LECTURE 2:
OBJECTS AND CLASSES

CS 2110
Fall 2021

LECTURE 2:
OBJECTS AND CLASSES

PART 1: SHORT REVIEW.
WHY OOP?

CS 2110
Fall 2021

Agenda

Previously in 2110:

- Strong typing
- Java’s primitive types
- Casting among primitive types
- Recitation on strings

```
int v;
v= “abc”;  // illegal
```

Most-used primitive types

int	4-byte integer
long	8-byte integer
double	8-byte floating point
char	Unicode character
boolean	true, false

```
(int) ‘a’
(double) (int) 6.5
```

Quiz?

Object-Oriented Programming

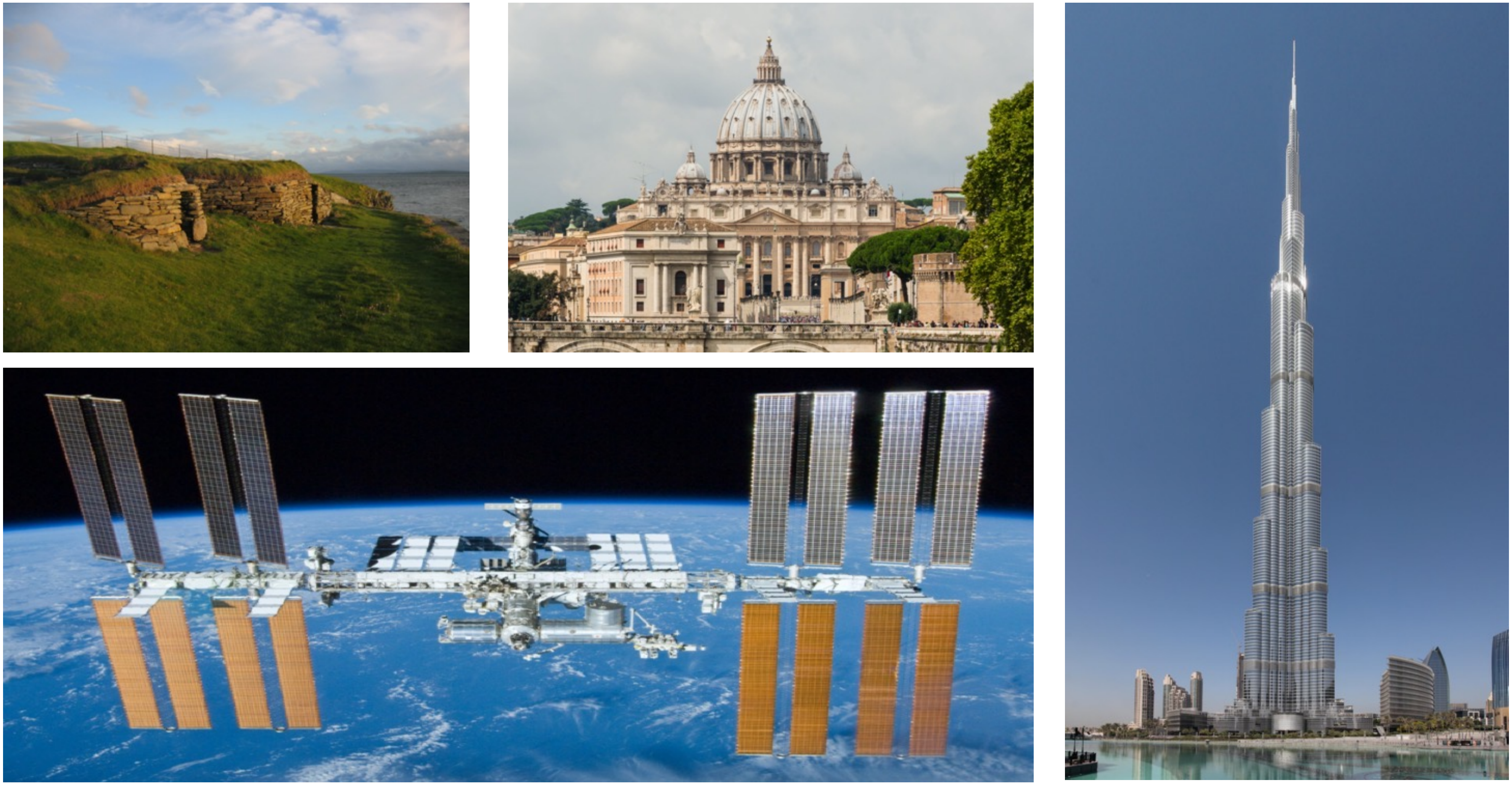
Today:

What is OOP?

Objects and classes

Methods and fields

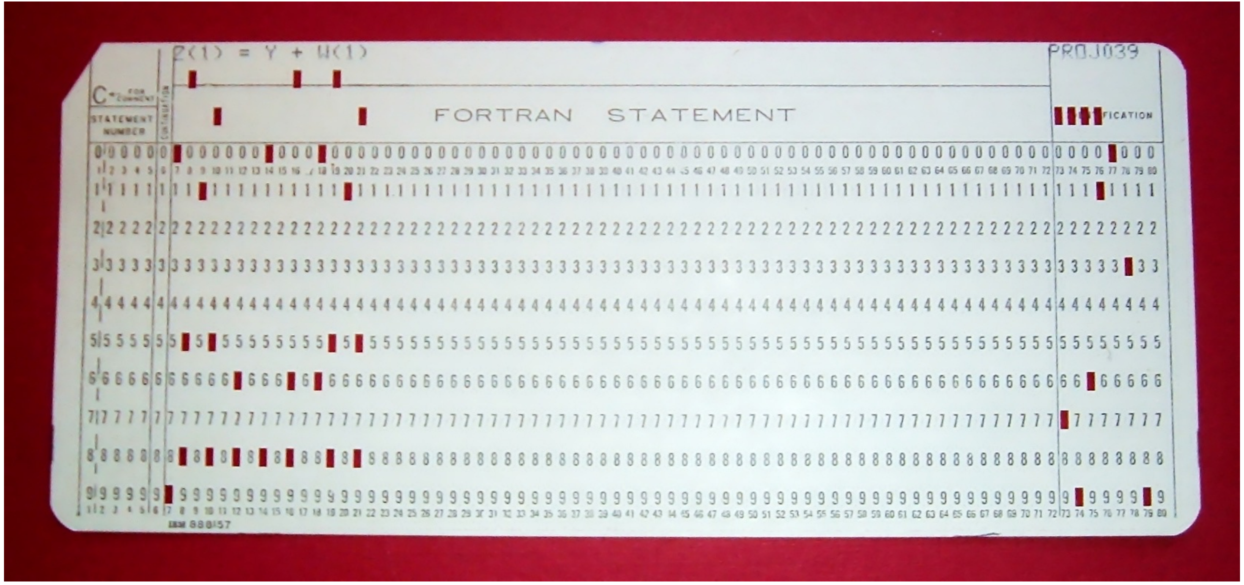
Building Bigger



Clockwise: Knap of Howar, St Peter’s Basilica, Burj Khalifa, ISS; all images in public domain

Programming and programming languages

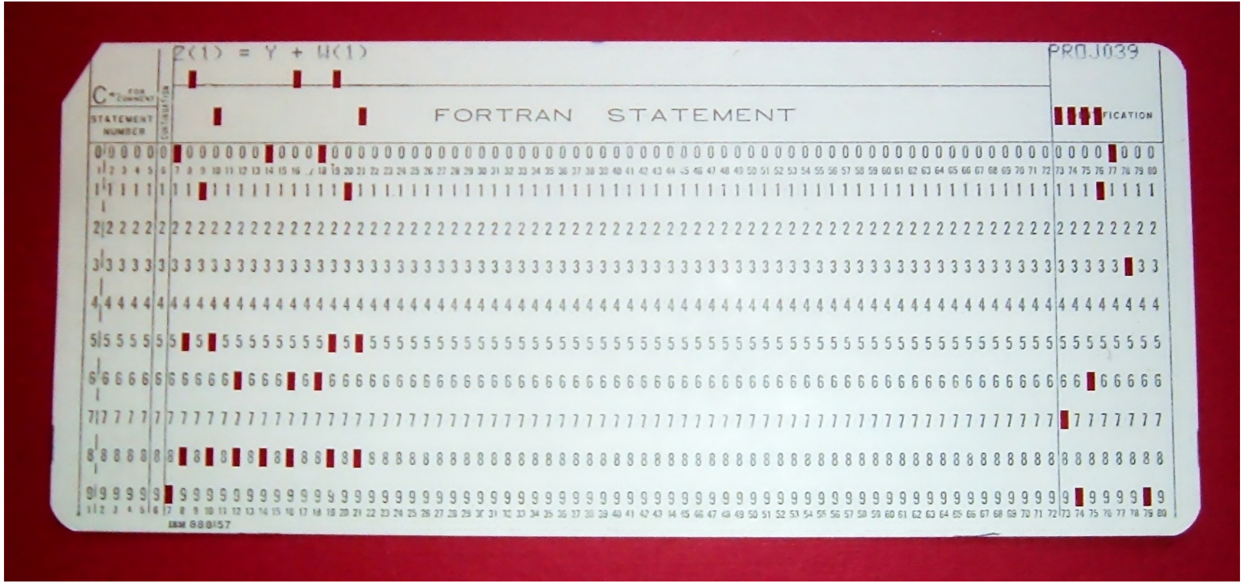
First high-level language:
Mid 1950s: Fortran



In June 1960, after graduating with a BS in Math from Queens College in NY, Gries started working for the US Naval Weapons Lab (as a civilian) as a *mathematician programmer*. They taught us Fortran in *one* week. We were then professional programmers.

Programming and programming languages

First high-level language:
Mid 1950s: Fortran



1960:
Algol 60, Lisp, Cobol

1970: Pascal
1972-73: C

None of these were object-oriented! People began feeling the need for more scalable languages, with better features for reuse and ability to change

Different forms of "modules"
Modula
ML
Ada --after the first programmer, a woman, Lady Ada Lovelace (1800's)

OOP: Building Bigger

Simula 67: arguably the first OO language: objects, classes, inheritance, etc.

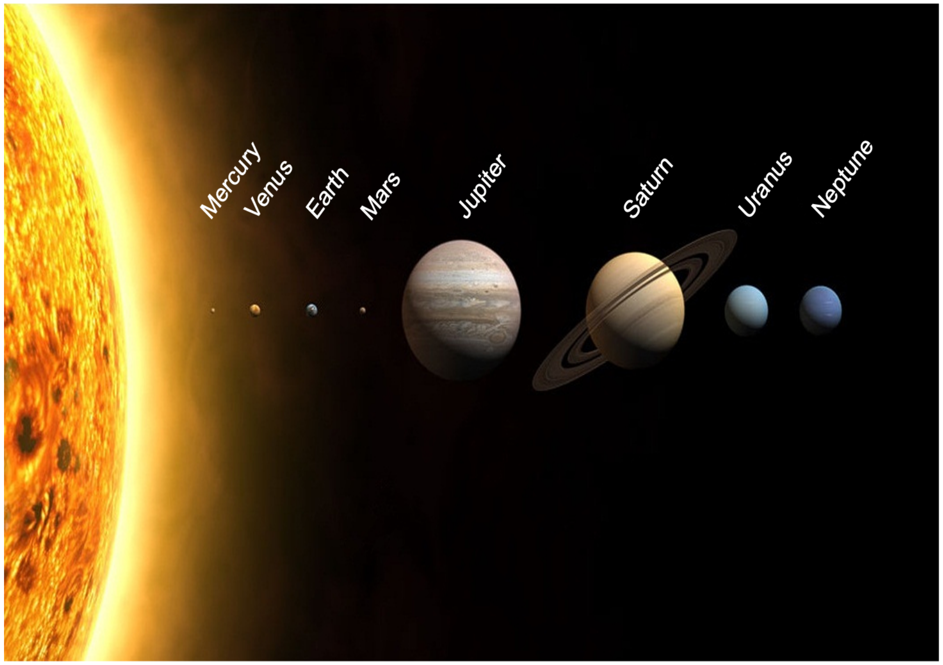
BYTE
the small system
AUGUST 1981 Vol. 6, No. 8
$2.50 in USA/$2.95 in Canada
A McGraw-Hill Publication
SMALLTALK

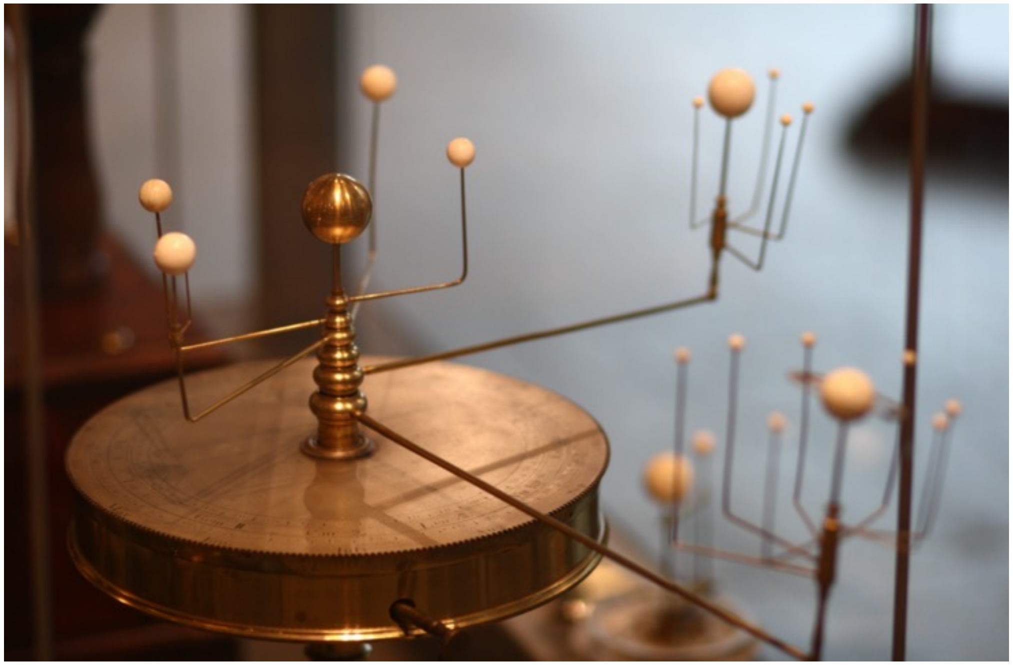
Smalltalk-80: based on Simula, huge influence on Java, Python, etc.

Images: http://www.simula67.info/, https://archive.org/details/byte-magazine-1981-08

OOP’s key insight:

Model objects from the real world

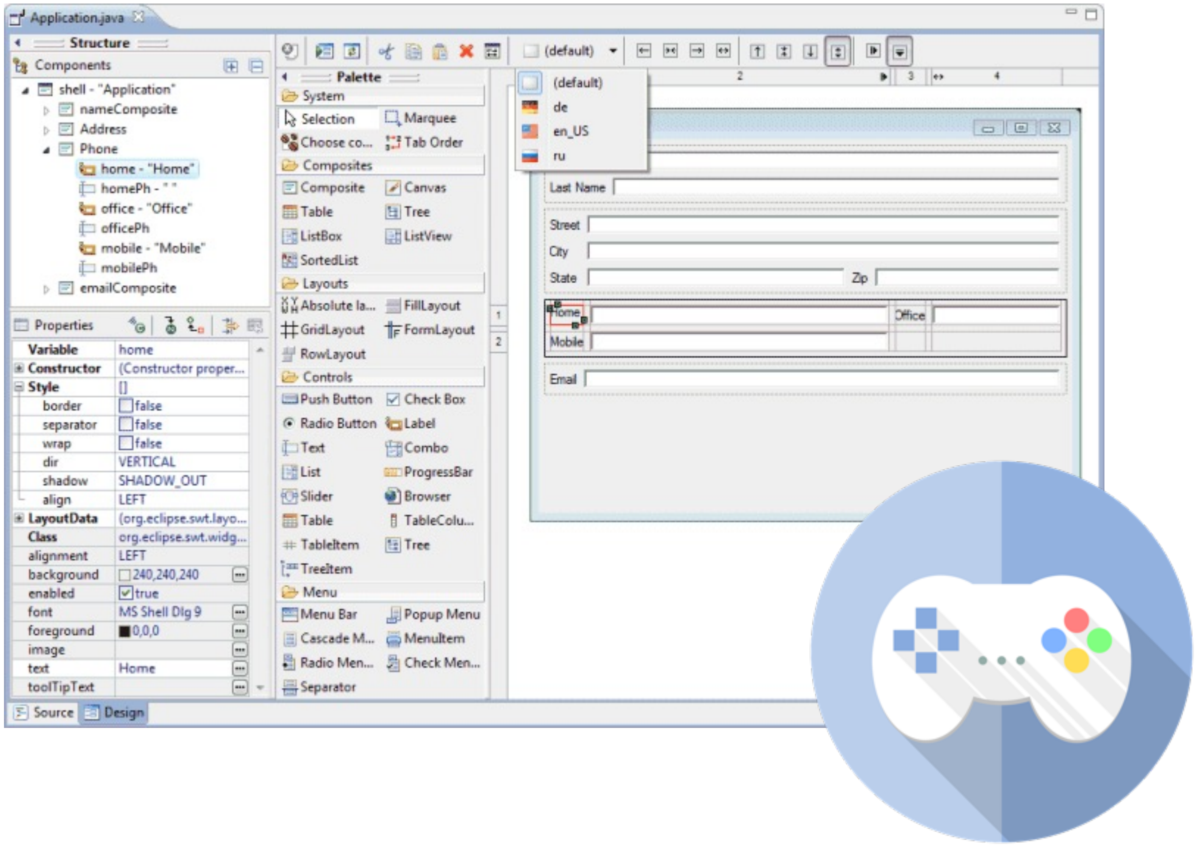




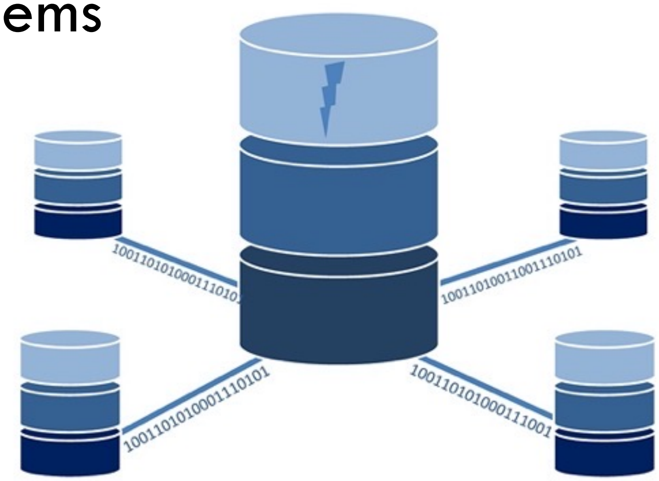
Images: https://upload.wikimedia.org/wikipedia/commons/c/cb/Planets2013.svg, https://commons.wikimedia.org/wiki/File:Planetarium_in_Putnam_Gallery_2,_2009-11-24.jpg

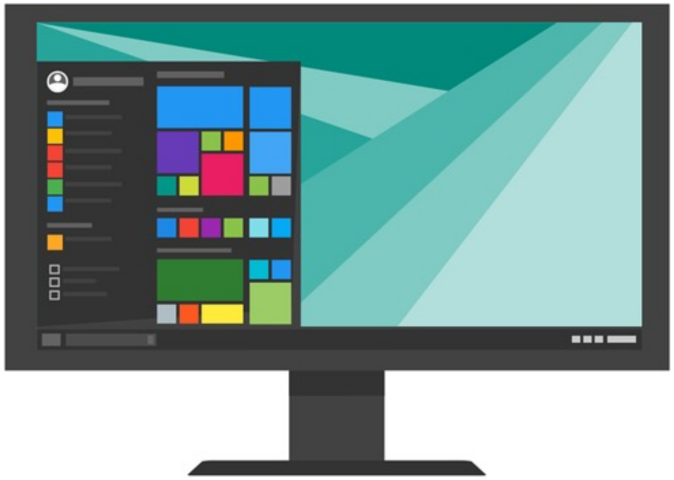
OOP: Beyond Physical Systems

GUIs, Games, Databases, Operating Systems



Variable	home
Constructor	(Constructor proper...
Style	[]
border	□ false
separator	□ false
wrap	□ false
dir	VERTICAL
shadow	SHADOW_OUT
align	LEFT
LayoutData	(org.eclipse.swt.layo...
Class	org.eclipse.swt.widg...
alignment	LEFT
background	□ 240,240,240
enabled	☑ true
font	MS Shell Dlg 9
foreground	■ 0,0,0
image	
text	Home
toolTipText	





Images:
https://www.eclipse.org/windowbuilder/images/wb_summary_shot.gif
https://pixabay.com/illustrations/game-gaming-gaming-console-gamer-1926905/
https://www.needpix.com/photo/833250/database-data-computer-network-cloud-storage-server-security
https://pixabay.com/vectors/operating-system-windows-os-1995434/

Why OOP?

- **Analysis:** OOP helps identify features
- **Design:** OOP improves resilience to change
- **Implementation:** OOP enables re-use

…none of these unique to OOP

But still a successful packaging!

Our Trajectory

- Lec 2: **Objects and classes:** the building blocks
- Lec 3: **Encapsulation:** a design principle
- Lec 4: **Inheritance:** achieving re-use
- Lec 5: **Subtyping:** achieving re-use
- Lec 6: **Abstraction:** a design principle
- (then we start studying data structures)

The end

LECTURE 2:
OBJECTS AND CLASSES

PART 2: OBJECT: HAS STATE AND BEHAVIOR
CLASS: DEFINES THE COMPONENTS OF OBJECTS

CS 2110
Fall 2021

Objects

- **Behavior:** response to stimulus
- **State:** condition of being; changeable

Examples:

- Battery
- BRB account
- David Gries

Cornell Dining
Towering Above the Rest

Image: https://svgsilh.com/image/1379208.html

Exercise

What are possible states and behaviors of some object around you?

Please raise your hand

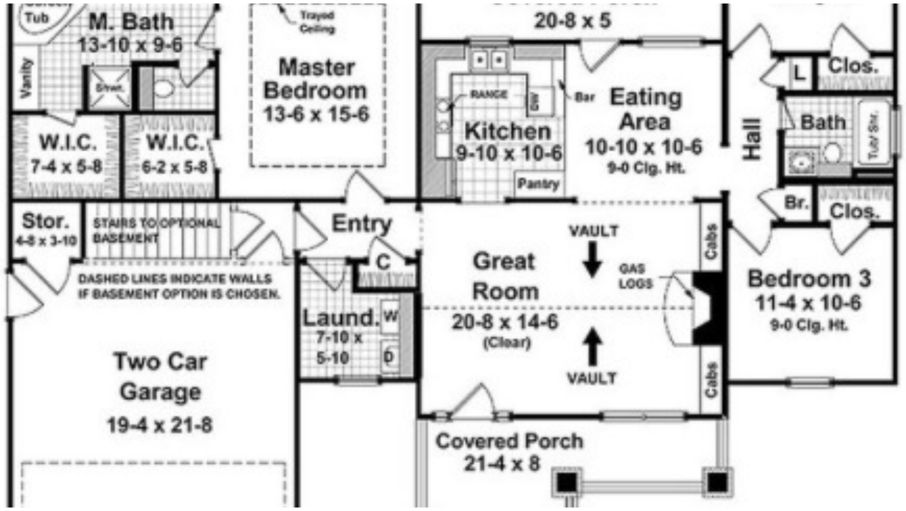
Classes

- **Class:** a blueprint for making new objects
- An object is an **instance** of a class

Image: https://twitter.com/SaranacBrewery/status/986339299435122688/photo/1

Class vs. Object

A blueprint, design, plan
A class



A house built from the blueprint: **An object**

Same class, different objects

Images: https://www.houseplans.com/plan/1509-square-feet-3-bedrooms-2-bathroom-cottage-house-plans-2-garage-28803

Example: a counter

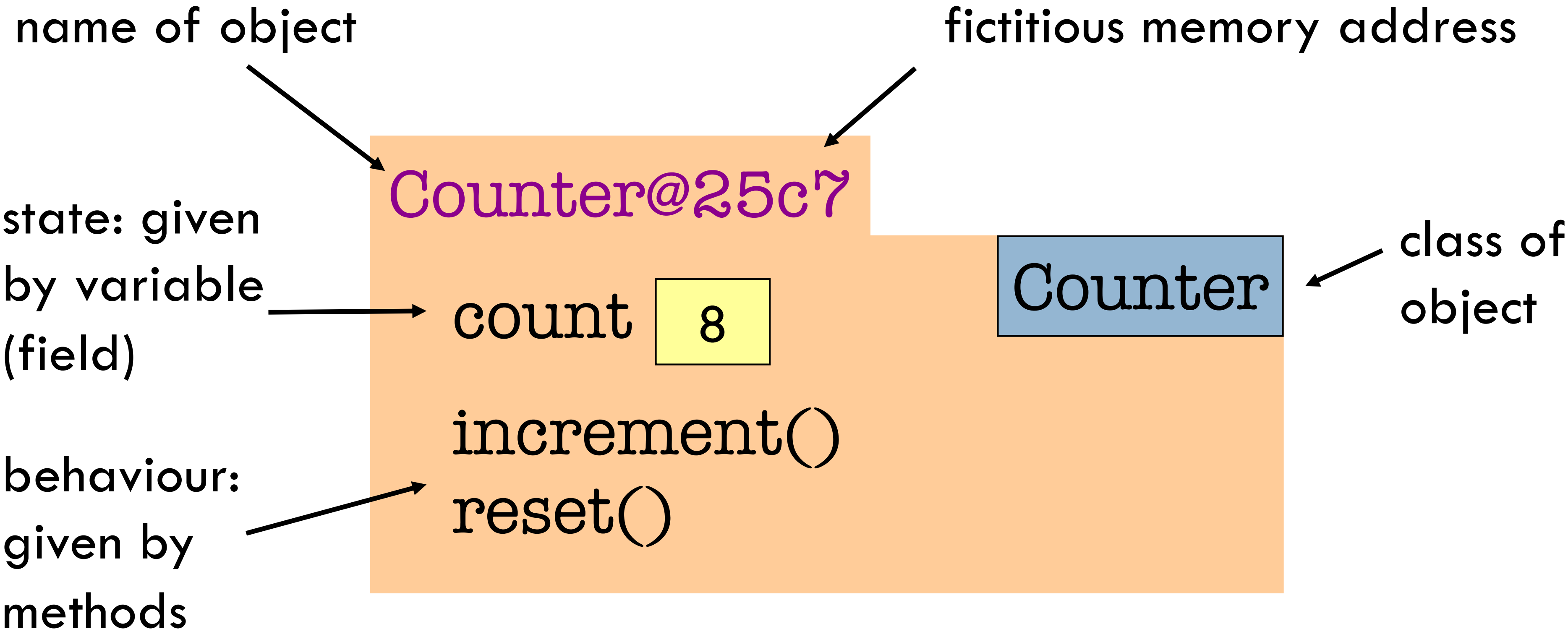
Counter

State: the value of the counter

Behavior: buttons to
(1) Set counter to 0,
(2) Increment the counter by 1.

Image: https://www.amazon.com/gp/product/B075QFK6DG

Folders: Depicting a Counter object



name of object

fictitious memory address

Counter@25c7

state: given by variable (field)

count 8

class of object

Counter

increment()
reset()

behaviour: given by methods

Computer creates an object: It allocates space in memory for the object, including space for the fields. Think of the methods also as residing in the object.

You create an object: This is how you draw it.

Declaration of class Counter

```
class Counter {






}
```

{ … } is called a block.

In this context, the block will contain declarations of fields and methods that belong in each instance of the class.

Counter

```
class Counter {
   int count;



}
```

Declaration of variable count.

Variable count is a field. It will appear in every instance (i.e. object) of class Counter.

Counter

```
class Counter {
  int count;

  void increment() {
    count= count + 1;
  }

}
```

Declaration of method increment.

Each time it is called (invoked),
1 is added to field count.

Counter

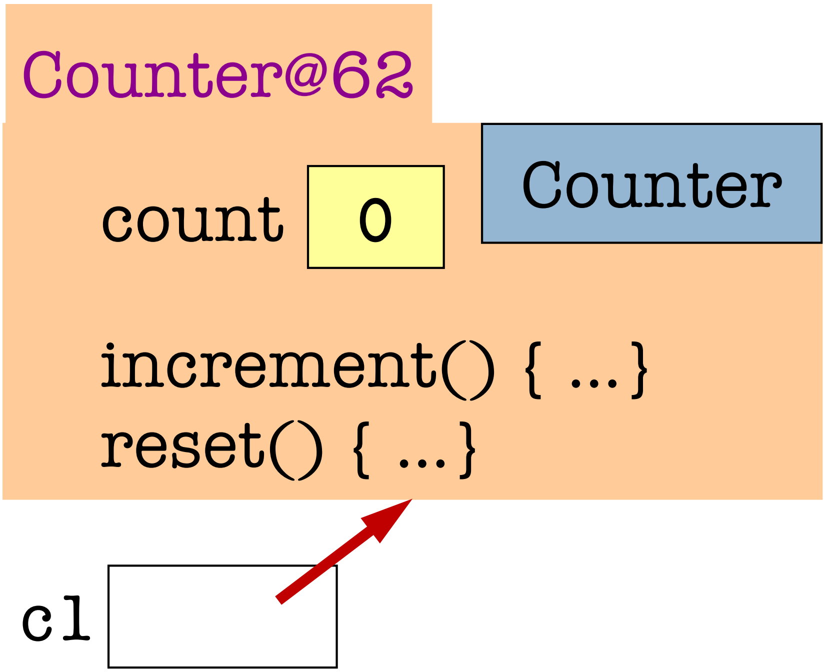
```
class Counter {
   int count;

   void increment() {
      count= count + 1;
   }

  void reset() {
     count= 0;
  }
}
```

Declaration of method reset.

Each time it is called,
field count is set to 0.

The new-expression

```
Counter c1;     @62
c1= new Counter();
```



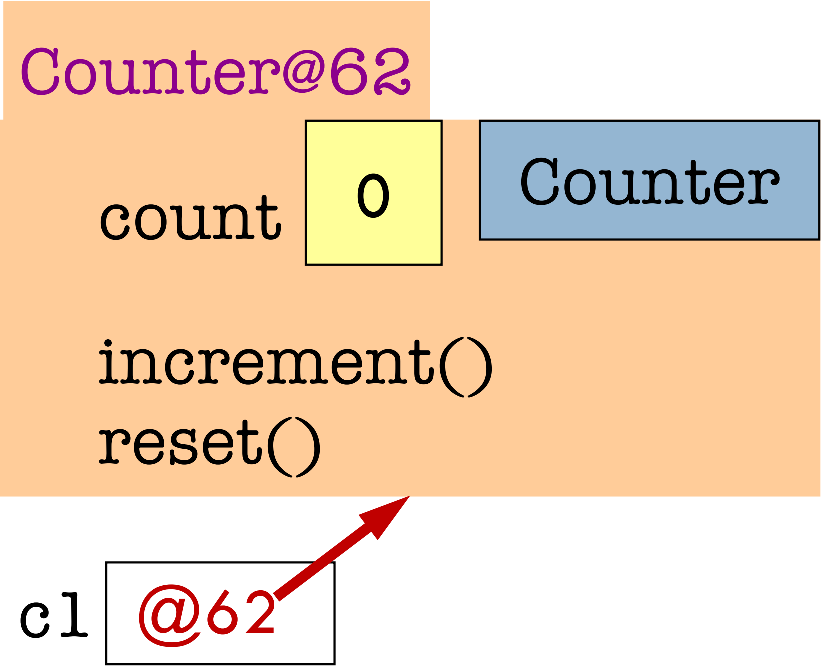
```
class Counter {
   int count;
   void increment() {
      count= count + 1;
   }
  void reset() {
     count= 0;
  }
}
```

Calling a method in an object

```
Counter c1;  c1= new Counter();
c1.increment();
```

```
class Counter {
   int count;
   void increment() {
      count= count + 1;
   }
   void reset() {
      count= 0;
   }
}
```

1

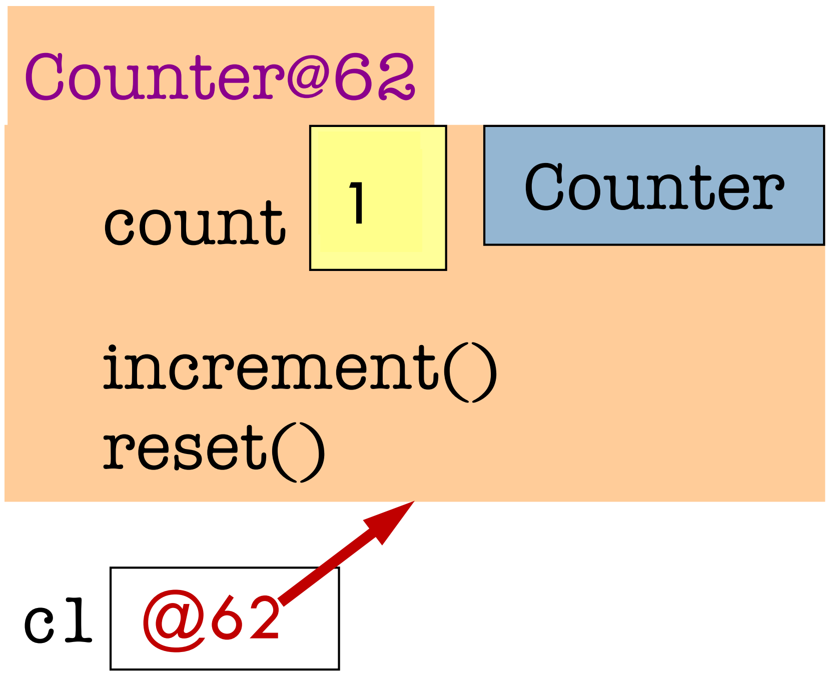


Referencing a field of an object

Counter c1; c1= new Counter();

c1.increment();

c1.count



```
class Counter {
   int count;
   void increment() {
      count= count + 1;
   }
  void reset() {
     count= 0;
  }
}
```

Language features just used

- **class:** Counter
  - **field:** count
  - **methods:** increment(), reset()
- **new-expression:** new Counter()
- **field access:** c.count
- **method call or invocation:** c.increment()

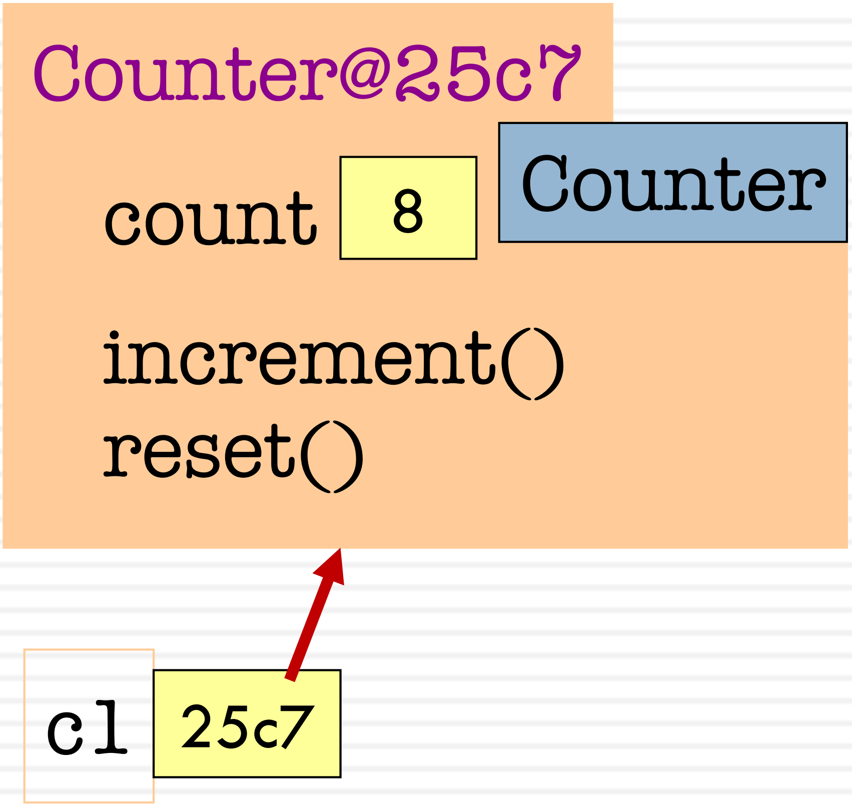
The end

LECTURE 2:
OBJECTS AND CLASSES

PART 3: OBJECT: HAS STATE AND BEHAVIOR
CLASS: DEFINES THE COMPONENTS OF OBJECTS
DEMO USING ECLIPSE AND JSHELL

CS 2110
Fall 2021

Develop class Counter



Counter@25c7

count 8

Counter

increment()

reset()

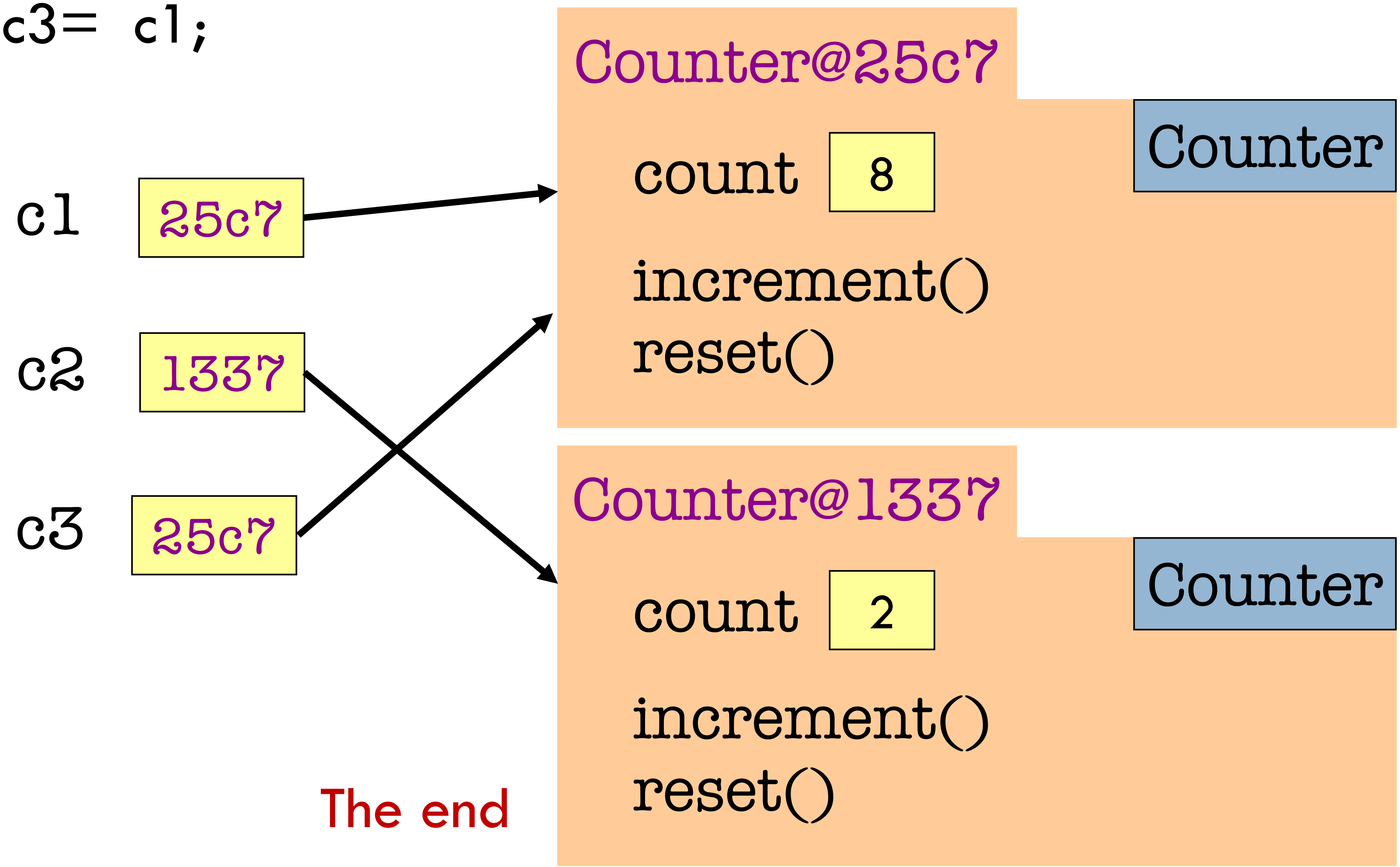
c1 25c7

Demo in Eclipse and JShell

Image: https://www.amazon.com/gp/product/B075QFK6DG

Names: Pointers to Objects

```
c3= c1;
```



The end

LECTURE 2:
OBJECTS AND CLASSES

PART 4: FIELDS AND METHODS
SCOPE AND THE INSIDE-OUT RULE
NEW-EXPRESSION

CS 2110
Fall 2021

Fields

- □ Fields are variables that live in an object. They constitute the *state* of the object
- □ Could be many in a class. Each has a default value depending on its type. See JavaHyperText.

  ```
  int count;
  String manufacturer;
  int serialNumber;
  ```

- □ Could initialize:

  ```
  int serialNumber= 8675309;
  ```

- □ *Java syntax is rich! See JavaHyperText for much more than we can cover in lecture.*

Methods

- Methods define the *behavior* of the object

- Definition:

```
type methodName(parameter declarations) {
        …
}
```

Methods: Procedures

Procedure: return type is void

```
void setTo(int i) {
      count= i;
}
```

```
c1.setTo(5);
```

No value is returned by this call

About the return statement

```
void setTo(int i) {
      if (i < 0) return;
      count= i;
}
```

Execution of   return;
terminates execution of the method body, so if i is negative, count is not assigned a value

Counter with life-time count

```
class Counter {
    int count;
    int lifetimeCount;
    void increment() {
        count= count + 1;
        lifetimeCount= lifetimeCount + 1;
    }
    void reset() { count= 0; }
}
```

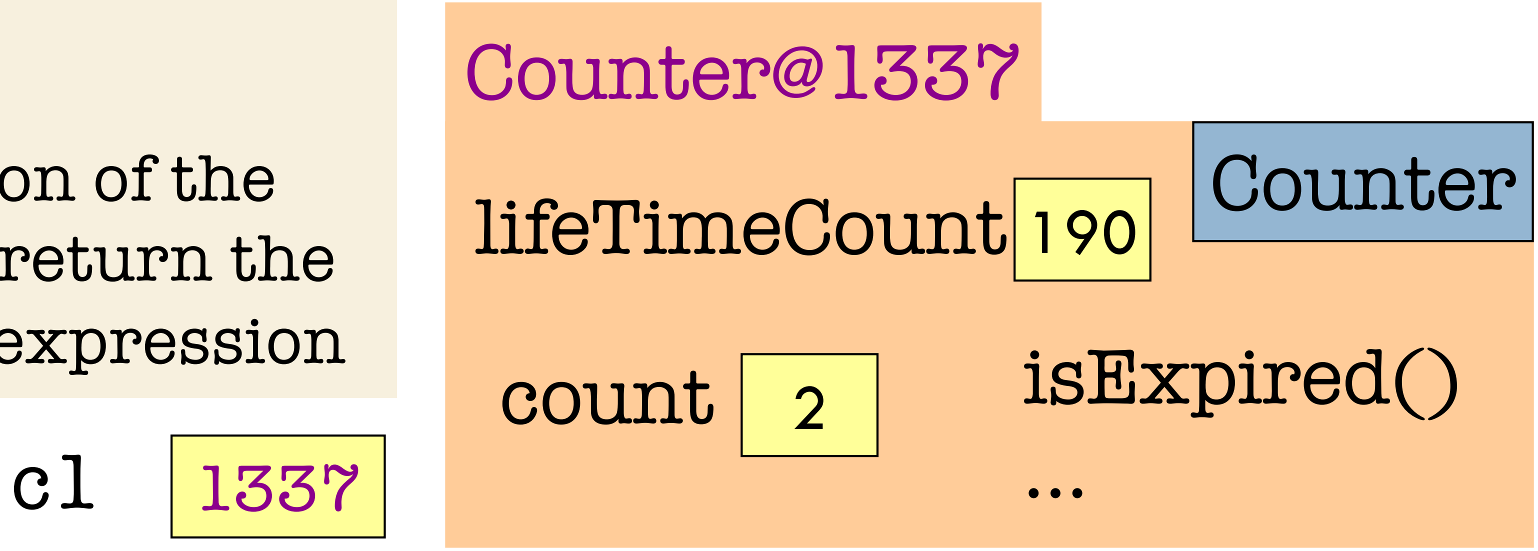
Methods: Functions

Function: return type is not void

```
boolean isExpired() {
        return lifetimeCount > 2,000,000;
   }
```

Execution of the return statement:

Stop execution of the method and return the value of the expression



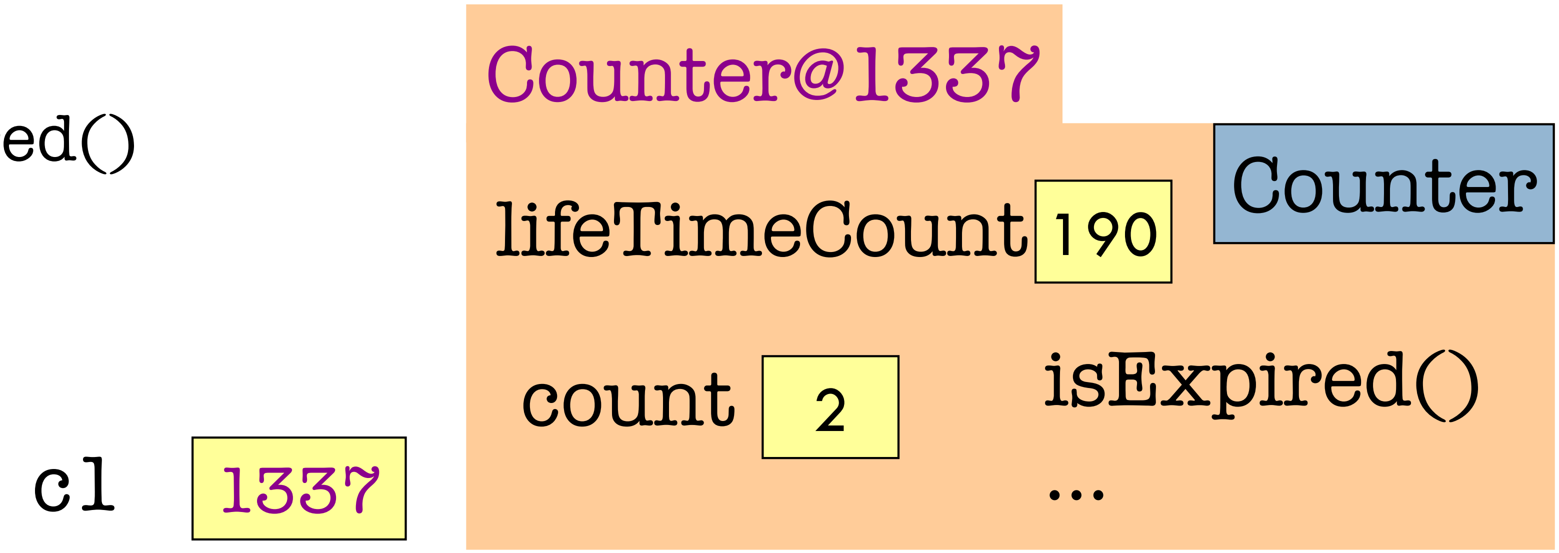
Methods: Functions

Function: return type is not void

```
boolean isExpired() {
        return lifetimeCount > 2,000,000;
  }
```

false

cl.isExpired()



cl 1337

Counter@1337

Counter

lifeTimeCount 190

count 2

isExpired()

…

Scope

Scope refers to the lifetime and accessibility of a variable —where in a program it can be used.

```
void setTo(int i) {
        if (i < 0) return;
        count= i;
   }
```

The scope of a parameter like i is the body of the method

The parameter is created when the method is called

e.g.   setTo(5);

And it is initialized to the argument value. (e.g. 5)

The parameter is destroyed when the method call ends

Scope

Scope of fields and methods:

- Every object created from a class C contains the fields and method declared in C.
- The scope of the fields and methods is the entire class (or object of the class)
- Fields and methods are created when the object is created …

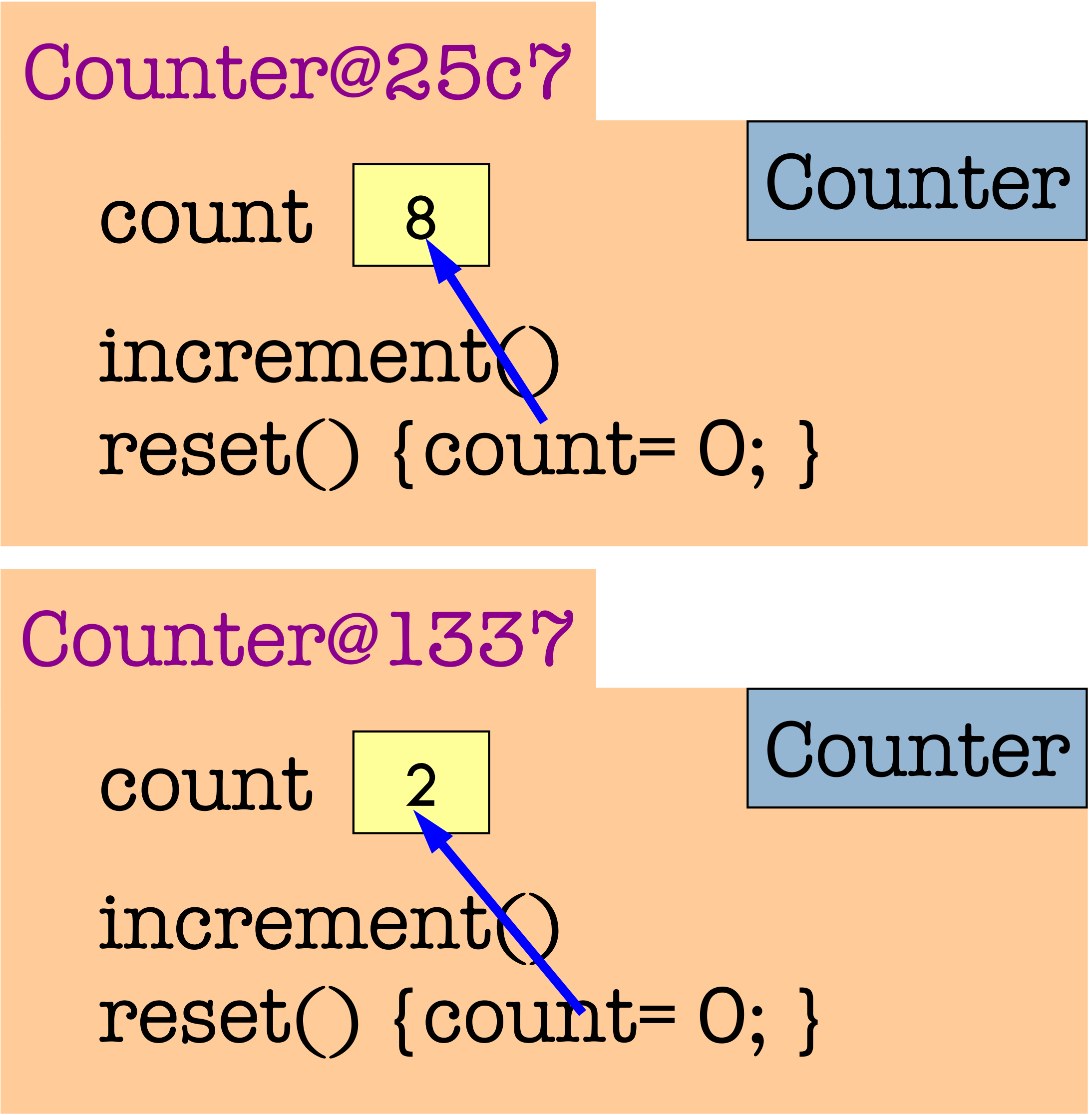
Inside-out rule:

to find the declaration of a name, look inside out

- start in closest enclosing scope
- then surrounding scope
- then surrounding
- etc.

Scoping

```
class Counter {
   int count;

   void increment() {
      count= count + 1;
   }

   void reset() {
      count= 0;
   }
}
```



New-expression

- **Instantiate** an object from a class: `new ClassName()`
- Evaluation:
  - Create a new object of that class
  - Yield object's name (address) as value of expression
- In context of assignment:
  - `Counter c= new Counter();`

The value of the new-expression is a pointer to the new object. The value is stored in variable c.

The end

LECTURE 2:
OBJECTS AND CLASSES

PART 5: OVERLOADING

CS 2110
Fall 2021

Overloading

"There are only two hard things in computer science: cache invalidation and naming things."

—attributed to Phil Karlton (Netscape developer)

Methods that reset count

```
class Counter {
   int count;
   …
   void reset() {
      count= 0;
   }
   void setTo(int i) {
      count= i;
   }
}
```

Overloading: use the same name

```
class Counter {
    int count;
    …
    void reset() {
        count= 0;
    }
    void reset(int i) {
        count= i;
    }
}
```

Q: reset now could mean one of two things to recipient object: how does it know which one to use?

Signatures

- Signature of method:
  - **name**, and
  - **argument types**, but
  - not its return type

- **Method:** void reset() { count= 0; }
- **Signature:** reset()

- **Method:** void reset(int i) { count= i; }
- **Signature:** reset(int)

Overloading: use the same name

```
class Counter {
    int count;
    ...
    void reset() {
        count= 0;
    }
    void reset(int i) {
        count= i;
    }
}
```

Q: reset message now could mean one of two things to recipient object: how does it know which one to use?

A: It uses the one with the right signature for the arguments in the method call.

c1.reset()   c1.reset(5)

Overloading happens often!

Class Math, which comes with Java, has lots of methods for basic numeric operations.

```
int      abs(int a) { … }
long     abs(long a) { … }
double  abs(double a) { … }
float    abs(float a) { … }
```

Demo in jshell

Space used: remember, a byte is 8 bits.

int:	4 bytes	long: 8 bytes
float:	4 bytes	double: 8 bytes

Overloading happens often!

Class Math, which comes with Java, has lots of methods for basic numeric operations.

```
int       abs(int a) { … }
long      abs(long a) { … }
double  abs(double a) { … }
float     abs(float a) { … }
```

Math.abs(-5)	5 (an int)
Math.abs(2147483648L)	2147483648 (a long)
Math.abs(-3.14)	3.14 (a double)
Math.abs(-3.14F)	3.14 (a float)

Your Turn: Read in JavaHyperText

- □ Class definition, object
- □ Reference, pointer
- □ Field
- □ Method, parameter, argument, method call
- □ Return statement
- □ Scope, Inside-out rule
- □ New expression, instantiate
- □ Overload, signature

THIS IS YOUR TEXTBOOK. USE IT REGULARLY

You'll find lots of links to concepts we haven't explored yet. Don't panic!

The end