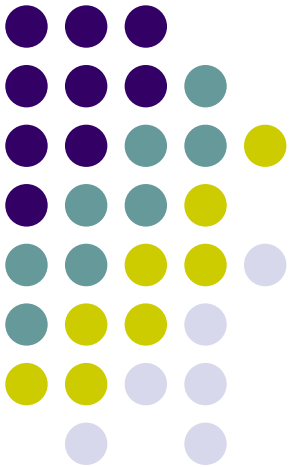
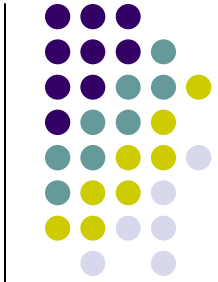
# 6.092: Java for 6.170

Lucy Mendel
MIT EECS



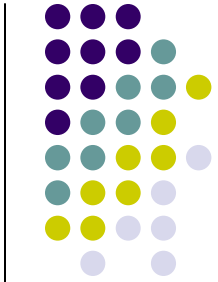
# Course Staff



- Lucy Mendel
- Corey McCaffrey
- Rob Toscano
- Justin Mazzola Paluska
- Scott Osler
- Ray He

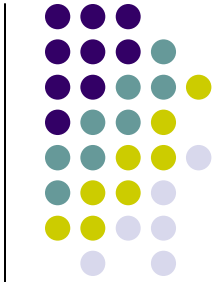
Ask us for help!

# Class Goals



- Learn to program in Java
  - Java
  - Programming (OOP)
- 6.170 problem sets are not supposed to take you 20 hours!
  - Tools, concepts, thinking

# Logistics



- 5 days long, optional second week
- 2 hrs lecture, 1 hr lab
  - End of week might be 1 hr lecture, 2 hr lab
  - Breaks!
- Labs
  - Work on homework with staff assistance (like LA hours in 6.170)
  - Mandatory even for listeners
  - Each is expected to take ~1-2 hrs

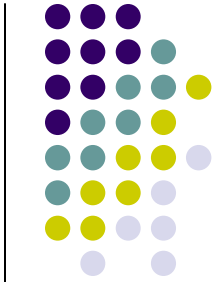
# Object Oriented Programming

- Objects have state
  - A person is an object and has a name, age, SS#, mother, &e.
- Programmers call methods on objects to compute over and potentially modify that state
  - programmer: How old are you?
  - object: I am 22.
  - programmer: Today is your birthday!
  - object: I have incremented my age by 1.

# Java Program

```
package hello;
import java.util.System;

class HelloWorld {
    String myString;

    void shout() {
        myString = new String("Hello, World!“);
        System.out.println(myString);
    }

    public static void main(String[] args) {
        HelloWorld myHelloWorld = new HelloWorld();
        myHelloWorld.shout();
    }
}
```

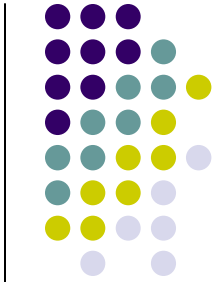
# Class



- Template for making objects
- Java is about objects → everything is in a class

```
class HelloWorld {              // classname
   … <everything> …
}
```

# Field

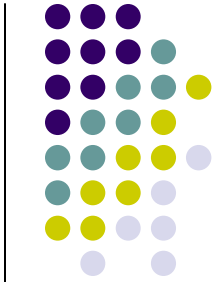


- Object state

```
class Human {
   int age;
}
```

<class type> <variable name>;

# Making objects

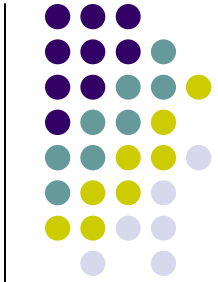


**Human lucy = new Human();**

- All object creation requires a "**new**"
- objects = instances (of classes)
- **lucy** is a pointer to the object
- We **assign** the constructed object to lucy

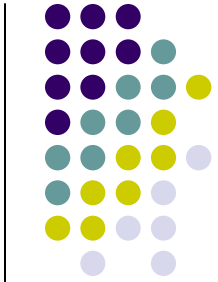
<type> <variable name> = <new object>;

# Using Objects



```
Human lucy = new Human();
lucy.age = 22;              // use '.' to access fields
Human david = new Human();
david.age = 19;
System.out.println(lucy.age);     // prints 22
System.out.println(david.age);    // prints 19
```

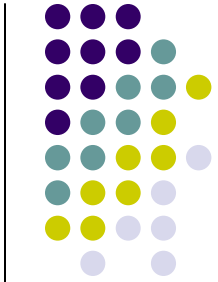
# 22



- Why did we not have to write

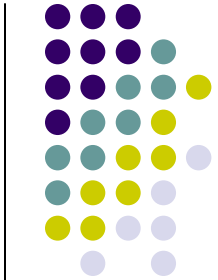
lucy.age = new int(22);  ??

# Primitives



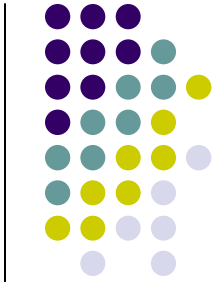
- Not *everything* is an object
- Some things are too simple and too frequently used to be bothered with objects:
  - boolean, byte, short, int, long, double, float, char

# Field *myString*



```
class HelloWorld {
    String myString;

    void shout() {
        myString = new String("Hello, World!");
        System.out.println(myString);
    }

    public static void main(String[] args) {
        HelloWorld myHelloWorld = new HelloWorld();
        myHelloWorld.shout();
    }
}
```

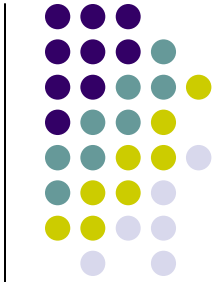
# Methods



- Process object state

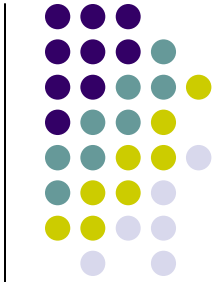
```
<return type> <method name>(<parameters>) {
   <method body>
}

myHelloWorld.shout();
                    // use '.' to access methods
```

# Constructors



- Constructors are special methods
  - no return type
  - use them to initialize fields
  - take parameters, normal method body (but no **return**)

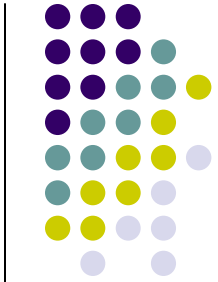
# Method Body



```
String firstname(String fullname) {
   int space = fullname.indexOf(“ ”);
   String word = fullname.substring(0, space);
   return word;
}
```

- Any number of parameters
- declare **local** variables
- return one thing (void = return nothing)

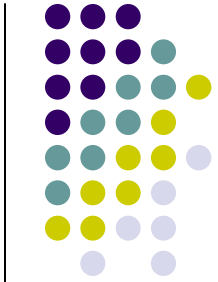
# Control Flow



```
if (lucy.age < 21) {
    // don't do stuff
} else if (lucy.hasCard()) {
    // do other stuff
} else {
    // doh
}
```

```
if (<predicate1>) {
    …
} else if (<predicate2>) {
    …
} else if (<predicate3>) {
….
} else if (<predicateN>) {
    …
} else { … }
```

# Predicates



- predicate = true or false (boolean)
- <, >, ==, <=, >=, !

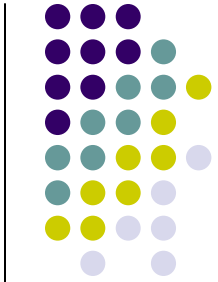
box.isEmpty()

box.numberBooks() == 0

!(box.numberBook() > 1)

box.numberBooks != MAX_NUMBER_BOOKS

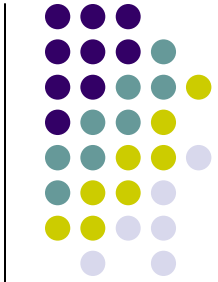
# For Loop



```
for (int i = 0; i < 3; i++) {
   System.out.println(i);       // prints 0 1 2
}
```

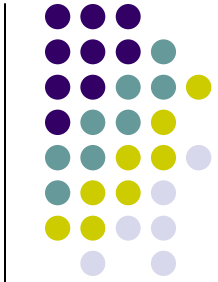
```
for (<initialize> ; <predicate> ; <increment>) {
     execute once                    every time
Stop when predicate is false
```

# While Loop



```
int i = 0;
while (i < 3) {
   System.out.println(i);      // prints 0 1 2
}

while (<predicate>) {
   …
}
```

# Combining Predicates



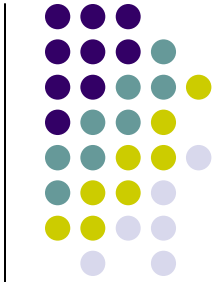
- && = logical and
- || = logical or

a. lucy.age >= 21 && lucy.hasCard

b. !someone.name.equals(“Lucy”))

c. (!true || false) && true

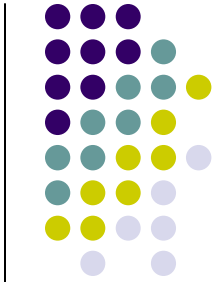
# Arrays



- Objects, but special like primitives

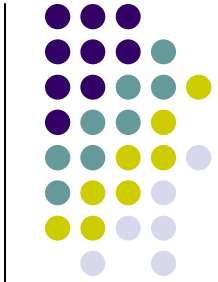
```
String[] pets = new String[2];
pets[0] = new String("Fluffy");
pets[1] = "Muffy";        // String syntactic sugar

String[] pets = new String[] {"Fluffy", "Muffy"};
System.out.println(pets.length);  // print 2
```

# Whoa, how many types are there?



- primitives
  - int a = 3 + 5;
- Objects
  - Integer a = new Integer(3);
  - Integer sum = a.add(5);
  - Arrays

# Objects Cause Trouble!!



- pets[3] >> halt program, throw ArrayOutOfBoundsException
- String[] str;
- str.length >> halt, throw NullPointerExceptoin

- Integer a = new Integer(3); // a→[3]
- a.add(5); // a→[3]
- a = a.add(5); // a→[8]

# Break (10 min)

- When we get back, more on Objects from Corey