CSSE 220 Day 14

Interfaces and Event Based Programming

Check out *EventBasedProgramming* from SVN

Plan for Today

- Interfaces
- Event Listeners
- Java Swing

What Interfaces Do

- Express common operations that multiple classes might have in common
- Make “client” code more reusable
- Provide method signatures and documentation
- Do **NOT** provide method implementations or fields

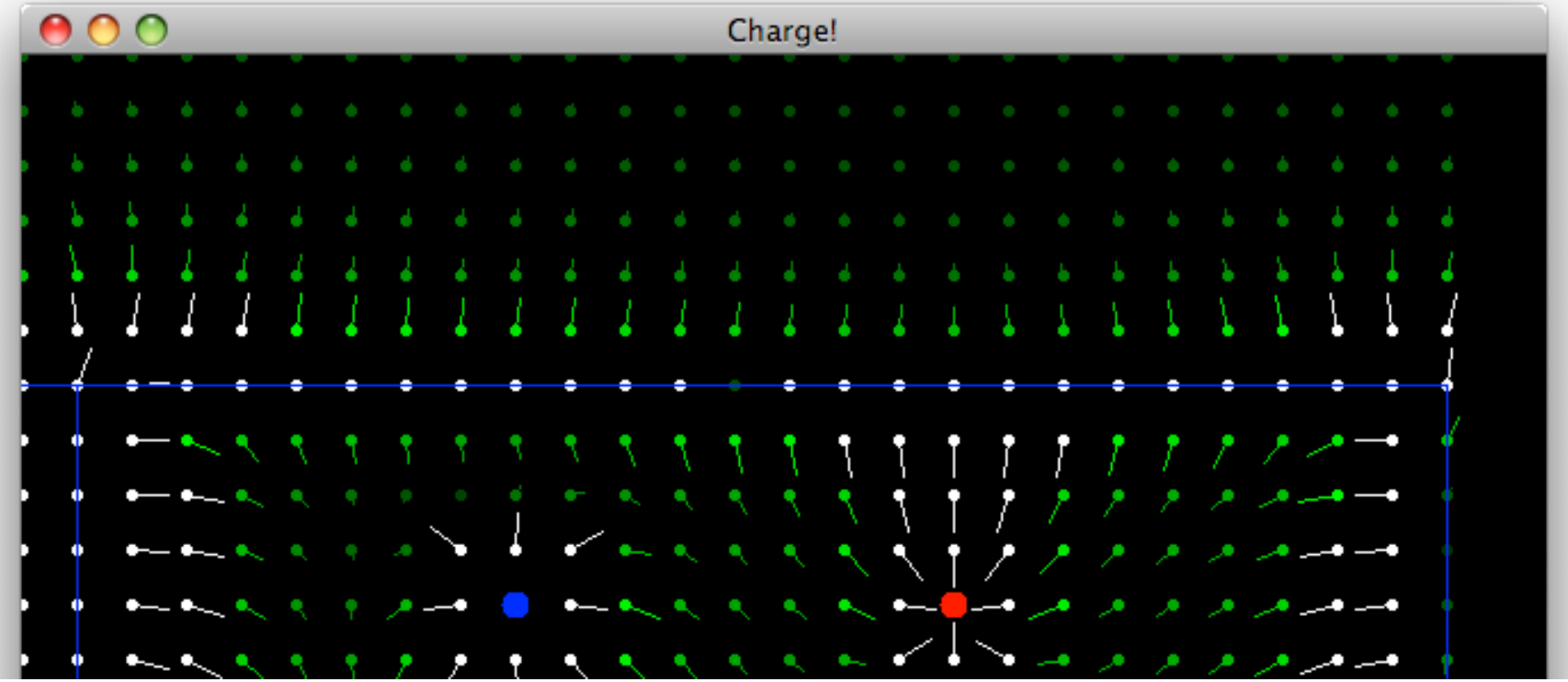
Q1&2

Interface Types: Key Idea

- Interface types are like **contracts**
  - A class can promise to **implement** an interface
    - That is, implement every method
  - Client code knows that the class will have those methods
    - Compiler verifies this
  - Any client code designed to use the interface type can automatically use the class!

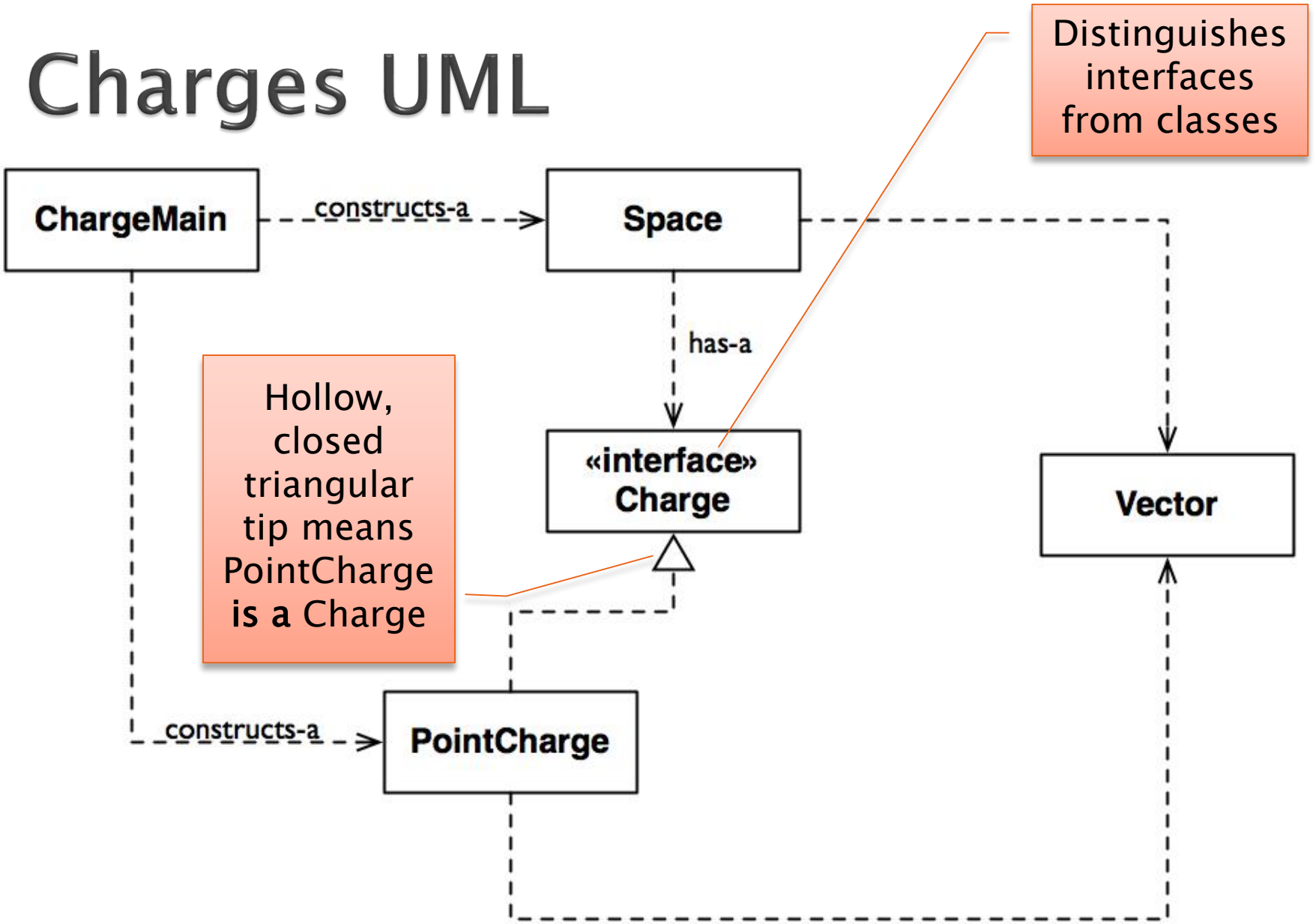
Q3

Live Coding



Charges Demo

Charges UML



Q4

Notation: In Code

interface, not class

```
public interface Charge {
        /**
         *  regular javadocs here
         */
        Vector forceAt(int x, int y);

        /**
         *  regular javadocs here
         */
        void drawOn(Graphics2D g);
    }

public class PointCharge implements Charge {
}
```

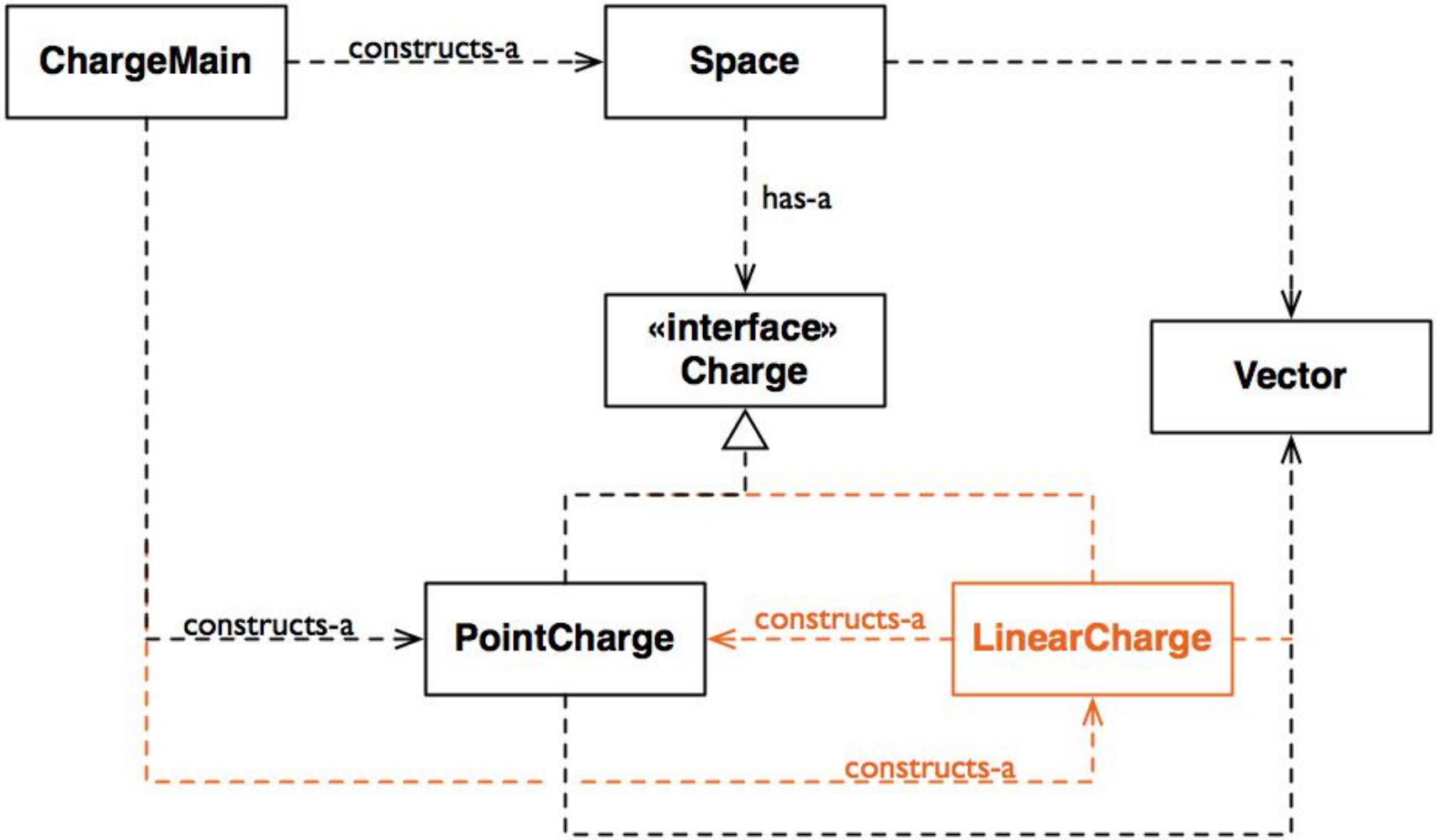
No "public", automatically are so

No method body, just a semi-colon

PointCharge promises to implement all the methods declared in the **Charge** interface

Q5

Updated Charges UML



Interfaces reduce coupling!

How does all this help reuse?

- Can pass an **instance** of a class where an interface type is expected
  - But only *if the class* `implements` *the interface*
- We passed `LinearCharge`s to `Space`'s `addCharge(Charge c)` method without changing `Space`!

- **Use interface types** for field, method parameter, and return types whenever possible

Q6

Polymorphism

- Origin:
  - Poly → many
  - Morphism → shape
- Classes implementing an interface give **many differently "shaped" objects for the interface type**

- **Late Binding**: choosing the right method based on the actual type of the implicit parameter **at run time**

Why is this OK?

```
Charge c = new PointCharge(…);
Vector v1 = c.forceAt(…);
c = new LinearCharge(…);
Vector v2 = c.forceAt(…);
```

- The type of the **actual object** determines the method used.

Break Time, then Java Swing!

Graphical User Interfaces in Java

- We say what to draw
- Java windowing library:
  - Draws it
  - Gets user input
  - Calls back to us with events
- We handle events

Handling Events

- Many kinds of events:
  - Mouse pressed, mouse released, mouse moved, mouse clicked, button clicked, key pressed, menu item selected, …
- We create **event listener objects**
  - that implement the right **interface**
  - that handle the event as we wish
- We **register** our listener with an **event source**
  - Sources: buttons, menu items, graphics area, …

Q7

Using Inner Classes

- Classes can be defined **inside** other classes or methods
- Used for "smallish" helper classes

- Example: `Ellipse2D.Double`

- Often used for `ActionListener`s…

Inner Classes and Scope

- **Inner classes can access any variables in surrounding scope**

- Caveats:
  - Local variables must be `final`
  - Can only use instance fields of surrounding scope if we're inside an instance method

- Example:
  - Prompt user for what porridge tastes like

Anonymous Classes

- Sometimes very small helper classes are only used once
  - This is a job for an anonymous class!

- **Anonymous** → no name
- A special case of inner classes

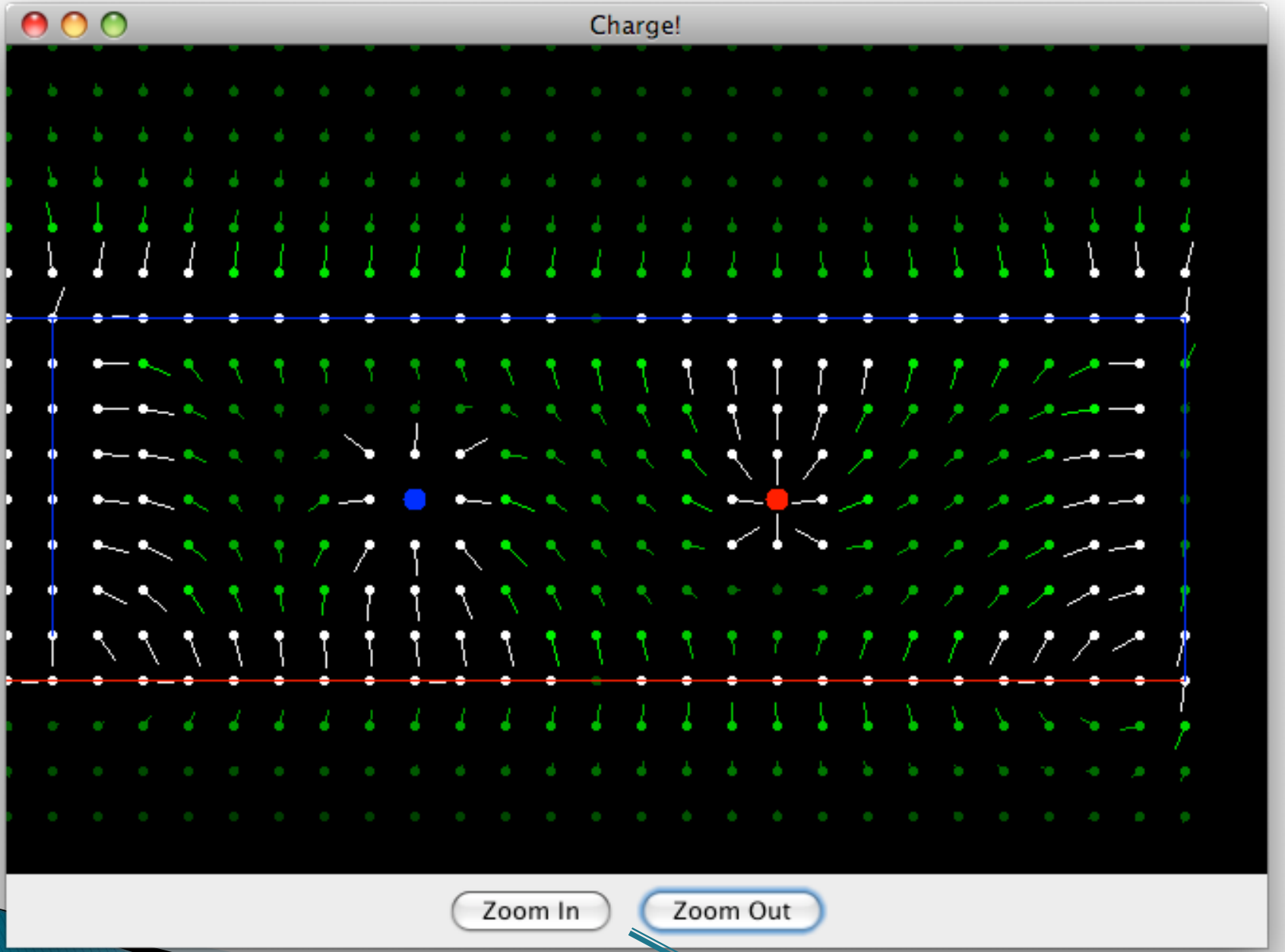
- Used for the simplest **`ActionListener`**s…

Time to Make the Buttons

- Layout in Java windows

Key Layout Ideas

- JFrame's add(Component c) method
  - Adds a new component to be drawn
  - Throws out the old one!
- JFrame also has method add(Component c, Object constraint)
  - Typical constraints:
    - BorderLayout.NORTH, BorderLayout.CENTER
  - Can add one thing to each "direction", plus center
- JPanel is a container (a thing!) that can display multiple components



So, how do we do this?

Repaint (and thin no more)

- To update graphics:
  - We tell Java library that we need to be redrawn:
    - `space.repaint()`
  - Library calls `paintComponent()` when it's ready

- **Don't call `paintComponent()` yourself! It's just there for Java's call back.**

Q11

Mouse Listeners

```
public interface MouseListener {
   public void mouseClicked(MouseEvent e);
   public void mouseEntered(MouseEvent e);
   public void mouseExited(MouseEvent e);
   public void mousePressed(MouseEvent e);
   public void mouseReleased(MouseEvent e);
}
```

Q12

Work Time

- Linear Lights Out