Introduction to Computer Science II
COMP 1406Z Fall 2022

Introduction to Classes and Objects

Dave McKenney
david.mckenney@carleton.ca

Learning Objectives

By the end of this lecture, you should be able to:

Explain difference between class and instance

Design a program with several interacting classes

Understand different types of memory in Java

Understand the role of constructors

Implement methods to specify class behaviour

Understand difference between static and instance

Previous code examples were not good examples of OOP

Why not?

What is a class?

A class specifies a type of object within our application

Think of an object as a bundle of data and associated behaviour (i.e., methods) that can be applied to that data

What is a class?

Class – the design of a type of object, specifying what state and behaviour the class has (a template)

Object/instance – a specific ‘physical’ copy of a specific class/type

Think cookie cutter (class) vs. cookie (instance)

Class Creation

The attributes of a class define the state of that type of object

All the variables required to define an instance of that type

What is a class?

What if we wanted to work with address data?

We could define an Address class

What data would it have?

Object Creation

Now that we have created a class, we can create instances of that class

Three steps:

1. **Declaration**
2. **Instantiation**
3. **Initialization**

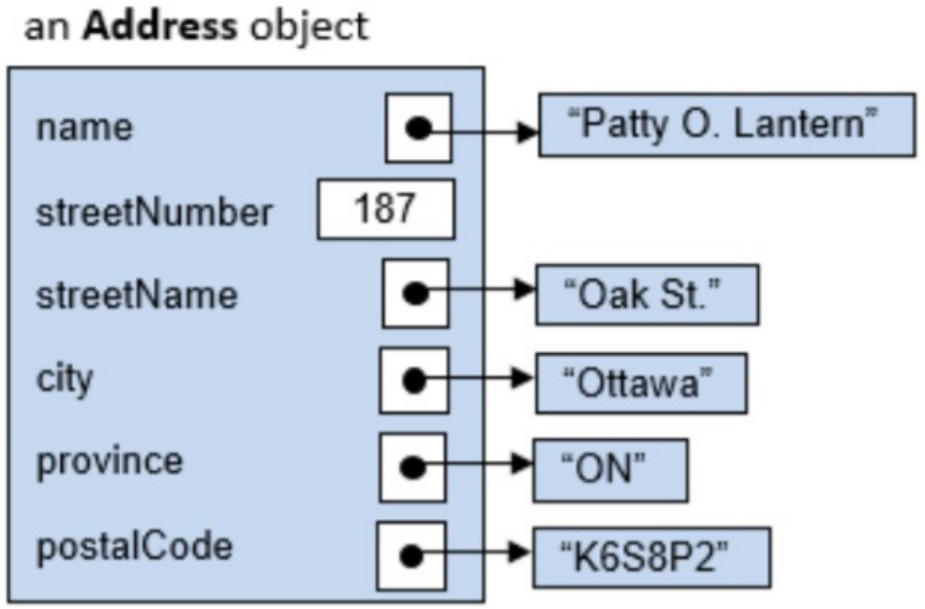
Primitive vs. Object

This has implications related to memory/storage

Object Storage Visualization

```
Address  addr;

addr = new Address();
addr.name = "Patty O.Lantern'
addr.streetNumber = 187;
addr.streetName = "Oak St.";
addr.city = "Ottawa";
addr.province = "ON";
addr.postalCode = "K6S8P2";
```



Note: non-primitives (i.e., objects) are 'pointed' to

Storing References

Objects are stored somewhere in memory
(not *inside* the object they are a part of)

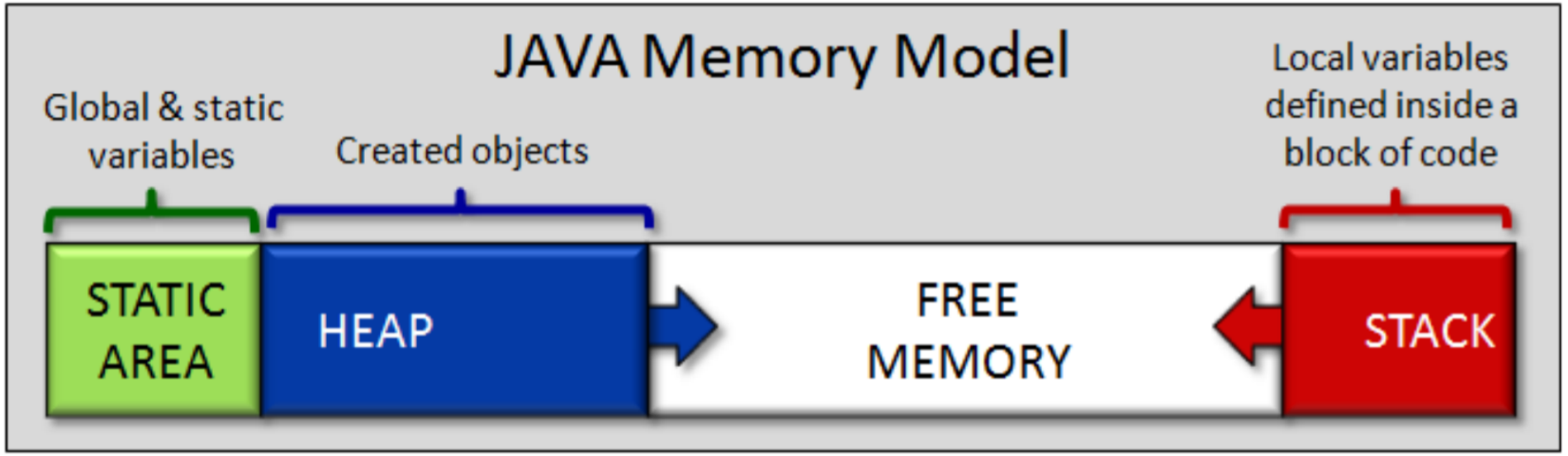
The object they belong to stores a memory reference
(i.e., a ‘pointer’ to that location)

Storing References

A memory address can be represented with a fixed amount of space (e.g., 32 bits or 64 bits)

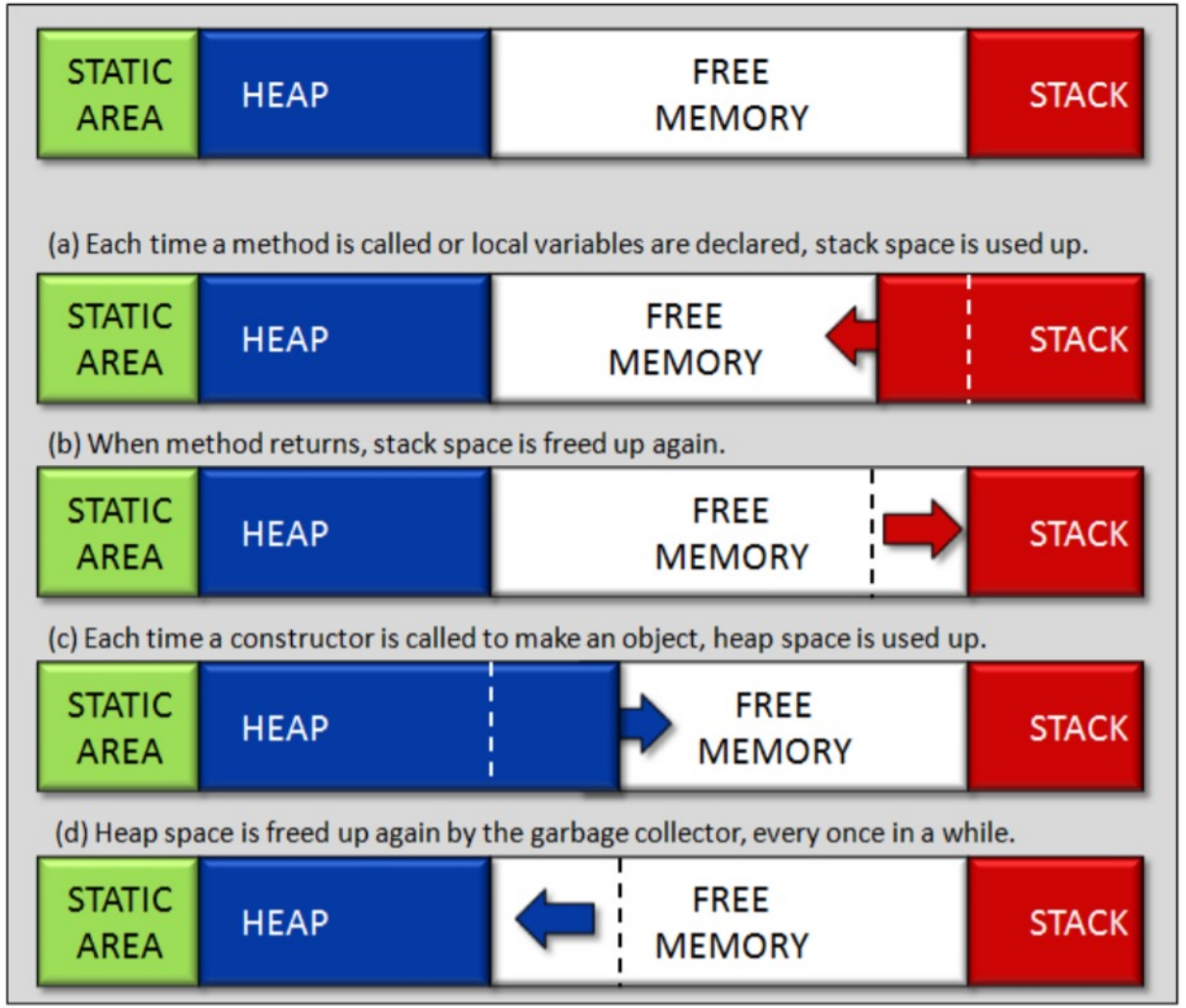
So we can calculate how much space we need to store an Address object, regardless of string size (or other attribute object size)

Java Memory Model



Four areas/types of memory space in a Java program

Memory Changes Over Time



Class Creation

The methods of a class define the classes behaviour

These methods typically modify the underlying state of an object and may have different behaviour based on the current state

Address example does not have any behaviour (yet)

What happens if we print an Address object out?

Address@72f5e3c4

What do you think this is?

The toString method is called automatically to convert an object to a String (e.g., when printing)

This is an example of object behaviour

Look at the code for creating and initializing several Address objects

Do you have any criticisms of this code?

Constructors

Constructors are a special type of method

Used when creating instances of a class

Responsible for initialization

Multiple Interacting Classes

Our classes can reference other classes
(i.e., they don't only have primitive attributes)

Adding More Behaviour

We will add methods to build out the Bank example

One thing to realize – methods are example of abstraction

Method Overloading

Method Overloading: having more than one method with same name

Useful when multiple forms of the same verb exist

Overloaded methods must have different signatures

Java Method Signatures

Method signature consists of:

Name of the method

Number/type/order of arguments

Instance vs. Static

Variables in our class have been ‘instance variables’

Methods in our class have been ‘instance methods’

These are methods that belong to or are called by one specific instance

Static vs. Instance

You may have noticed ‘static’ in front of the main method…

Static is a modifier

It indicates something belongs to the class at a class-wide level (i.e., it does not belong to an instance)

Static vs. Instance

Static variables track class-wide state

A single copy for entire class

Static vs. Instance

Static methods are class-wide methods (i.e., not called by an instance)

Typically utility methods (i.e., Math class)

Cannot access/modify any instance variables (i.e., non-static variables)

One ‘final’ Modifier

The last modifier (get it?) we will talk about is:
final

A variable specified as final cannot be re-assigned
(this is not the same as “it can’t be changed”!)

Summary

A class defines the state/behaviour of an object type

An instance/object is one specific manifestation of the object's class

Each object has its own attributes – instance methods can act on these attributes

Summary

Java organizes memory into several different areas

Heap stores actual object instances

Stack stores local variables (object references and primitives)

Summary

Classes can store references to other classes

We can have interactions between our objects

Summary

We can add constructors and methods to our classes

Allow us to define behaviour the class supports

Methods provide abstraction, which can make it easier to reason about our programs

Summary

Static variables/methods allow us to define class-level state and behaviour

We will use this less commonly

It is an important concept to understand to prevent yourself from introducing errors!

Next Steps

Next, we will begin to cover the principles of OOP and start to see the advantages that OOP can provide

This will make up the core of the 1406 course

The latter half of the course will be looking at these OOP principles being applied to different problem areas