Optimizations in Java JIT Compilers Based on Value Types

Bhavya Hirani
Sardar Vallabhbhai National Institute of Technology, Surat

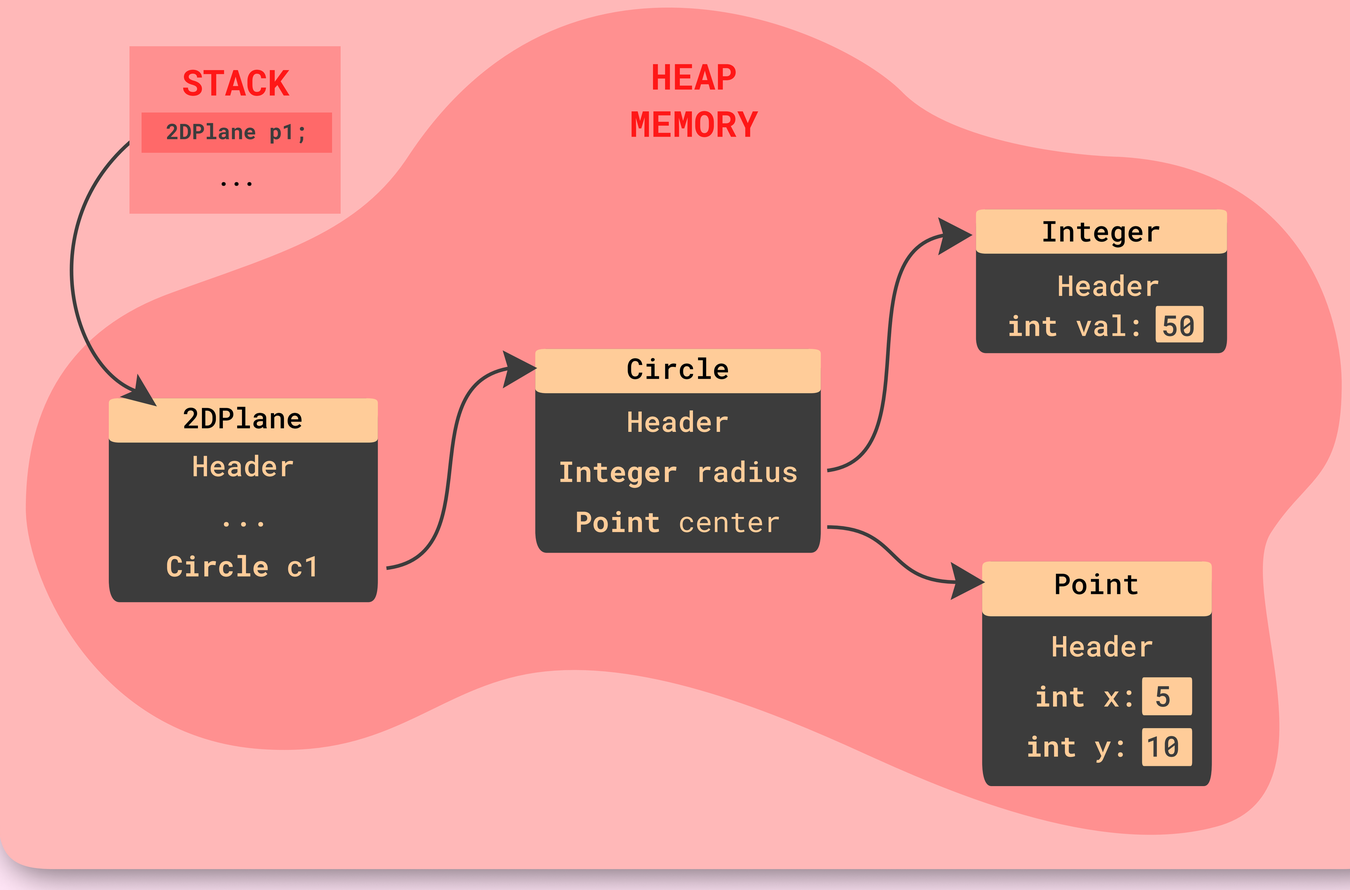
Arjun Harikumar
AMD, Bangalore

Prof. Manas Thakur
Indian Institute of Technology Bombay

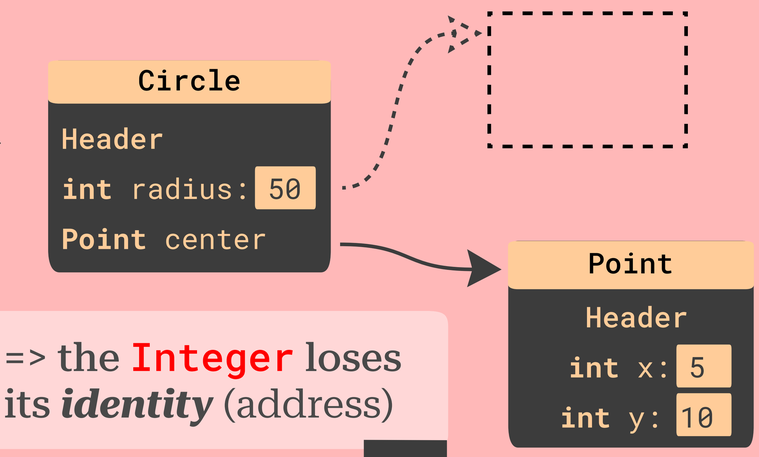
IIT Bombay

What are "Value Type" Objects ?

Consider This Running Example



We commonly see integers in Java being declared as `int`s rather than `Integer` objects.

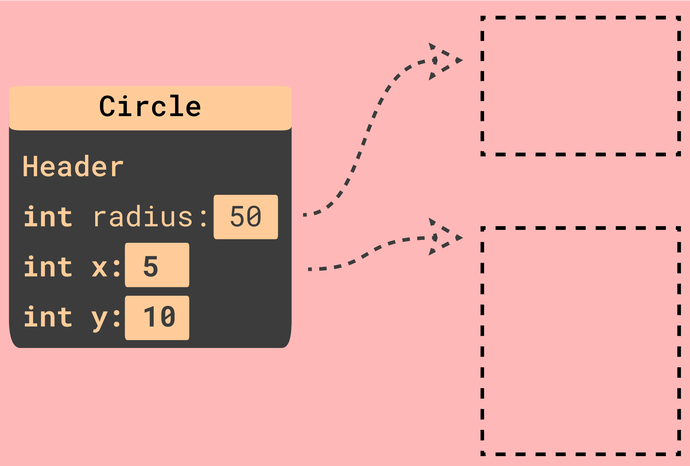


=> the `Integer` loses its ***identity*** (address)

We do it because it has latency-related benefits --- fewer cache misses, for example.

It *works* because Integers are merely ***wrappers over numerical values***, and *usually*, no other objects are trying to access/mutate them synchronously...

But we still haven't addressed the elephant in the room...**Point** is also merely a wrapper over a bunch of values, isn't it?

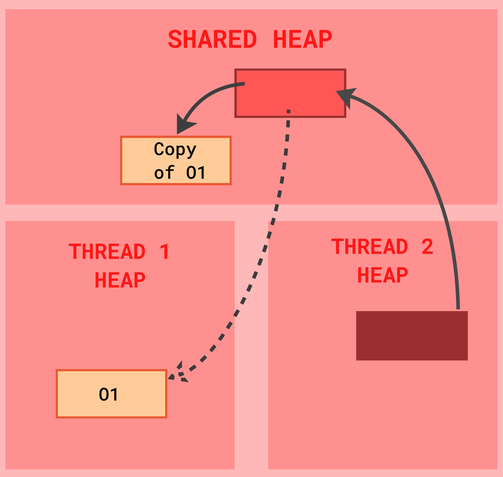


And what about **Circle**? If **Circle** objects aren't synchronized, we can treat them as a bunch of int values as well!

1. For Integers, we sacrifice object representation because it's a minor inconvenience.
2. But for other objects which do not synchronize or mutate, a similar feature would be useful, as long as there's not much programmer effort.
3. **Project Valhalla** [1] addresses this by allowing objects to be declared as "value type": final objects which acts as wrappers over primitives.

Better Garbage Collection?

The immutability of value types could allow each thread to run an independent GC algorithm on local value type objects through an optimization called *copy on update* [2]



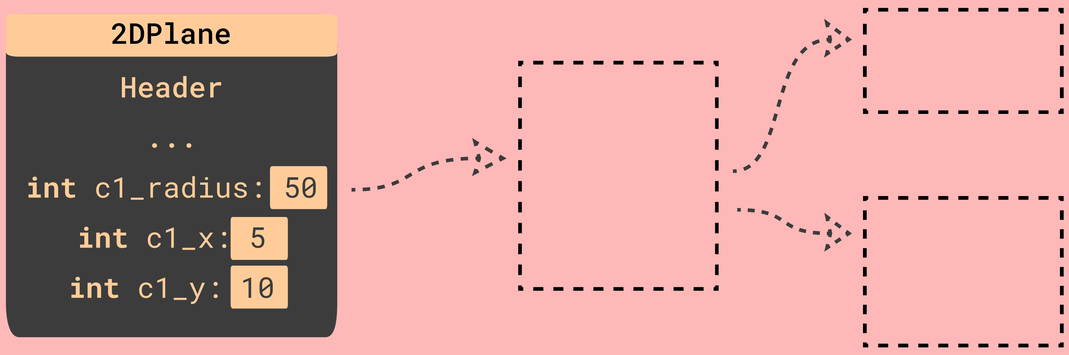
TAKEAWAY

Get used to noticing when certain classes do not need an *identity*, because in the future, you'll be able to leverage it to make your java code run faster with minimal effort!

PLATO

Object Inlining

This feature can lead to several cool optimizations! The one we have been seeing is called **Object Inlining**. If we declare `Circle` to be a **value type** object, its entire depth will be *inlined* within `2DPlane`.



Saved **36 bytes** (header) and two levels of Indirection

1. OpenJDK. 2023. Project Valhalla. https://openjdk.org/projects/valhalla/
2. Damien Doligez and Xavier Leroy. A concurrent, generational garbage collector for a multithreaded implementation of ML. In Proceedings of the 20th ACM SIGPLAN- SIGACT Symposium on Principles of Programming Languages - POPL '93, pages 113–123, Charleston, South Carolina, United States, 1993. ACM Press.