Software Engineering in the Arts and Humanities

Object-Oriented Programming; React, Part I

October 7, 2019

Reminders

- Lab 4 due Wed 10/9.
- Office hours tomorrow, Tue 10/8.
- Lab 5 releasing on Wed 10/9, due in **two weeks** on Wed 10/23.
- No Lab 6 anymore; this is the final one!
- Project proposals due Sun 10/20 (form coming next week).
  - Now is the time to start thinking about:
    - What field you want to be your domain
    - What data sets you might want to find/construct
    - Who your team will be (size 3-4 is ***required***, no exceptions)

Agenda

- Programming Paradigms
- OOP
  - Classes
  - Objects
  - Methods and properties
  - Abstraction
  - Inheritance
- React: client-side

Bank Accounts

- A canonical real-world example of a type of problem that is difficult to model programmatically without the use of objects is a bank account.

Bank Accounts

- A canonical real-world example of a type of problem that is difficult to model programmatically without the use of objects is a bank account.

- What are some of the things that come to mind when you think about what an account is?

Bank Accounts

- A canonical real-world example of a type of problem that is difficult to model programmatically without the use of objects is a bank account.

- What are some of the things that come to mind when you think about what an account is?

- What about some of the interactions that one can have with an account?

Paradigms

Paradigms

- Especially if this is only your second computer science course, odds are most of the programming you have done follows an *imperative*, *procedural* paradigm.

Paradigms

- Especially if this is only your second computer science course, odds are most of the programming you have done follows an *imperative*, *procedural* paradigm.

- By *imperative*, we mean that we as programmers explicitly tell the program how to manipulate its state.

Paradigms

- Especially if this is only your second computer science course, odds are most of the programming you have done follows an *imperative*, *procedural* paradigm.

- By *imperative*, we mean that we as programmers explicitly tell the program how to manipulate its state.

- By *procedural*, we mean that our code is typically organized into a series of procedure (function) calls, and those procedures manipulate data/state.

Paradigms

- By contrast, React (which we'll be talking about later today and Wednesday) uses a *declarative* paradigm. It's results-oriented, and less detail-oriented.
  - SQL is the same way, if you think about it!

Paradigms

- By contrast, React (which we'll be talking about later today and Wednesday) uses a *declarative* paradigm. It's results-oriented, and less detail-oriented.
  - SQL is the same way, if you think about it!

- Object-oriented programming, while still *imperative*, is rather the opposite of procedural.

Paradigms

- By contrast, React (which we'll be talking about later today and Wednesday) uses a *declarative* paradigm. It's results-oriented, and less detail-oriented.
  - SQL is the same way, if you think about it!

- Object-oriented programming, while still *imperative*, is rather the opposite of procedural.

- Our focus will be on fundamentals. For more, CS51 et al.

Classes

Classes

- The concept of a class allows to conceptualize models to define what are effectively templates for instances of those models.

Classes

- The concept of a class allows to conceptualize models to define what are effectively templates for instances of those models.

- Let's think about defining an actual class for those bank accounts we talked about earlier.

Objects

Objects

- Reframed in this context, when named classes are in play, and object is not just a collection of methods and properties; it is a manifestation of an instance of a class.

Objects

- We can still use objects in a procedural manner. JavaScript can behave as an imperative, procedural language, as we've seen.

Objects

- We can still use objects in a procedural manner. JavaScript can behave as an imperative, procedural language, as we've seen.

```
function(object);
```

Objects

- We can still use objects in a procedural manner. JavaScript can behave as an imperative, procedural language, as we've seen.

~~`function(object);`~~

Objects

- We can still use objects in a procedural manner. JavaScript can behave as an imperative, procedural language, as we've seen.

```
object.function();
```

Objects, recap

- Objects generally have two main "things" that we care about.
  - Properties
  - Methods

Objects, recap

- Objects generally have two main "things" that we care about.
  - Properties
  - Methods

- We can use this combination to use objects to model something about the world, such that you can imagine an object as being something physically manipulable.

Objects, recap

- Objects generally have two main "things" that we care about.
  - Properties
  - Methods

- We can use this combination to use objects to model something about the world, such that you can imagine an object as being something physically manipulable.
  - By way of analogy, consider a vehicle, a bank account, a person.

Objects, recap

- Properties are data fields that describe information about the object itself in its current state.

Objects, recap

- Properties are data fields that describe information about the object itself in its current state.

- Methods are functions that affect the object in some way, either by reading a property, modifying a property, or causing the object to interact with something else. (another object, for instance).

Objects, recap

- Properties are data fields that describe information about the object itself in its current state.

- Methods are functions that affect the object in some way, either by reading a property, modifying a property, or causing the object to interact with something else. (another object, for instance).

- Objects can be manifestations of classes which predefine a standard set of properties and methods as a template.

Abstraction

Abstraction

- The example programs we ran earlier in *account0* and *account1* are, while technically correct, flawed. Why?

Abstraction

- The example programs we ran earlier in *account0* and *account1* are, while technically correct, flawed. Why?

- It would be nice for accounts to have methods of "self-defense," protecting against user error and disallowing invalid actions.

Abstraction

- The example programs we ran earlier in *account0* and *account1* are, while technically correct, flawed. Why?

- It would be nice for accounts to have methods of "self-defense," protecting against user error and disallowing invalid actions.

- For this reason, it is generally considered good practice to abstract property manipulation away to methods.
  - Some languages force this (Java), other languages/libraries *very strongly suggest* it (React).

Inheritance

Inheritance

- Some classes of objects are very similar to others; so much so that redefining that class would be redundant.

Inheritance

- Some classes of objects are very similar to others; so much so that redefining that class would be redundant.

- In most object-oriented languages, a mechanism that can be used to handle exactly this situation is known as object *inheritance*, whereby one object class can effectively be built off of another.

Inheritance

- Some classes of objects are very similar to others; so much so that redefining that class would be redundant.

- In most object-oriented languages, a mechanism that can be used to handle exactly this situation is known as object *inheritance*, whereby one object class can effectively be built off of another.

- The newly-defined class *inherits* all of the properties and methods of the parent, and can define more beyond.

Inheritance

```
class Student extends Person {
   constructor() {
         super(first, last);
         this.id = id;
   }

   reportInfo() {
         // stuff
   }

   introduce() {
         // stuff
   }

}
```

Inheritance

superclass

```
class Student extends Person {
    constructor() {
        super(first, last);
        this.id = id;
    }

    reportInfo() {
        // stuff
    }

    introduce() {
        // stuff
    }

}
```

Inheritance

subclass

```
class Student extends Person {
    constructor() {
            super(first, last);
            this.id = id;
    }

    reportInfo() {
            // stuff
    }

    introduce() {
            // stuff
    }

}
```

Inheritance

superclass constructor

```
class Student extends Person {
    constructor() {
        super(first, last);
        this.id = id;
    }

    reportInfo() {
        // stuff
    }

    introduce() {
        // stuff
    }

}
```

Inheritance

subclass property

```
class Student extends Person {
    constructor() {
        super(first, last);
        this.id = id;
    }

    reportInfo() {
        // stuff
    }

    introduce() {
        // stuff
    }

}
```

Inheritance

subclass method

```
class Student extends Person {
    constructor() {
        super(first, last);
        this.id = id;
    }

    reportInfo() {
        // stuff
    }

    introduce() {
        // stuff
    }

}
```

Inheritance

override

```
class Student extends Person {
    constructor() {
        super(first, last);
        this.id = id;
    }

    reportInfo() {
        // stuff
    }

    introduce() {
        // stuff
    }

}
```

React

React

- JavaScript library created in 2013 and maintained by Facebook.

React

- JavaScript library created in 2013 and maintained by Facebook.

- Goal is to make development of single-page application front-ends much cleaner, relying on *declarative* programming techniques to reduce tedium.

React

- JavaScript library created in 2013 and maintained by Facebook.
- Goal is to make development of single-page application front-ends much cleaner, relying on *declarative* programming techniques to reduce tedium.
- Heavily built around the techniques of object-oriented programming and inheritance.

JSX

JSX

- "JavaScript XML" – looks very similar to HTML itself, but allows us to basically treat HTML-like syntax as JavaScript objects themselves.

JSX

- "JavaScript XML" – looks very similar to HTML itself, but allows us to basically treat HTML-like syntax as JavaScript objects themselves.

```
const element = (
    <div>
        Hello, world!
    </div>
);
```

JSX

- "JavaScript XML" – looks very similar to HTML itself, but allows us to basically treat HTML-like syntax as JavaScript objects themselves.

```
const element = (
    <div>
        Hello, world!
    </div>
);
```

JSX

- Also possible to nest JavaScript expressions and embed them in JSX.

```
const name = "Doug";
const element = (
    <div>
        Hello, {name}!
    </div>
);
```

JSX

- Also possible to nest JavaScript expressions and embed them in JSX.

```
const name = "Doug";
const element = (
    <div>
        Hello, {name}!
    </div>
);
```

Properties

`this.props`

State

`this.state`

Lifecycle Methods

`componentDidMount()`

`componentWillUnmount()`

...