COMP2511

Object-Oriented Design and Programming

User Interface Design

Wayne Wobcke

w.wobcke@unsw.edu.au

Today's Lecture

- Usability
- User Centred Design
- Java GUI Programming
- Observer Pattern
- Model View Controller

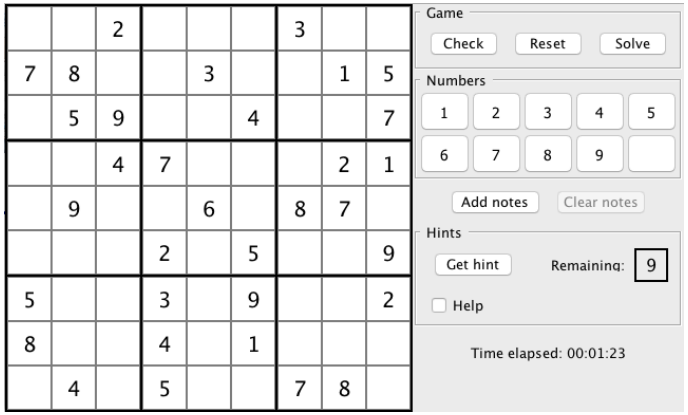
Usability Heuristics

1. Visibility of system status
2. Match between system and real world
3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Help users recognize, diagnose, recover from errors
10. Help and documentation

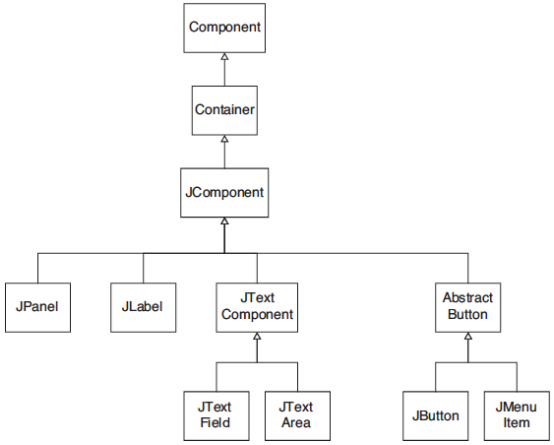
User Centred Design

- Personas
  - Who are the users?
  - What are their goals/skills/environment?
- Scenarios
  - How/when will they use the system?
- Use Cases
  - How will they interact with the system?

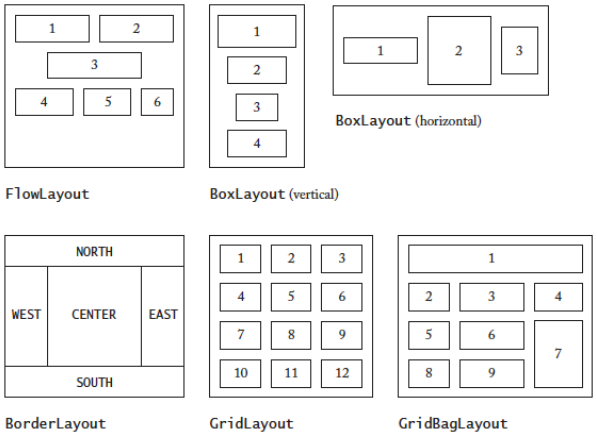
Sudoku Interface Example



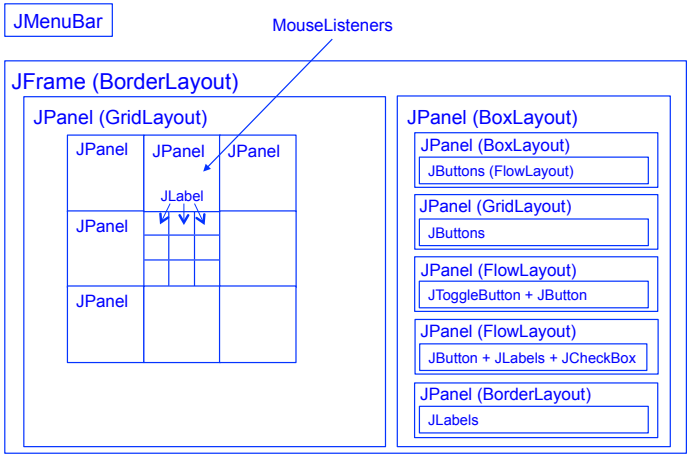
Java GUI Programming: Swing



Layout Managers



Sudoku Interface Components



Sudoku View

```
JFrame sudoku = new JFrame();
sudoku.setLayout(new BorderLayout());

SudokuGrid grid = new SudokuGrid(game);
SudokuController sudokuController = new SudokuController(grid, game);
grid.setController(sudokuController);
sudoku.add(sudokuGrid, BorderLayout.CENTER);

ButtonPanel buttonPanel = new ButtonPanel(game, grid);
ButtonController buttonController = new ButtonController(grid, game);
buttonPanel.setController(buttonController);
sudoku.add(buttonPanel, BorderLayout.EAST);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.pack();
frame.setVisible(true);
```

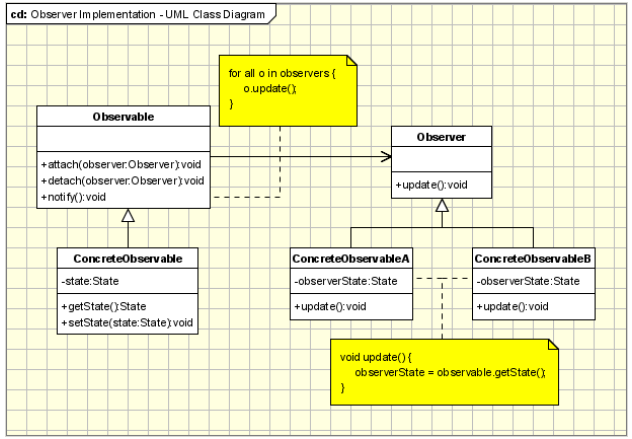

Simple Example: Timer

```
JFrame frame = new JFrame();

final JLabel label = new JLabel();
label.setText("Time: " + Calendar.getInstance().getTime);
frame.setLayout(new FlowLayout());
frame.add(label);

Timer t = new Timer(DELAY, new ActionListener() {
   public void actionPerformed(ActionEvent event) {
      label.setText("Time: " + Calendar.getInstance().getTime());
   }
t.start();

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.pack();
frame.setVisible(true);
```

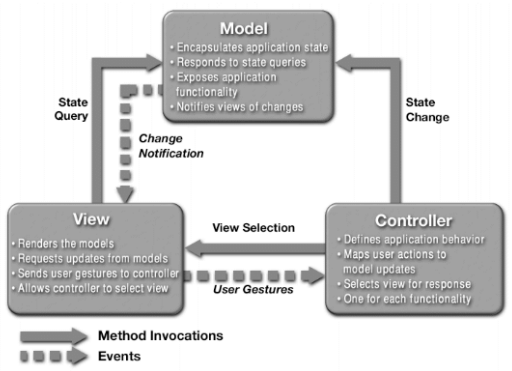

Observer Pattern

- Motivation
  - Need a way for a number of objects to be informed of changes in another object
- Intent
  - Have the object maintain a list of the objects that need to be informed, and notify them whenever any change occurs

Observer Pattern Implementation



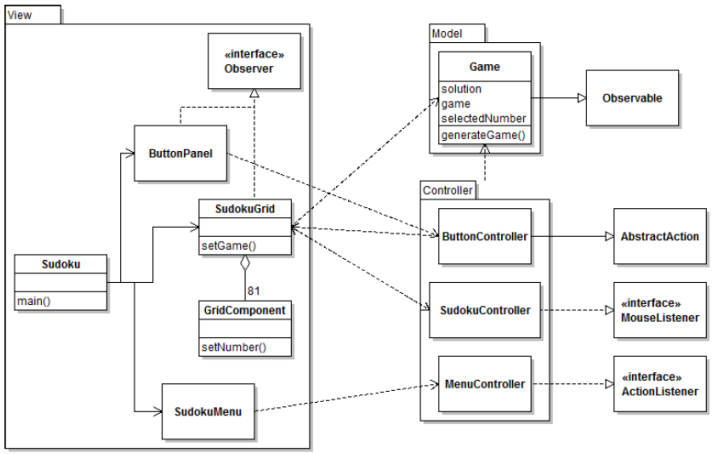
MVC Architectural Pattern



■ Not necessarily needed in the project

Image http://lkubaski.wordpress.com/2012/12/20/java-se-application-design-with-mvc-with-images/

Sudoku UML Class Diagram



Sudoku Controllers

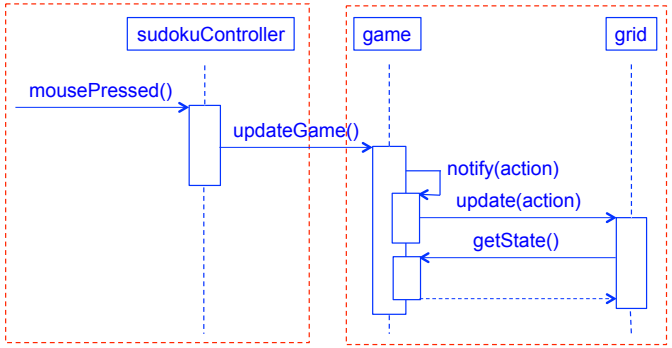
```
JFrame sudoku = new JFrame();
sudoku.setLayout(new BorderLayout());

SudokuGrid grid = new SudokuGrid(game);
SudokuController sudokuController = new SudokuController(grid, game);
grid.setController(sudokuController);
sudoku.add(sudokuGrid, BorderLayout.CENTER);

ButtonPanel buttonPanel = new ButtonPanel(game, grid);
ButtonController buttonController = new ButtonController(grid, game);
buttonPanel.setController(buttonController);
sudoku.add(buttonPanel, BorderLayout.EAST);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.pack();
frame.setVisible(true);
```


Sudoku Controller Example

```
public class SudokuController implements MouseListener {

public void mousePressed(MouseEvent e)
  {
    JPanel panel = (JPanel) e.getSource();
    Component component = panel.getComponentAt(e.getPoint());
    if (component instanceof GridComponent)
      {
        GridComponent selected = (GridComponent) component;
        // either set selected on view or update model
      }
  }
}
```

Sudoku Controller

SudokuController implements MouseListener
Game extends Observable; SudokuGrid implements Observer
grid.setController(sudokuController); game.addObserver(grid);



Observer Pattern

Next Week

- Design Patterns and Refactoring
  - Decorator & Composite Patterns