Topic 7

Asynchronous Pattern

- Demand Driven
- Singular in nature
- Callbacks are required
  - Expectation that the callback will eventually run

Observer Pattern

- Request Driven Pattern
- Handle 0 or more processes
- You can setup a listener, but that listener is not guaranteed to fire

Key Features of Observer Pattern

- Subject <=> Observer
  - Each subject can have many observers
  - Observer can't have many subject
- Defining method of a subject is the emit method
  - `emit('eventName'<string>, [...data]);`
- Defining method of an observer is the on method
  - `.on('eventName'<string>, listener<function>)`
  - listener function receives any data passed to emit
- Observer cannot exist without subject (tightly coupled)

Example

```
// index.js

bambie = new Deer();
//photographer observer
bambie.on("jump", (height) => take_photo(height));
--------------------------------------------------
class{
    ...


    behvaior(){
        if(....){
            this.emit("jump", 20);
        }
    }
}
```

```
// Deer.js
```

Another example

```
const readline = require('readline');
const r1 = readline.createInterface({input: process.stdin, output:
process.stdout});

//both are the same
r1.on("line", (input), => console.log(`received ${123}`))
r1.on("SIGSTOP", () => console.log("done with ctrl-z"))
// function ask(){
//     r1.question("Enter some inout: ", (input) => {
//         console.log(`Received: ${input}`)
//         ask();
//     })
// }

// ask();
```

Recap Class 2

Observer Pattern

- Request Oriented
- 0 or more relation with events
- You have a subject with no observers (default state)

Event Emitter

- Emitter library in Node.JS
- `.emit(eventName <string>, [...data]);`
- `.on(eventName <string>, listener <function>);`

```
const EventEmitter = require("events");
class DayEmitter extends EventEmitter {
    constructor (update_time = 240){
        super();
        this.day = new Date();
        this.update_time = update_time
    }
    start(){
        this.day.setDate(this.day.getDate() + 1);
        let mm = `${(this.day.getMonth() + 1 + "").padStart(2, "0")}`
        let dd = `${(this.day.getDate() + 1 + "").padStart(2, "0")}`
        let temp = (Math.floor(Math.random()*70)).toString()
```

```
            this.emit('newday', {mm_dd: `${mm}/${dd}`, temp})
        }

    }


    =======================
    const day_emitter = new DayEmitter();

    day_emitter.on('newday', ({mm_dd}) => {
        process.stdout.cursorTo(0, 0);
        process.stdout.clearLine();
        process.stdout.write(mm_dd);
        process.stdout(cursorTo(0, 2));
    })

    day_emitter.on('newday', ({temp}) => {
        process.stdout.cursorTo(0, 0);
        process.stdout.clearLine();
        process.stdout.write(temp);
        process.stdout(cursorTo(0, 2));
    })
```