Basics of Java: Values & Expressions & Variables

Nathaniel Osgood

MIT 15.879

April 4, 2012

Recall: Method Bodies & “Statements”

- Method bodies consist of
  - √ Comments (mostly ignored by “build”)
  - √ Variable Declarations
  - – Statements (most involving “Expressions”)

What is a Value?

- A value is a single quantity (that cannot be further evaluated)
- Values can be of different "types" (where a type describes the legal universe of possible values)
  - Integers (*e.g.* 2, -10)
  - Floating point values (*e.g.* 2.5, 3.14159, 0.0, 1.0)
  - Characters (*e.g.* 'a', 'b')
  - Boolean (*e.g.* true/false)
  - Reference to objects that are instances of classes (*e.g.* Person, String, Main, Simulation, etc.)

Java “Expressions”

- A Java Expression *computes (“evaluates to”) a **value**, typically based on other values of the same of different types*
  - *This is like a “formula”* to compute a value
  - *For years, you have likely been writing expressions in algebra, using them in Excel, in calculators, etc.*
- *Examples: “2*5”, “1.0/3”, “(a+b)*c”, “a>b”, “this.getConnectedAgent(i).name”*
- In the process of computing the value, it may cause some changes to “program state” (e.g. change the value of a variable”
  - Assignment expression
  - Calling a method

In most places AnyLogic wants a value, we can give it a Java *Expression*.

Common Java Expressions

- Literal (3.5, 1, “my string”, { 1, 2.71, 3.14, 0}, null)
- Comparison (a>b,a==b, a<=b)
- Mathematical Operators (+,-,/,*) *Can be “overloaded” to mean other things (e.g. + as concatenation)*
- “Dereferencing”: Looking up field or value *b* in a reference to an object *a:* (*a*.*b*) (*a* is a reference to an instance of a class; *b* is a name of a field or method associated with the class of *a*, and thus with the object)
- Ternary operator: (predicate ? a : b)
- Potentially causes changes (Side effecting)
  - Assignment (a=b) *Left hand side is some location (variable, field, etc.) and variants (a++, ++a, a+=2, a*=5)*
  - Method call (function call): this.get_Main()

Additional Common Operators

- Boolean expressions
  - a&&b (logical and), a||b (logical or), !a (logical not)
- Indexing: a[20], a[getConnectionsNumber()-1]
  - Must value preceding must denote an array
- Method call: f(2,3)
- For strings

  strA+strB (concatenates strings)

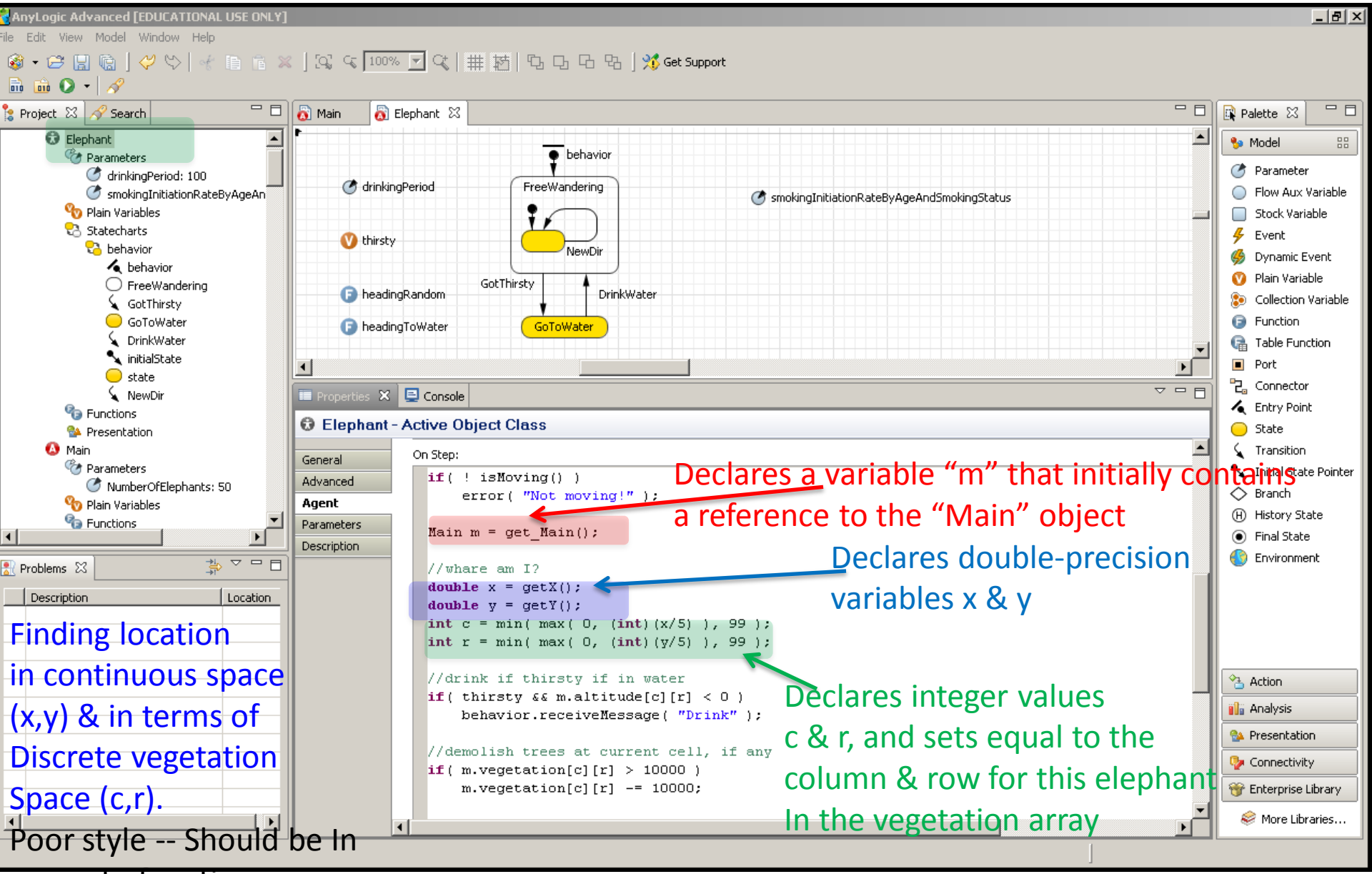
Reading Java Expressions

- Generally, expressions are calculated from "inside out", and left-to-right
  - e.g. a.getConnectedAgent(i).getName().length
- Expressions are routinely "strung together" in this way (e.g. where the left components return values used by the right components)

Variable Declarations

- Most java variable denotes location that contains a value (exceptions: constant vars, type parameters)
- Variables are associated with “types”
  - The types describe the sort of values that a variable can contain (the set of possible values), e.g.
    - double: Double precision floating point numbers
    - int: (positive & negative): Integer values within some range
    - boolean: A dichotomous value, holding “true”, or “false”
    - String: A (reference to a) text sequence
- When we “declare” a variable, we indicate its name & type – and possibly an initial value
- In Java, the value associated with a variable can change over time (as location holds different values)

Example Variable Declarations



Location (L-Values) and Assignment

- Some expressions ("L-values") denote locations
  - A variable name ("this", "mother", etc.)
  - A field name off of some reference (e.g. this.color, p.ethnicity, this.get_Main().populationSize)
  - Array references (a[20], b[i])
- The assignment operator (and its variants) puts a new value into the specified location
  - age=0
  - this.color=Color.red
  - p.ethnicity=randomEthnicity()
  - this.get_Main().populationSize=100

Varieties of Variables

- Java variables can be found in many different contents.
- These variables exhibit a uniformity of general use, but differ in terms of their
  - Lifetimes (scope)
  - Accessibility

Some Common Varieties of Variables

- Variables associated with a method
  - (Java) Formal parameters
  - Local variables (associated with statement blocks)
- Fields ("Instance variables"): Variables associated with objects
- Class variables: Variables associated with the class rather than objects of that class
- NB: There are other types of variables not covered here (e.g. type parameters for generics)

Variable Scope

- The location associated with a variable only exists for a certain length of time
- In many – but not – all cases, the lifetime of the variable's location can be considered the same as the time over which we can access this location
- We term the "scope" of the variable the region of a Java program that can "see" the variable
- If we have a variable that refers to an object (i.e. that holds a reference to an object), and the variable disappears, the object need not disappear!
  - There may also be many other references to this object !

Variable Lifetimes/Scopes

Variable variety	Lifetime
Parameter	Method
Local variable	Enclosing statement
Field	As long as object exists (until last reference is eliminated)
Static (“class”) variable	As long as class is loaded (most commonly, entire duration of execution)

Variables and Assignment Expressions

- Because (most) variables denote locations, such variables can be assigned to
  - Exceptions: Const variables, type parameters
- As an inheritance from the “C” language, Java supports a rich repertoire of assignment expressions

  a=b, a+=b, a*=b, a/=b, a-=b, a++, ++a, a--, --a

Variables and Assignment Expressions Assignment Variants

- These variants provide useful shorthand to
  - Get a value just before/after modifying it (danger!)
  - Update a value based on its previous value & return the resulting value

Post- and Pre- Increment / Decrement

- Suppose before any of the below, we have a=3
- a++ (post-increment: returns current value of a, but increments a by 1 immediately thereafter). This will evaluate to 3, but a will be 4)
- ++a (pre-increment: increments a by 1 and returns resulting value of a). This will return 4, and a will be 4)
- a-- (post-decrement: returns current value of a, but decrements a by 1 immediately after). This will evaluate to 3, but a will be 2)
- --a (pre-decrement: decrements a by 1 and returns resulting value of a). This will return 2, and a will be 2)

Operate & Assign

- These operators perform some calculation based on the existing value in a location, update the value in that location with the resulting value, and return that result
- Here, the location is always updated with the value that results from the calculation
- Suppose before any of the below, we have a=3
  - a*=2 (will evaluate to 6; a will be 6) a = (a * 2)
  - a/=3 (will evaluate to 1; a will be 1) a = (a / 3)
  - a+=2 (will evaluate to 5; a will be 5) a = (a + 2)
  - a-=2 (will evaluate to 1; a will be 1) a = (a – 2)

NB: Generalizability of the Preceding

- Because all of the preceding operate on l-values (names for locations), they can be applied to other l-values, e.g.

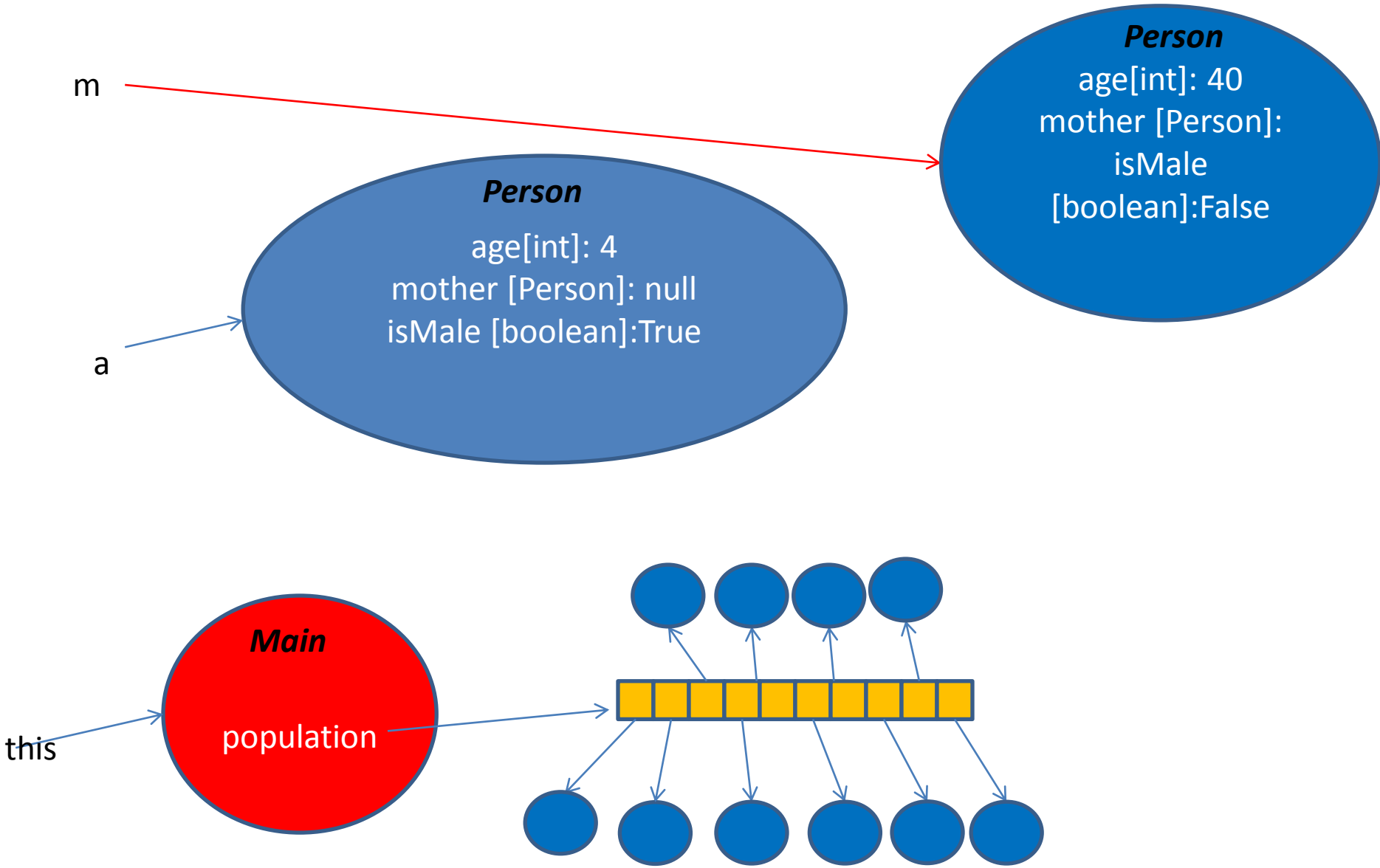
  a[20]++

  p.getConnectedNeighbor(2).income *= 2

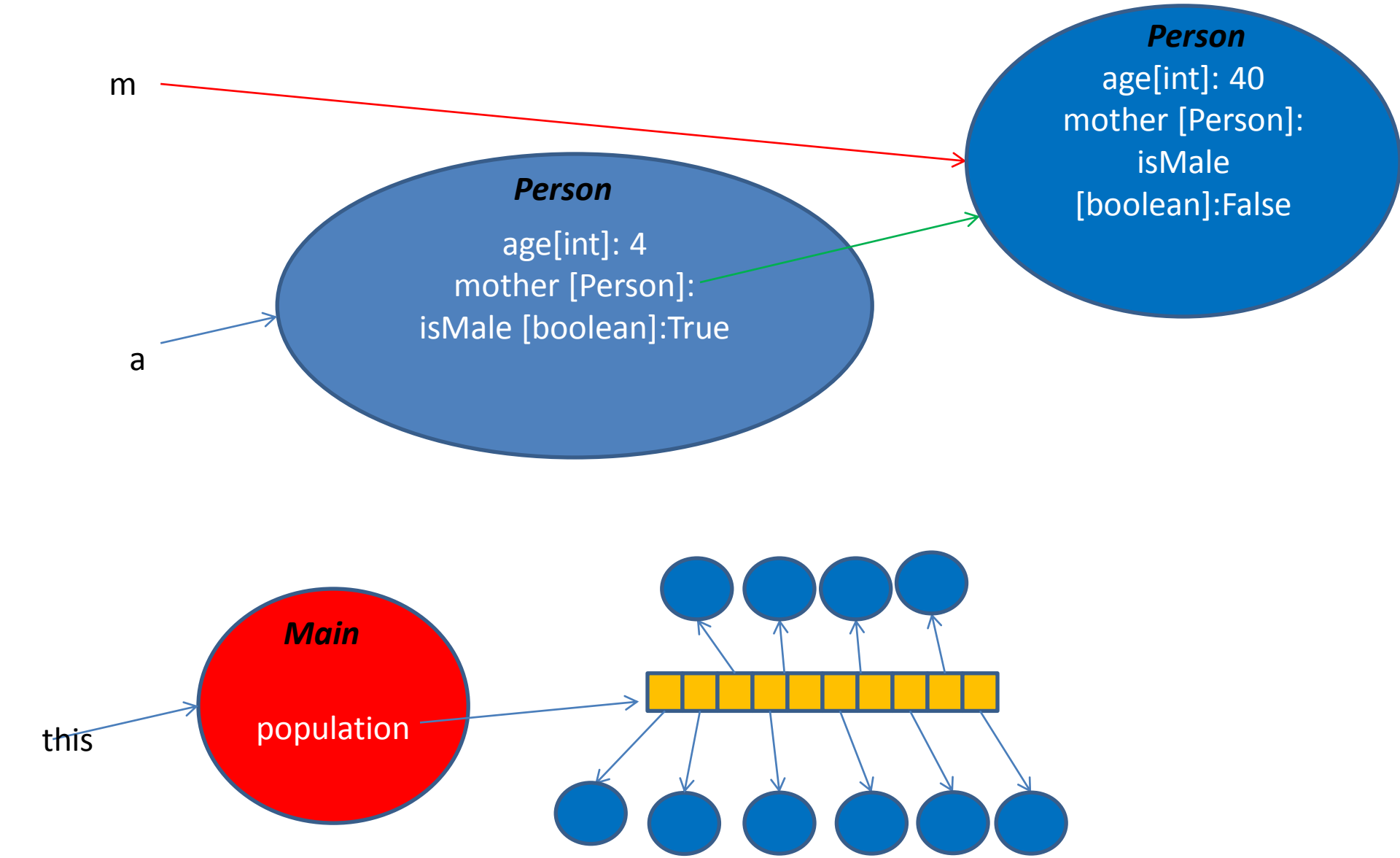
Assignments and References

- When we have a location whose associated value is a reference to an object, and assign another reference (possibly null) to that location , the stored reference (not the object) that is changed
- Suppose we have
  - a.mother=m
  - a.mother=this.get_Main().population[2] // This assinment changes the references value stored in a.mother – but doesn't affect the reference of m

At Start


m
Person
age[int]: 40
mother [Person]:
isMale
[boolean]:False
Person
age[int]: 4
mother [Person]: null
isMale [boolean]:True
a
Main
population
this

After a.mother=m



After
a.mother=this.get_Main().population[2]