CSE 142, Spring 2013

Chapter 8

Lecture 8-3: Encapsulation, `this`

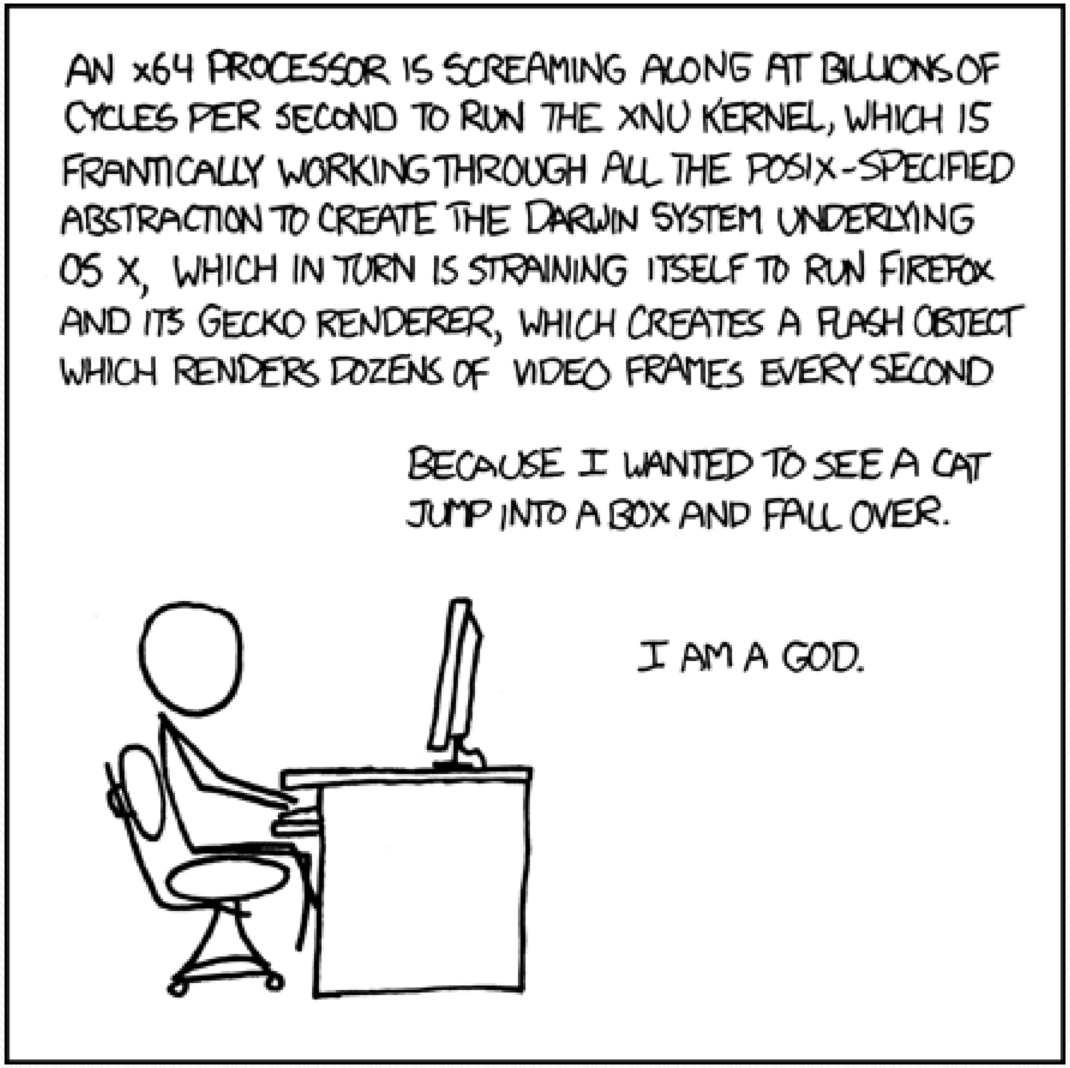
reading: 8.5 - 8.6

self-checks: #13-17

exercises: #5

Abstraction

Don't need to know this


AN x64 PROCESSOR IS SCREAMING ALONG AT BILLIONS OF CYCLES PER SECOND TO RUN THE XNU KERNEL, WHICH IS FRANTICALLY WORKING THROUGH ALL THE POSIX-SPECIFIED ABSTRACTION TO CREATE THE DARWIN SYSTEM UNDERLYING OS X, WHICH IN TURN IS STRAINING ITSELF TO RUN FIREFOX AND ITS GECKO RENDERER, WHICH CREATES A FLASH OBJECT WHICH RENDERS DOZENS OF VIDEO FRAMES EVERY SECOND
BECAUSE I WANTED TO SEE A CAT JUMP INTO A BOX AND FALL OVER.
I AM A GOD.

Can focus on this!!

The `toString` method

tells Java how to convert an object into a `String`

```
Point p1 = new Point(7, 2);
System.out.println("p1: " + p1);

// the above code is really calling the following:
System.out.println("p1: " + p1.toString());
```

- Every class has a `toString`, even if it isn't in your code.
  - Default: class's name `@` object's memory address (base 16)

```
Point@9e8c34
```

`toString` syntax

```
public String toString() {
    code that returns a String representing this object;
}
```

- Method name, return, and parameters must match exactly.

- Example:

```
// Returns a String representing this Point.
public String toString() {
    return "(" + x + ", " + y + ")";
}
```

Private fields

- A field can be declared *private*.
  - No code outside the class can access or change it.

```
private type name;
```

  - Examples:

```
private int id;
private String name;
```

- Client code sees an error when accessing private fields:

```
PointMain.java:11: x has private access in Point
System.out.println("p1 is (" + p1.x + ", " + p1.y + ")");
                                  ^
```

Accessing private state

- We can provide methods to get and/or set a field's value:

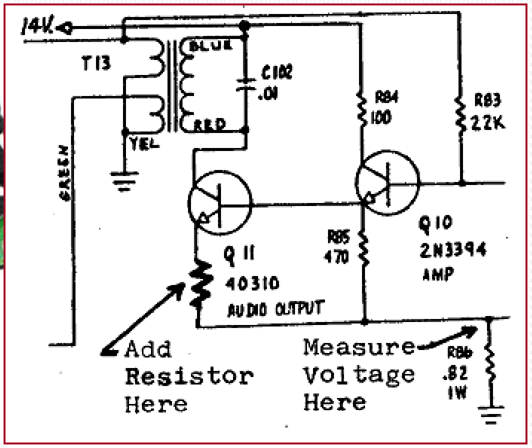
```
// A "read-only" access to the x field ("accessor")
public int getX() {
    return x;
}

// Allows clients to change the x field ("mutator")
public void setX(int newX) {
    x = newX;
}
```

  - Client code will look more like this:

```
System.out.println("p1: (" + p1.getX() + ", " + p1.getY() + ")");
p1.setX(14);
```

Encapsulation

- **encapsulation**: Hiding implementation details of an object from its clients.
  - Encapsulation provides *abstraction*.
    - separates external view (behavior) from internal view (state)
  - Encapsulation protects the integrity of an object's data.



Point class, version 4

```
// A Point object represents an (x, y) location.
public class Point {
    private int x;
    private int y;

    public Point(int initialX, int initialY) {
        x = initialX;
        y = initialY;
    }

    public double distanceFromOrigin() {
        return Math.sqrt(x * x + y * y);
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public void setLocation(int newX, int newY) {
        x = newX;
        y = newY;
    }

    public void translate(int dx, int dy) {
        x = x + dx;
        y = y + dy;
    }
}
```

Client code, version 4

```
public class PointMain4 {
    public static void main(String[] args) {
        // create two Point objects
        Point p1 = new Point(5, 2);
        Point p2 = new Point(4, 3);

        // print each point
        System.out.println("p1: (" + p1.getX() + ", " + p1.getY() + ")");
        System.out.println("p2: (" + p2.getX() + ", " + p2.getY() + ")");

        // move p2 and then print it again
        p2.translate(2, 4);
        System.out.println("p2: (" + p2.getX() + ", " + p2.getY() + ")");
  }
}
```

OUTPUT:

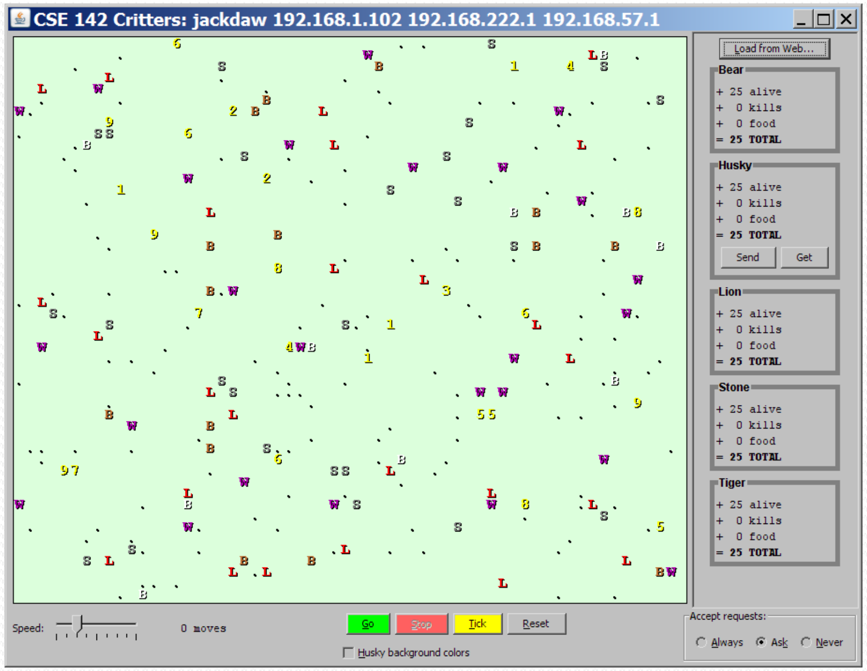
```
p1 is (5, 2)
p2 is (4, 3)
p2 is (6, 7)
```

CSE 142 Critters

- Ant
- Bird
- Hippo
- Vulture
- Husky (creative)

- behavior:

eat	eating food
fight	animal fighting
getColor	color to display
getMove	movement
toString	letter to display



CSE 142 Critters: jackdaw 192.168.1.102 192.168.222.1 192.168.57.1

```
Bear
+ 25 alive
+  0 kills
+  0 food
= 25 TOTAL

Husky
+ 25 alive
+  0 kills
+  0 food
= 25 TOTAL

Lion
+ 25 alive
+  0 kills
+  0 food
= 25 TOTAL

Stone
+ 25 alive
+  0 kills
+  0 food
= 25 TOTAL

Tiger
+ 25 alive
+  0 kills
+  0 food
= 25 TOTAL
```

Speed: 0 moves

Husky background colors

Accept requests: Always [✓ Ask] Never

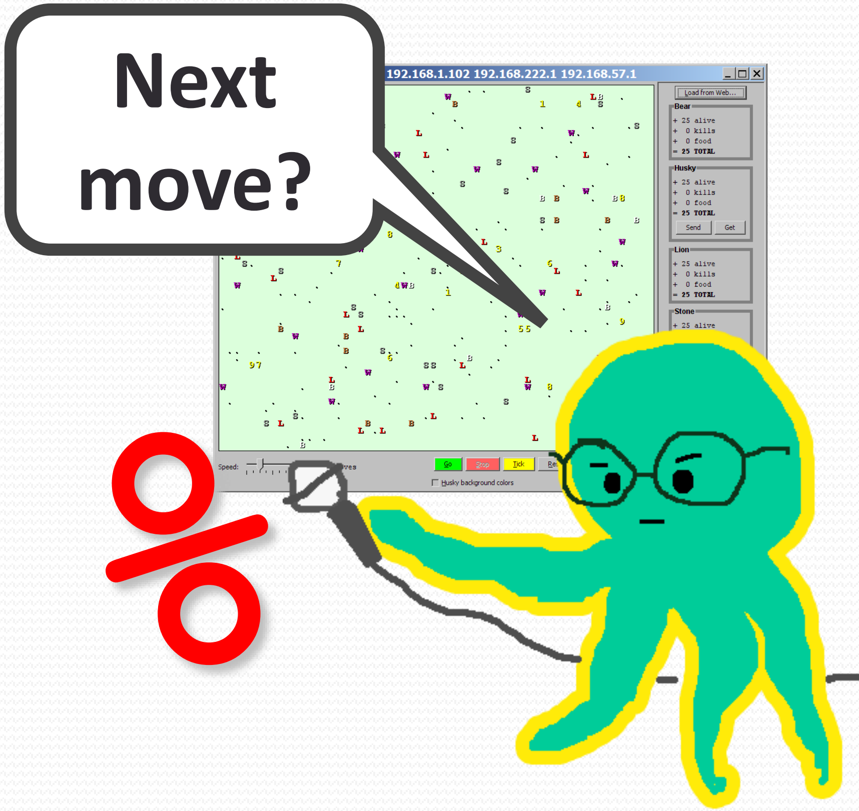
A `Critter` subclass

```
public class name extends Critter { ... }

public abstract class Critter {
    public boolean eat()
    public Attack fight(String opponent)
           // ROAR, POUNCE, SCRATCH
    public Color getColor()
    public Direction getMove()
           // NORTH, SOUTH, EAST, WEST, CENTER
    public String toString()
}
```

How the simulator works

- "Go" → loop:
  - move each animal (`getMove`)
  - if they collide, `fight`
  - if they find food, `eat`

- Simulator is in control!
  - `getMove` is <u>one move</u> at a time
    - (*no loops*)
  - Keep <u>state</u> (fields)
    - to remember future moves



Development Strategy

- Do one species at a time
  - in ABC order from easier to harder (Ant → Bird → ...)
  - debug `println`s

- Simulator helps you debug
  - smaller width/height
  - fewer animals
  - **"Tick"** instead of "Go"
  - **"Debug"** checkbox
  - drag/drop to move animals

Critter exercise: `Cougar`

- Write a critter class `Cougar`:

Method	Behavior
constructor	`public Cougar()`
`eat`	Always eats.
`fight`	Always pounces.
`getColor`	Blue if the `Cougar` has never fought; red if he has.
`getMove`	Walks west until he finds food; then walks east until he finds food; then goes west and repeats.
`toString`	`"C"`

Ideas for state

- You must not only have the right state, but update that state properly when relevant actions occur.

- Counting is helpful:
  - How many total moves has this animal made?
  - How many times has it eaten? Fought?

- Remembering recent actions in fields is helpful:
  - Which direction did the animal move last?
    - How many times has it moved that way?
  - Did the animal eat the last time it was asked?
  - How many steps has the animal taken since last eating?
  - How many fights has the animal been in since last eating?

Cougar solution

```
import java.awt.*;  // for Color

public class Cougar extends Critter {
    private boolean west;
    private boolean fought;

    public Cougar() {
        west = true;
        fought = false;
    }

    public boolean eat() {
        west = !west;
        return true;
    }

    public Attack fight(String opponent) {
        fought = true;
        return Attack.POUNCE;
    }

    ...
```

`Cougar` solution

```
    ...

    public Color getColor() {
        if (fought) {
            return Color.RED;
        } else {
            return Color.BLUE;
        }
    }

    public Direction getMove() {
        if (west) {
            return Direction.WEST;
        } else {
            return Direction.EAST;
        }
    }

    public String toString() {
        return "C";
    }
}
```