UCI University of California, Irvine

Informatics 134

Software User Interfaces
Winter 2021

Mark S. Baldwin
baldwinm@ics.uci.edu

2/1/2022

Agenda

1. This Week
2. Basic Structured Graphics Recap
3. Application of Structured Graphics
4. Assignment 3: Custom Graphical Toolkit
5. References

This Week

This Week

- Lecture on Thursday
- Keep working on T2 (Part 1 DUE tonight, Part 2 due 2/8)
- Start working on A3

Basic Structured Graphics Recap

Basic Structured Graphics Recap

Structured Graphics Overview

Encapsulate a primitive (rectangles, lines, images, icons, etc.)

Expose reusable code for rendering how and when.

Enable programmer to create the what (e.g., a button)

Basic Structured Graphics Recap

Advantages of Structured Graphics

- Less code, more reusable
- Encapsulation of common mechanisms enables automation of required actions like redraw and refresh
- Hierarchical model supports custom encapsulation as well

Basic Structured Graphics Recap

Some Trade-offs

- Supporting reuse increases memory consumption
- Redraw and refresh can take more time
- Combined, can effect ’snappiness’ of UI

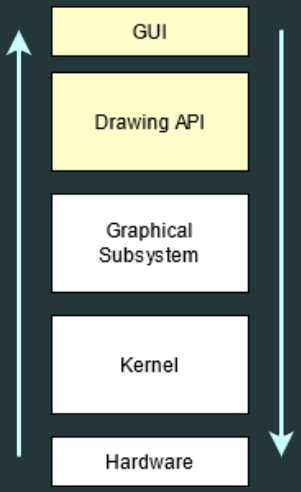
Though modern computing power negates most of these concerns

Application of Structured Graphics

Application of Structured Graphics

Build your own graphical toolkit?

- Ground up using 'low-level' graphics primitives
- Work in the browser (avoid intricacies of OS)
- Build on event propagation model of the DOM (rather create our own!)



Application of Structured Graphics

Two Options:

Drawing

- Objects retain specification after draw
- Objects can be moved, changed, and scaled
- Better suited for user interfaces, high resolution
- In browser: Scalable Vector Graphics (SVG)

Painting

- Objects become pixels after draw
- Editable through change
- Better suited for games/graphics, low or fixed resolution
- In browser: Canvas

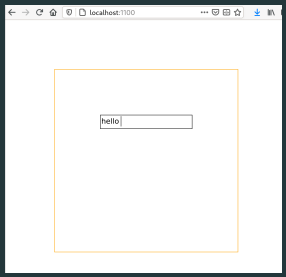
Application of Structured Graphics

We will be using SVG

- SVG.js (https://svgjs.dev/)
- No dependencies–fast
- Choose your own editor or IDE

Hierarchy in SVG

Let’s start with a look at hierarchy in raw SVG using SVG.js




```
1  import {SVG} from './svg.min.js';
2
3  SVG.on(document, 'DOMContentLoaded', function(){
4      var draw = SVG().addTo('body').size('1000px','1000px');
5      var window = draw.group();
6      window.rect(400,400).stroke("orange").fill("white");
7
8      var group = draw.group();
9      var rect = group.rect(200, 30).fill("white").stroke("black");
10     var text = group.text("hello").move(2,4);
11     var caret = group.line(45, 2.5, 45, 25).
12         stroke({ width: 1, color: "black" });
13
14     group.move(100,100);
15
16     window.add(group);
17     window.move(100,100);
18 });
```

SVG and the DOM

Which language?

- Javascript or Typescript?
- Typescript is built on strong types and object oriented principles
- Javascript does not require transpiling

SVG and the DOM

Object Oriented Model

- Already popular with GUI toolkits
- Conceptually similar to primitives and aggregates
- Easier (IMO) to reason about how a hierarchy should be structured.

Object Oriented Hierarchy in Typescript

- Abstraction
- Class == graphical object
- Instances
- Inheritance

```
interface IWidgetEvent{
    ...
}

class Window implements IWidgetEvent {
    private _objects: Widget[];

    constructor(height:any, width:any){
        ...
    }
    public addWidget(widget:Widget){
        this._objects.push(widget);
    }
}
```

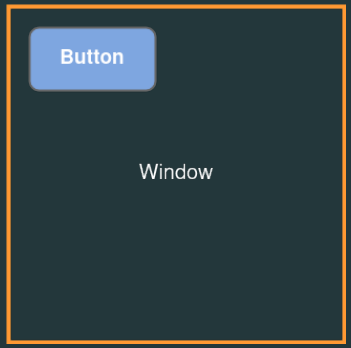
Object Oriented Hierarchy in Typescript

- Abstraction
- Class == graphical object
- Instances
- Inheritance

```
abstract class Widget implements IWidgetEvent {
    private _backcolor: string;

    constructor(height:any, width:any){
        ...
        this._backcolor = "black";
    }
    get backColor(): string{
        return this._backcolor;
    }
    set backColor(color:string){
        this._backcolor = color;
    }
}

class Button extends Widget{
    constructor(){
        super(100, 50);
        ...
    }
}
```




```
let w = new Window(500,500);
let btn = new Button();
btn.backColor = "blue";
w.addWidget(btn);
```

Object Oriented Model

Already, you can see how decisions need to be made.

- Design and optimize classes and interfaces to avoid duplicate code (costly space/performance)
- Encapsulate when possible (the Button class should not need to worry about backcolor change)
- Sensible names and parameters

Assignment 3: Custom Graphical Toolkit

DEMO

A3: Custom Graphical Toolkit

You will create the following widgets:

- Button (use starter code, customize)
- Check Box
- Radio Button
- Text Box
- Scroll Bar
- Progress Bar
- Custom (your choice)

A3: Custom Graphical Toolkit

You will be responsible for all of the following:

- Decide what functionality users of your widgets should be able to access.
- Apply a custom theme across all of your widgets
- Create a state chart for each widget
- Create a small GUI program that makes use of all of your widgets

A3: Custom Graphical Toolkit

Getting started:

- Full assignment will be release after class
- Start looking at the SVG.js documentation
- Start working on your state charts
- We will be covering more over the next few lectures

Let’s Dive In!

References

References i