

Principles of Software Construction: Objects, Design, and Concurrency

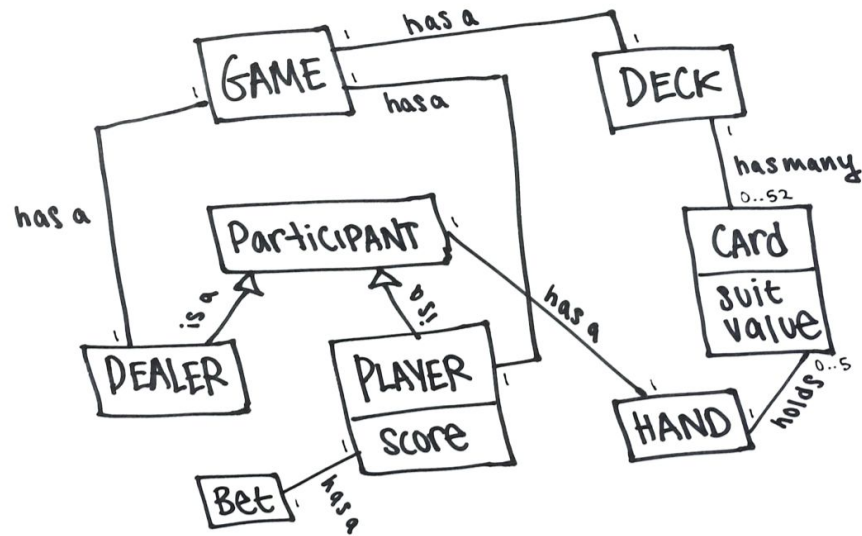
Introduction to GUIs

Jonathan Aldrich

Bogdan Vasilescu



We have done: a backend with no explicit interaction



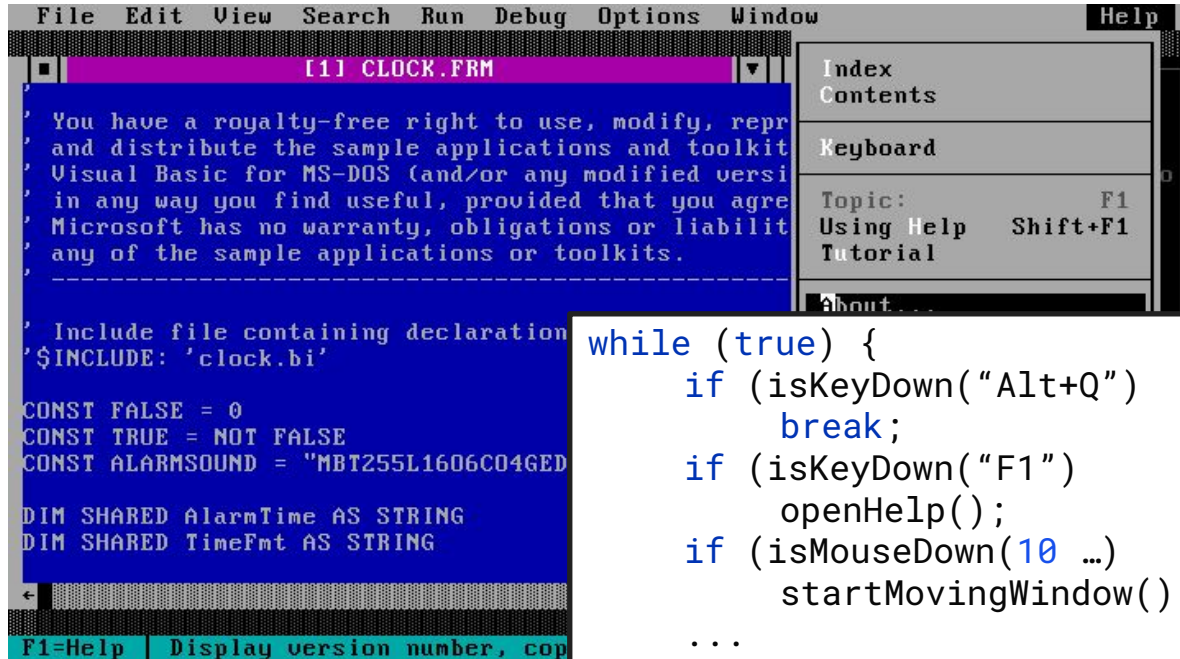
One Possible
Domain model
this is NOT a reference solution, it's
an example of what a domain model
looks like

Interaction with CLI

```
Terminal
File Edit View Search Terminal Help
scripts/kconfig/conf arch/x86/Kconfig
*
* Linux Kernel Configuration
*
* General setup
*
Prompt for developer
Local version - app
Automatically append
0) [N/y/?] y
Kernel compression
> 1. Gzip (KERNEL_C
  2. Bzip2 (KERNEL_
  3. LZMA (KERNEL_L
  4. LZO (KERNEL_LZ
choice[1-4?]: 3
Support for paging
System V IPC (SYSVI
POSIX Message Queues (POSIX_MESSAGE_QUEUES) [Y/n/?]
BSD Process Accounting (BSD_PROCESS_ACCT) [Y/n/?] n
Export task/process statistics through netlink (EXPERIMENTAL) (TASKSTATS) [Y/n/?]
1] y
Enable per task delay accounting (EXPERIMENTAL) (TASK_DELAY_ACCT) [Y/n/?]
```

```
Scanner input = new Scanner(System.in);
while (questions.hasNext()) {
    Question q = question.next();
    System.out.println(q.toString());
    String answer = input.nextLine();
    q.respond(answer);
}
```

How do you wait?



The screenshot shows a Visual Basic IDE window titled 'CLOCK.FRM'. The main editor area contains the following code:

```
' You have a royalty-free right to use, modify, repr  
' and distribute the sample applications and toolkit  
' Visual Basic for MS-DOS (and/or any modified versi  
' in any way you find useful, provided that you agre  
' Microsoft has no warranty, obligations or liabilit  
' any of the sample applications or toolkits.  
-----  
' Include file containing declaration  
' $INCLUDE: 'clock.bi'  
  
CONST FALSE = 0  
CONST TRUE = NOT FALSE  
CONST ALARMSOUND = "MBT255L1606C04GED"  
  
DIM SHARED AlarmTime AS STRING  
DIM SHARED TimeFmt AS STRING
```

A help window is open on the right, showing a table of contents with the following entries:

Index	
Contents	
Keyboard	
Topic:	F1
Using Help	Shift+F1
Tutorial	

The status bar at the bottom of the IDE shows 'F1=Help | Display version number, cop'.

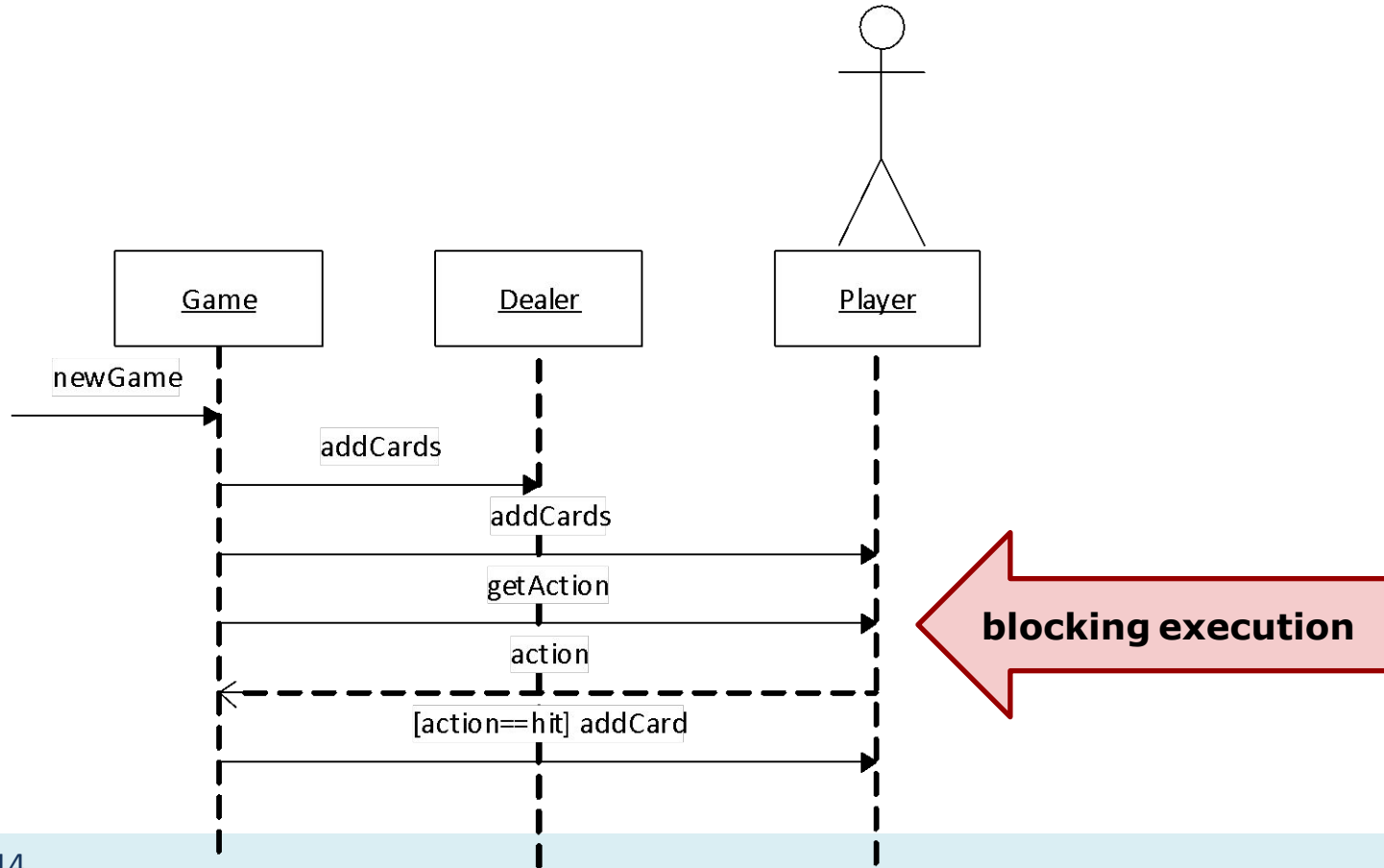
```
while (true) {  
    if (isKeyDown("Alt+Q"))  
        break;  
    if (isKeyDown("F1"))  
        openHelp();  
    if (isMouseDown(10 ...))  
        startMovingWindow();  
    ...  
}
```

How do you GUI? Multiplayer?



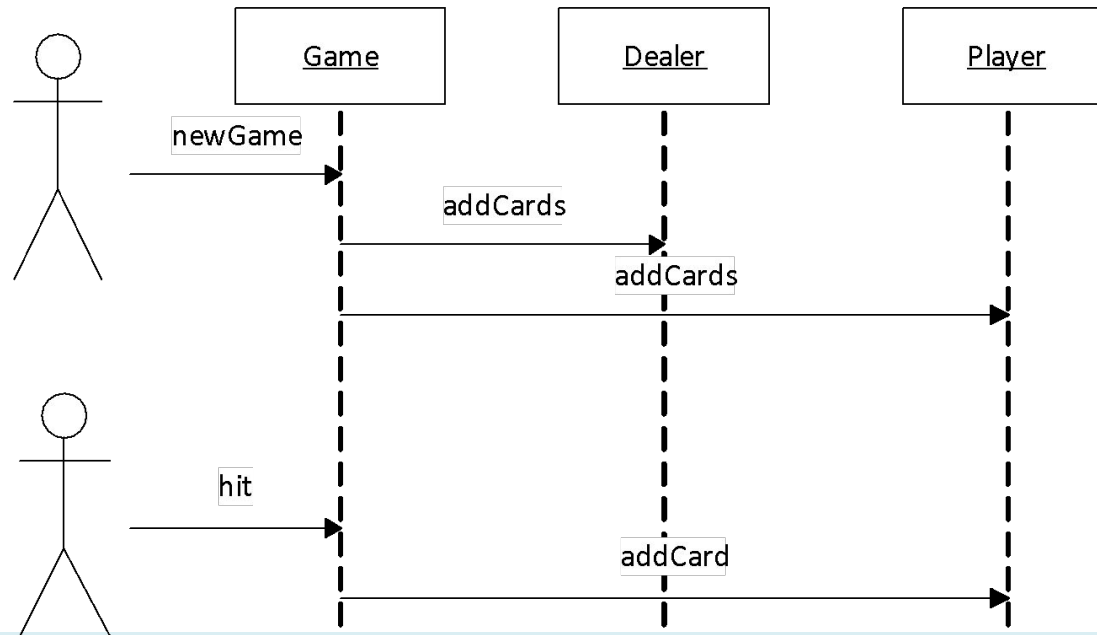
```
while (true) {  
    if (player === "player1") {  
        hasWon = play("player1");  
        if (hasWon) break;  
        player = "player2";  
    } else (player === "player2") {  
        hasWon = play("player2");  
        if (hasWon) break;  
        player = "player1";  
    }  
}
```

Potential issue: Blocking interactions with users



Interactions with users through events

- Do not block waiting for user response
- Instead, react to user events



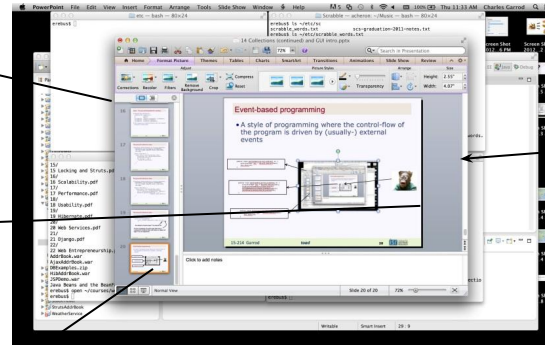
Event-based programming

Style of programming where control-flow is driven by (usually external) events

```
public void performAction(ActionEvent e) {  
    List<String> lst = Arrays.asList(bar);  
    foo.peek(42)  
}
```

```
public void performAction(ActionEvent e) {  
    bigBloatedPowerPointFunction(e);  
    withANameSoLongIMadeItTwoMethods(e);  
    yesIKnowJavaDoesntWorkLikeThat(e);  
}
```

```
public void performAction(ActionEvent e) {  
    List<String> lst = Arrays.asList(bar);  
    foo.peek(40)  
}
```



Event-based GUIs

Form Preview [ContactEditor]

Name

First Name: Last Name:

Title: Nickname:

Display Format:

E-mail

E-mail Address:

Item 1
Item 2
Item 3
Item 4
Item 5

Mail Format:
 HTML Plain Text Custom

OK Cancel

```
//static public void main...  
JFrame window = ...  
window.setDefaultCloseOperation(  
    WindowConstants.EXIT_ON_CLOSE);  
window.setVisible(true);
```

```
//on add-button click:  
String email = emailField.getText();  
emaillist.add(email);
```

```
//on remove-button click:  
int pos = emaillist.getSelectedItemId();  
if (pos >= 0) emaillist.delete(pos);
```

So, what about a frontend?

...in fact, let's start with basically just a frontend without an explicit backend.

(and we'll come back to that backend later.)

How To Make This Happen?

- Be comfortable with object-oriented concepts and with programming in the Java or JavaScript language
- Have experience designing medium-scale systems with patterns
- Have experience testing and analyzing your software
- Understand principles of concurrency and distributed systems

See a more detailed list of [learning goals](#) describing what we want students to know or be able to do by the end of the semester. We evaluate whether learning goals have been achieved through assignments and exams.

Coordinates

Tu/Th 3:05 - 4:25 p.m. in PH 100

As an IPE class, we will be teaching remotely for the first two weeks of the semester. Zoom links are available via Canvas. We will share those links with the waitlisted students for the first week or so while the waitlist is sorted out.

[Claire Le Goues](#), clegoues@cs.cmu.edu, TCS 363, office hours TBA (see calendar)

[Bogdan Vasilescu](#), TCS 326, office hours TBA (see calendar)

Our TAs also provide an additional 18h of office hours each week, usually in TCS 310, see details in the calendar.

The instructors have an open door policy: If the instructors' office doors are open and no-one else is meeting with us, we are happy to answer any course-related questions. Feel free to email us for appointments; we are also available over Zoom.

Course Calendar

17214 S22								
Today	Feb 28 – Mar 6, 2022					Week	Month	Agenda
	Mon 2/28	Tue 3/1	Wed 3/2	Thu 3/3	Fri 3/4	Sat 3/5	Sun 3/6	
9am			9:05 – 9:55 17214 Description A		9 – 11 Li Guo's OH https://cmu.zoom.us/j/6593343031			
10am			10:10 – 11 17214 Description B		11 – 12p Claire's OH (in person: TCS)			
11am			11:15 – 12:05p 17214 Description C		12:10p – 2:10p Deyuan's OH TCS 310, 4665 Forbes Ave, Pittsburgh, PA 15213, USA			
12pm			12:20p – 1:10p 17214 Description D	1p – 3p Lihao's OH https://cmu.zoom.us/j/921577524207 pwd=VG1BN244ck NKU3dGWTRFNW4y			4p – 3p Jake OH https://cmu.zoom.us/j/myjzych	
1pm			1:25p – 2:15p 17214 Description E	3:05p – 4:25p 17214 Lecture https://cmu.zoom.us/j/945133412687				
2pm			2:30p – 3:20p 17214 Description F	5p – 7p Katrina's OH (Remote) https://cmu.zoom.us/j/925144540677	6:05p – 7:05p Haoan OH https://cmu.zoom.us/j/myjzhr1723			
3pm	3p – 5p Julia OH TCS 432	3:05p – 4:25p 17214 Lecture https://cmu.zoom.us/j/945133412687						
4pm		4:45p – 6:45p Jessica OH TCS 310	4:30p – 5:30p Isabel OH TCS 310					
5pm	5p – 7p Michael OH TCS432, TCS Hall, 4665 Forbes Ave, Pittsburgh, PA							
6pm								

GUI Design: what do we want?

- Nested Elements
- Style Vocabulary
- Interactivity

GUI Design: what do we want?

- Nested Elements
 - HTML
- Style Vocabulary
 - CSS
- Interactivity
 - JavaScript

Anatomy of an HTML Page

Predefined elements

Root/\'document\'*

Header

Body

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body> == $0
    <nav id="navigation" class="hidden">...</nav>
    <header id="top" class="container">...</header>
    <div id="main" class="container">...</div>
  </body>
</html>
```

html body

Styles Computed Layout Event Listeners DOM Breakpoint

Filter

```
element.style {
}

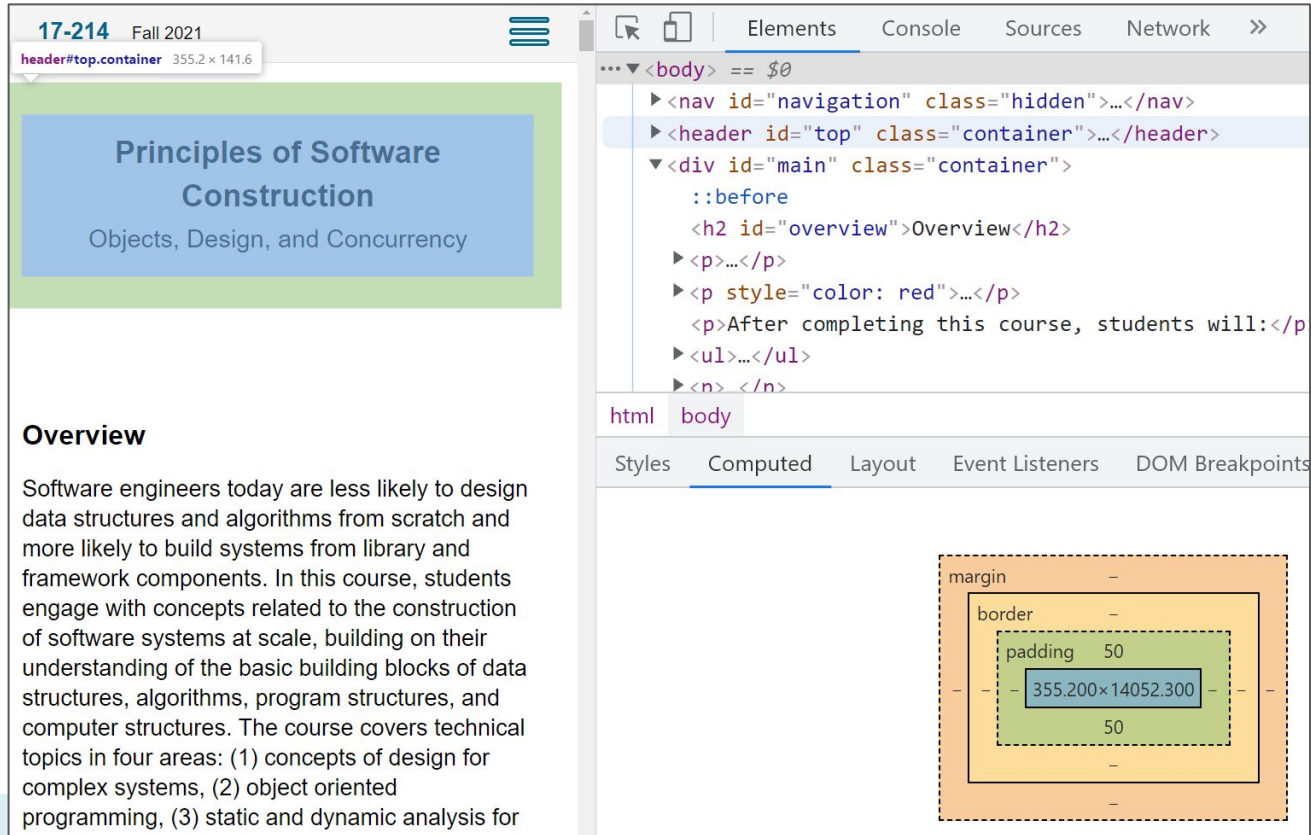
body {
  font-family: 'Helvetica Neue', Helvetica, sans-serif;
  font-size: .9em;
  padding: 50px 0;
}
```

body {

Anatomy of an HTML Page

Nested elements

- Sizing
- Attributes
- Text



The screenshot shows a web browser window with a page titled "17-214 Fall 2021". The page content includes a header with the text "Principles of Software Construction" and "Objects, Design, and Concurrency", and an "Overview" section. The developer tool is open, showing the HTML structure of the page. The HTML structure is as follows:

```
<body>
  <nav id="navigation" class="hidden">...</nav>
  <header id="top" class="container">...</header>
  <div id="main" class="container">
    ::before
    <h2 id="overview">Overview</h2>
    <p>...</p>
    <p style="color: red">...</p>
    <p>After completing this course, students will:</p>
    <ul>...</ul>
    <n> </n>
```

The developer tool also shows a diagram of the box model for the selected element, with the following dimensions:

- margin: -
- border: -
- padding: 50
- width: 355.200 x 14052.300
- height: 50

A few Tags

- `<html>`
 - The root of the visible page
- `<head>`
 - Stores metadata, imports
- `<p>`
 - A paragraph
- `<button>`
 - Attributes include ``name``, ``type``, ``value``
- `<div>`
 - Generic section -- very useful
- `<table>`
 - The obvious
- Many more; dig into a real page!

Anatomy of an HTML Page

Nested elements

- Sizing
- Attributes
- Text

You can write these out directly, or compose and modify them programmatically!

- Or, both! (we'll see in a minute).

The screenshot shows a web browser window with the URL `17-214 Fall 2021` and a page titled "Principles of Software Construction" with the subtitle "Objects, Design, and Concurrency". The page content includes an "Overview" section with text about software engineering education. The developer tools are open, showing the HTML structure of the page. The HTML structure is as follows:

```
<body>
  <nav id="navigation" class="hidden">...</nav>
  <header id="top" class="container">...</header>
  <div id="main" class="container">
    ::before
    <h2 id="overview">Overview</h2>
    <p>...</p>
    <p style="color: red">...</p>
    <p>After completing this course, students will:</p>
    <ul>...</ul>
  </div>
</body>
```

The developer tools also show a box model diagram for the selected element, illustrating the margin, border, padding, and content areas. The diagram shows a content area of 355.200x14052.300 pixels, surrounded by a padding of 50 pixels, a border, and a margin.

Anatomy of a GUI/HTML Page

GUIs are typically trees

- Nested elements, recursively
- Some fixed positions (html, body)

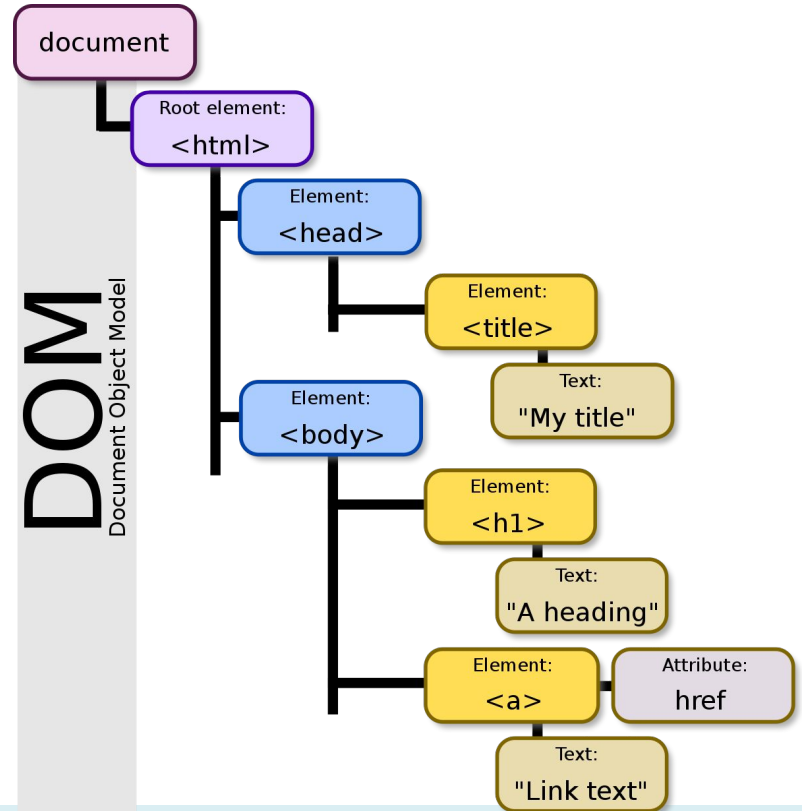
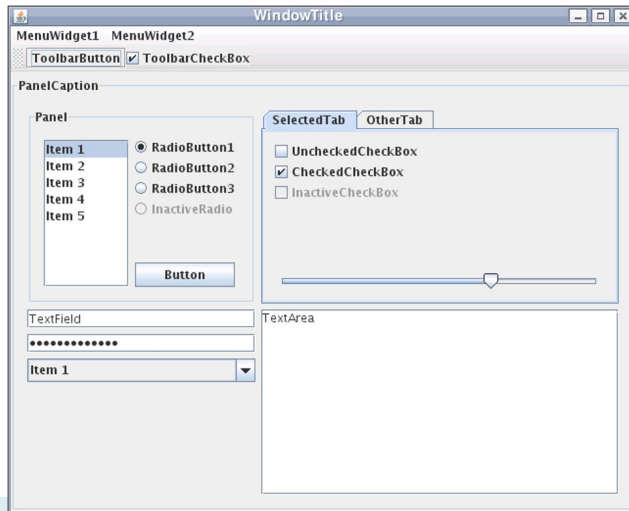
How to implement this?

JFrame

JPanel

JTextField

...



The composite pattern

- Problem: Collection of objects has behavior similar to the individual objects
- Solution: Have collection of objects and individual objects implement the same interface
- Consequences:
 - Client code can treat collection as if it were an individual object
 - Easier to add new object types
 - Design might become too general, interface insufficiently useful

Composite

- Elements can contain elements
 - With restrictions
 - Need to deal with style, interaction
- In JS: HTML`Element`
 - With child-classes e.g. `HTMLDivElement`, `HTMLBodyElement`
 - Navigation:
 - `getElement*`: locate by tag name, id, class, etc.
 - `next/prev(Element)Sibling`
 - `childNodes`, `parent`

Let's start with a very simple example.

Style

Tags come with inherent & customizable style

- Inherent:
 - `<div>` is a `block` (full-width, with margin)
 - `` is in-line
 - `<h1>` is large
- Customizable: add and override styles
 - Change font-styles, margins, widths
 - Modify groups of elements

Style: CSS

- Cascading Style Sheets
 - Reuse: styling rules for tags, classes, types
 - Reuse: not just at the leafs!

```
<span style="font-weight:bold">Hello again!</span>
```

VS.

```
<style type="text/css">
  span {
    font-family: arial
  }
</style>
```


Style: CSS

- Cascading Style Sheets
 - Reuse: styling rules for tags, classes, types
 - Reuse: not just at the leafs!
- What if there are conflicts?

```
<div style="font-weight:normal">  
  <span style="font-weight:bold">Hello again!</span>  
</div>
```

- Innermost element wins*

*Technically, there's a whole scoring system

Style: CSS

What is happening here?

The screenshot shows a browser's developer tools interface. The top-left pane displays the rendered text: "Hi there!" followed by "Hello again!" on a new line. The top-right pane shows the DOM tree with the following HTML structure:

```
<span style="font-style:bold">Hi there!</span>
<br>
<div style="font-weight:normal">
  ... <span style="font-weight:bold">Hello again!</span> == $0
</div>
</body>
</html>
```

The bottom-right pane shows the "Styles" tab with the following CSS rules:

```
div > span {
  font-family: 'Times New Roman', Times, serif;
}
span {
  font-family: arial;
}
span {
  font-family: arial;
}
```

The first rule, `div > span { font-family: 'Times New Roman', Times, serif; }`, is highlighted with a black rounded rectangle. The source for this rule is listed as `main.css:13`. The second rule, `span { font-family: arial; }`, is listed as `index.html:6`. The third rule, `span { font-family: arial; }`, is listed as `main.css:9`. The "Inherited from" section shows `div` and the "style attribute" section is partially visible.

Style: CSS

- Cascading Style Sheets
 - Reuse: styling rules for tags, classes, types
 - Reuse: not just at the leafs!
- What if there are no conflicts?

```
<div style="font-family:arial">  
  <span style="font-weight:bold">Hello again!</span>  
</div>
```

- How would you implement this?

Decorator

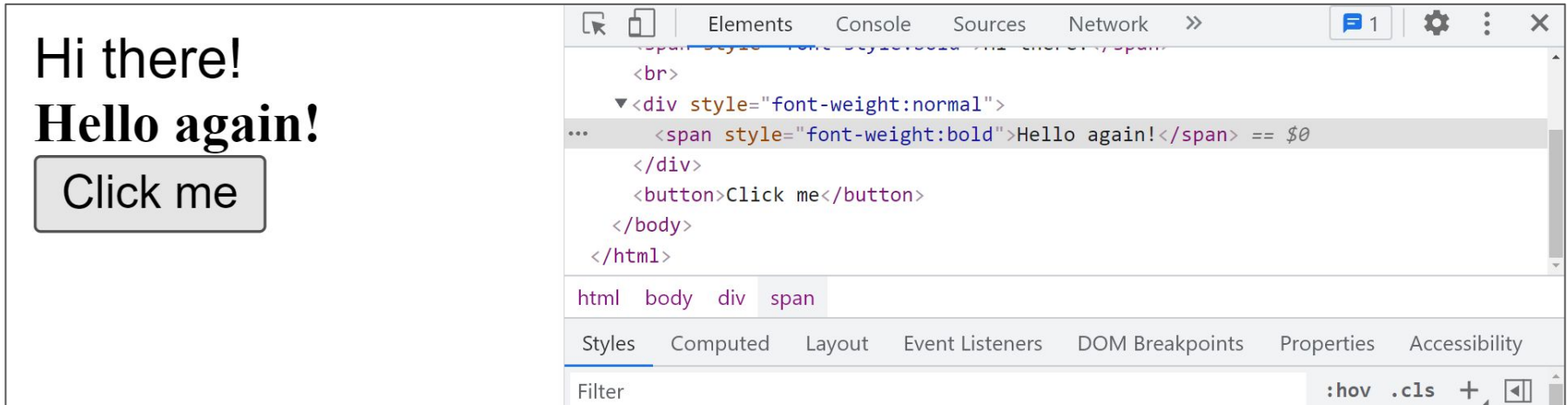
What is happening here?

- To compute the style of an element:
 - Apply its tag-default style
 - **Wrap** in added style rules (tag-specific or general)
 - Text: font-family, weight, etc.
 - Inherit parents' style
 - Conflicts lead to overrides
- Makes *themes* really powerful

Technically, HTML is streamed top-to-bottom; CSS works bottom-up

Interactivity: A GUI is more than just a document

- How do we make it “work”?
- This is a two-part answer: (1) we can attach scripts to elements, but (2) ...how? [Design question!]



The screenshot shows a web browser window with a simple GUI on the left and the Chrome DevTools 'Elements' panel on the right. The GUI contains the text 'Hi there!' and 'Hello again!' in a large font, followed by a button labeled 'Click me'. The 'Elements' panel shows the DOM tree with the following structure:

```
<html>  
  <body>  
    <div style="font-weight:normal">  
      <span style="font-weight:bold">Hello again!</span> == $0  
    </div>  
    <button>Click me</button>  
  </body>  
</html>
```

The 'span' element is selected, and the breadcrumb below it reads 'html > body > div > span'. The 'Styles' panel at the bottom shows a filter box and some default styles like ':hov' and '.cls'.

That's extremely simple, let's try something
slightly more complicated.

Consider: TicTacToe

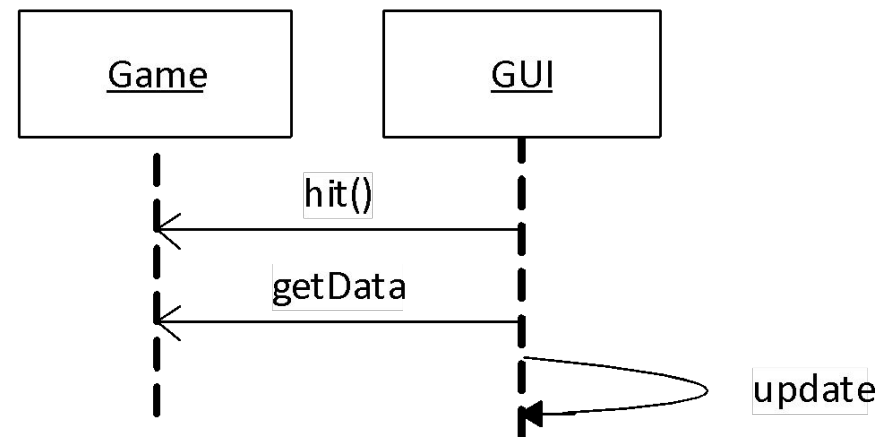
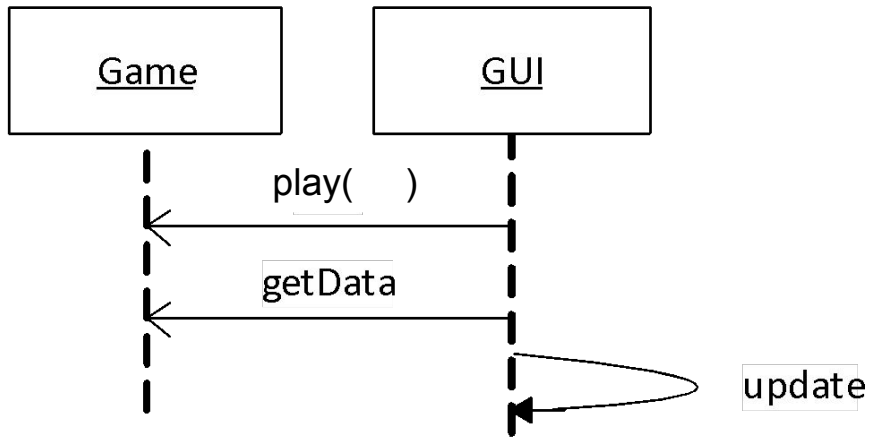
(note that this is NOT the same code you'll see in recitation next week,
but the game itself will look basically the same.)

A design challenge

DECOUPLING THE GUI

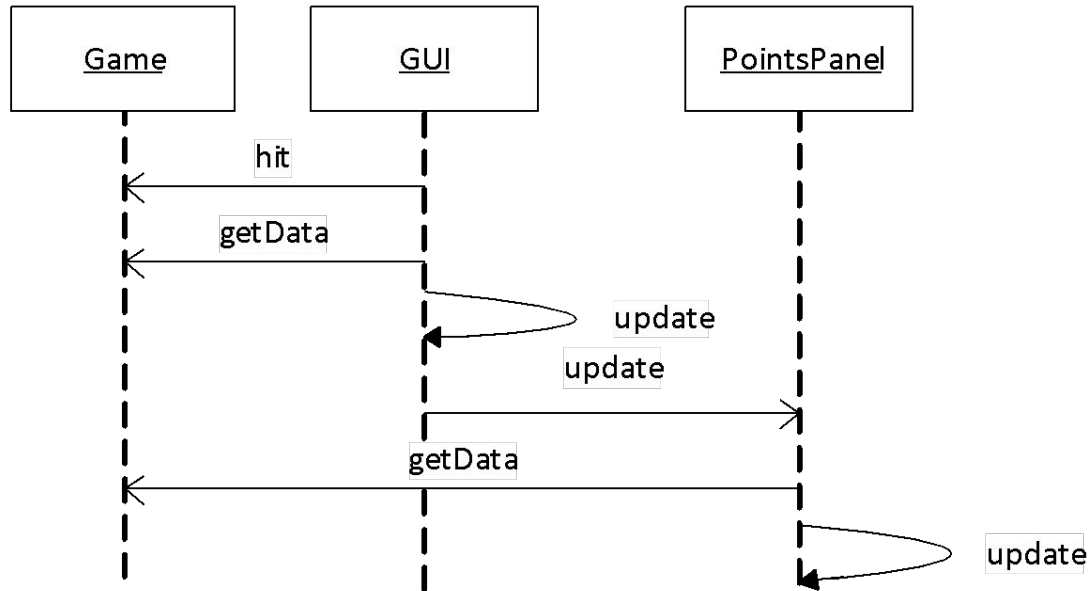
GUI design challenge

- Consider TicTacToe or Blackjack game, implemented by Game class:
 - Player clicks a space, expects it to update; clicks “hit” and expects a new card
 - When should the GUI update the screen?



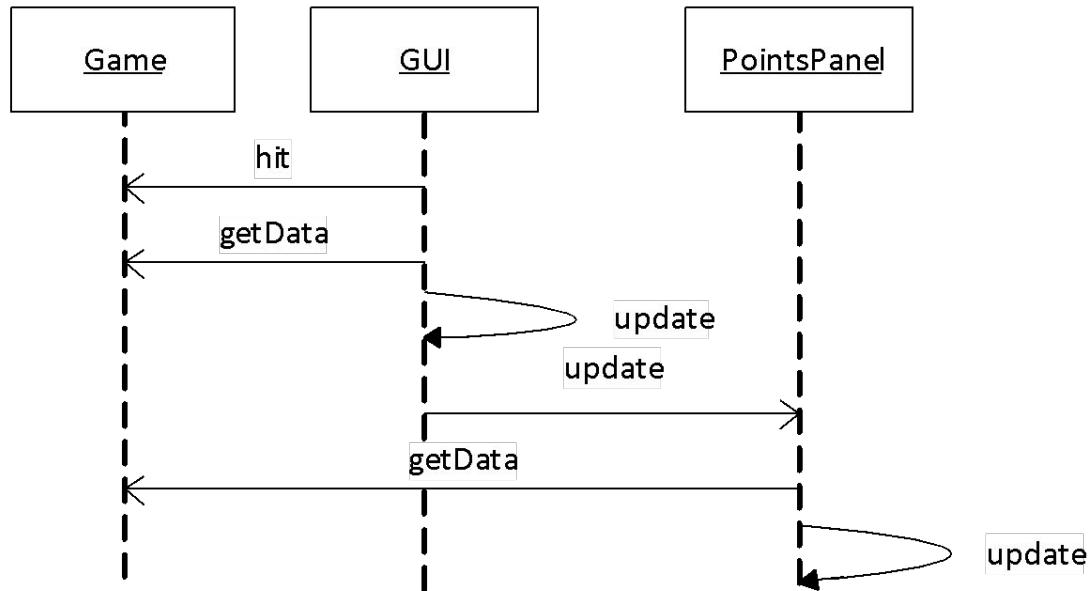
A GUI design challenge, extended

- What if we want to show the points won?



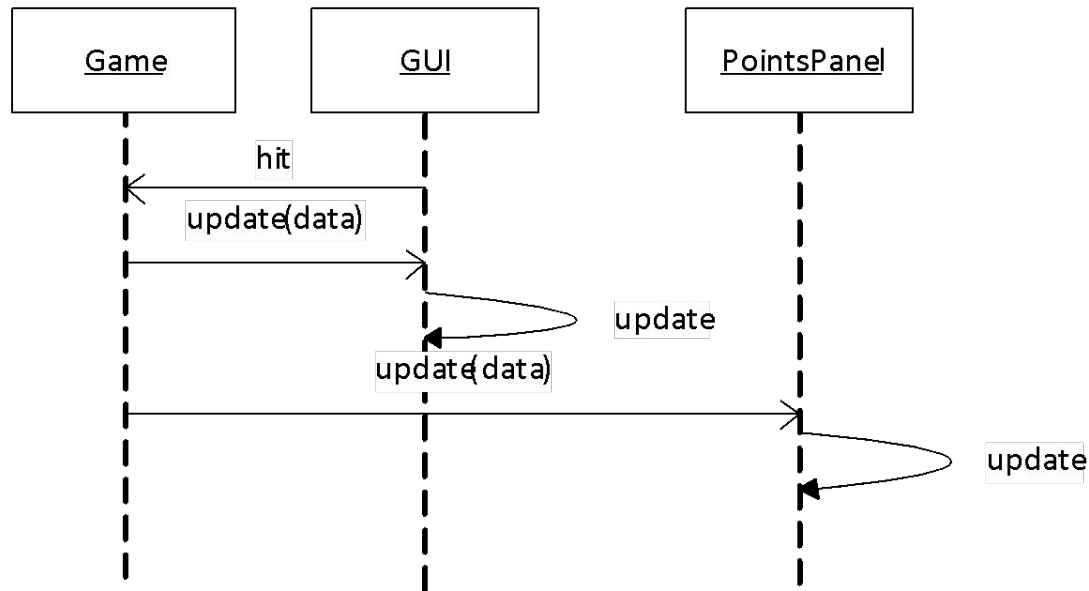
Game updates GUI?

- What if points change for reasons not started by the GUI?
(or computations take a long time and should not block)



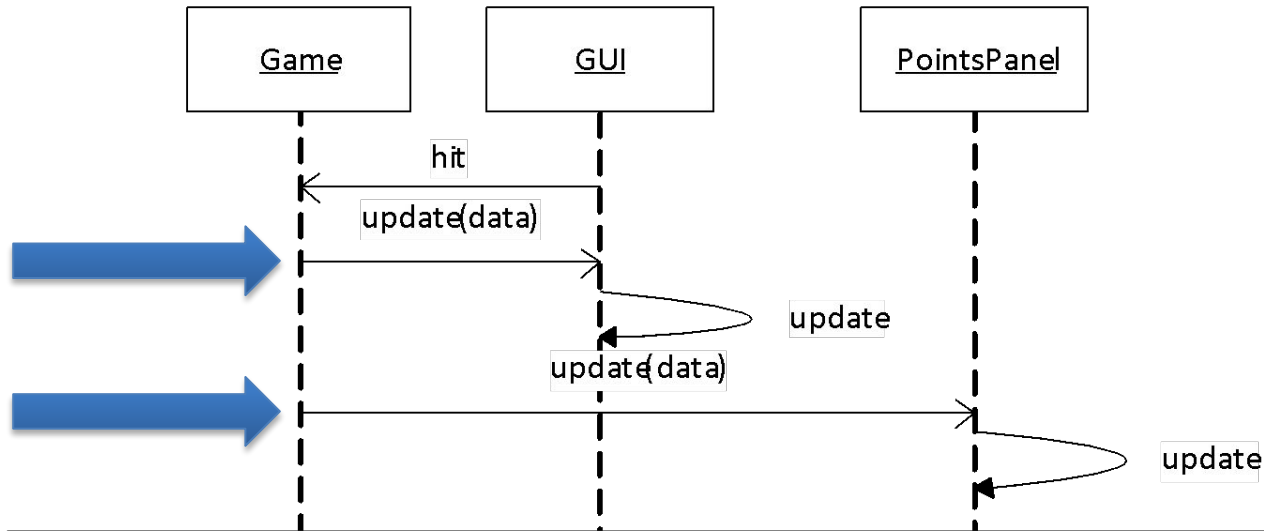
Game updates GUI?

- Let the Game tell the GUI that something happened



Game updates GUI?

- Let the Game tell the GUI that something happened



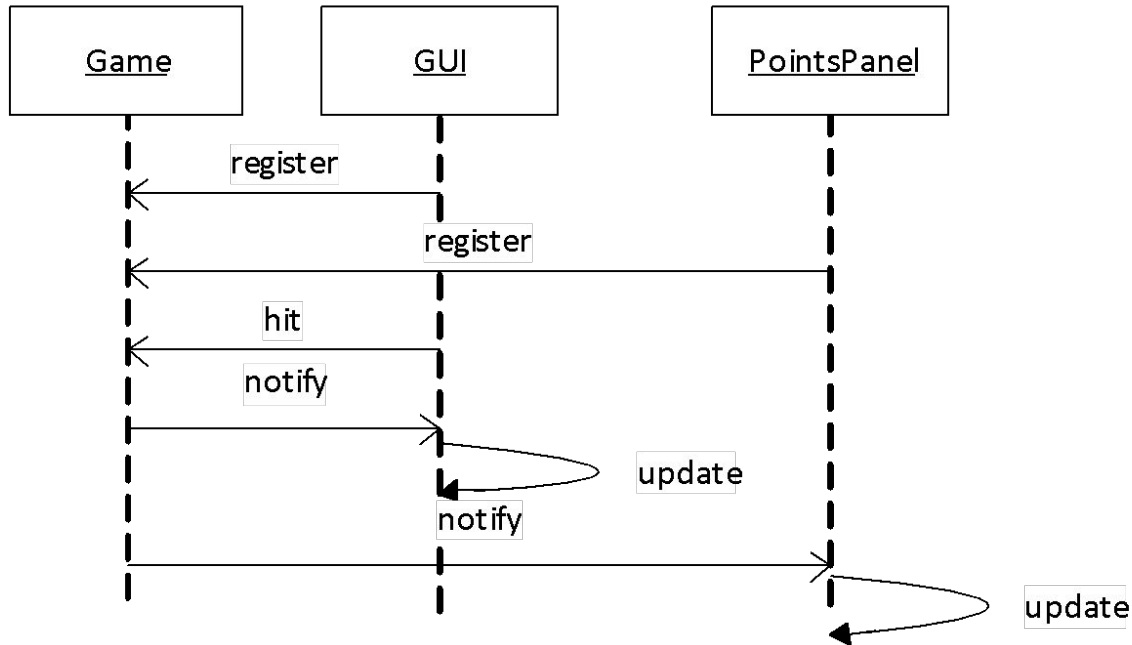
Problem: This couples the world to the GUI implementation.

Core implementation vs. GUI

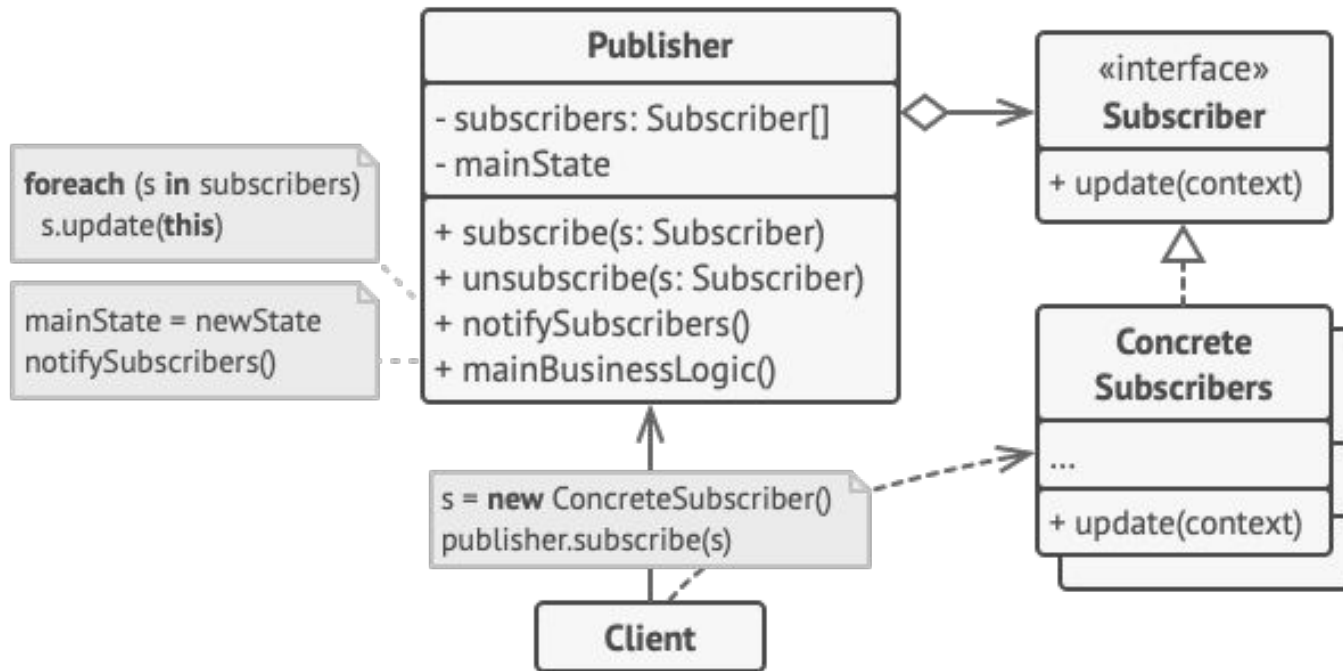
- Core implementation: Application logic
 - Computing some result, updating data
- GUI
 - Graphical representation of data
 - Source of user interactions
- Design guideline: *Avoid coupling the GUI with core application*
 - Multiple UIs with single core implementation
 - Test core without UI
 - *Design for change, design for reuse, design for division of labor; low coupling, high cohesion*

Decoupling with the Observer pattern

- Let the Game tell *all* interested components about updates



Recall the Observer



<https://refactoring.guru/design-patterns/observer>

Observer Pattern

- Manages publishers and subscribers
 - Here, button publishes its 'click' events
 - `buttonClicked` subscribes to 1+ updates
- Flexibility and Reuse
 - Multiple observers per element
 - Shared observers across elements

Actions: JavaScript

- Key: event listeners/the Observer Pattern
- (frontend) JS is highly event-driven
 - Respond to window `onLoad` event, content loads (e.g., ads)
 - Respond to clicks, moves
- This is what happened with our simple button!

The screenshot shows a web browser window with the following content:

Hi there!
Hello again!

The browser's developer tools are open, showing the Network tab. The selected resource is a button element with the following HTML structure:

```
<span style="font-style:italic">Hi there!</span>  
<br>  
<div style="font-weight:normal">  
  <span style="font-weight:bold">Hello again!</span>  
</div>  
... <button onclick="buttonClicked()">Click me</button> == $0
```

The Console tab is also open, showing the following JavaScript code in main.js:

```
1 function buttonClicked() {  
2   alert('You did it!')  
3 }
```

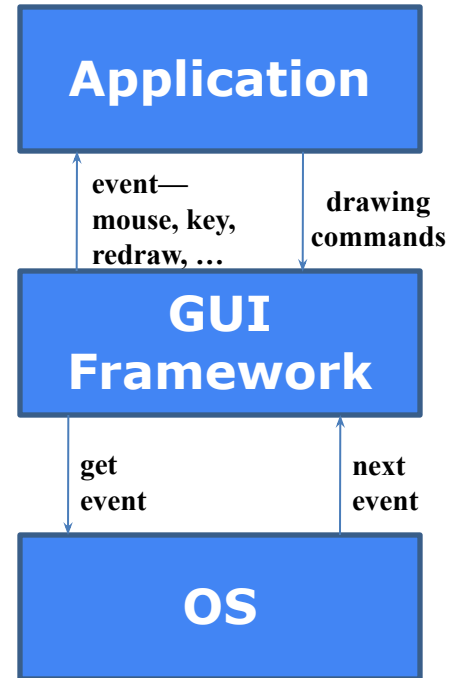
What does this look like in TicTacToe?

Let's go look!

Important note! just because TTT is implemented in a static web page all in the frontend, does *not* mean that the GUI and the Game are hopelessly entangled or that we're violating the design principle to keep them separate!

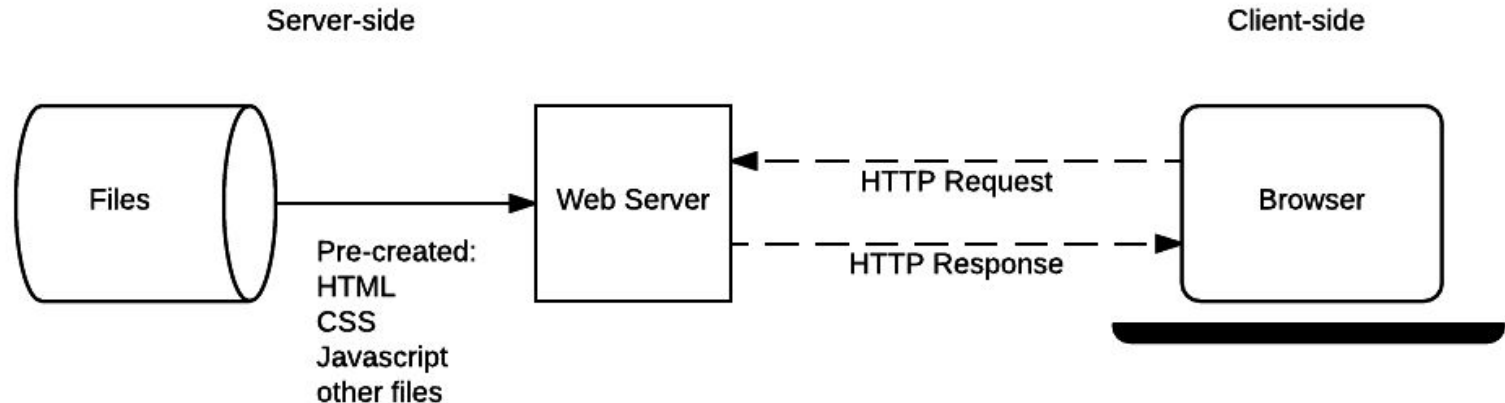
An event-based GUI with a GUI framework

- Setup phase
 - Describe how the GUI window should look
 - Register observers to handle events
- Execution
 - Framework gets events from OS, processes events
 - Your code is mostly just event handlers



Static Web Pages

- Delivered as-is, final
 - Consistent, often fast
 - Cheap, only storage needed
- “Static” a tad murky with JavaScript
 - We can still have buttons, interaction
 - But it won’t “go” anywhere -- the server is mum



Static Web Pages

- Delivered as-is, final
 - Consistent, often fast
 - Cheap, only storage needed
- Can be maintained with *static website generators*
 - Or you'll be doing a lot of copying
 - Coupled with themes => rapid development, deployment
 - Quite popular, e.g. hosting on GH Pages

Static Web Pages

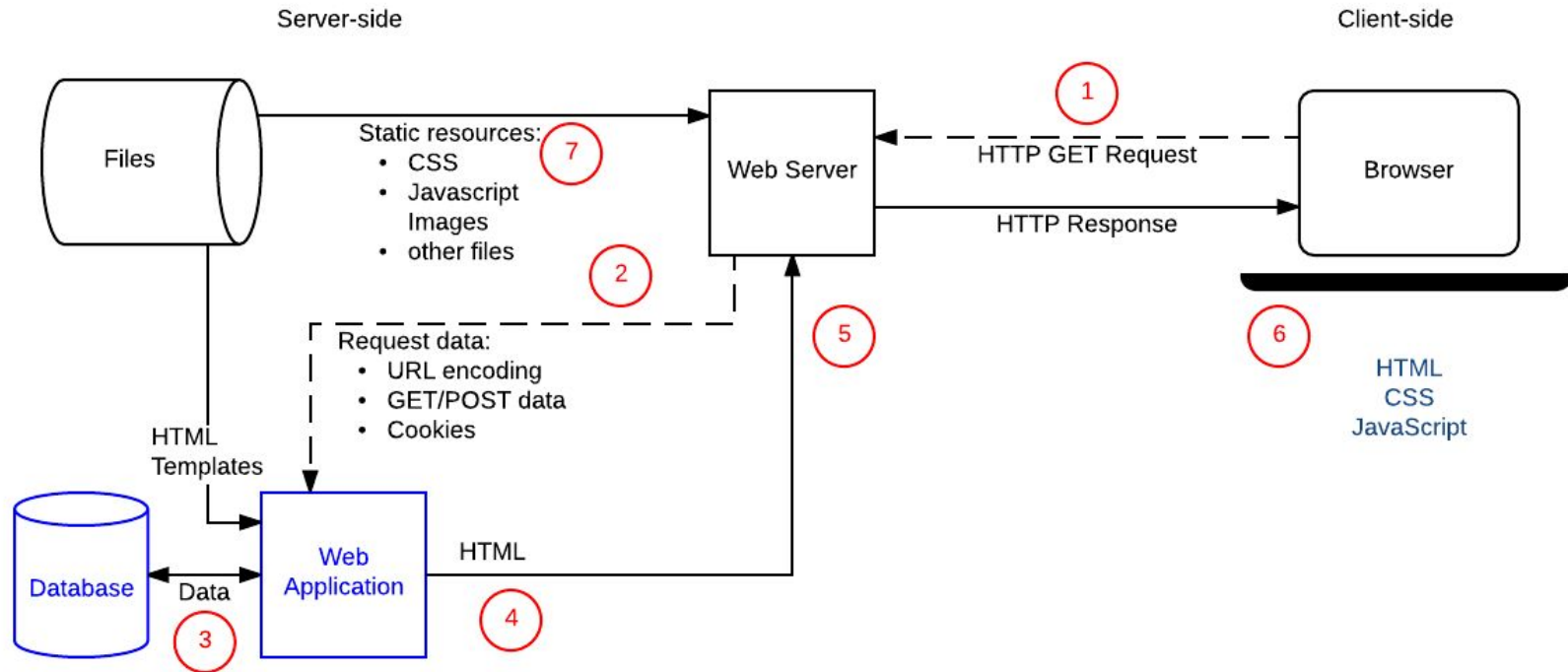
- But ...
 - No data from elsewhere (where does your email come from?)
 - No persistence (at least, not obviously)
 - No customizability (e.g., accounts)
 - No communication (payment, chat, etc)
 - Realistically, no intensive jobs

Dynamic Web Pages

- Client/Server
 - Someone needs to answer the website's calls
 - Doesn't need to be us!
 - Host a webserver
 - Serves pages, handles calls
 - For static pages too!
- We'll show you more in recitation tomorrow (Wednesday)

Web Servers

Dynamic sites can do more *work*



https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Client-Server_overview#anatomy_of_a_dynamic_request

Web Servers

- Communicate via HyperText Transfer Protocol
 - URL (the address)
 - Method:
 - GET: retrieve data. Parameters in URL ``...?key=value&key2=value2`` and message body
 - POST: store/create data. Parameters in request body
 - Several more, rarely used
 - Responses:
 - *Status Code*:
 - We probably all know 404.
 - 2XX family is OK.
 - And possible data. E.g., entire HTML page.

Web Servers

- Communicate via HyperText Transfer Protocol
 - URL (the address)
 - Method:
 - GET: retrieve data. Parameters in URL ``...?key=value&key2=value2`` and message body
 - POST: store/create data. Parameters in request body
 - Several more, rarely used
 - Responses:
 - *Status Code*. We all know 404. 2XX family is OK.
 - And possible data. E.g., entire HTML page.
 - POST makes no sense for static sites!
 - As do GETs with parameters

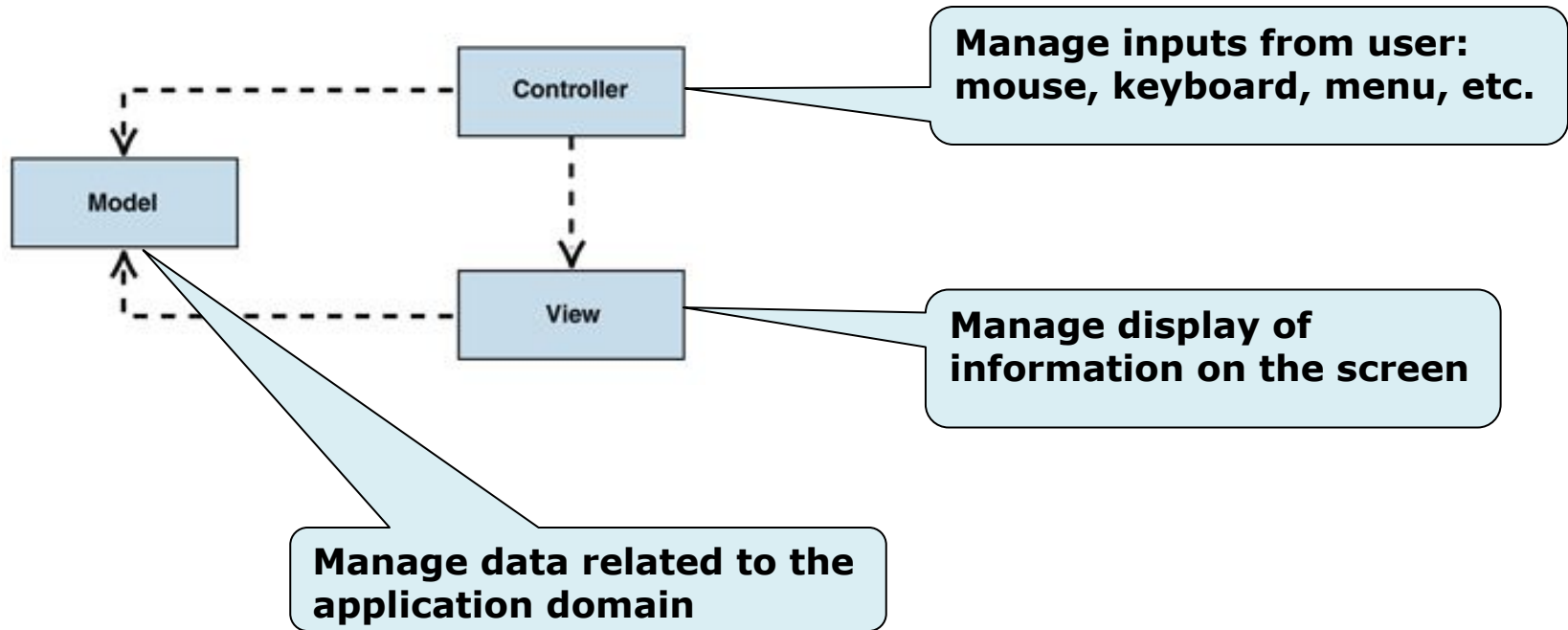
We can implement TicTacToe this way, too!

Let's go see.

(network tab of inspect will show us messages!)

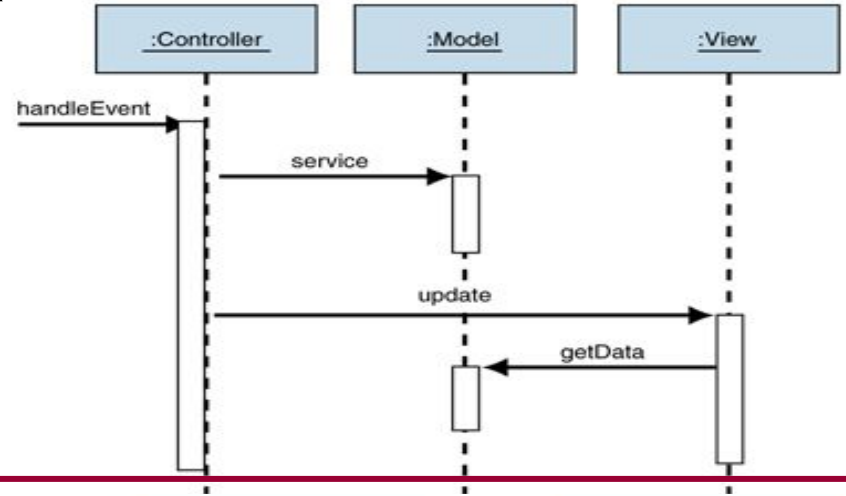
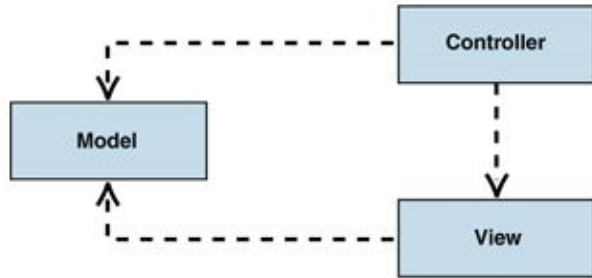
But notice we've begun to more explicitly separate out the HTML from the logic.

An architectural pattern: Model-View-Controller (MVC)

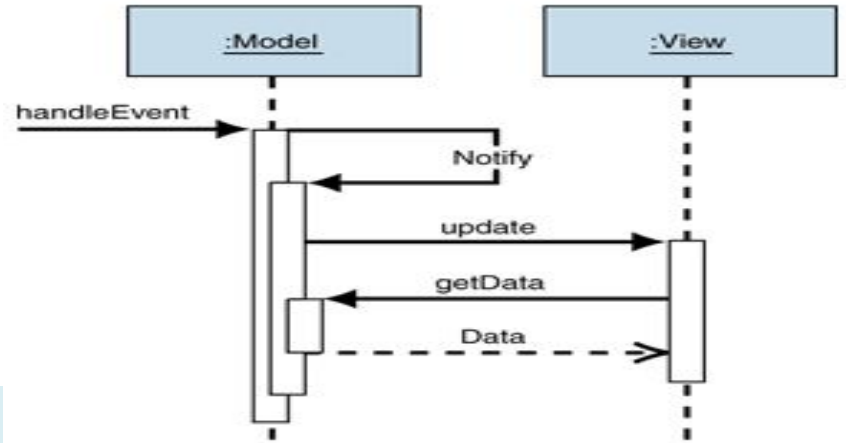
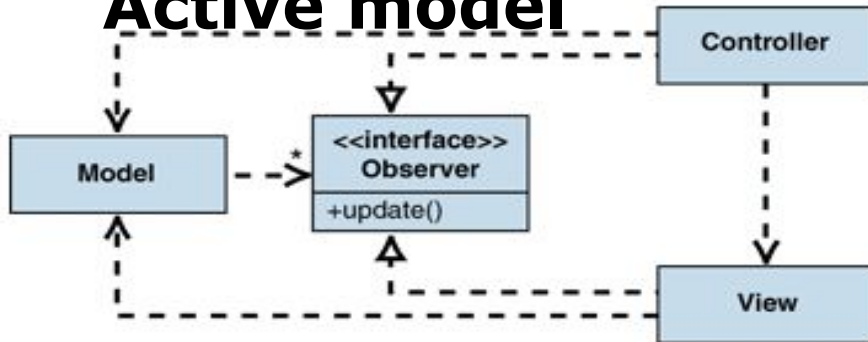


Model-View-Controller (MVC)

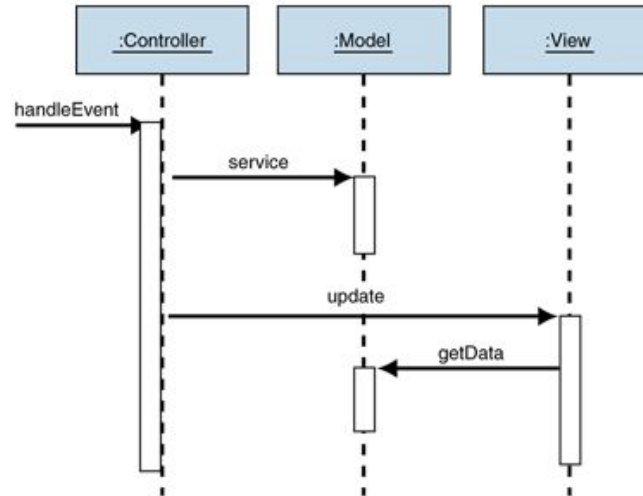
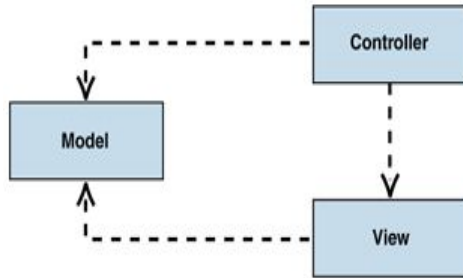
Passive model



Active model



Model View Controller Dependencies



MVC is ubiquitous

Separates:

- Model: data organization
 - Interface to the database
- View: visual representation (typically HTML)
 - Often called *templates* in web-dev; “view” is a bit overloaded
- Controller: intermediary between client and model/view
 - Typically asks model for data, view for HTML

How to Web App?

- Let's avoid generating HTML from scratch on every call
 - Map requests to handler code
 - Fetch data, process
 - Generate and return HTML
- Historically: PHP
 - Modifies HTML pages server-side on request; strong ties to SQL

```
<?php
// The global $_POST variable allows you to access the data sent with the POST method by name
// To access the data sent with the GET method, you can use $_GET
$say = htmlspecialchars($_POST['say']);
$to  = htmlspecialchars($_POST['to']);

echo $say, ' ', $to;

?>
```

Summary

- GUIs are full of design patterns
 - Helpful for reuse, delegation in complex environments
- Covered the basics of HTML, CSS, JS, servers
 - Needed for dynamic web pages
 - Decouple the GUI; architect your backend
 - A lot more to learn (security, performance, privacy), but this will do
- You will build this
 - At a small scale