Last of Maze Game

CS151

Chris Pollett

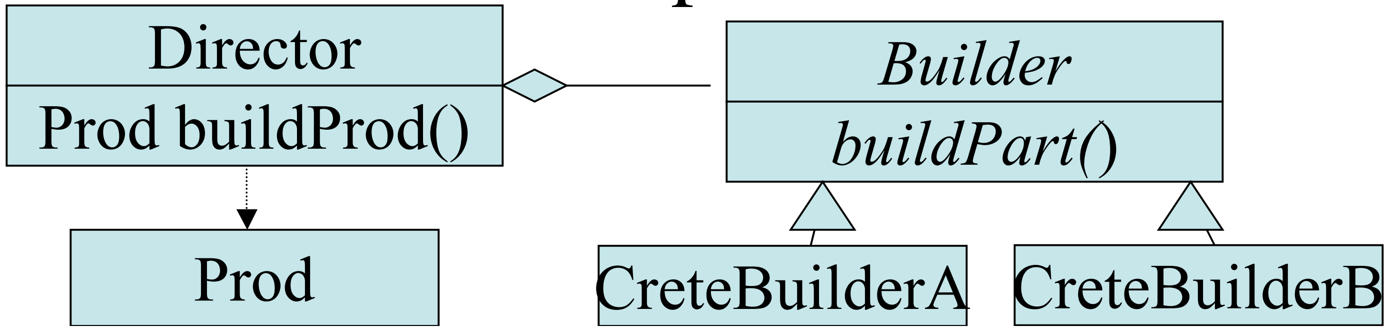
Nov. 21, 2005.

Outline

- Builder
- Command Pattern
- Adapter Pattern
- Model View Controller

Builder Design Pattern

- Last day we talked about how to do themes in our maze game.
- We had two methods createMaze and createLargeMaze which were quite long and were repetitive.
- We can use a Builder pattern to reduce the amount of code duplication



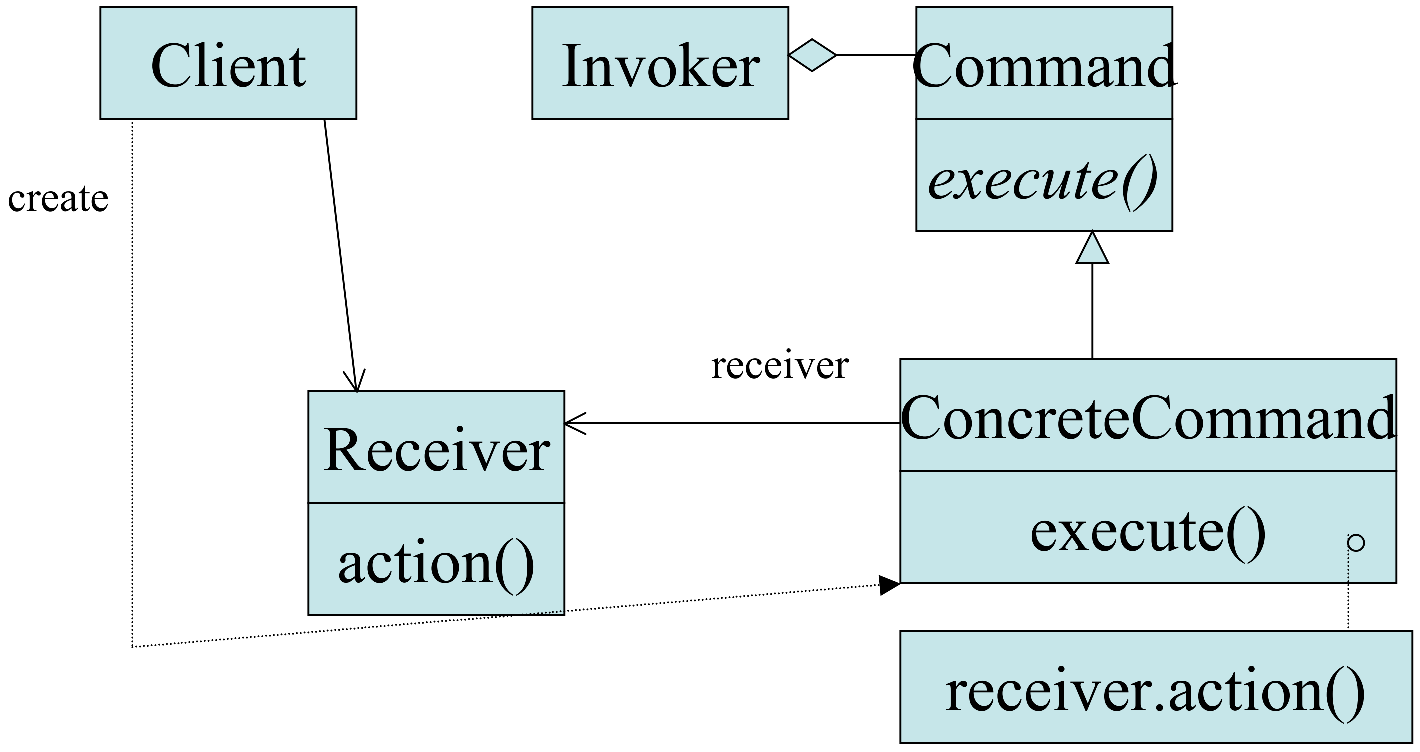
Builder in Maze Game

- In the maze game the Director of the previous slide is given by MazeGameBuilder and Prod is the Maze.
- The class MazeBuilder corresponds to Builder and it is an interface with methods newMaze(), getMaze(), buildRoom(int roomNumber) and buildDoor(int roomNumber, roomNumber2, dir, open).
- This is subclassed into SimpleMazeBuilder and FactoryMazeBuilder.

Intro to the Command Pattern

- Beyond the layout of the maze we also want to support different player actions.
- For instance, move left, right, up, and down. Also, we might want to support undoing moves.
- One object-oriented way of supporting undoing of actions is to use the command pattern.

The Command Pattern



More Command Pattern

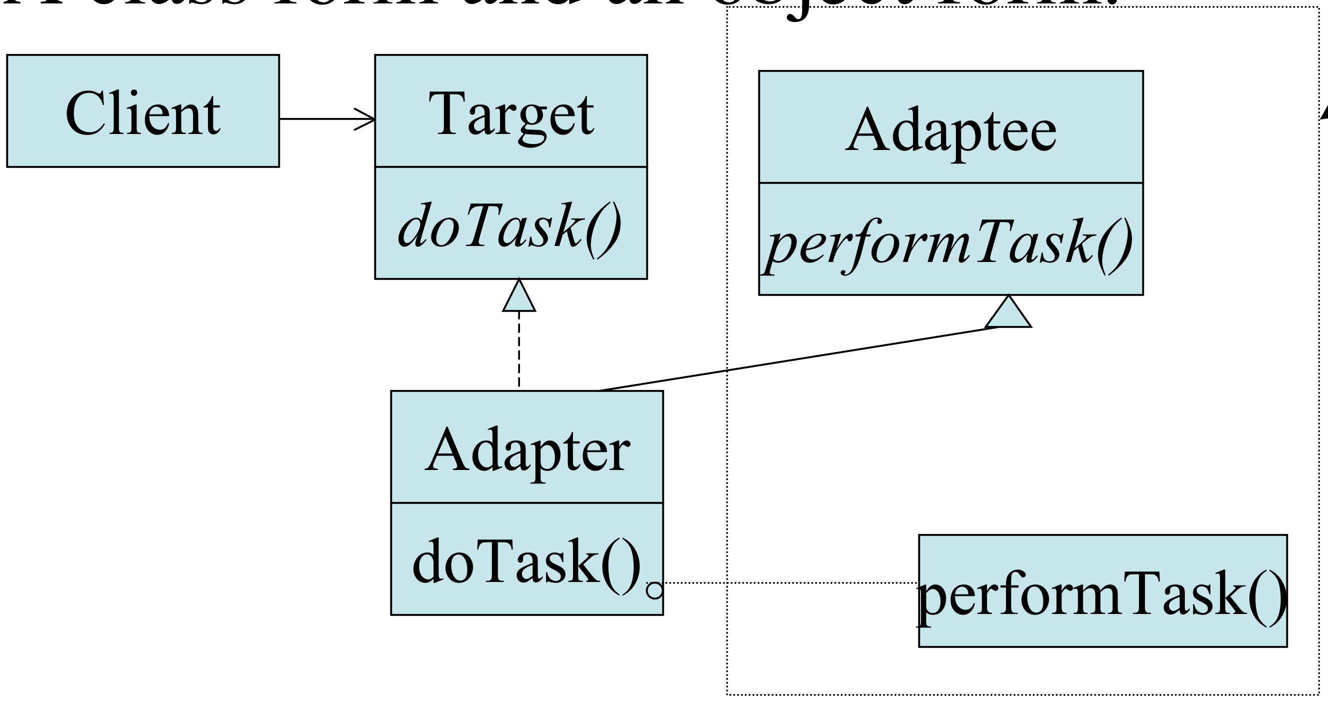
- Command -- is the interface from the book Command (has execute) or its its subinterface UndoableCommand (also has undo) -- the latter defines the interface to perform an undo action
- Receiver -- in this case Maze -- knows how to perform actions
- ConcreteCommand -- MazeMoveCommand -- implements Command interface so has an execute method, delegates execution to the receiver.
- Client -- Maze.MazeKeyListener -- creates concrete commands and binds the commands to their receivers.
- Invoker (Maze) which asks the command to carry out their actions.

Intro to the Adapter Pattern

- Suppose we have a nice reusable component with interface and we have a client which could potentially make use of this component if the interface corresponded to one that the client expected.
- Then we’d be in a situation where we could use the Adapter pattern…

The Adapter Pattern

- There are two forms of the adapter pattern: A class form and an object form.

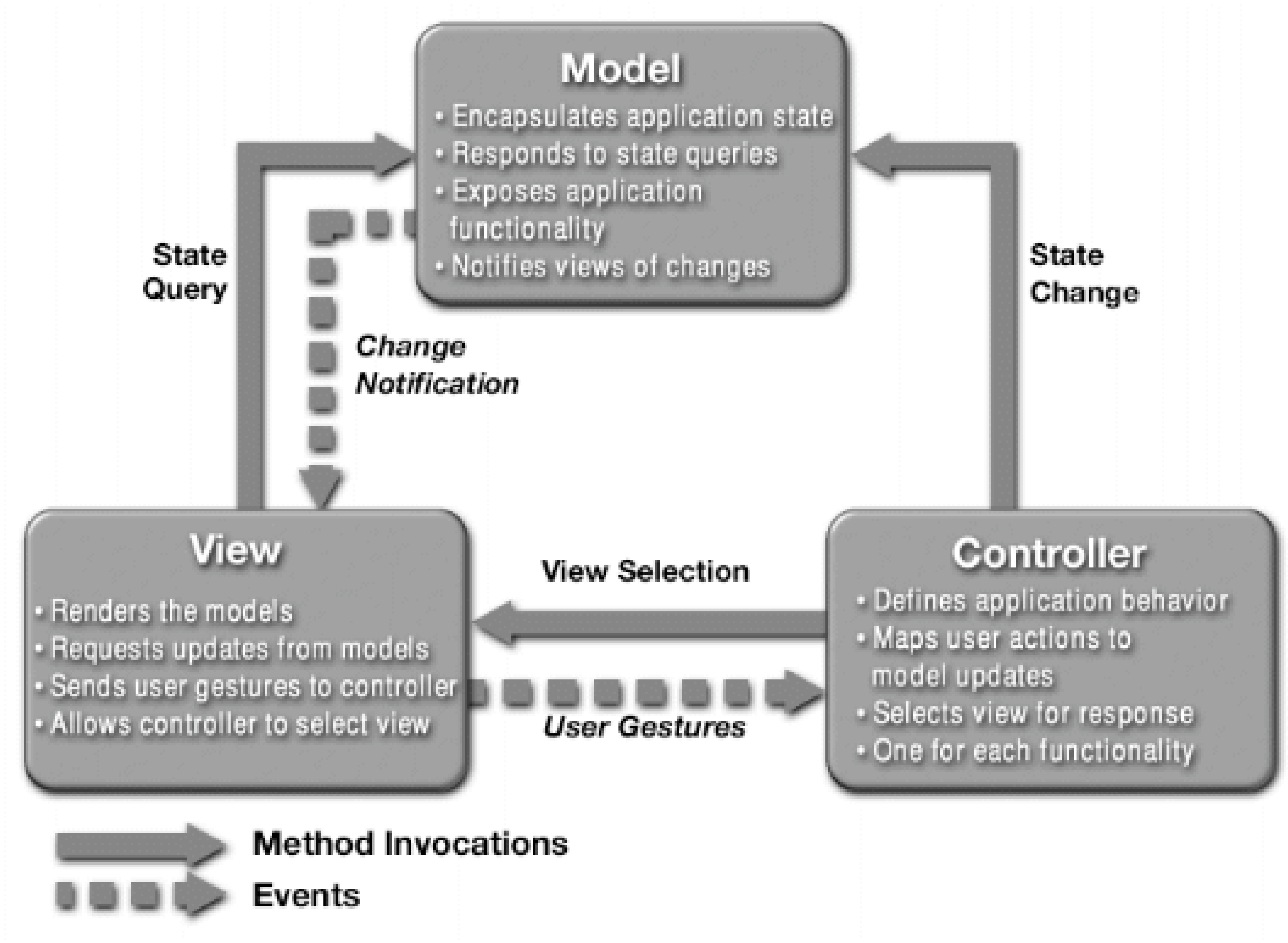


This is the class version. The object version has Adapter which has an Adaptee and delegates the method call.

Example Adapter Pattern from book

- Target -- TableEntry in a table of students program from book. Has interface getColumnCount, getColumnName, getColumnTip, getColumnClass, getColumnComparator, getColumnWidth
- Client -- Table which makes use of TableEntry’s
- Adaptee -- Student a class with student info to be reused.
- Adapter -- StudentEntry, StudentEntry which adapts the interface of Student so that it can be used as a TableEntry

Model View Controller Pattern


Model
• Encapsulates application state
• Responds to state queries
• Exposes application functionality
• Notifies views of changes
State Query
Change Notification
State Change
View
• Renders the models
• Requests updates from models
• Sends user gestures to controller
• Allows controller to select view
View Selection
User Gestures
Controller
• Defines application behavior
• Maps user actions to model updates
• Selects view for response
• One for each functionality
Method Invocations
Events