* Lightning component library
  + Library of pre-built components that we can use in the LCF
  + <lightning:card>
    - Provides opaque container for the nested elements
  + <aura:if>
    - Allows for flow control within our component to conditionally render markup
  + <lightning:require>
    - Allows you to load external css and js files that you’ve uploaded as static resources
  + <lightning:icon>
    - Displays an icon available from SLDS
* Lightning data service
  + Same as standard controllers in visualforce
  + Lets our aura components access the database In a really fast efficient way
  + It can all be done with markup
  + It us really efficient because each record is instantiated for our app
  + Parameters = attributes
  + Parameters
    - Field
      * Comma separated list of the field API names we want to work with
    - Layout type
      * Full or compact
    - Columns
      * Determines the number of columns on our form
    - Mode
      * Determine which operation we are doing
      * View allows edit and read
      * Readonly only allows read
      * Edit allows edit and create
    - Object API name
      * Only required attribute and it simply specifies the object API name that we want to work with
    - Record ID
      * Holds the record we are retrieving or updating
    - Record type ID
      * If we want to specify a certain record type to work with, we can pass the ID of that here or it will be passed in from the service
  + <lightning:recordForm>
  + <lightning:recordViewForm>
  + <lightning:recordEditForm>
  + <force:recordData>
* Javascript controller and helper
  + Helper is only instantiated once for the component bundle vs the controller which is instantiated many times
  + Controller should only have logic to handle events and logic to call helper methods
* Apex controller
  + If we want to use an apex class as our server side controller, we need to annotate the method with @AuraEnabled and it must be public static
* Aura events/Lightning events
  + A lightning event is an action that happens inside of a lightning component
  + They are handled very differently than regular events
  + We have several different types of events
    - Browser events
      * On click
      * OnMouseOver
    - System events
      * Like on load
    - Ones we define/create
      * Custom component events
      * Custom application events
  + Communication between components
    - Communication is done by passing information from parent to child components with attributes
    - The way to pass information up to other component and application events
      * Component events are how we communicate from child to parent
      * Application events are how we communicate to any component that isn’t or might not be directly related to the triggering component
  + These events serve as our method of async communication between sibling/cousin components or from clid to parent components
  + The difference between application and component event is how they propagate in the DOM
  + Component events are very similar to DOM events from Javascript
    - They go through a capturing phase that starts at the root component of the application and travels down the component hierarchy to the triggering component
    - They go through a bubbling phase where they travel from the triggering component to the root component
  + Component events fire in the bubbling phase by default but we can have them invoke in the capturing phase if we want
  + Application events can be broadcast to every component in the application no matter the component hierarchy
  + They have 3 phases
    - Capturing phase
    - Bubbling phase
    - Default phase
      * Default way our event handler events occur
      * No set order in which the event propagates through the component family tree
        + It will travel to components that are not ancestors of the triggering component
      * Because of this, these events take longer to fire
      * The only time you should use an application event is if you need to communicate with a component that isn’t a direct ancestor
  + Creating events
    - To create an event, we create a new lightning event
    - To determine if its an application or controller event, we set the type attribute of the aura event tag to APPLICATION or COMPONENT
    - These events will use aura attributes to hold information that it needs to pass between components
    - Aura attributes in events are called event parameters and they are used to hold the information you might want to pass between components
  + Firing events
    - Are done asynchronously so sometimes it can take a bit
    - We must register that the component that will fire the event can actually fire it
      * Do this with <aura:registerEvent name”nameOfEvent” type=”namespace:EventName”>
    - We then actually fire the event in the Javascript controller
      * We do so by getting the event
        + Component.getEvent(“Event Name”)
      * Set any parameters
      * Fire using the fire() method
    - For lightning component library component events, we use $A.get(“e.force:eventName”)
  + Handling events
    - Once fired, an event will propagate and it will look for components that will handle the event
    - Any component that we want to be able to handle that kind of event must declare that they can handle it
      * We do this with <aura;handler name=”nameOfEvent” event=”namespace:EventName” action=”{!c.controllerMethod}”
        + This is for component events
      * For application events we do not need to specify a name attribute
    - Optional phase attribute
      * Will determine in which phase of propagation the event will be handled
    - System events
      * Name is set to name of system event we want to handle
      * The value attribute is set to “{!this}”
      * And the action is the same