Design principles and patterns
Design Goals

1. Make it easy to build
2. Make it easy to test
3. Make it easy to maintain
4. Make it easy to change

SIMPLE

FLEXIBLE

INTUITIVE
Design

How do we go about design and what do we produce?

- Practices
- Principles
- Patterns

LAST TIME

What are the characteristics of good design?

What are good solutions to common design problems?
Practices: Forms of Models

- Diagrams (UML)
- Text (hyperlinked)
- Prototypes – Sudoku versions
- Mathematical models
- Charts, graphs

All Driven by User Stories and Requirements

LAST TIME
Design

• Principles

What are the characteristics of good design?

TODAY: Principles of design
Are there any?
Do they matter?
Can we evaluate them in action?
User stories

Initial grid and number pad appear. Original entries are black. Current number "1" is highlighted in number pad.

User clicks on number pad cell. Current number de-highlighted. New number is highlighted.

User clicks on cell. Cell is blank or has user entered value. Current number is consistent for that cell.

Current number appears in that cell in blue.
Start with the following Design
Develop this design using “good” design principles
Key design principle

Single responsibility principle:
classes and methods should have a single, focused responsibility.

Foundation of Software Development
Several related Principles:
  Encapsulation
  Abstraction, etc.
Developed historically: “Spagetti Code” “Goto
Considered Harmful”
What is the responsibility of this class?

- Creates and stores grid values
- Answers questions about data e.g. consistency
What is the responsibility of this class?

- Creates and stores grid values
- Answers questions about data e.g. consistency

- Creates and stores grid cells (buttons)
- Responds to clicks
- Updates button titles
What is the responsibility of this class?

- **GridModel**
  - Creates and stores grid values
  - Answers questions about data e.g. consistency

- **GridView**
  - Creates and stores grid cells (buttons)
  - Responds to clicks
  - Updates button titles

- **NumPadView**
  - Creates and stores numpad cells (buttons)
  - Responds to clicks
  - Controls highlights
GridModel
• Creates and stores grid values
• Answers questions about data e.g. consistency

ViewController

GridView
• Creates and stores grid cells (buttons)
• Responds to clicks
• Updates button titles

NumPadView
• Creates and stores numpad cells (buttons)
• Responds to clicks
• Controls highlights

What is the responsibility of this class?

• Creates model & views
• Mediates input
Key design principle

Law of Demeter – or Law of Least Knowledge

Classes should know as little as possible about anything beyond themselves.

Why?????
The fundamental notion is that a given object should assume as little as possible about the structure or properties of anything else (including its subcomponents).

**Law of Demeter**

only talk to your friends

Each unit should have only limited knowledge about other units:
only units "closely" related to the current unit.

Each unit should only talk to its friends; don't talk to strangers.

Only talk to your immediate friends.

no no no: myGame->theBoard->cells[0][0].update()
Should not know details of theBoard
Knows about the model and views. Doesn’t know or care about how they do their job.

GridModel

Knows about its data. Doesn’t know or care about how its data is used.

ViewController

GridView

NumPadView

Knows about its cells. Doesn’t know or care what its data “means”; how it is generated or managed

Knows about its cells. Understands when cells are highlighted. Doesn’t know or care about what highlighting “means”.

9/16/13
Key design principles

Classes and methods should have high cohesion – low coupling

- Focused on a single responsibility, Uniform level of abstraction
- Doesn’t know or care about many other classes How to evaluate?
Key design principles

Classes and methods should have high cohesion – low coupling

- Focused on a single responsibility,
  Uniform level of abstraction

- Doesn’t know or care about many other classes
Cohesion: Examples

- Functional cohesion (SRP): grouped because they contribute to a single well-defined task
- Coincidental cohesion: grouped together because
- Logical cohesion: grouped because they fall into some logical category, i.e. I/O
- Temporal/sequential cohesion: grouped because they are processed at the same time or in sequence; i.e. error handling (create, notifies user, etc.)
- Communicational: grouped because they operate on same data, e.g., operate on same record
Coupling: Examples

- **Content coupling:** a module or class relies on the implementation of another; i.e. accessing private data
- **Common coupling:** modules or classes share global data
- **Control coupling:** one module controls the logic of another (e.g. passing a what-to-do flag)
- **Data-structure coupling:** modules share composite data structure
- **Message coupling:** modules communicate through interface
User Story

Initial grid and number pad appear. Original entries are black. Current number “1” is highlighted in number pad.

What is the sequence
How does it know what?
Use Sequence Diagram

9/16/13
Sequence diagram: display

Who has initial data?
Who needs initial data?
Sequence diagram: display

GridModel
 ViewController
 GridView
 NumPadView

initialize

get value at row, column

set value at row, column

What type is value?

Initialize with frame

Initialize with frame

9/16/13
Sequence diagram: display

GridModel
ViewController
GridView
NumPadView

initialize

get value at row, column

set value at row, column

Initialize with frame

Return values are often implicit

9/16/13
- Creates model & views
- Mediates input

**ViewController:** UIViewcontroller
GridModel* gridModel
GridView* gridView
NumPadView* numPadView

**GridModel:** NSObject
int cells[9][9]
(void) generateGrid
(int) getValueAtRow: column

- Creates and stores grid values
- Answers questions about data e.g. consistency

**GridView:** UIView
NSMutableArray* cells
(void) setValueAtRow: column: to:

- Creates and stores grid cells (buttons)
- Responds to clicks
- Updates button titles

**NumPadView:** UIView
NSMutableArray* cells

- Creates and stores numpad cells (buttons)
- Responds to clicks
- Controls highlights

9/16/13
User clicks on number pad cell.

Current number de-highlighted. New number is highlighted.
Use UML Notation

NumPadView: UIView
NSMutableArray* cells

9

UIButton
A has a B

A has one or more Bs

A has a n Bs

UML Notation
Who should know that numPad cells are buttons?
Law of least knowledge. NumPadView should not reveal implementation details. ViewController should not care about implementation details.

- Creates model & views
- Mediates input

ViewController: UIViewcontroller
GridModel* gridModel
GridView* gridView
NumPadView* numPadView

- Creates and stores grid values
- Answers questions about data e.g. consistency

GridModel: NSObject
int cells[9][9]
(void) generateGrid
(int) getValueAtRow: column

- Creates and stores grid cells (buttons)
- Responds to clicks
- Updates button titles

GridView: UIView
NSMutableArray* cells
(void) setValueAtRow: column: to:

- Creates and stores numpad cells (buttons)
- Responds to clicks
- Controls highlights

NumPadView: UIView
NSMutableArray* cells

9/16/13
Who should know that numPad cells are buttons?
Design principle

INTUITIVE
Use real world objects

Is a number pad a collection of buttons?
Is a sudoku grid a collection of buttons?
Design principle

INTUITIVE
Use real world objects
Who should know that numPad cells are buttons?
Sequence diagram: numPad cell tapped

NumPadView

(UIButton)

(cellSelected)

setBackgroundColor to white (old cell)

setBackgroundColor to yellow (new cell)
• Creates model & views
• Mediates input

**ViewController (UIViewController)**
- **GridModel**: NSObject
  - int cells[9][9]
  - (void) generateGrid
  - (int) getValueAtRow: column

**GridView (UIView)**
- **GridModel**: GridModel*
- GridView*: gridView
- (void) setValueAtRow: column: to:

**NumPadView**: UIView
- **NumPadView**: NumPadView*
- NSMutableArray* cells
- Int currentValue
- (void) cellSelected:

- Creates and stores grid values
- Answers questions about data e.g. consistency
- Creates and stores grid cells (buttons)
- Responds to clicks
- Updates button titles
- Creates and stores numpad cells (buttons)
- Responds to clicks
- Controls highlights
User clicks on cell. Cell is blank or has user-entered value. Current number is consistent for that cell.

Current number appears in that cell in blue.
GridView: UIView
NSMutableArray* cells
(void) setValueAtRow: column: to:
(void) cellSelected

UIButton
addTarget: action

9x9
Sequence diagram: gridView tapped

GridView: now what?
Sequence diagram: gridCell tapped
Sequence diagram: gridCell tapped

- GridModel
- ViewController
- GridView
- UIButton
- NumPadView

- Check if cell mutable
- Cell is mutable
- Check if value m consistent
- Value m is consistent
- cellSelected
- Get current value
- Current value is m
Sequence diagram: gridCell tapped

- GridModel
  - Check if cell mutable
  - Cell is mutable
  - Check if value m consistent
  - Value m is consistent
  - Set value to m

- ViewController
  - cellSelected
  - cell Selected
  - Get current value
  - Current value is m
  - Value m is consistent
  - Set value to m

- GridView
  - cellSelected

- UIButton

- NumPadView

9/16/13
- Creates model & views
- Mediates input

**ViewController**: UIViewController
- GridModel* gridModel
- GridView* gridView
- NumPadView* numPadView
- (void) gridCellSelected:

**GridView**: UIView
- NSMutableArray* cells
- Int currentValue
- (void) setValueAtRow: column to:
- (void) cellSelected:

**NumPadView**: UIView
- NSMutableArray* cells
- Int currentValue
- (void) cellSelected:
- (int) getCurrentValue:

- Creates and stores grid values
- Answers questions about data e.g. consistency

- Creates and stores grid cells (buttons)
- Responds to clicks
- Updates button titles

- Creates and stores numpad cells (buttons)
- Responds to clicks
- Controls highlights

9/16/13
Design principle

SIMPLE, INTUITIVE, FLEXIBLE

Don’t repeat yourself (D.R.Y.)

data/code should occur once and only once
- Creates model & views
- Mediates input

**ViewController**: `UIViewController`  
`GridModel* gridModel`  
`GridView* gridView`  
`NumPadView* numPadView`  
`(void) gridCellSelected:

- Creates and stores grid values
- Answers questions about data e.g. consistency

**GridModel**: `NSObject`  
`int cells[9][9]`  
`(void) generateGrid`  
`(int) getValueAtRow: column`  
`(void) setValueAtRow: column: to:`  
`(bool) isMutableAtRow: column:`  
`(bool) isConsistentAtRow:column:for:

**GridView**: `UIView`  
`NSMutableArray* cells`  
`(void) setValueAtRow: column: to:`  
`(void) cellSelected:

- Creates and stores grid cells (buttons)
- Responds to clicks
- Updates button titles

**NumPadView**: `UIView`  
`NSMutableArray* cells`  
`Int currentValue`  
`(void) cellSelected:`  
`(int) getCurrentValue:

- Creates and stores numpad cells (buttons)
- Responds to clicks
- Controls highlights
sudokuView: UIView

NSMutableArray* cells (id) initWithFrame:

gridView: sudokuView

numPadView: sudokuView

Is this better?
Summary

- Single responsibility principle
- Law of demeter (talk only to your friends)
- High cohesion/low coupling
- Use real world objects
- Don’t repeat yourself (D.R.Y)
Key design principle

Single responsibility principle:
classes and methods should have a single, focused responsibility.
GridModel: NSObject
int cells[9][9]
(void) generateGrid
(int) getValueAtRow: column
to:
(void) setValueAtRow: column:
to:
(bool) isMutableAtRow: column:
(bool) isConsistentAtRow:column:for:

ViewController: UIViewController
GridModel* gridModel
GridView* gridView
NumPadView* numPadView
(void) gridCellSelected:

Manage grid data

Manage grid cell UI

NumPadView: UIView
NSMutableArray* cells
Int currentValue
(void) cellSelected:
(int) getCurrentValue:

Manage numPad cell UI

Coordinates view & model
Design principle

Law of Demeter

Principle of Least Knowledge
Knows about its objects but doesn’t know or care about other they do their jobs.

**ViewController**: UIViewController
- GridModel* gridModel
- GridView* gridView
- NumPadView* numPadView
- (void) gridCellSelected:

**GridModel**: NSObject
- int cells[9][9]
- (void) generateGrid
- (int) getValueAtRow: column
- (void) setValueAtRow: column: to:
- (bool) isMutableAtRow: column:
- (bool) isConsistentAtRow:column:for:

**GridView**: UIView
- NSMutableArray* cells
- (void) setValueAtRow: column to:
- (void) cellSelected:

**NumPadView**: UIView
- NSMutableArray* cells
- Int currentValue
- (void) cellSelected:
- (int) getCurrentValue:

Knows about its data.
Doesn’t know or care about how its data is used.

Knows about its cells.
Doesn’t know or care about how clicks are handled other than to pass the message along.

Knows about its cells and current value.
Doesn’t know or care about how current value is used.

9/16/13
Key design principles

Classes should have
high cohesion – low coupling

Focused on a single responsibility

Doesn’t know about, care about, depend on too many other classes
Knows about its objects: gridModel, gridView, numPadView
But doesn’t know or care about how they do their job.

ViewController: UIViewController
GridModel* gridModel
GridView* gridView
NumPadView* numPadView
(void) gridCellSelected:

GridModel: NSObject
int cells[9][9]
(void) generateGrid
(int) getValueAtRow: column
(void) setValueAtRow: column: to:
(bool) isMutableAtRow: column:
(bool) isConsistentAtRow:column:for:

GridView: UIView
NSMutableArray* cells
(Int) currentValue
(void) setValueAtRow: column to:
(void) cellSelected:

NumPadView: UIView
NSMutableArray* cells
Int current value
(void) cellSelected:
(int) getCurrentValue:

Knows about its data.
Doesn’t know or care about how its data is used.

Knows about its cells.
Doesn’t know or care about how clicks are handled other than to pass the message along.

Knows about its cells and current value.
Doesn’t know or care about how current value is used.

9/16/13
Design principle

INTUITIVE

Use real world objects

Domain model \[\rightarrow\] Design model
Our design often augments domain classes with controller and utilities classes; e.g. viewController

- Creates model & views
- Mediates input

viewController: UIViewController
GridModel* gridModel
GridView* gridView
NumPadView* numPadView
(void) gridCellSelected:

GridModel: NSObject
int* cells[9][9]
(void) generateGrid
(int) getValueAtRow: column
(void) setValueAtRow: column: to:
(bool) isMutableAtRow: column:
(bool) isConsistentAtRow: column:for:

GridView: UIView
NSMutableArray* cells
(void) setValueAtRow: column:to:
(void) cellSelected:

NumPadView: UIView
NSMutableArray* cells
Int currentValue
(void) cellSelected:
(int) getCurrentValue:

- Creates and stores grid values
- Answers questions about data e.g. consistency

- Creates and stores grid cells (buttons)
- Responds to clicks
- Updates button titles

- Creates and stores numpad cells (buttons)
- Responds to clicks
- Controls highlights
CS 121 Motto

UNDER-PROMISE
and
OVER-DELIVER
Design Principle

Dependency-inversion principle:
Abstraction should not depend on details, details should depend on abstraction.

Interface should not depend on implementation, implementation should realize interface.
Sequence diagram: gridCell tapped

GridModel

ViewController

GridView

UIButton

NumPadView

Check if cell mutable

Cell is mutable

Check if value consistent

Value is consistent

Set value to m

cellSelected

TARGET-ACTION design pattern

This is what we want to do. Can we do it?

Get current value

Current value is m

Set value to m

Value is consistent

Set value to m

Check if cell mutable

Cell is mutable

ViewController on GridModel

Get current value

Current value is m

Set value to m

This is what we want to do. Can we do it?
Design Patterns are good solutions to common problems.
Observer design pattern

Objects $A_1, A_2, \ldots, A_n$ need to know about $B$’s state

B notifies all observers when its state changes.
B doesn’t need to know what the $A$’s are, why they care, or what they do in response.
iOS Observers

iOS has a few ways to implement the observer-like design pattern:

- Target-action
- Delegation
- Notification

UIControl (incl. UIButton) uses target-action
Button doesn’t know anything about its context. During runtime it is told where to send messages for certain events.
ViewController

GridView

UIButton

setTarget: action:
cellSelected

cell Selected
ViewController: UIViewController
GridModel* gridModel
GridView* gridView
NumPadView* numPadView
(void) gridCellSelected:

GridModel: NSObject
int cells[9][9]
(void) generateGrid
(int) getValueAtRow: column
(void) setValueAtRow: column to:
(bool) isMutableAtRow: column:
(bool) isConsistentAtRow:column:for:

GridView: UIView
NSMutableArray* cells
(int) getCurrentValue
(void) cellSelected:
(void) setValueAtRow: column to:
(void) setTarget: action:

NumPadView: UIView
NSMutableArray* cells
Int currentValue
(int) cellSelected:
(int) getCurrentValue:
Implementation details

UIViewController has a cellSelected method (I called my gridCellSelected)!
We can pass objects to this method but not ints. I chose to get the current row and column from gridView.
Implementation details

GridView has data members to store the object and selectory to notify when a cell is selected.

GridView has a setTarget: action: method.
Implementation details

This is the call that notifies the ViewController that a grid cell has been selected.

This is the setTarget:action: method.
More design principles and patterns to come
Summary: design principles

• Single responsibility principle
• Law of demeter (principle of least knowledge)
• High cohesion/low coupling
• Use real world objects
• Don’t repeat yourself (D.R.Y)
• Favor composition over inheritance
• Dependency-inversion principle
Summary: design patterns

- Observer (Target-Action)
Sequence diagram: gridCell tapped

GridModel

ViewController

GridView

UIButton

NumPadView

Check if cell mutable

Cell is mutable

Check if value consistent

Value is consistent

Set value to m

Set value to m

Get current value

Current value is m

PROBLEM

Cell is mutable

Cell Selected

cellSelected
What does a button need to know to identify itself?
Ans. Its row and column.
Cell (int row, column) has a UIButton. UIButton has a Cell (int row, column).
Favor composition over inheritance to reduce cohesion

BLACK box reuse

composition

WHITE box reuse

inheritance

Is a
Design principle

Favor composition over inheritance

Caveat: sometime inheritance is the right thing (i.e. gives us polymorphism)
Sequence diagram: gridCell tapped

What does a button need to know to identify itself?
Ans. Its row and column.

We can use the button tag to give each button and identifier. (Map row, column to button id.)
Sequence diagram: gridCell tapped

- GridModel
- ViewController
- GridView
- UIButton
- NumPadView

Check if cell mutable

Cell is mutable

Check if value consistent

Value is consistent

Set value to m

Get current value

Current value is m

Set value to m

Cell Selected

cellSelected

PROBLEM

9/16/13
Sequence diagram: gridCell tapped

ONE SOLUTION: have buttons report to view controller.

Increases cohesion:
ViewController knows cells are implemented as buttons.
Another solution:

Now GridView has to know about ViewController and its interface