CSE115 / CSE503
Introduction to Computer Science I

Dr. Carl Alphonce
343 Davis Hall
alphonce@buffalo.edu

Office hours:
Thursday 12:00 PM – 2:00 PM
Friday 8:30 AM – 10:30 AM
OR request appointment via e-mail
Turn off and put away electronics:

- cell phones
- pagers
- laptops
- tablets
- etc.
DATE: Tuesday March 8
TIME: 9:15 PM – 10:15 PM
LOCATION: various rooms in NSC
specific room/seat assignments to come

COVERAGE:
lecture material up to and including 2/26
lab material up to and including lab 4
readings up to and including lesson 27

HAVE A CONFLICT?
I will ask for documentation 2/22 – 2/26

BRING: your UB card

NO ELECTRONICS: cell phone, calculator, etc.
Last time
- class definitions in detail
- variables revisited

Today
- variable scope
- variable lifetime
- methods

Coming up
- class relationships
REVIEW
We will labeled each part of the class definition.

```java
package lab2;

public class Farm {
    public Farm() {
    }
}
```
A variable exists at a storage location in memory. For example, location 12207:
A variable has:

- a name ➔ in the HLL (Java)
- a location ➔ in memory
- a type ➔ representation scheme/size
- a value ➔ contents
- a scope
- a lifetime

We’ll discuss these next
SCOPE
(no, not the mouthwash...)

© Dr. Carl Alphonce
The **scope** of a variable is the part of a program where a variable declaration is in effect.

Variables declared in different ways have different scope:

- local variables
- instance variables
A variable declared within a constructor (or a method) is called a *local variable*.

The *scope* of a local variable is from the point of the declaration to the end of the brace-delimited block containing the declaration.
a constructor is called *during object creation* (only during the evaluation of a ‘new’ expression)

once an object exists a constructor cannot be called (on *that* object)

a method is called *after* object creation

a method is always invoked *on an existing object*
package lab2;

public class Farm {
    private example1.BarnYard _t;

    public Farm() {
        _t = new example1.BarnYard();
    }

    public void addTwoChickens() {
        example1.Chicken c1;
        c1 = new example1.Chicken();
        example1.Chicken c2;
        c2 = new example1.Chicken();
        _t.addChicken(c1);
        _t.addChicken(c2);
        c1.start();
    }
}

We’ll return to method definitions in more detail soon.
package lab2;

public class Farm {
    public Farm() {
        example1.Terrariumium t;
        t = new example1.Terrariumium();

        example1.Chicken c;
        c = new example1.Chicken();
        t.addChicken(c);
        c.start();
    }
}

end of block containing declaration

Declaration
A variable declared within a class but outside of any method is called an *instance variable*.

The **scope** of an instance variable is the entire class body.
package code;

public class Dog {

    private Tail _tail;

    public Dog() {
        _tail = new Tail();
    }

}
LIFETIME

(sorry, no pun here)
The *lifetime* of a variable is the period of time during execution of a program that the variable exists in memory. This is a dynamic property (one relating to runtime).

Variables declared in different ways have different lifetimes:

- local variables
- instance variables
Memory organization

Process A

Process B

Process C

STATIC SEGMENT

HEAP

FREE/AVAILABLE MEMORY

RUNTIME STACK

dynamically allocated memory
A local variable comes into existence when a method is called, and disappears when the method is completed.

Space for a local variable is allocated in a special region of memory, called the *runtime stack*.

All the local variables of a method are allocated space in the same area, called a *stack frame* (or *invocation record*).
Local variables are stored on the runtime stack. Each method invocation (call) results in an invocation record (stack frame) being added to the top of the stack. When a method exits, its invocation record is removed from the top of the stack.
Instance variables are created when a class is instantiated.

‘new’ allocates memory from the heap

Each object has its own set of instance variables.

the variables are the constituents of an object
instance variables therefore exist on the heap

Instance variables persist as long as their objects persist

as far as we know right now, objects persist until the end of the runtime of the program.
All memory allocated by ‘new’ comes from the heap.

Objects are allocated space by ‘new’, and their representations (which contain their instance variables) therefore exist on the heap.