CS 342 Software Design

Today: Java Members

- Fields
- Methods
- Nested Classes

Q & A: is there anything wrong with this code? Click A when you have posted.

```
class Point { int x, y; }
final class ColoredPoint extends Point { int color; }
class Colored3DPoint extends ColoredPoint { int z; }
```

Clicker Question: Which of the following are fundamental OOP concepts?

A. Internalization
B. Optimization
C. Encapsulation
D. Optimization and Inheritance
E. None of the above

Fields: Variables inside of a class

public: Available to anyone with access to the package

protected: Access only permitted inside the package containing class where protected member is declared

private: Available only in the body of top level class that encloses it, not inherited by subclass however they are in objects of subclasses

static: Only one exists for every instance of the class (class variable)

final: Once value is set it can not be changed. If a reference you can still modify the object

Methods:

Modifiers: **public, protected, private, static, final, abstract, synchronized**

Can override except for static, final and not inherited(private)

Only in a subclass, same arguments, same return type, access level is not more restrictive

```
class Point {
    int x, y;
    private Point() { reset(); }
    Point(int x, int y) { this.x = x; this.y = y; }
    private void reset() { this.x = 0; this.y = 0; }
}
class ColoredPoint extends Point {
    int color;
    void clear() { reset(); }
}
class Test {
    public static void main(String[] args) {
        ColoredPoint c = new ColoredPoint(0, 0);
        c.reset();
    }
}
```

Google Q&A:

Find the errors in this piece of code!

Post to Q&A
Click A when done

Hiding: Members that would otherwise be inherited but are not because of a declaration in a subclass

Clicker Question:

```
class Point {
    int x = 2;
}
class Test extends Point {
    double x = 4.7;
    public static void main(String[] args) {
        new Test().printX();
    }
    void printX() {
        System.out.println(x + " " + super.x);
    }
}
```

A: 2 2

B: 4.7 4.7

C: Doesn’t compile

D: 4.7 2

E: 2 4.7

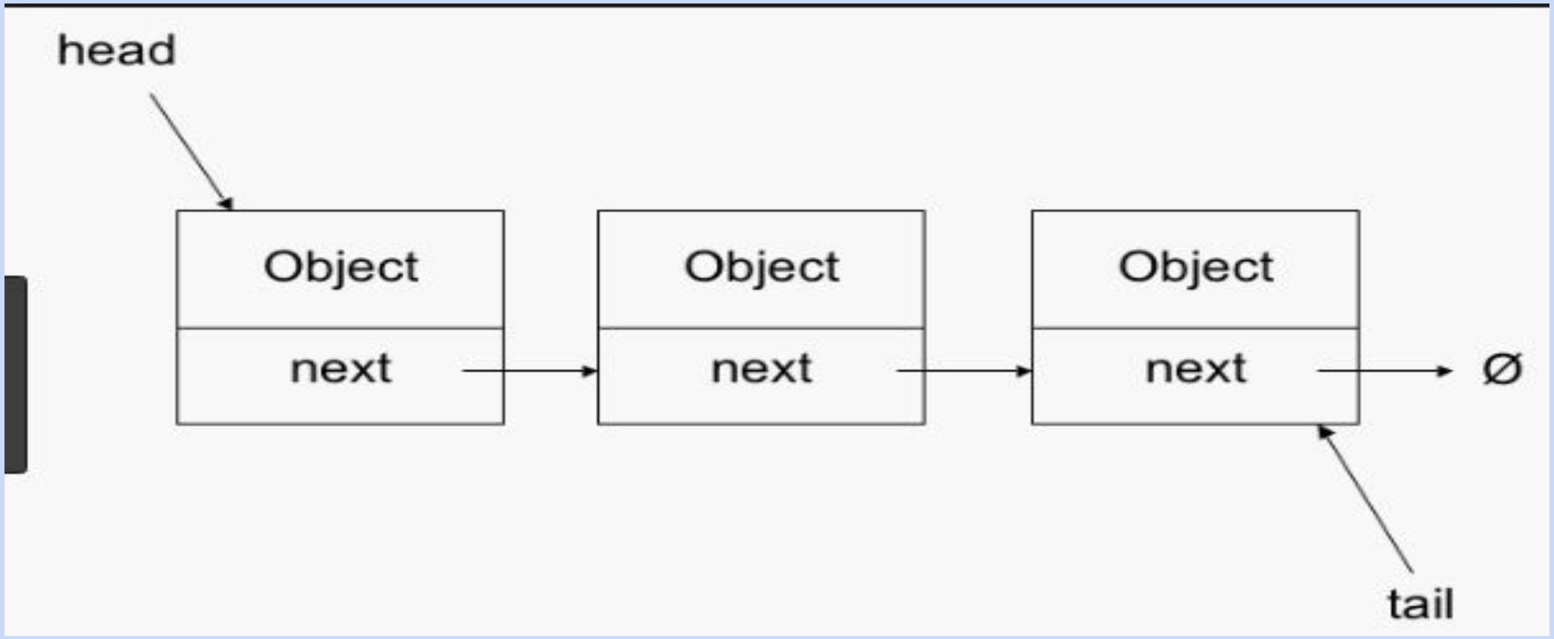
Nested Classes: Static and Non-Static

Why do we need nested classes?

- If one class is only useful to one other class, it makes sense to embed the “helper class”
- If class B needs access to the private members of class A, nesting class B gives it access and hides it from other code. (Encapsulation!)

When Would We Need a Nested Class?

A node for a linked list!



Nested Classes: Non-Static

- Associated with an instance of the enclosing class; can not define any static members
- Have access to members of enclosing class even if they are private.
- To instantiate, first instantiate object of outer class
- Modifiers: private, public, protected

Example:

```
JungleCat.MedReport mr = regis.new MedReport();
```

Nested Classes: Static

- Like a static method, static classes can not refer directly to instance variables or methods of the enclosing class.
- You can instantiate directly, you **DO NOT** need to first instantiate object of outer class

Example:

```
JungleCat.SIC scE = new JungleCat.SIC();
```

Nested Classes: Beware of Shadowing!

If a declaration of a type (such as a member variable or a parameter name) in a particular scope (such as an inner class or a method definition) has the same name as another declaration in the enclosing scope, then the declaration shadows the declaration of the enclosing scope. You cannot refer to a shadowed declaration by its name alone.