CS 5044
Object-Oriented Programming with Java

Q&A Session

GUI development lineage of Java

- Java v1.0 (1995) Abstract Window Toolkit (AWT)
  - The first major in-language attempt at cross-platform desktop GUI support
  - Looked different on each platform, but one code base behaved the same everywhere
- Java v1.2 (1998) Swing
  - Introduced a "pluggable" look and feel; can look native or platform-independent
  - Swing, which still uses AWT, has remained the primary GUI for Java applications
- Java v7 (2008) JavaFX
  - Introduced as the successor to Swing, but is still far less popular (future is uncertain)
  - JavaFX was bundled with Java v8-v10, but has been spun off (again) starting with Java v11
    - Much more sophisticated event model and property handling
      - APIs are fully consistent with functional programming (more on this later!)
    - Includes support for touch, sensors, native packaging, and 3D (and a lot more)
    - Open source port runs (most) JavaFX applications on both iOS and Android
  - This is likely to be the future, but Swing will probably stick around for quite a long time
    - Even AWT is still alive and well, with no signs of being deprecated
      - Most Swing components essentially "extend" AWT classes and use AWT events
    - JavaFX can easily incorporate Swing components (and vice versa)
      - New applications should probably use only JavaFX at this point

Graphical user interfaces in general

- Most commonly associated with a Model-View-Controller (MVC) architecture
  - *Model* - Not a GUI; the underlying system which is being represented by the interface
    - Accesses or mutates the state of the objects within the system
    - Not generally developed as part of the interface; may not even be aware of the GUI
  - *View* - the facade as presented by all the visible user interface components
    - Represents the visual layout of all the components on the screen
  - *Controller* - elements that can be manipulated by the user (via the View, or otherwise)
    - Handles the various events (mouse, action, and so forth) that can be triggered
    - Updates the View as necessary to reflect any resultant state changes
- Consider some of our projects as simple examples
  - Project 3: Tetris
    - Model: the Board object and all related objects/states (shapes, rotations, etc.)
    - View: the playfield and its blocks, the scores at the top, and a start message
    - Controller: the keyboard and timer handlers that update the display and mode
  - Project 6: DAB GUI
    - Model: the DABGame from Project 4
    - View: the button, labels, menu bar, combo boxes, and DABGrid component
    - Controller: The draw button and menu item handlers (plus others within DABGrid)

MVC View (Swing)

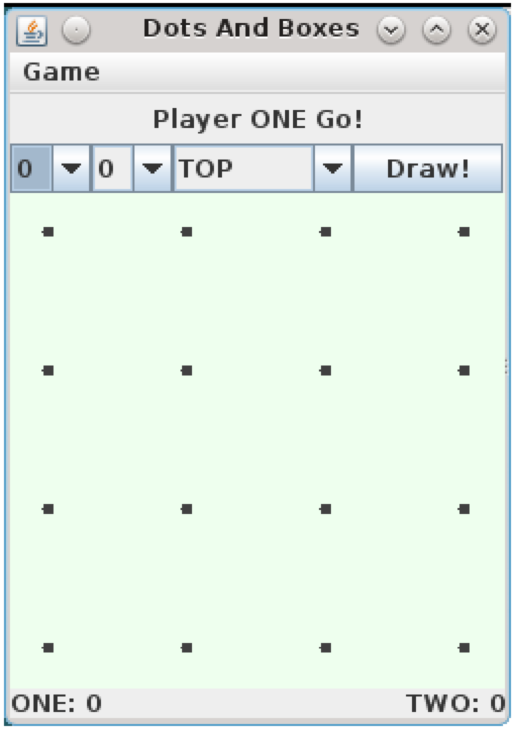
- Visual layout of components; relies upon layout managers for each container
  - Built-in layout managers Includes: Flow, Border, Box, Grid, GridBag, and Group
    - Often complex layouts are designed with help from GUI builders
      - Helps with layout and connecting to methods/fields in your code
      - Usually you still need to do at least some work by hand
        - You can always design the whole thing by hand
  - Generally we only need to combine built-in widget types (such as JButton, JLabel, etc.)
    - You can also define custom-drawn components, to render other parts of the interface
      - This is usually only necessary for games or other specialized systems
      - Tetris (JavaFX) and the Project 6 DABGrid (Swing) both take this approach
- The Swing toolkit is fairly robust
  - Containers contain components or other containers, forming a tree
    - Components cover all the familiar user interface widgets and menu systems
  - You can choose native or one of several platform-independent look and feel choices
  - Swing has been around, in the same essential form, for about 20 years now
    - Many tutorials, best practices, and lessons learned are readily available

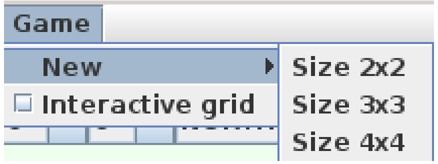
MVC Controller (Swing)

- Includes all input to the application, often as presented by the View component
- The Swing system itself is in (nearly) complete control of the application
  - Events are managed as objects in an event queue, each handled by your controller
  - Your controller code is developed as a set of call-back methods that react to events
    - Your code handles the event, then implicitly returns control back to Swing
    - We're somewhat used to this; in TDD, the test methods "control" the application
- GUI applications are necessarily event-driven in nature
  - Components trigger various events upon user interaction via mouse/keyboard
  - Timers can also be set to trigger events without user interaction
- Controller code can access (and mutate) any part of the View or the Model
  - However, note that long-running tasks can become very problematic
    - The UI will remain unresponsive while your controller is handling any event
    - The solution is asynchronous processing on one or more parallel execution threads
      - This is actually fairly easy to achieve in Java (but that's another course!)

Project 6: DAB GUI

- We're adding a graphical user interface to our DABGame!

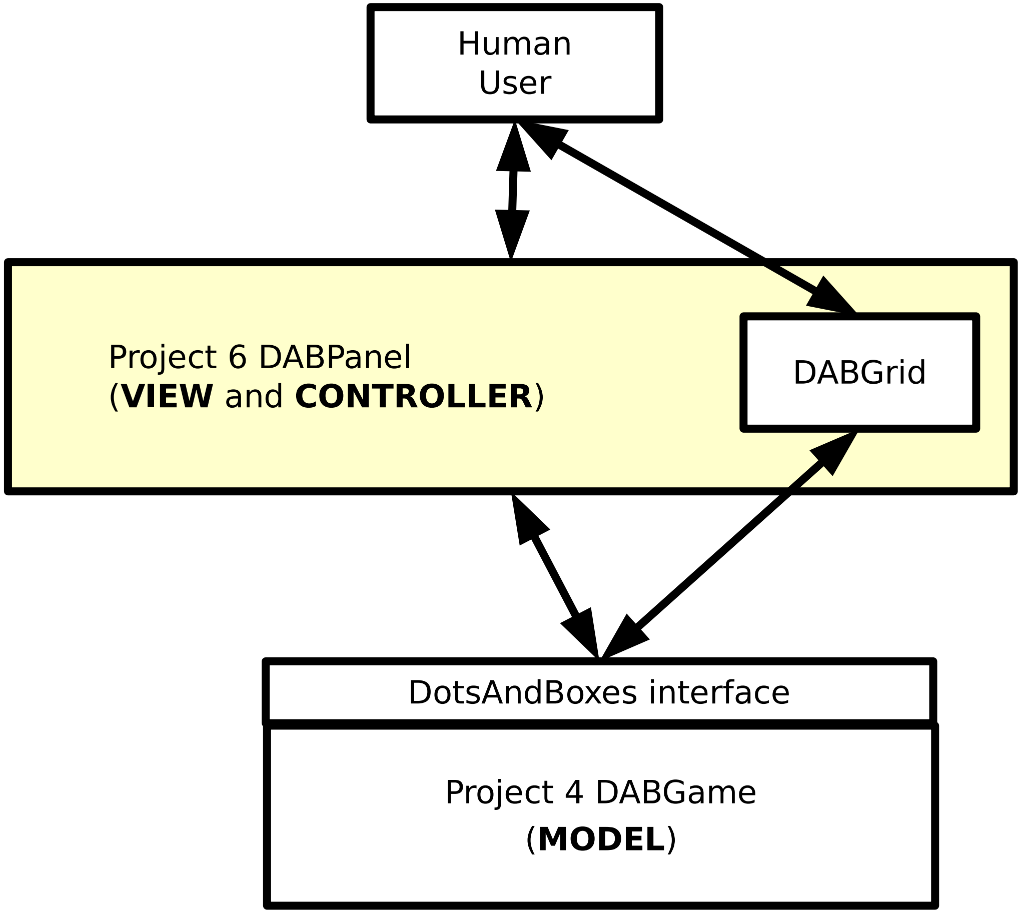




Project 6: DABGame (MVC "Model")

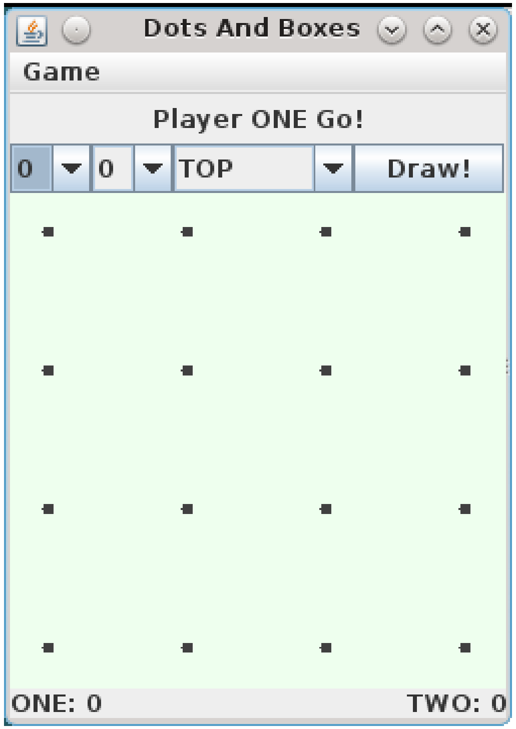
- Project 6 depends entirely upon the Project 4 DABGame as the Model
  - Please don't worry if your Project 4 is incomplete or otherwise not working properly
    - A fully operational reference implementation is available for all to download
    - The reference solution passes all the Web-CAT tests (source code not included)
    - It's easy to switch between your implementation and the reference implementation
- DABGame doesn't need any changes to act as the model
  - View and Controller both interact with the Model via the DotsAndBoxes interface

Project 6: MVC

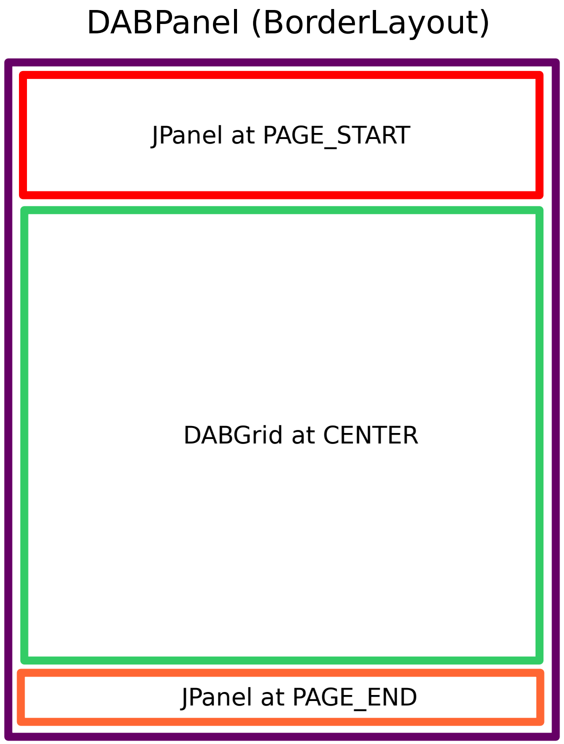


Project 6: Component layout (MVC "View")

- This is just a **suggestion** (you don't need to replicate this exact layout)
  - Any reasonable arrangement of the components is perfectly acceptable
  - Layout must be reasonably responsive to resizing of the frame



Player ONE Go!



DABPanel (BorderLayout)

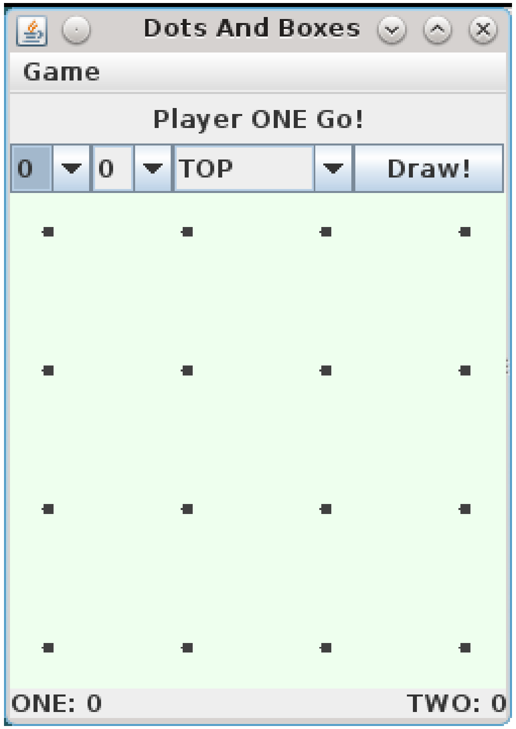
JPanel at PAGE_START

DABGrid at CENTER

JPanel at PAGE_END

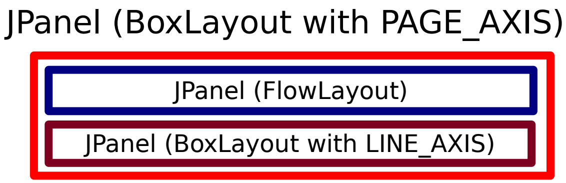
Project 6: Component layout (MVC "View")

- This is just a **suggestion** (you don't need to replicate this exact layout)
  - Any reasonable arrangement of the components is perfectly acceptable
  - Layout must be reasonably responsive to resizing of the frame



Player ONE Go!

ONE: 0 TWO: 0

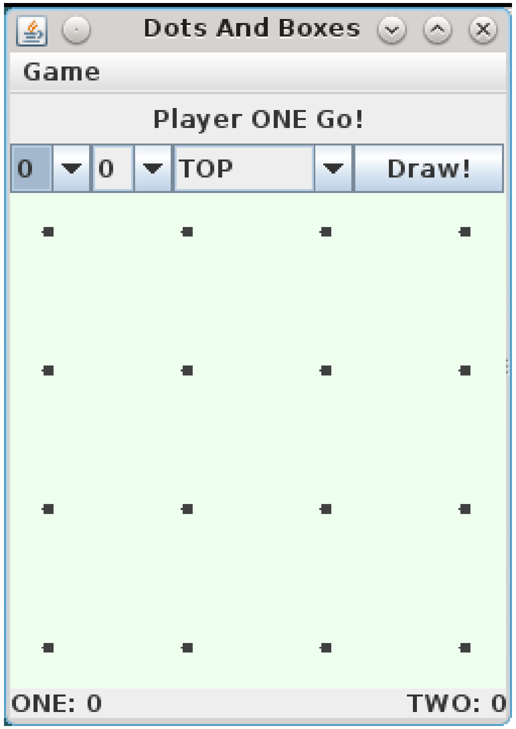

JPanel (BoxLayout with PAGE_AXIS)

JPanel (FlowLayout)

JPanel (BoxLayout with LINE_AXIS)

Project 6: Component layout (MVC "View")

- This is just a **suggestion** (you don't need to replicate this exact layout)
  - Any reasonable arrangement of the components is perfectly acceptable
  - Layout must be reasonably responsive to resizing of the frame



Player ONE Go!

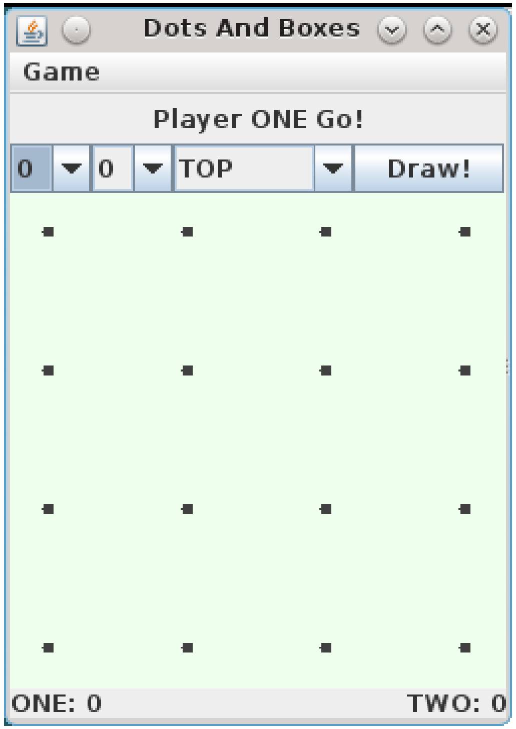
ONE: 0

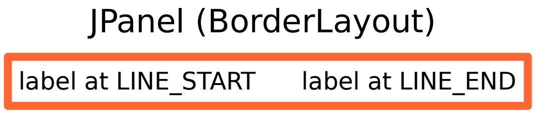
TWO: 0

DABGrid component (no layout)

Project 6: Component layout (MVC "View")

- This is just a **suggestion** (you don't need to replicate this exact layout)
  - Any reasonable arrangement of the components is perfectly acceptable
  - Layout must be reasonably responsive to resizing of the frame





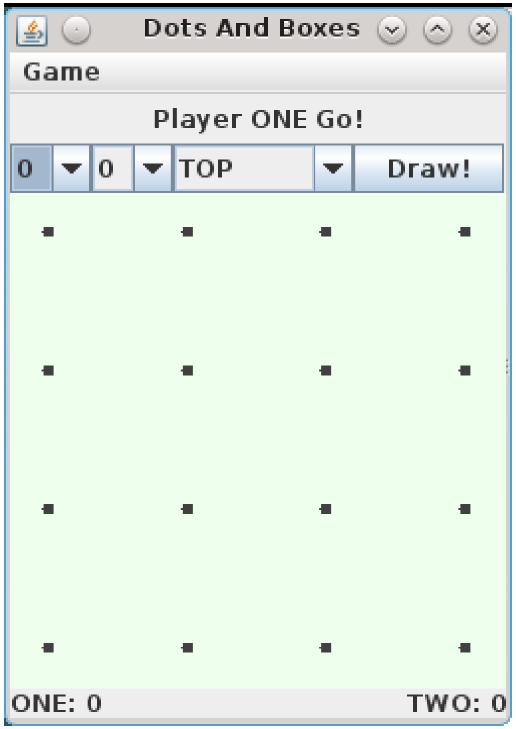
JPanel (BorderLayout)

label at LINE_START

label at LINE_END

Project 6: Component layout (MVC "View")

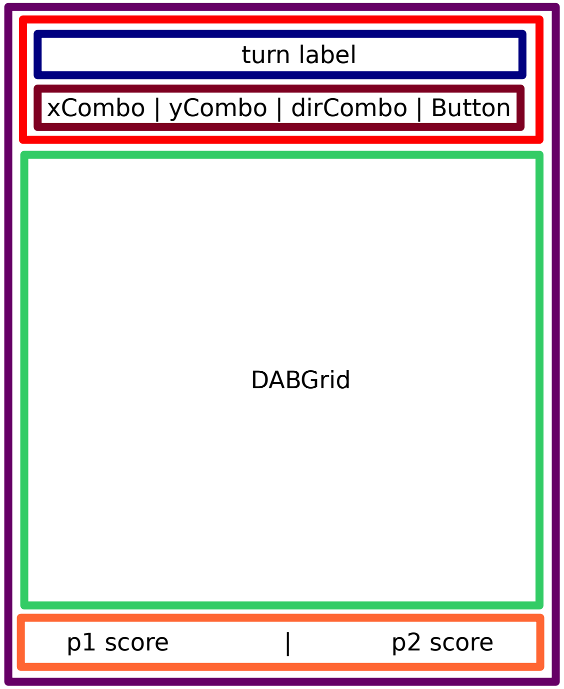
- This is just a **suggestion** (you don't need to replicate this exact layout)
  - Any reasonable arrangement of the components is perfectly acceptable
  - Layout must be reasonably responsive to resizing of the frame



Player ONE Go!

ONE: 0

TWO: 0



turn label

xCombo | yCombo | dirCombo | Button

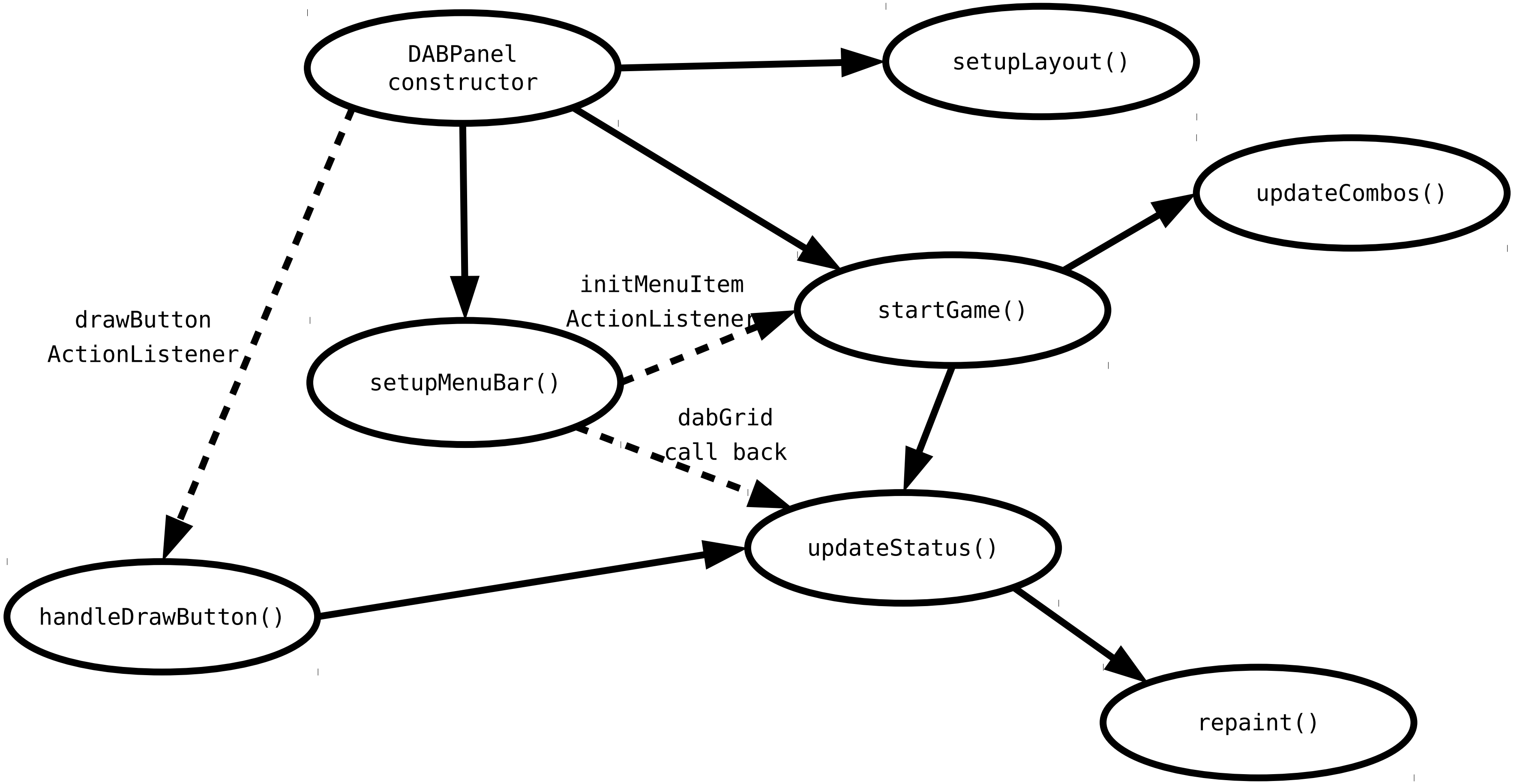
DABGrid

p1 score | p2 score

Project 6: Event handling (MVC "Controller")

- Components that trigger events:
  - Menu items
    - New Game (three possible sizes)
    - Interactive Mode (toggles to activate/deactivate)
  - Draw button
    - Reads combo box selections for coordinate and direction values
- Each of these components posts an ActionEvent which we must handle
  - New Game items invoke our startGame() helper, each with an appropriate size parameter
  - Interactive Mode is slightly more complex
    - Proceed based on the isSelected() state of the checkbox item
      - Selected?
        - Call setCallback() on the DABGrid to notify us when an edge is drawn
        - The callback implementation will just call our updateStatus() helper
      - Unselected?
        - Call setCallback(null) on the DABGrid to disable interactivity
  - Draw Button just invokes our handleDrawButton() method
    - This method reads the X, Y, and Direction combo boxes, then calls game.drawEdge()
      - If the return from drawEdge() is true, we must call our updateStatus() helper

Project 6: Method interaction overview



Adding event listeners

- We'll cover the functional programming aspects of Java 8 in depth after the break
  - These aspects introduce an entirely new syntax into the Java language
  - Two of these features will be particularly convenient for Project 6
    - Lambda Expressions
      - Can take the place of certain classes that implement certain interfaces
        - Useful for anonymous inner classes, which are very common in GUI code
    - Method References
      - Can take the place of certain Lambda Expressions, when parameters can be inferred
        - Again quite useful and common in GUI code
- Don't worry about the details at this point
  - You can (and should!) start development without any functional programming features
    - Later, we can easily simplify the code by leveraging functional programming syntax
    - You can read ahead (section 19.5) in the textbook if you're interested now
      - We'll cover this much more thoroughly after the break