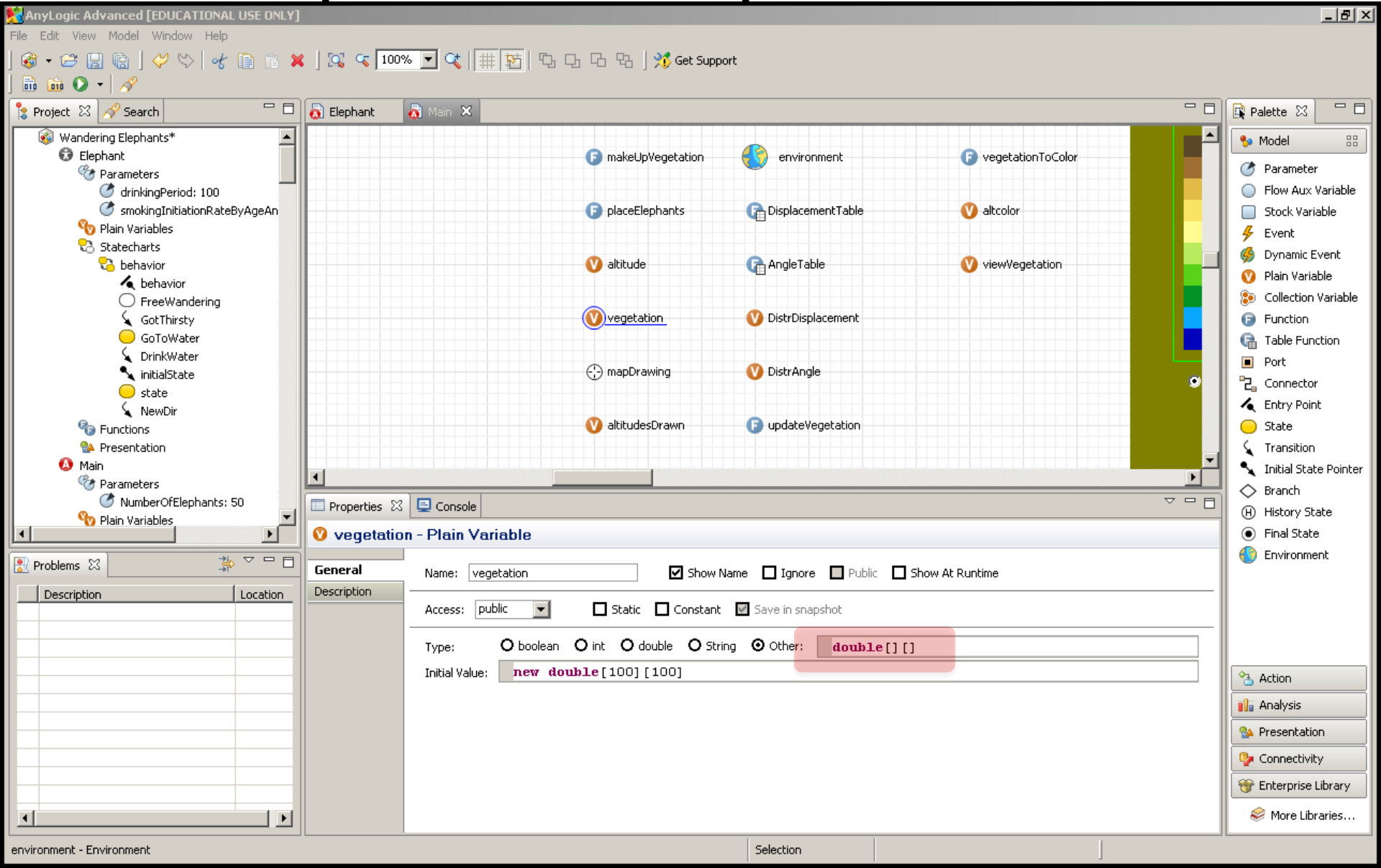
References Vs. Values

- The “type” of a variable indicates the sort of data to which it can refer
- Looking at a variable’s type will tell you much about how it can be used
  - Whether primitive or reference
  - Sort of operations that are possible on the data it holds

Arrays

- Java supports collections called “Arrays”
  - These store collections of values in an “indexed” fashion
    - By giving an “index”, we can get back an element
- These arrays can be of 1 or more “dimensions”
  - An array of dimension 2 is just a (1D) array of references to (1D) arrays

Example: Landscape Information



Good Models to Examine for Better Software Engineering Elements

- ABMClinicModelV7
- ABMModelWithBirthDeath