Lesson 41: Craps Lab #1 (W12D2)

Balboa High School

Michael Ferraro

November 3, 2015

Do Now

1. Load up your `BookOrder` program (from the HW)
2. Compare your solutions to mine here

Aim

Students will get an overview of the *Craps Lab* and get an explanation of its design and architecture.

Notes about PS #6

- At this point, you should be (mostly!) done with §§1-4
- Today's lesson enables you to start §5
- This lesson and the next one should be enough support for you to do all of §5

Our Goal

Our goal is, as part of a (virtual) 3-person programming team, to build a Craps application.

Litvin provides a working version of the application for us

- You'll be playing with the JAR file that contains a working version while you work on the problem set
- Let's see what it does...

Rules of Craps

When you do the reading in Litvin §6.9, you'll see the rules for the game of Craps, and you'll have to summarize them in PS #6, §5.

Application Design: Teamwork

- There are multiple components and they **need** to be separate, as a team of 3 will be dividing up the labor

Application Design: Teamwork

- There are multiple components and they **need** to be separate, as a team of 3 will be dividing up the labor
- Division of labor:

Application Design: Teamwork

- There are multiple components and they **need** to be separate, as a team of 3 will be dividing up the labor
- Division of labor:
  - **Architecture, Animation:** Maria Litvin, author

Application Design: Teamwork

- There are multiple components and they **need** to be separate, as a team of 3 will be dividing up the labor
- Division of labor:
  - **Architecture, Animation:** Maria Litvin, author
  - **GUI Interface:** Aisha

Application Design: Teamwork

- There are multiple components and they **need** to be separate, as a team of 3 will be dividing up the labor
- Division of labor:
  - **Architecture, Animation:** Maria Litvin, author
  - **GUI Interface:** Aisha
  - **. . . the leftovers:** You!

Modeling a Real-World Game

- A Java program will be *modeling* a real-world system (game)

Modeling a Real-World Game

- A Java program will be *modeling* a real-world system (game)
- What are the elements of *Craps* that the program will need to represent/model?

Modeling a Real-World Game

- A Java program will be *modeling* a real-world system (game)
- What are the elements of *Craps* that the program will need to represent/model?
    - Things you see:

Modeling a Real-World Game

- A Java program will be *modeling* a real-world system (game)
- What are the elements of *Craps* that the program will need to represent/model?
  - Things you see:
    - dice

Modeling a Real-World Game

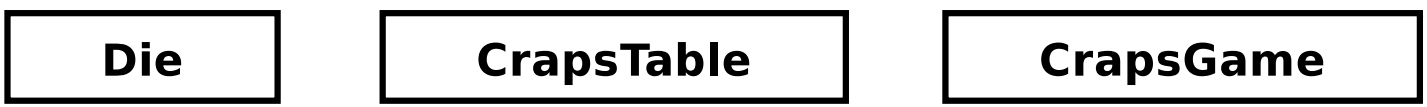
- A Java program will be *modeling* a real-world system (game)
- What are the elements of *Craps* that the program will need to represent/model?
  - Things you see:
    - dice
    - table

Modeling a Real-World Game

- A Java program will be *modeling* a real-world system (game)
- What are the elements of *Craps* that the program will need to represent/model?
  - Things you see:
    - dice
    - table
  - What you don't: the game's logic (i.e., the rules)

Modeling a Real-World Game

- A Java program will be *modeling* a real-world system (game)
- What are the elements of *Craps* that the program will need to represent/model?
  - Things you see:
    - dice
    - table
  - What you don't: the game's logic (i.e., the rules)



Modeling a Real-World Game

- The idea of the `Die` class is that it should be relatively simple

Modeling a Real-World Game

- The idea of the `Die` class is that it should be relatively simple
  - `Die` will have a method called `roll()`, which simulates a die's roll by setting an `int` field to a random integer from the set $\{1, 2, 3, 4, 5, 6\}$

Modeling a Real-World Game

- The idea of the `Die` class is that it should be relatively simple
  - `Die` will have a method called `roll()`, which simulates a die's roll by setting an `int` field to a random integer from the set $\{1, 2, 3, 4, 5, 6\}$
  - an accessor method, `getNumDots()`, returns the value of the local variable holding the result of `roll()`

Modeling a Real-World Game

- The idea of the `Die` class is that it should be relatively simple
  - `Die` will have a method called `roll()`, which simulates a die's roll by setting an `int` field to a random integer from the set $\{1, 2, 3, 4, 5, 6\}$
  - an accessor method, `getNumDots()`, returns the value of the local variable holding the result of `roll()`
- Based on what you saw earlier from the finished product, a `Die` object is apparently more complicated: It's animated!

Modeling a Real-World Game

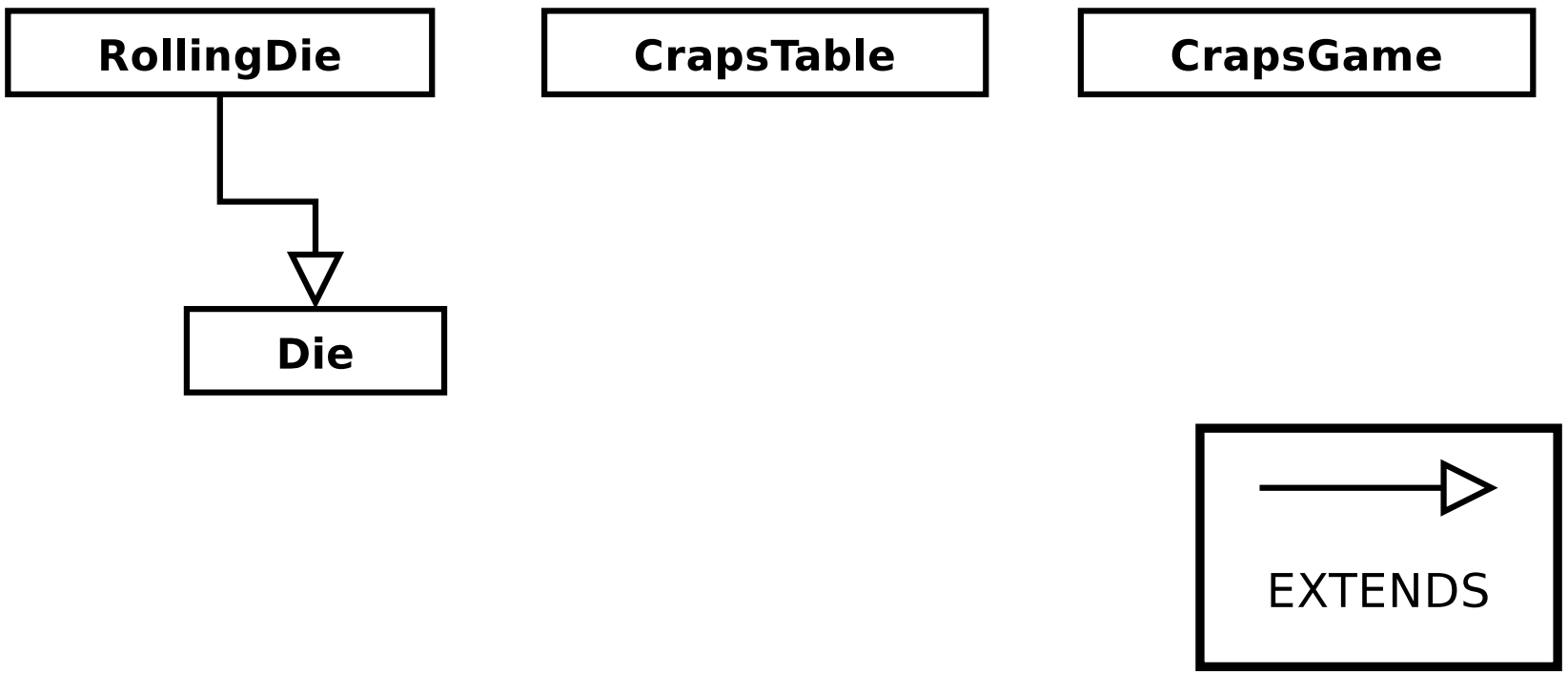
- The idea of the `Die` class is that it should be relatively simple
  - `Die` will have a method called `roll()`, which simulates a die's roll by setting an `int` field to a random integer from the set $\{1, 2, 3, 4, 5, 6\}$
  - an accessor method, `getNumDots()`, returns the value of the local variable holding the result of `roll()`
- Based on what you saw earlier from the finished product, a `Die` object is apparently more complicated: It's animated!
  - The class `RollingDie` will be a subclass of `Die`, extending its functionality by adding animation

Modeling a Real-World Game

- The idea of the `Die` class is that it should be relatively simple
  - `Die` will have a method called `roll()`, which simulates a die's roll by setting an `int` field to a random integer from the set $\{1, 2, 3, 4, 5, 6\}$
  - an accessor method, `getNumDots()`, returns the value of the local variable holding the result of `roll()`
- Based on what you saw earlier from the finished product, a `Die` object is apparently more complicated: It's animated!
  - The class `RollingDie` will be a subclass of `Die`, extending its functionality by adding animation
  - Maria Litvin will write `RollingDie`; you just need to make sure that the simpler class, `Die`, is working correctly

Modeling a Real-World Game

Updated class diagram showing the classes that **model** the game:



Viewing the Model

- What you see in the UI is the view

Viewing the Model

- What you see in the UI is the view
- In *Craps*, the view shows

Viewing the Model

- What you see in the UI is the view
- In *Craps*, the view shows
  - # games won

Viewing the Model

- What you see in the UI is the view
- In *Craps*, the view shows
  - # games won
  - # games lost

Viewing the Model

- What you see in the UI is the view
- In *Craps*, the view shows
  - # games won
  - # games lost
  - the "point" value

Viewing the Model

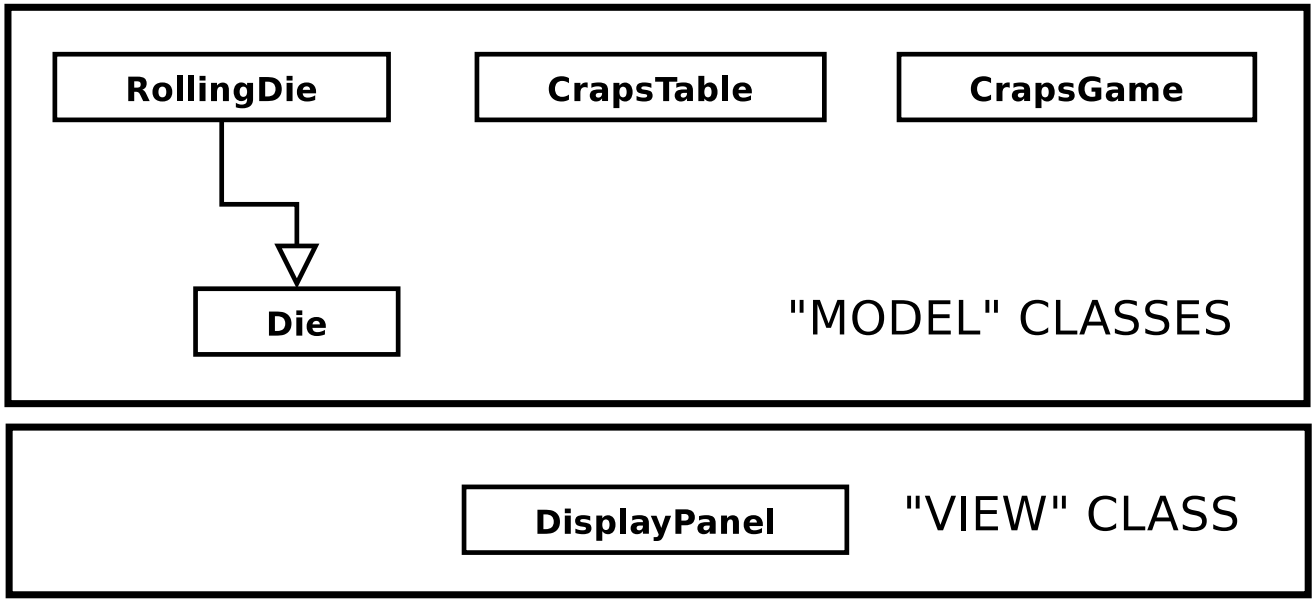
- What you see in the UI is the view
- In *Craps*, the view shows
  - # games won
  - # games lost
  - the "point" value
- Think of the *view* in this game as the "scoreboard"

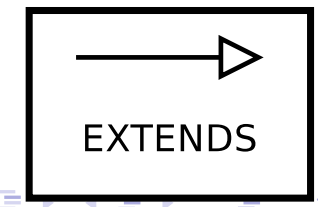
Viewing the Model

- What you see in the UI is the view
- In *Craps*, the view shows
  - # games won
  - # games lost
  - the "point" value
- Think of the *view* in this game as the "scoreboard"
- Litvin decided to have the *view* part of this app be in a class called `DisplayPanel`

Model and View Classes

Updated class diagram showing the **model** and **view** classes:





What's Left: The Controller

- By now, we've modeled the physical elements of the game and its rules — *model classes*

What's Left: The Controller

- By now, we've modeled the physical elements of the game and its rules — *model classes*
- The *view* class lets us see the game's state at any given time (again, think of this as the scoreboard)

What's Left: The Controller

- By now, we've modeled the physical elements of the game and its rules — *model classes*
- The *view* class lets us see the game's state at any given time (again, think of this as the scoreboard)
- What's missing? Give the user a way to *control* the game!

What's Left: The Controller

- By now, we've modeled the physical elements of the game and its rules — *model classes*
- The *view* class lets us see the game's state at any given time (again, think of this as the scoreboard)
- What's missing? Give the user a way to *control* the game!
- *Controller* class: `ControlPanel`

What's Left: The Controller

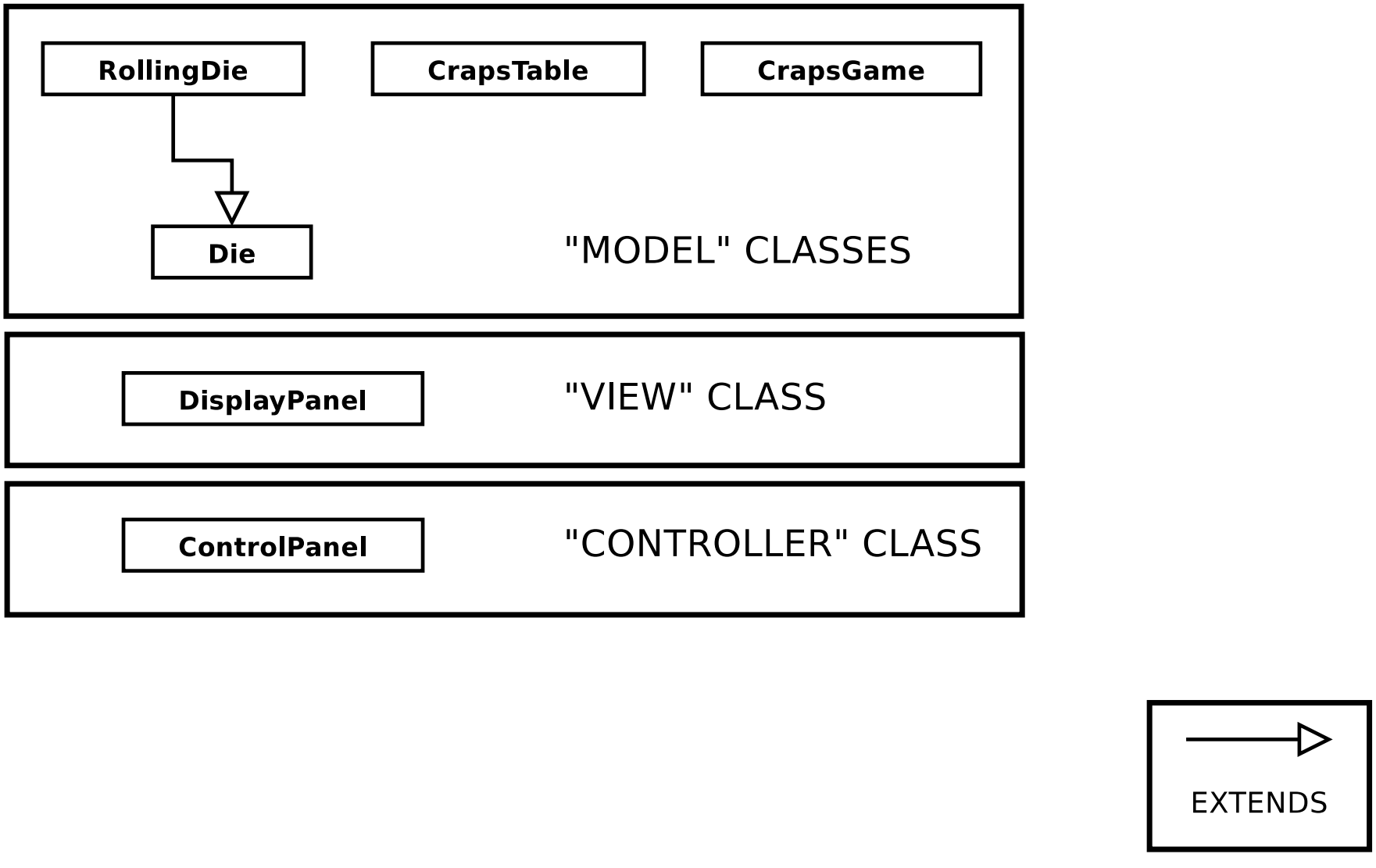
- By now, we've modeled the physical elements of the game and its rules — *model classes*
- The *view* class lets us see the game's state at any given time (again, think of this as the scoreboard)
- What's missing? Give the user a way to *control* the game!
- *Controller* class: `ControlPanel`
  - Give the user a single button, `Roll`, to play the game

What's Left: The Controller

- By now, we've modeled the physical elements of the game and its rules — *model classes*
- The *view* class lets us see the game's state at any given time (again, think of this as the scoreboard)
- What's missing? Give the user a way to *control* the game!
- *Controller* class: `ControlPanel`
  - Give the user a single button, `Roll`, to play the game
  - What is the **role** of this button? Think about how the game would operate in its absence...

Model, View, & Controller Classes

Updated class diagram showing the **model**, **view**, and **controller** classes:



Your First Design Pattern: MVC

- A common GUI application *design pattern* is called **Model-View-Controller**, or **MVC** for short

Your First Design Pattern: MVC

- A common GUI application *design pattern* is called **Model-View-Controller**, or **MVC** for short
- Design patterns are common ways to accomplish certain kinds of tasks

Your First Design Pattern: MVC

- A common GUI application *design pattern* is called **Model-View-Controller**, or **MVC** for short
- Design patterns are common ways to accomplish certain kinds of tasks
- MVC happens to be a convenient way to design certain kinds of GUI applications in an OO language like Java

Your First Design Pattern: MVC

- A common GUI application *design pattern* is called **Model-View-Controller**, or **MVC** for short
- Design patterns are common ways to accomplish certain kinds of tasks
- MVC happens to be a convenient way to design certain kinds of GUI applications in an OO language like Java
- *Design Patterns* are discussed in Litvin Ch. 27 (available online only, see it here)

How MVC Works

- Element(s) in **controller** send message(s) to **model**.

How MVC Works

- Element(s) in **controller** send message(s) to **model**.

  Ex: `ControlPanel`: "I will tell `CrapsTable` to `rollDice()`."

How MVC Works

- Element(s) in **controller** send message(s) to **model**.

  Ex: `ControlPanel`: "I will tell `CrapsTable` to `rollDice()`."

- **Model** takes that information, makes the necessary changes to its representation of the real-world situation, and then tells **view** to update itself.

How MVC Works

- Element(s) in **controller** send message(s) to **model**.

  Ex: `ControlPanel`: "I will tell `CrapsTable` to `rollDice()`."

- **Model** takes that information, makes the necessary changes to its representation of the real-world situation, and then tells **view** to update itself.

  Ex: `CrapsTable`: "OK, let's roll! I will take my two `RollingDies` and tell them to `roll()`. Next, I'll start a clock ticking and let the dice animation run. After the clock is up, I'll ask the `CrapsGame` class to figure out what the results mean by calling `processRoll()`. Now it's time to ask `DisplayPanel`, the **view**, to update the scoreboard."

How MVC Works, cont'd

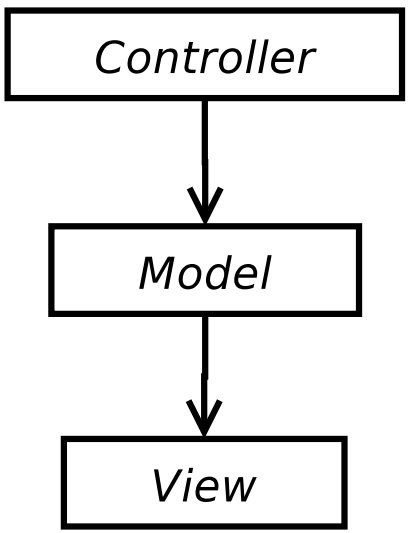
- **View** gets the message that it's time to update itself.

  Ex: `DisplayPanel`: "`CrapsTable` just sent me a result (win/lose/keep rolling) and a point value. I'll update the variables I have that keep track of these items. Next, it's time to `repaint()` myself!"

How MVC Works, cont'd

- **View** gets the message that it's time to update itself.

  Ex: `DisplayPanel`: "`CrapsTable` just sent me a result (win/lose/keep rolling) and a point value. I'll update the variables I have that keep track of these items. Next, it's time to `repaint()` myself!"



All *Craps* Classes

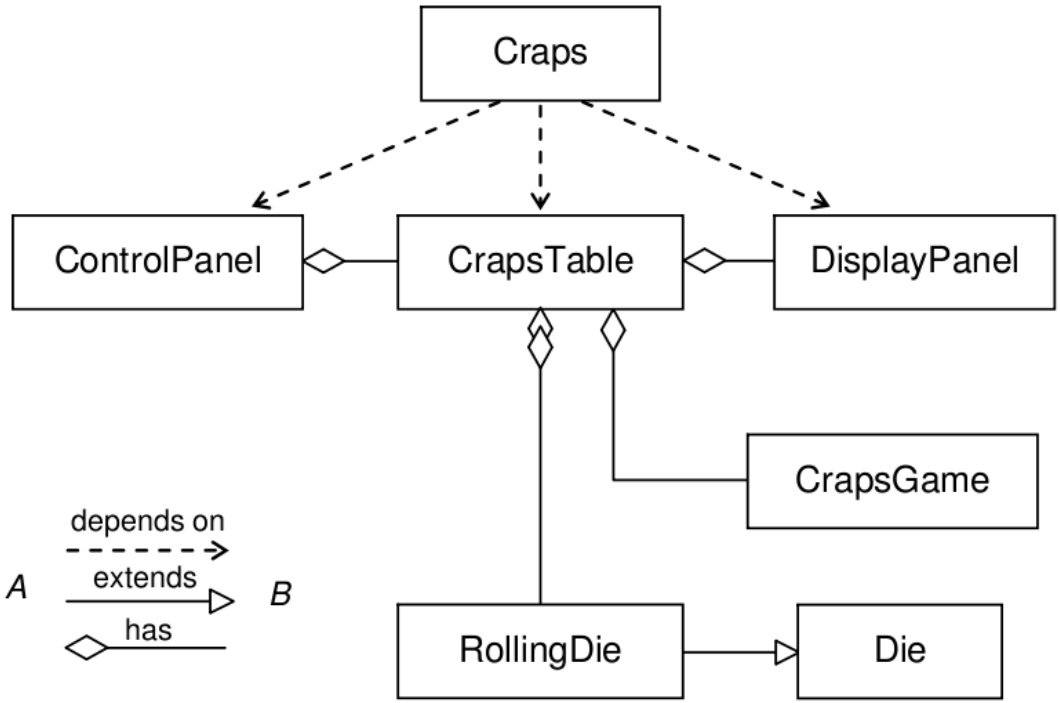


Figure 7-4. ***Craps*** **classes and their relationships**

From Litvin §6.9

Who's Responsible for What?

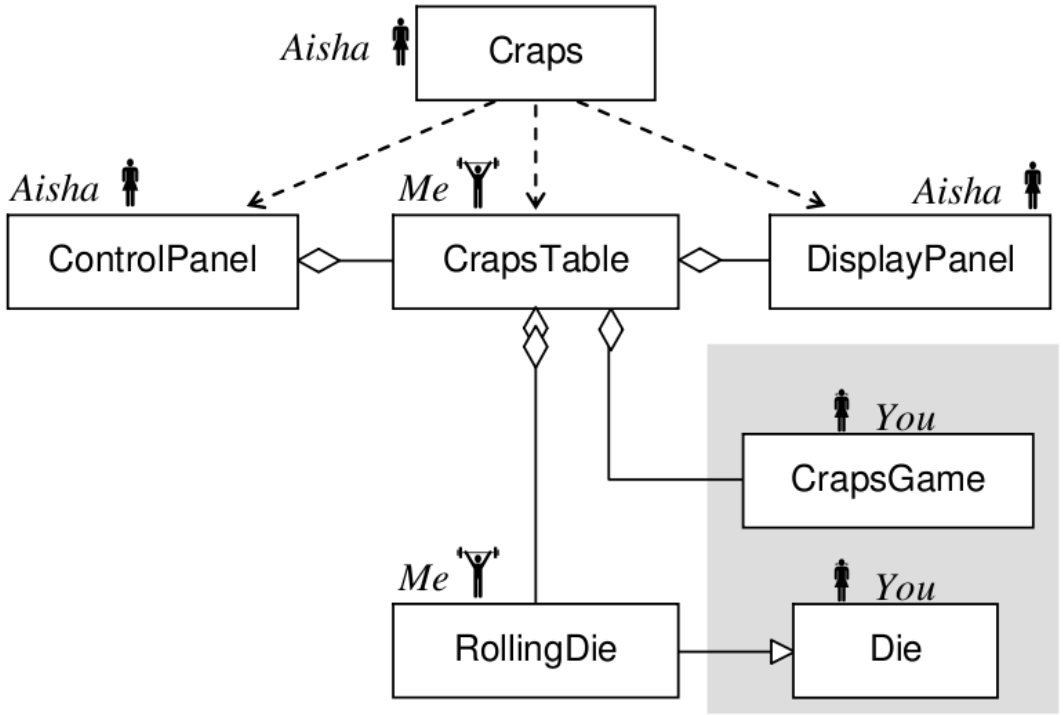


Figure 7-5. Task assignments in the *Craps* project

From Litvin §6.9

Next. . .

- You've gotten a high-level overview of the architecture of *Craps*
- Begin working on §5 of PS #6
  - Read both the problem set AND the textbook carefully!
  - You'll have the rest of today and next class
- Next class: How to generate a random number for `roll()`

HW

Make sure you spend sufficient time working on §5 of PS #6!