Lecture 10

Graphics Part II – Animations & Shapes
Recall: ColorChanger Application

- Button changes the color of label with every click
  - EventHandlers enable app to respond to user input
  - Can use lambda expressions in place of separate classes for EventHandlers
- PaneOrganizer factors out graphical logic into separate class
- Code can be found on [GitHub](https://github.com/)

Stage
Scene
This is the grey background. All elements in the Scene Graph will show up within the Scene
Pane (e.g., VBox)
This is the structure that contains the label and the Button
Button
Label
CS15 Rocks
Random Color
Logical vs. Graphical Containment/Scene Graph

- **Graphically**, VBox is a pane contained within Scene, but **logically**, VBox is contained within PaneOrganizer.

- **Graphically**, Button and Label are contained within VBox, but **logically**, Button and Label are contained within PaneOrganizer, which has no graphical appearance.

- **Logical** containment is based on where instances are instantiated, while **graphical** containment is based on JavaFX elements being added to other JavaFX elements via `getChildren.add(…)` method, and on the resulting scene graph.
Outline

• Animation

• Layout Panes

• Java FX Shapes
Animation – Change Over Time

• Suppose we have an alien Shape we would like to animate (e.g. make it move across the screen)

• As in film and video animation, we can create apparent motion with many small changes in position (e.g., Flipbook Animation: https://www.youtube.com/watch?v=ntD2qiGx-DY)

• If we move fast enough and in small enough increments, we get smooth motion

• Same goes for size, orientation, shape change, etc…

• How to orchestrate a sequence of incremental changes?
  o coordinate with a Timeline where change happens at defined instants
Introducing **Timelines** (1/3)

- The **Timeline** sequences one or more **KeyFrames**
  - a **KeyFrame** can be thought of as a singular snapshot
  - in our simple use of JavaFX **KeyFrames**, each lasts for its entire **Duration** without making any changes
  - when the **Duration** ends, the **EventHandler** updates variables to affect the animation
Introducing **Timelines** (2/3)
Introducing **Timelines** (3/3)

We can do simple animation using a single **KeyFrame** that is repeated a fixed or indefinite number of times **EventHandler** is called. **EventHandler** makes incremental changes to time-varying variables (e.g., (x, y) position of a shape).
Using JavaFX **Timelines** (1/2)

- `javafx.animation.Timeline` is used to sequence one or more `javafx.animation.KeyFrames` or run through them cyclically
  - each `KeyFrame` lasts for its entire `Duration` until its time interval ends and `EventHandler` is called to make updates

- First we instantiate a `KeyFrame`, and pass in
  - a `Duration` (e.g. `Duration.seconds(0.3)` or `Duration.millis(300)`), which defines time that each `KeyFrame` lasts
  - an `EventHandler` of type `ActionEvent` that defines what should occur upon completion of each `KeyFrame`

- `KeyFrame` and `Timeline` work together to control the animation, but our application’s `EventHandler` is what actually causes variables to change

- As we saw from last lecture, we can use lambda expressions to represent the `EventHandler`s instead of creating a separate class
Using JavaFX Timelines (2/2)

• Next we instantiate our Timeline, setting its CycleCount property
  o defines number of cycles in Animation
  o setting CycleCount to Animation.INDEFINITE will let Timeline run forever or until we explicitly stop it
• We pass our new KeyFrame into Timeline
• In order for Timeline to work, we must then call Timeline.play();
Our First JavaFX animation: Clock

• Simple example of discrete (non-smooth) animation

• Specifications: App should display current date and time, updating every second

• Useful classes:
  o `java.util.Date`
  o `javafx.util.Duration`
  o `javafx.animation.KeyFrame`
  o `javafx.animation.Timeline`
Process: Clock

1. Write an App class that extends javafx.application.Application and implements start (Stage)

2. Write a PaneOrganizer class that instantiates root node and returns it in a public getRoot() method. Instantiate a Label and add it as root node’s child. Factor out code for Timeline into its own method.

3. In our own setupTimeline(), instantiate a KeyFrame passing in Duration and a lambda expression (defined later) as our EventHandler. Then instantiate Timeline, passing in our KeyFrame, and play Timeline

4. Define lambda expression to represent our EventHandler – for every ActionEvent, update the text on the Label
Clock: App class (1/3)

Note: Exactly the same process as in ColorChanger’s App [Lecture 8]

1a. Instantiate a PaneOrganizer and store it in the local variable organizer

```java
public class App extends Application {
    @Override
    public void start(Stage stage) {
        PaneOrganizer organizer = new PaneOrganizer();
    }
}
```
Clock: App class (2/3)

Note: Exactly the same process as in ColorChanger’s App [Lecture 8]

1a. Instantiate a PaneOrganizer and store it in the local variable organizer

1b. Instantiate a Scene, passing in organizer.getRoot(), and desired width and height of Scene
Clock: App class (3/3)

Note: Exactly the same process as in ColorChanger’s App [Lecture 9]

1a. Instantiate a PaneOrganizer and store it in the local variable organizer

1b. Instantiate a Scene, passing in organizer.getRoot(), desired width and height of the Scene

1c. Set the Scene, set the Stage’s title, and show the Stage!

```java
public class App extends Application {
    @Override
    public void start(Stage stage) {
        PaneOrganizer organizer = new PaneOrganizer();
        Scene scene = new Scene(organizer.getRoot(), 200, 200);
        stage.setScene(scene);
        stage.setTitle("Clock!");
        stage.show();
    }
}
```
Process: **Clock**

1. Write App class that extends `javafx.application.Application` and implements `start(Stage)`

2. **Write a PaneOrganizer class** that instantiates root node and returns it in a public `getRoot()` method. Instantiate a `Label` and add it as root node’s child. Factor out code for `Timeline` into its own method, which we’ll call `setupTimeline()`

3. In our own `setupTimeline()`, instantiate a `KeyFrame` passing in `Duration` and a lambda expression (defined later) as our `EventHandler`. Then instantiate a `Timeline`, passing in our `KeyFrame`, and play the `Timeline`

4. Define lambda expression to represent our `EventHandler` – for every `ActionEvent`, update the text on the `Label`
Clock: PaneOrganizer Class (1/3)

2a. In the PaneOrganizer class’ constructor, instantiate a root VBox and set it as the return value of a public getRoot() method

```java
public class PaneOrganizer {
    private VBox root;

    public PaneOrganizer() {
        this.root = new VBox();
    }

    public VBox getRoot() {
        return this.root;
    }
}
```
Clock: PaneOrganizer Class (2/3)

2a. In the PaneOrganizer class’ constructor, instantiate a root VBox and set it as the return value of a public getRoot() method

```java
public class PaneOrganizer {
    private VBox root;
    private Label label;

    public PaneOrganizer() {
        this.root = new VBox();
        this.label = new Label();
        this.root.getChildren().add(this.label);
    }

    public VBox getRoot() {
        return this.root;
    }
}
```

2b. Instantiate a Label and add it to the list of the root node’s children
Clock: PaneOrganizer Class (3/3)

2a. In the PaneOrganizer class’ constructor, instantiate a root VBox and set it as the return value of a public getRoot() method.

```java
public class PaneOrganizer {
    private VBox root;
    private Label label;
    public PaneOrganizer() {
        this.root = new VBox();
        this.label = new Label();
        this.root.getChildren().add(this.label);
        this.setupTimeline();
    }

    public VBox getRoot() {
        return this.root;
    }
}
```

2b. Instantiate a Label and add it to the list of the root node’s children.

```java
public VBox getRoot() {
    return this.root;
}
```

2c. Call setupTimeline(); this is another example of delegation to a specialized “helper method” which we’ll define next!
Process: **Clock**

1. Write an `App` class that extends `javafx.application.Application` and implements `start(Stage)`

2. Write a `PaneOrganizer` class that instantiates the root node and returns it in a public `getRoot()` method. Instantiate a `Label` and add it as the root node's child. Factor out code for `Timeline` into its own method.

3. In `setupTimeline()`, instantiate a `KeyFrame`, passing in `Duration` and a lambda expression (defined later) as our `EventHandler`. Then instantiate a `Timeline`, passing in our `KeyFrame`, and play the `Timeline`.

4. Define lambda expression to represent our `EventHandler` – for every `ActionEvent`, update the text on the `Label`. 

---

Andries van Dam © 2021 10/12/21
Within `setupTimeline()`:

3a. Instantiate a `KeyFrame`, which takes two parameters: Duration and `EventHandler`

```java
public class PaneOrganizer {
    //other code elided

    public void setupTimeline() {
        KeyFrame kf = new KeyFrame( , );
    }
}
```
Clock: PaneOrganizer class - setupTimeline() (1/4)

Within setupTimeline():

3a. Instantiate a KeyFrame, which takes two parameters: Duration and EventHandler

  ○ want to update text of label each second — therefore make Duration of the KeyFrame 1 second

```java
public class PaneOrganizer {
    // other code elided
    private void setupTimeline() {
        KeyFrame kf = new KeyFrame(Duration.seconds(1), // how long
            ...)
    }
}
```
Clock: PaneOrganizer class - setupTimeline() (1/4)

Within setupTimeline():

3a. Instantiate a KeyFrame, which takes two parameters: Duration and EventHandler

  ○ want to update text of label each second – therefore make Duration of the KeyFrame 1 second

  ○ for the EventHandler parameter, pass a lambda expression (to be defined later)

Note: JavaFX automatically calls this.updateLabel at end of each KeyFrame, which in this case changes the label text, and then lets the next 1 second cycle of KeyFrame start
public class PaneOrganizer {
    //other code elided

    private void setupTimeline() {
        KeyFrame kf = new KeyFrame(
            Duration.seconds(1),
            (ActionEvent e) ->
            this.updateLabel() //event handler
        );

        Timeline timeline = new Timeline(kf);
    }
}
**Clock: PaneOrganizer class- setupTimeline() (3/4)**

**Within setupTimeline():**

3a. Instantiate a **KeyFrame**

3b. Instantiate a **Timeline**, passing in our new **KeyFrame**

3c. **Set CycleCount to INDEFINITE**

```java
public class PaneOrganizer {
    //other code elided

    private void setupTimeline() {
        KeyFrame kf = new KeyFrame(
            Duration.seconds(1),
            (ActionEvent e) ->
                this.updateLabel()); //event handler

        Timeline timeline = new Timeline(kf);

        timeline.setCycleCount(Animation.INDEFINITE);
    }
}
```
Within `setupTimeline()`:

3a. Instantiate a `KeyFrame`

3b. Instantiate a `Timeline`, passing in our new `KeyFrame`

3c. Set `CycleCount` to `INDEFINITE`

3d. Play, i.e. start `Timeline`
Process: **Clock**

1. **Write an App class that extends** `javafx.application.Application` **and implements** `start(Stage)`
2. **Write a PaneOrganizer class that instantiates the root Node** and returns it in public `getRoot()` method. Instantiate a `Label` and add it as root node’s child. Factor out code for `Timeline` into its own method.
3. **In setupTimeline()**, instantiate a `KeyFrame` passing in a `Duration` and a lambda expression (defined later) as our `EventHandler`. Then instantiate a `Timeline`, passing in our `KeyFrame`, and play the `Timeline`.
4. **Define a lambda expression to represent our EventHandler** – for every `ActionEvent`, update the text on the Label.
Clock: EventHandler: lambda expression (1/3)

4a. The last step is to define a lambda expression to represent our EventHandler, specifying what should occur at the end of each KeyFrame — called automatically by JavaFX.

```java
public class PaneOrganizer {
    //other code elided

    private void setUpTimeline () {
        KeyFrame kf = new KeyFrame(
            Duration.seconds(1),
            (ActionEvent e) ->
                this.updateLabel()); //event handler
        //other code elided
    }

    private void updateLabel() {

    }
}
```
4a. The last step is to create our `TimeHandler` and implement `handle()`, specifying what should occur at the end of each `KeyFrame` — called automatically by JavaFX.

4b. `java.util.Date` represents a specific instant in time. `Date` is a representation of the time, to the nearest millisecond, at the moment the `Date` is instantiated.

```java
public class PaneOrganizer {
    // other code elided

    private void setUpTimeline() {
        KeyFrame kf = new KeyFrame(
            Duration.seconds(1),
            (ActionEvent e) ->
                this.updateLabel()); // event handler

        // other code elided
    }

    private void updateLabel() {
        Date now = new Date();
    }
}
```
4a. The last step is to create our `TimeHandler` and implement `handle()`, specifying what to occur at the end of each `KeyFrame` – called automatically by JavaFX.

4b. `java.util.Date` represents a specific instant in time. `Date` is a representation of the time, to the nearest millisecond, at the moment the `Date` is instantiated.

4c. Because our `Timeline` has a `Duration` of 1 second, each second a new `Date` will be generated, converted to a `String`, and set as the label's text. This will appropriately update `label` with correct time every second!

```java
public class PaneOrganizer {
    //other code elided

    private void updateLabel() {
        Date now = new Date();
        //toString converts the Date into a String with year, day, time etc.

        //this.label instantiated in //constructor of PO
        this.label.setText(now.toString());
    }
}
```
public class App extends Application {

    @Override
    public void start(Stage stage) {
        PaneOrganizer organizer = new PaneOrganizer();
        Scene scene = new Scene(organizer.getRoot(), 200, 200);
        stage.setScene(scene);
        stage.setTitle("Clock");
        stage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}

public class PaneOrganizer {
    private VBox root;
    private Label label;

    public PaneOrganizer() {
        this.root = new VBox();
        this.label = new Label();
        this.root.getChildren().add(this.label);
        this.setupTimeline();
    }

    public VBox getRoot() {
        return this.root;
    }

    private void setupTimeline() {
        KeyFrame kf = new KeyFrame(Duration.seconds(1),
            (ActionEvent e) -> this.updateLabel());
        Timeline timeline = new Timeline(kf);
        timeline.setCycleCount(Animation.INDEFINITE);
        timeline.play();
    }

    private void updateLabel() {
        Date now = new Date();
        this.label.setText(now.toString());
    }
}
Outline

• Animation

• Layout Panes

• Java FX Shapes
Layout Panes

• Until now, we have been adding all our GUI components to a VBox
  o VBoxes lay everything out in one vertical column
• What if we want to make some more interesting GUIs?
• Use different types of layout panes!
  o VBox is just one of many JavaFX panes – there are many more options
  o we will introduce a few, but check out our documentation or Javadocs for a complete list
HBox

- Similar to VBox, but lays everything out in a horizontal row (hence the name)

- Example:

```
// code for setting the scene elided
HBox buttonBox = new HBox();
Button b1 = new Button("Button One");
Button b2 = new Button("Button Two");
Button b3 = new Button("Button Three");
buttonBox.getChildren().addAll(b1, b2, b3);
```

- Like VBox, we can set the amount of horizontal spacing between each child in the HBox using the setSpacing(double) method

```
buttonBox.setSpacing(20);
```
**BorderPane (1/2)**

- **BorderPane** lays out children in top, left, bottom, right and center positions
- To add things visually, use `setLeft(Node), setCenter(Node),` etc.
  - this includes an implicit call to `getChildren().add(…)`
- Use any type of **Node** – **Panes** (with their own children), **Buttons**, **Labels**, etc.!
BorderPane (2/2)

• Remember our **VBox** example from earlier?

```java
VBox buttonBox = new VBox();
Button b1 = new Button("Top");
Button b2 = new Button("Middle");
Button b3 = new Button("Bottom");
buttonBox.getChildren.addAll(b1,b2,b3);
buttonBox.setSpacing(8);
buttonBox.setAlignment(Pos.TOP_CENTER);
```

• We can make our **VBox** the center of this **BorderPane**

```java
BorderPane container = new BorderPane();
container.setCenter(buttonBox);
```

• No need to use all regions – could just use a few of them

• Unused regions are “compressed”, e.g. could have a two-region (left/right) layout without a center

Note: we didn’t have to call `container.getChildren.add(buttonBox)`, as this call is done implicitly in the `setCenter()` method!
Absolute Positioning

• Until now, all layout panes we have seen have performed layout management for us
  o what if we want to position our GUI components freely ourselves?
• Need to set component’s location to exact *pixel location* on screen
  o called *absolute positioning*
• When would you use this?
  o to position shapes – stay tuned!
**Pane**

- **Pane** allows you to lay things out completely freely, like on an art canvas – DIY graphics! More control, more work 😊
- It is a **concrete** superclass to all more specialized layout panes seen earlier that do automatic positioning
  - we can call methods on its graphically contained children (panes, buttons, shapes, etc.) to set location within pane
    - for example: use `setX(double)` and `setY(double)` to position a **Rectangle**, one of the primitive shapes
  - **Pane** performs no layout management, so coordinates you set determine where things appear on the screen
Creating Custom Graphics

• We’ve now introduced you to using JavaFX’s native UI elements
  o ex: Label and Button
• Lots of handy widgets for making your own graphical applications!
• What if you want to create your own custom graphics?
• This lecture: build your own graphics using the `javafx.scene.shape` package!
Outline

• Animation

• Layout Panes

• Java FX Shapes
javafx.scene.shape Package

• JavaFX provides built-in classes to represent 2D shapes, such as rectangles, ellipses, polygons, etc.

• All these classes inherit from abstract class Shape, which inherits from Node
  o methods relating to rotation and visibility are defined in Node
  o methods relating to color and border are defined in Shape
  o other methods are implemented in the individual classes of Ellipse, Rectangle, etc.
Shape Constructors

- Rectangle(double width, double height)
- Ellipse(double radiusX, double radiusY)
- Polygon(double … points)
  - the “…” in the signature means that you can pass in as many points as you would like to the constructor
  - pass in Points (even number of x and y coordinates) and Polygon will connect them for you
  - passing points will define and position the shape of Polygon - this is not always the case with other Shapes (like Rectangle or Ellipse)
  - Example: new Polygon(0,10,10,10,5,0)

- Each of these Shape subclasses have multiple constructors (same name, different parameter lists) This is called method overloading – will come back to it during Design Patterns. Check out the JavaFX documentation for more options!
  - for example, if you wanted to instantiate a Rectangle with a given position and size: Rectangle(double x, double y, double width, double height)
  - you could also instantiate a Rectangle with a given width, height, and color: Rectangle(double width, double height, Paint fill)

Default position for Shape with this constructor would be (0,0)
Shapes: Setting Location

- JavaFX Shapes have different behaviors (methods) for setting their location within their parent’s coordinate system
  - Rectangle: use `setX(double)` and `setY(double)`
  - Ellipse: use `setCenterX(double)` and `setCenterY(double)`
  - Polygon: use `setLayoutX(double)` and `setLayoutY(double)`

- JavaFX has *many* different ways to set location
  - from our experience, these are the most straightforward ways
  - if you choose to use other methods, be sure you fully understand them or you may get strange bugs!
  - check out our [JavaFX documentation](https://openjfx.io/javadoc/17/) and the [Javadocs](https://openjfx.io/javadoc/17/) for more detailed explanations!
Shapes: Setting Size

- JavaFX Shapes also have different behaviors (methods) for altering their size
  - **Rectangle**: use `setWidth(double)` and `setHeight(double)`
  - **Ellipse**: use `setRadiusX(double)` and `setRadiusY(double)`
  - **Polygon**: use `setScaleX(double)` and `setScaleY(double)`
    - multiplies the original size in the X or Y dimension by the **scale factor**

- Again, this is not the only way to set size for Shapes but it is relatively painless
  - reminder: [JavaFX documentation] and [Javadoc]!
Accessors and Mutators of all Shapes

- **Rotation:**
  - `public final void setRotate(double rotateAngle);`
  - `public final double getRotate();`

- **Visibility:**
  - `public final void setVisible(boolean visible);`
  - `public final boolean getVisible();`

- **Color:**
  - `public final void setStroke(Paint value);`
  - `public final Paint getStroke();`
  - `public final void setFill(Paint value);`
  - `public final Paint getFill();`

- **Border:**
  - `public final void setStrokeWidth(double val);`
  - `public final double getStrokeWidth();`

Rotation is about the center of the Shape’s “bounding box”; i.e., the smallest rectangle that contains the entire shape. To have a Shape rotate about an arbitrary center of rotation, add a Rotate instance with a new center of rotation to the Shape’s transform list (see Javadocs).

Generally, use a Color, which inherits from Paint. Use predefined color constants `Color.WHITE`, `Color.BLUE`, `Color.AQUA`, etc., or define your own new color by using the following syntax:

```
Paint color = Color.color(0.5, 0.5, 0.5);
OR:
Paint color = Color.rgb(100, 150, 200);
```

The stroke is the border that outlines the Shape, while the fill is the color of the interior of the Shape.
Announcements (1/2)

- **Code from today’s lecture** is available on GitHub – mess around for practice!
- Fruit Ninja deadlines (all due 11:59 PM ET):
  - On-time handin: *today 10/12*
  - Late handin: *Thursday 10/14*
- Java FX Lab
  - Pre-lab [video](#) and pre-lab [quiz](#)
Announcements (2/2)

- **Collaboration Policy Phase 2** starting at Cartoon
  - can debug each other’s terminal-produced errors
  - fill out mandatory [collaboration phase 2 quiz](#)

<table>
<thead>
<tr>
<th></th>
<th>Phase 1 Debugging Policy</th>
<th>Phase 2 Debugging Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discuss lecture material and general concepts</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Collaborate on mini-assignments and labs</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Get help from TAs at TA Hours and on Ed</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Discuss high-level project-specific concepts and all material provided in handouts and section</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Help another student debug a terminal-produced error message, as long as your own computer is closed</td>
<td>✗</td>
<td>✔️</td>
</tr>
<tr>
<td>Help another student debug a logical code error</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Share or compare code with another student</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Discuss project-specific implementation details</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
Topics in Socially Responsible Computing

Privacy & Surveillance I
Early American surveillance

• J. Edgar Hoover, first Director of the FBI, set groundwork for surveillance state
• Served under eight (!!) Presidents (Coolidge, Hoover, FDR, Truman, Eisenhower, Kennedy, LBJ, Nixon)
• FBI mission: “Protect the American people and uphold the Constitution of the United States”
• Investigate criminal enterprise but also domestic terrorism, subversion of the state, etc
A brief primer on American surveillance

- FBI (founded 1908): focus on domestic cases, investigate crimes, works with Department of Justice

- CIA (founded 1947): foreign intelligence to policymakers for national security decisions, prohibited from collecting info on “US Persons”

- NSA (founded 1952): counterintelligence against adversaries and foreign intelligence

A history of recent American surveillance

• Increase in US Govt’s surveillance apparatus after 9/11
  • broad rollback of requirements of a court order to get certain info
  • Patriot Act (2001)
    • allows wiretapping domestic/international phones, facilitates more sharing of intelligence across agencies
    • as national surveillance has increased, blurring of responsibilities between CIA and NSA, both of which grew 50+% between 2004 and 2013
  • Protect America Act (2007)
    • National Security Agency can monitor electronic communications between people both inside/outside of the US without court order or oversight
    • previously NSA restricted to non-Americans!

• Began under George W. Bush, largely unchanged in successor administrations (Obama, Trump, Biden)
A history of recent American whistleblowers

- **Daniel Ellsberg: The Pentagon Papers (1973)**
  - showed that multiple administrations had lied to the American public and to Congress about involvement in Vietnam (NYT)
  - no formal declaration of war, govt overstepped!
  - failed attempt by the government to stop the leak

- **Watergate (1973)**
  - Nixon administration had wiretapped Democratic Party offices
  - leaker “Deep Throat” revealed to journalists
  - captured national attention for awhile + caused Nixon to resign

An NYT cover when chargers for leaking the Pentagon Papers were dropped (1973). Credit: Zinn Education Project

Snowden / the NSA & CIA

- Edward Snowden: employee of CIA & later NSA subcontractor, increasingly disillusioned as agency ignored his raising flags about ethics
- Snowden leaked a variety of documents to journalists Laura Poitras and Glenn Greenwald (Pulitzer for their work!)
- PRISM: top-secret NSA program leaked by Snowden in 2013
  - NSA can “reach directly into servers” from, Facebook, Apple, Microsoft Google/YouTube, among others
  - NSA able to “obtain both stored communications as well as perform real time collection on targeted users”
- Siloed view of systems
  - many parts of PRISM were top secret / hidden even from people working on it
  - Snowden pursued them — was one of ~1000 NSA system admins who had sufficient access to the system (The Guardian)
PRISM / Snowden

- Govt had access to
  - Chat (video and voice), voice calling, video conferencing
  - Videos, photos, file transfers

- Critique of Snowden
  - did ‘grave damage’ to American intelligence capacities
  - put American personnel overseas in danger, exposed intelligence operations + partnerships (Associated Press)

- Snowden cannot return to the United States

- United States vs. Moalin (Sep 2020) — data collection leaked by Snowden ruled unconstitutional

- See also: Chelsea Manning (pardoned by Obama), Julian Assange (more controversial…), Frances Haugen (ex-FB, Anabelle’s presentation last Thursday)
In summary

• Surveillance has existed for a long time/is not new!
• Still physical surveillance everywhere
  • “Brown Has One Surveillance Camera For Every 9 Undergrads” – GoLocalProv
  • increasing digitally, both by govts and companies
• Technology enables surveillance at scale!
More reading that may be of interest!

- Glenn Greenwald: Why privacy matters (TEDGlobal 2014)
- Permanent Record (2019), Snowden’s phenomenal autobiography
- “Taking Back Our Privacy” (2020), article about Signal & building surveillance-resistant technologies (we’ll talk about Signal later in the course!)
- CITIZENFOUR, documentary (dir. Laura Poitras)
- “The Black Budget: Covert action. Surveillance. Counterintelligence. The U.S. ‘black budget’ spans over a dozen agencies that make up the National Intelligence Program.” — The Washington Post
- J. Edgar Hoover and the Great American Inquisitions (2006), documentary, Dir. Denis Mueller
- “How Edward Snowden went from loyal NSA contractor to whistleblower” — The Guardian
- Chelsea Manning — Wikipedia
- “Wrenn GS: The surveillance school” – Brown Daily Herald
- ”Brown has one surveillance camera for every 9 undergrads, student raises concerns” – GoLocalProv
- WikiLeaks — Wikipedia
- “Costs of Snowden leak still mounting 5 years later” (2018), Deb Riechmann, Associated Press