6.005 Software Construction
Fall 2011
Prof. Rob Miller

# A Guide to Snapshot Diagrams

## Purpose

Snapshot diagrams represent the internal state of a program at runtime – its stack (methods in progress and their local variables) and its heap (objects that currently exist).

Here's why we use snapshot diagrams in 6.005:

- To talk to each other through pictures (in lecture, recitation, and team meetings)
- To illustrate concepts like primitive types vs. object types, immutable values vs. immutable references, pointer aliasing, stack vs. heap, abstractions vs. concrete representations.
- To help explain your design for your team project (to each other and to your TA)
- To pave the way for richer design notations in subsequent courses. Snapshot diagrams generalize into object models in 6.170.

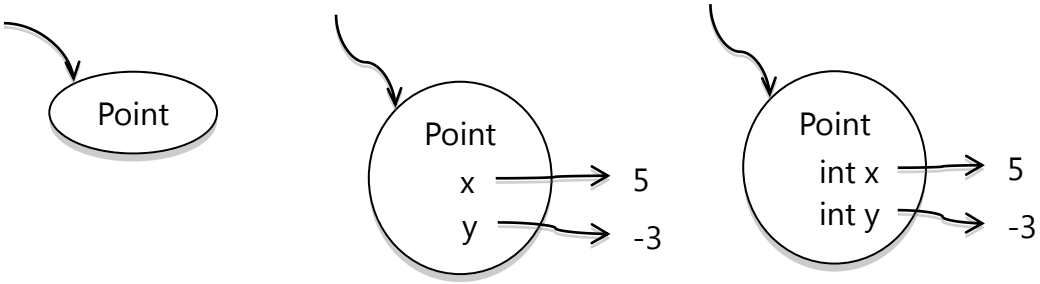
Although the diagrams below use examples from Java, the notation can be applied to most object-oriented programming languages, e.g. Python, Javascript, C++, Ruby.

## Primitive values

Primitive values are represented by bare constants. (The incoming arrow is a reference from a variable or an object field.)

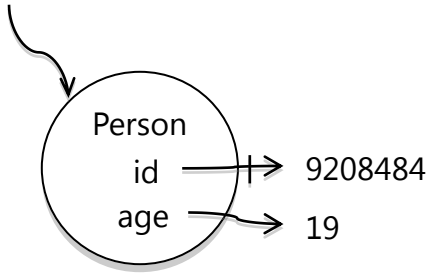

## Object values

An object value is a circle labeled by its type. When we want to show more detail, we write field names inside it, with arrows pointing out to their values. For still more detail, the fields can include their declared types. Some people prefer to write *x:int* instead of *int x*; both are fine.

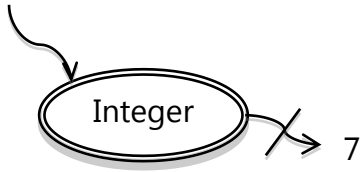


# Immutability

Immutable references (called *final* in Java) are denoted by a crossbar just behind the arrowhead. Here's an object whose *id* never changes (it can't be reassigned to a different number), but whose *age* can change.



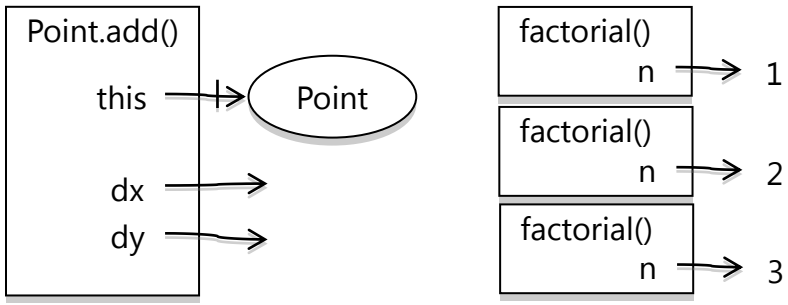

Immutable objects (intended by their designer to always represent the same value) are denoted by a double border. For example, here's an Integer object, an object value that represents the "boxed" form of an int:




Note that we also omitted the field name here, because it isn't necessary to understand the picture.

# Stack frames

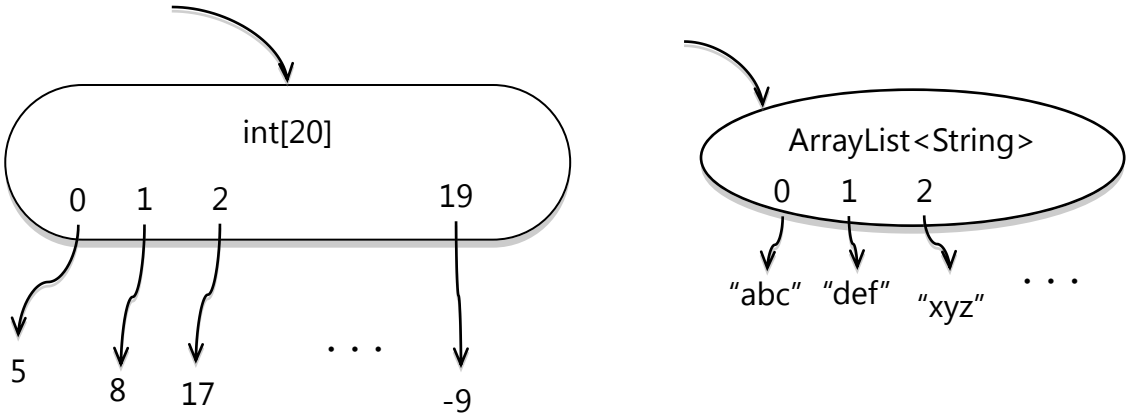
A stack frame is a method in progress. It is labeled by the method name (including its class name or parameter types if necessary to be clear), and contains references for parameters and local variables. Examples:




The left example shows Java's *this* reference, which refers to the object on which the method was called. The right example shows that stacks are drawn upwards: when a method calls another method, the new stack frame is drawn on top. This factorial function recursively calls itself with steadily decreasing values for the parameter *n*.

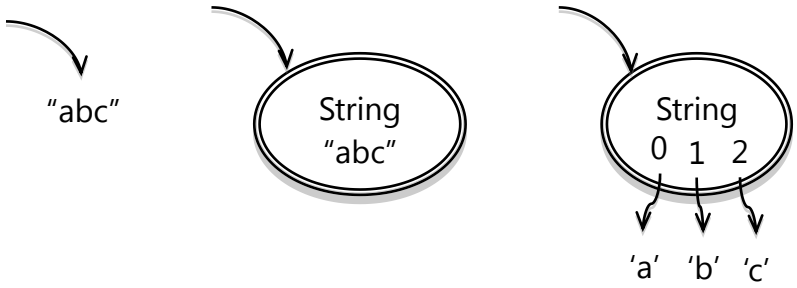
# Arrays and Lists

Like other object values, arrays and lists are labeled with their type. In lieu of field names, we label the outgoing edges with indexes (0, 1, ...). When the sequence of elements is obvious, we may omit the index labels.

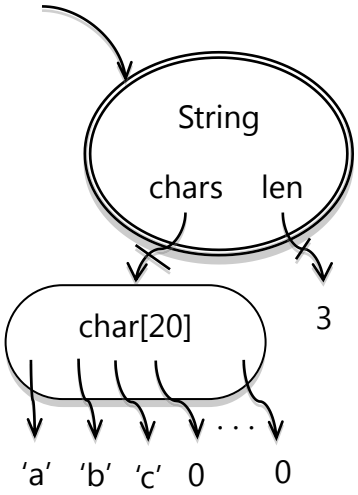



## Strings

Here are several ways to write strings that emphasize different things. The leftmost treats a string as a primitive, which we'll often do simply because strings are immutable and frequent, so it's convenient. The middle picture treats a string as an object value, but shows its abstract value. The right picture regards a string as a sequence of characters.



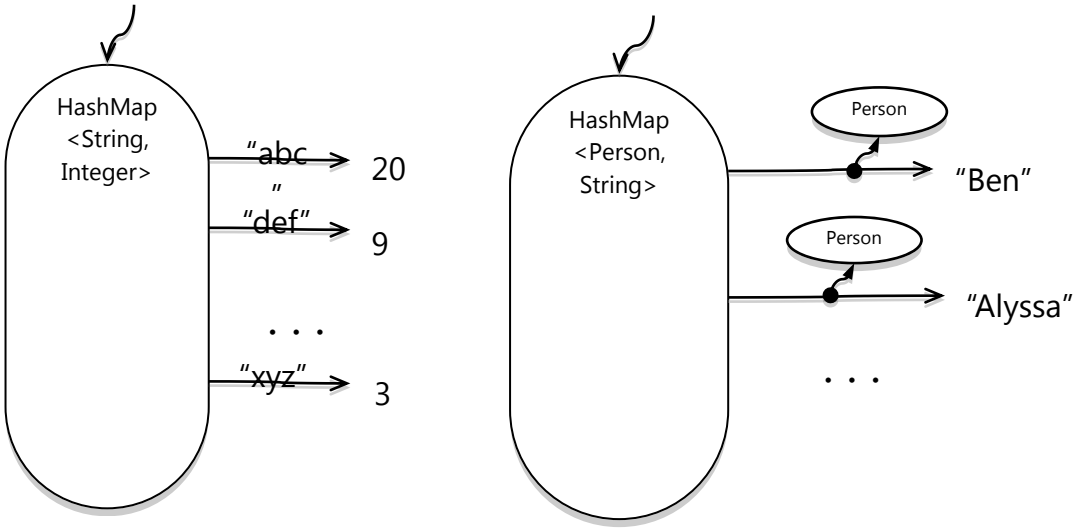

Finally, when we're talking about the concrete internal representation of a string in Java, we might draw a picture like this, which shows the string's character array:



# Maps

A map value maps one type (the keys) into another type (the values). We've already seen examples of these: arrays, strings and lists are maps with integer keys, which we often omitted writing on the edges because they could be inferred from the order of the edges.

For maps with other types of keys, we will label the edges with the abstract values of the keys. Here are two examples:




This example uses some abbreviation: strings and integers are drawn as their abstract primitive values, even though Java internally represents them as object values.

6.005 Elements of Software Construction

Fall 2011