CSC 222: Object-Oriented Programming

Fall 2017

Object-oriented design, revisited

- highly cohesive, loosely coupled
- HW6: Hunt the Wumpus
- enumerated types
- MVC pattern

OO design principles

recall from earlier:

- the object-oriented approach focuses on identifying the entities/objects that make up a problem solution, then build software models
- want code to be modular, so that it can be developed & tested independently
- also, want to be able to reuse useful code

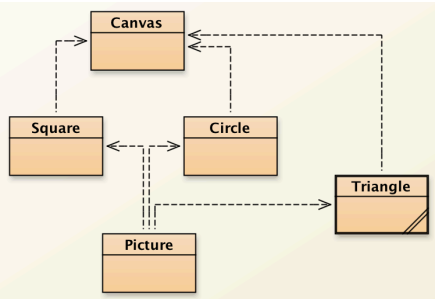
in a highly cohesive system:

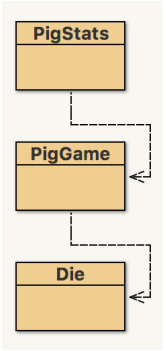
- each class maps to a single, well-defined entity – encapsulating all of its internal state and external behaviors
- each method of the class maps to a single, well-defined behavior
- h*ighly cohesive code is easier to read and reuse*

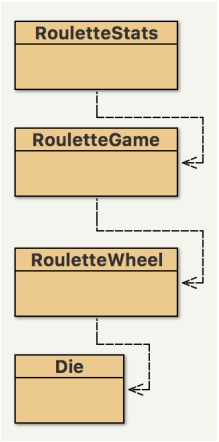
in a loosely coupled system:

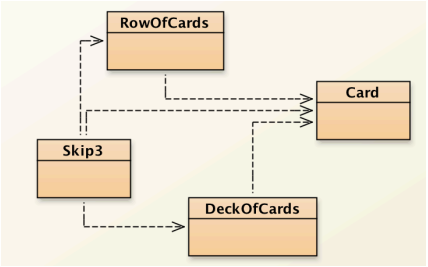
- each class is largely independent and communicates with other classes via a small, well-defined interface
- *loosely coupled code is easier to develop and modify*

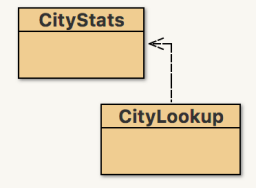
Previous examples











3

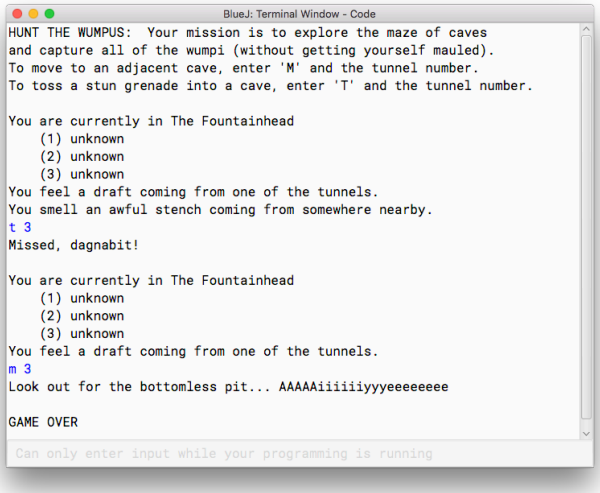
HW6: Hunt the Wumpus

you will implement a variant of one of the first text-based video games

- written in BASIC by Gregory Yob in 1972
- later ported to various PC's, (e.g., Commodore, TI) and UNIX
- named by Time Magazine as one of the All-Time 100 Video Games

4

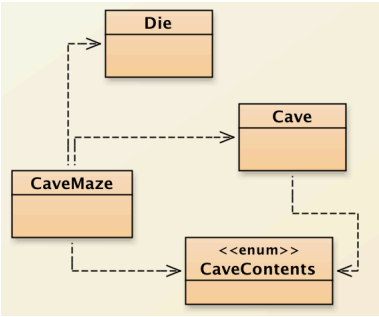
Game rules



- player explores a maze of caves, with each cave connected to 1-4 others
- randomly placed wumpi (1-3), bottomless pit (1) and bat swarm (1)
- player can sense when an obstacle is adjacent
- player can move or throw a stun grenade through a tunnel, wumpi move when hear an explosion

goal: avoid obstacles and capture all of the wumpi before they maul you!

Hunt the Wumpus design



Cave:

- contains all of the information about a given cave, including its contents

CaveContents:

- special type of class for specifying the possible contents of a cave

CaveMaze:

- models the maze of caves
- the caves stored in an ArrayList, linked together
- utilizes the Die class in order to select random locations in the maze

Cave class

you must implement a class that models a single cave

- each cave has a name & number, and is connected to other caves via tunnels
- by default, caves are empty & unvisited (although these can be updated)

how do we represent the cave contents?

- we could store the contents as a string: "EMPTY", "WUMPUS", "BATS", "PIT"

```
Cave c = new Cave("Cavern of Doom", 0, adjList);
c.setContents("WUMPUS");
```

- potential problems?

there are only 4 possible values for cave contents

- the trouble with using a String to represent these is the lack of error checking

```
c.setContents("WUMPIS");    // perfectly legal, but ???
```

7

Enumerated types

there is a better alternative for when there is a small, fixed number of values

- an enumerated type is a new type (class) whose value are explicitly enumerated

```
public enum CaveContents {
    EMPTY, WUMPUS, PIT, BATS
}
```

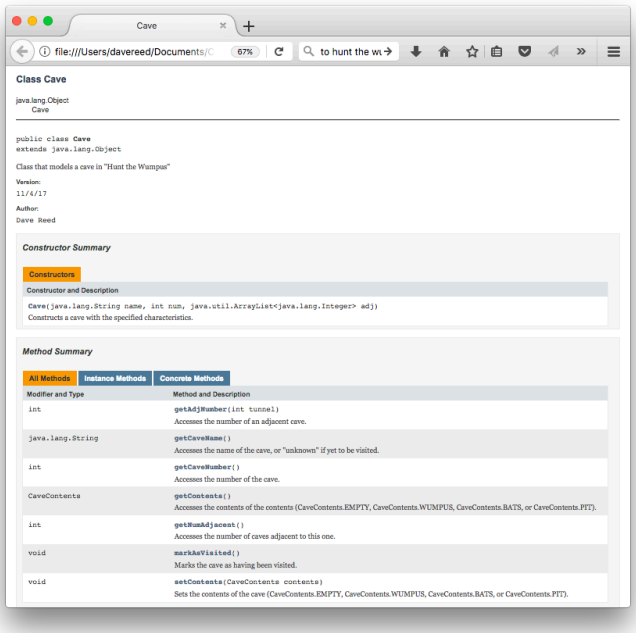
- note that these values are NOT Strings – they do not have quotes
- you specify a enumerated type value by ENUMTYPE.VALUE

```
c.setContents(CaveContents.WUMPUS);
```

since an enumerated type has a fixed number of values, any invalid input would be caught by the compiler

8

Cave javadoc



be sure your class follows the javadoc specifications

- will need to decide what fields are needed (and only those fields!)
- cave contents are defined by `CaveContents` enumerated type
- `getCaveName` will return "unknown" if that cave is unvisited, otherwise return the cave's name

9

CaveMaze

the CaveMaze class reads in & stores a maze of caves

- provided version uses an ArrayList (but could have used an array)
- the caves and their connections are defined in a file: `caves.txt`
- cave 0 is assumed to be the start cave

```
20
 0  3  1  4  9     The Fountainhead
 1  3  0  2  8     The Rumpus Room
 2  2  1  6        Buford's Folly
 3  3  7  4  5     The Hall of Kings
 4  4 12  0  8  3  The Silver Mirror
 5  4  8  3  9 11  The Gallimaufry
 6  4 14 13  2  7  The Den of Iniquity
 7  3  3  6 19     The Findledelve
 8  3  5  1  4     The Page of the Deniers
 9  4  0 10 12  5  The Final Tally
10  2  9 15        Ess four
11  3 14 16 5      The Trillion
12  2  4  9        The Scrofula
13  4  6 16  8 14  Ephemeron
14  3  6 13 11     Shelob's Lair
15  3 10 17 16     The Lost Caverns of the Wyrm
16  3 17 15 19     The Lost Caverns of the Wyrm
17  3 16 17 18     The Lost Caverns of the Wyrm
18  3 19 18 17     The Lost Caverns of the Wyrm
19  3 17 18 19     The Lost Caverns of the Wyrm
```

```
public class CaveMaze {
  private Cave currentCave;
  private ArrayList<Cave> caves;

  public CaveMaze(String filename) throws java.io.FileNotFoundException {
      Scanner infile = new Scanner(new File(filename));

      int numCaves = infile.nextInt();
      this.caves = new ArrayList<Cave>();
      for (int i = 0; i < numCaves; i++) {
          this.caves.add(null);
        }

      for (int i = 0; i < numCaves; i++) {
          int num = infile.nextInt();
          int numAdj = infile.nextInt();
          ArrayList<Integer> adj = new ArrayList<Integer>();
          for (int a = 0; a < numAdj; a++) {
              adj.add(infile.nextInt());
          }
          String name = infile.nextLine().trim();
          this.caves.set(num, new Cave(name, num, adj));
      }

      this.currentCave = this.caves.get(0);
      this.currentCave.markAsVisited();
  }
  . . .
```

10

CaveMaze (cont.)

currently,

- can move between caves
- only see the names of caves you have already visited

- you must add the full functionality of the game (incl. adding & reacting to dangers, winning/losing)

```
  . . .

  public String move(int tunnel) {
    if (tunnel < 1 || tunnel > this.currentCave.getNumAdjacent()) {
      return "There is no tunnel number " + tunnel;
    }

    int caveNum = this.currentCave.getAdjNumber(tunnel);
    this.currentCave = this.caves.get(caveNum);
    this.currentCave.markAsVisited();

    return "Moving down tunnel " + tunnel + "...";
  }

  public String showLocation() {
    String message = "You are currently in " +
                     this.currentCave.getCaveName();

    for (int i = 1; i <= this.currentCave.getNumAdjacent(); i++) {
      int caveNum = this.currentCave.getAdjNumber(i);
      Cave adjCave = this.caves.get(caveNum);
      message += "\n    (" + i + ") " + adjCave.getCaveName();
    }
    return message;
  }

  . . .

}
```

11

User Interface

using BlueJ, we have been able to manipulate objects directly

- create an object by right-clicking on the class icon (& providing inputs if necessary)
- call a method by right-clicking on the object icon (& providing inputs if necessary)

for an interactive application like a game, you want a class to automate the top-level control

- convention is to have a "driver" class with a static "main" method
  (main is automatically called using other development environments)

- in this case, the main method
  1. creates the `CaveMaze`
  2. loops to get each player action (e.g., move or toss)
  3. calls the `CaveMaze` method associated with that action
  4. displays the result of the action

12

Terminal driver

```
public class WumpusTerminal {

    public static void main(String[] args) throws java.io.FileNotFoundException {
        CaveMaze maze = new CaveMaze("caves.txt");

        System.out.println("HUNT THE WUMPUS:  Your mission is to explore the maze of caves");
        System.out.println("and capture all of the wumpi (without getting yourself mauled).");
        System.out.println("To move to an adjacent cave, enter 'M' and the tunnel number.");
        System.out.println("To toss a stun grenade into a cave, enter 'T' and the tunnel number.");

        Scanner input = new Scanner(System.in);
        while (maze.stillAble() && maze.stillWumpi()) {
            System.out.println("\n"+maze.showLocation());

            try {
                String action = input.next();
                if (action.toLowerCase().charAt(0) == 'q') {
                    System.out.println("Nobody likes a quitter.");
                    break;
                }
                if (action.toLowerCase().charAt(0) == 't') {
                    System.out.println(maze.toss(input.nextInt()));
                } else if (action.toLowerCase().charAt(0) == 'm') {
                    System.out.println(maze.move(input.nextInt()));
                } else {
                    System.out.println("Unrecognized command -- please try again.");
                }
            }
            catch (java.util.InputMismatchException e) {
                System.out.println("Unrecognized command -- please try again.");
            }
        }
        System.out.println("\nGAME OVER");
    }
}
```

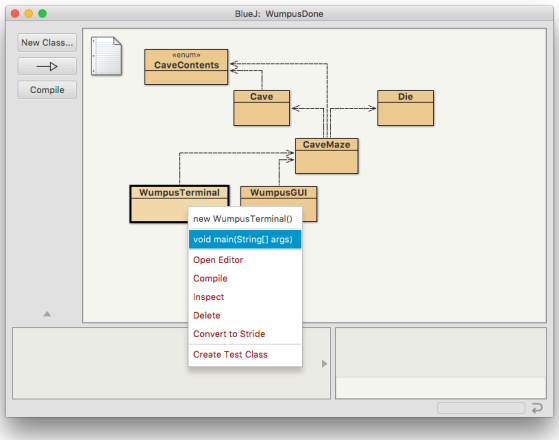
13

Calling main

since main us a static method, call the method on the class

- e.g., call main on WumpusTerminal for terminal-based interface

in other IDEs (e.g., NetBeans, Eclipse), main will automatically be executed if present



14

MVC pattern

model-view-controller is a software pattern used to develop reusable, modular software

- goal: isolate the application-specific logic from the user interface
- allows for independent testing & development, easy updates

consider past example:

- `CityStats` received inputs via constructor parameters, methods returned values
  → could use this class with any view & controlled
  e.g., driver that read info from a file or keyboard, displayed result in the terminal
  e.g., driver that read from text fields in a window, displayed results in a text area
- `CityLookup` received read city data from a file, displayed results in terminal
  → its behavior is closely tied to those input/output interfaces
  if we wanted to change it work with a GUI, would need to change most methods

tying the program logic to a particular interface is unnecessarily restrictive

15

MVC Wumpus

Hunt the Wumpus is a good example of how the MVC pattern can lead to flexible programs

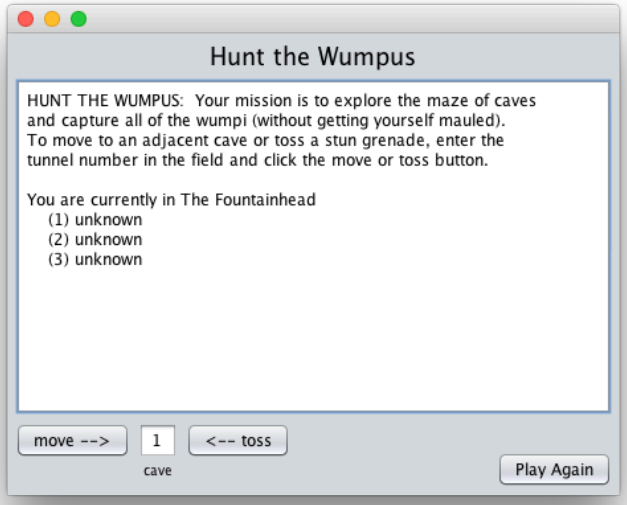
- Model (i.e., the logic): `Die`, `CaveContents`, `Cave`, `CaveMaze`
  none of the classes perform input/output (other than reading the maze data from a file)
  any data that a constructor/method needs is passed as parameters
  any result a method needs to report is returned
- View (i.e., the user interface): the Java terminal window
- Controller (i.e., the connector): `WumpusTerminal`
  input comes from System.in, read in using a Scanner
  output goes to System.out, displayed using System.out.println

by separating the logic from the interface, it makes it possible to take the same Model and connect it to a different View (using a new Controller)

- View: Graphical User Interface (window with text area, buttons, text field, …)
- Controller: `WumpusGUI`

16

GUI version



for Hunt the Wumpus

- your HW6 classes should work with either `WumpusTerminal` or `WumpusGUI`

17