Events & Objects

… delegates, encapsulation, polymorphism, inheritance

Objects

In the 1st lab we identified some patterns.

Things the computer is good at.

The original idea of patterns from Chris Alexander was a language for people to discuss independent of how experienced they are.

Sequence, Repetition, Selection, Exceptions … are all patterns.

Today we will add delegates, encapsulation, polymorphism and inheritance.

Design Patterns -> Extreme Programming

Kent Beck, Ward Cunningham and Ron Jeffries formulated extreme Programming in 1999.

Beck wrote his book, Implementation patterns after XP and the agile manifesto.

Even though Agile and XP don’t advocate big design up front the design patterns still seem relevant.

The patterns that we are discussing are foundational.

Delegate

A Handler for an Event

Sometimes called publish and subscribe

```
app.post("/sms", (req, res) =>{
        let sFrom = req.body.From || req.body.from;
        let sMessage = req.body.Body||
req.body.body;
        let d = new Date();
        d.setMinutes(d.getMinutes() + 20);
        let aReply = [`Thank you for your order
${sFrom}`, `Please pick it up at
${d.toTimeString()}`];
        res.setHeader('content-type', 'text/xml');
        let sResponse = "<Response>";
        for(let n = 0; n < aReply.length; n++){
        sResponse += "<Message>";
        sResponse += aReply[n];
        sResponse += "</Message>";
        }
        res.end(sResponse + "</Response>");
});
```

Encapsulation

```
module.exports = class Order{
        constructor(){
        this.stateCur =
OrderState.WELCOMING;
        this.orderItems = [];
        }
        handleInput(sInput){
        let aReturn = [];
        switch(this.stateCur){
        case
OrderState.WELCOMING:

this.orderItems.push(sInput)
                aReturn.push("Would
you like drinks with that?");
                this.stateCur =
OrderState.DRINKS;
                break;
```

Mix behaviour and data

Polymorphism

The same stimulus or method elicits different behaviour.

```
module.exports = class Order{
        constructor(){
        this.stateCur = OrderState.WELCOMING;
        this.sSize = "";
        this.sToppings = "";
        this.sDrinks = "";
        this.sItem = "pizza";
        }
        handleInput(sInput){
        let aReturn = [];
        switch(this.stateCur){
        case OrderState.WELCOMING:
                this.stateCur = OrderState.SIZE;
                aReturn.push("Welcome to Fratello's
pizza.");
                aReturn.push("What size pizza
would you like?");
                break;
```

Inheritance

```
        let sMessage =
req.body.Body|| req.body.body;
        let aReply =
oOrders[sFrom].handleInput(sMess
age);
        if(oOrders[sFrom].isDone()){
        delete oOrders[sFrom];
        }
```

Also generalization and specialization

These 4 Ideas

Take us to modern programming

And Away from the command line

We will look for more patterns as we dig into Progressive Web Apps