zuul-even-better

/**
 * This class is an abstract superclass for all command classes in the game.
 * Each user command is implemented by a specific command subclass.
 * Objects of class Command can store an optional argument word (a second
 * word entered on the command line). If the command had only one word,
 * then the second word is <null>.
 */

public abstract class Command {
    private String secondWord;

    /**
     * Create a command object. First and second word
     * must be supplied, but either one (or both) can be null. The command
     * word should be null to indicate that this was a command that is not
     * recognised by this game.
     */
    public Command() {
        secondWord = null;
    }

    /**
     * Return the second word of this command. If no
     * second word was entered, the result is null.
     */
    public String getSecondWord() {
        return secondWord;
    }

    /**
     * Define the second word of this command (the word
     * entered after the command word). Null indicates that
     * there was no second word.
     */
    public void setSecondWord(String secondWord) {
        this.secondWord = secondWord;
    }

    /**
     * Execute this command. A flag is returned indicating whether
     * the game is over as a result of this command.
     */
    public abstract boolean execute(Player player);

    * Check whether a second word was entered for this
    * command.
    */
    public boolean hasSecondWord() {
        return secondWord != null;
    }

    /**
     * Define the second word of this command (the word
     * entered after the command word). Null indicates that
     * there was no second word.
     */
    public void setSecondWord(String secondWord) {
        this.secondWord = secondWord;
    }

    /**
     * Execute this command. A flag is returned indicating whether
     * the game is over as a result of this command.
     */
    public abstract boolean execute(Player player);
import java.util.HashMap;
import java.util.Iterator;

/**
 * This class is the main class of the "World of Zuul"
 * application.
 * "World of Zuul" is a very simple, text based
 * adventure game.
 * This class holds a collection of all command words
 * known to the game.
 * It is used to recognise commands as they are typed
 * in.
 * @author Michael Kolling and David J. Barnes
 * @version 2.0 (December 2002)
 */

public class CommandWords
{
    private HashMap commands;

    /**
     * Constructor - initialise the command words.
     */
    public CommandWords()
    {
        commands = new HashMap();
        commands.put("go", new GoCommand());
        commands.put("help", new HelpCommand(this));
        commands.put("quit", new QuitCommand());
    }

    /**
     * Given a command word, find and return the
     * matching command object.
     * Return null if there is no command with this
     * name.
     */
    public Command get(String word)
    {
        return (Command)commands.get(word);
    }

    public void showAll()
    {
        for(Iterator i = commands.keySet().iterator();
            i.hasNext(); )
        {
            System.out.print(i.next() + " ");
        }
        System.out.println();
    }
}
This class is the main class of the "World of Zuul" application.
"World of Zuul" is a very simple, text based adventure game. Users can walk around some scenery. That's all. It should really be extended to make it more interesting!

To play this game, create an instance of this class and call the "play" method.

This main class creates and initialises all the others: it creates all rooms, creates the parser and starts the game.

@Author Michael Kolling and David J. Barnes
@Version 1.1 (December 2002)

class Game {
    private Parser parser;
    private Player player;

    /**
     * Create the game and initialise its internal map.
     */
    public Game() {
        player = new Player();
        parser = new Parser();
        createRooms();
    }

    /**
     * Create all the rooms and link their exits together.
     */
    private void createRooms() {
        Room outside, theatre, pub, lab, office;
        // create the rooms outside = new Room("outside the main entrance of the university");
        theatre = new Room("in a lecture theatre");
        pub = new Room("in the campus pub");
        lab = new Room("in a computing lab");
        office = new Room("in the computing admin office");

        // initialise room exits
        outside.setExit("east", theatre);
        outside.setExit("south", lab);
        outside.setExit("west", pub);

        theatre.setExit("west", outside);
        pub.setExit("east", outside);
        lab.setExit("north", outside);
        lab.setExit("east", office);
        office.setExit("west", lab);

        // the player starts the game outside
        player.setCurrentRoom(outside);
    }

    /**
     * Main play routine. Loops until end of play.
     */
    public void play() {
        printWelcome();

        // Enter the main command loop. Here we repeatedly read commands and execute them until the game is over.
        boolean finished = false;
        while(!finished) {
            Command command = parser.getCommand();
            if(command == null) {
                System.out.println("I don't understand...");
            } else {
                finished = command.execute(player);
            }
        }
    }
}
System.out.println("Thank you for playing. Good bye.");

/** * Print out the opening message for the player. */
private void printWelcome()
{
    System.out.println();
    System.out.println("Welcome to The World of Zuul!");
    System.out.println("The World of Zuul is a new, incredibly boring adventure game.");
    System.out.println("Type 'help' if you need help.");
    System.out.println();
    System.out.println(player.getCurrentRoom().getLongDescription());
}

/**
 * Implementation of the 'go' user command.
 * @author Michael Kolling
 * @version 1.0 (December 2002)
 */
public class GoCommand extends Command
{
    /**
     * Constructor for objects of class GoCommand
     */
    public GoCommand()
    {
    }

    /**
     * Try to go to one direction. If there is an exit, enter the new
     * room, otherwise print an error message. Returns always 'false'.
     */
    public boolean execute(Player player)
    {
        if(hasSecondWord())
        {
            String direction = getSecondWord();
            player.walk(direction);
        }
        else
        {
            // if there is no second word, we don't know where to go...
            System.out.println("Go where?");
        }
        return false;
    }
}
/**
 * Implementation of the 'help' user command.
 * @author Michael Kolling
 * @version 1.0 (December 2002)
 */
public class HelpCommand extends Command
{
    private CommandWords commandWords;

    /**
     * Constructor for objects of class HelpCommand
     */
    public HelpCommand(CommandWords words)
    {
        commandWords = words;
    }

    /**
     * Print out some help information. Here we print
     * some stupid, * cryptic message and a list of the command words.
     * Returns always false.
     */
    public boolean execute(Player player)
    {
        System.out.println("You are lost. ... at the university.");
        System.out.println();
        System.out.println("Your command words are:");
        commandWords.showAll();
        return false;
    }
}

/**
 * Implementation of the 'quit' user command.
 * @author Michael Kolling
 * @version 1.0 (December 2002)
 */
public class QuitCommand extends Command
{
    public QuitCommand()
    {
    
    }

    /**
     * "Quit" was entered. Check the argument to see
     * whether we really quit the game. Return true, if we should quit, false
     * otherwise.
     */
    public boolean execute(Player player)
    {
        if(getSecondWord() == null) {
            return true;
        }
        else {
            System.out.println("I cannot quit that...");
            return false;
        }  
    }
}
import java.io.BufferedReader; import java.io.InputStreamReader; import java.util.StringTokenizer; 

/*
 * This class is the main class of the "World of Zuul" application.
 * "World of Zuul" is a very simple, text based adventure game.
 * This parser reads user input and tries to interpret it as an "Adventure" command. Every time it is called
 * it reads a line from the terminal and tries to interpret the line as a two word command. It returns
 * the command as an object of class Command. 
 * The parser has a set of known command words. It checks user input against
 * the known commands, and if the input is not one of the known commands, it
 * returns a command object that is marked as an unknown command.
 * @author Michael Kolling and David J. Barnes
 * @version 1.1 (December 2002)
 */

class Parser
{
    private CommandWords commands; // holds all valid command words

    public Parser()
    {
        commands = new CommandWords();
    }

    public Command getCommand()
    {
        String inputLine = ""; // will hold the full input line
        String word1;
        String word2;
        System.out.print("> "); // print prompt

        BufferedReader reader =
            new BufferedReader(new InputStreamReader(System.in));
        try {
            inputLine = reader.readLine();
        } catch(java.io.IOException exc) {
            System.out.println("There was an error during reading: "+ exc.getMessage());
        }

        StringTokenizer tokenizer = new StringTokenizer(inputLine);
        if(tokenizer.hasMoreTokens())
            word1 = tokenizer.nextToken(); // get first word
        else
            word1 = null;
        if(tokenizer.hasMoreTokens())
            word2 = tokenizer.nextToken(); // get second word
        else
            word2 = null;

        Command command = commands.get(word1);
        if(command != null) {
            // Print out a list of valid command words.
            commands.showAll();
            System.out.println("<command>");
            command = commands.get(word1);
        }

        return command;
    }

    /**
     * Print out a list of valid command words.
     */
    public void showCommands()
    {
        commands.showAll();
    }
}
/**
 * This class represents players in the game. Each player has
 * a current location.
 * @author Michael Kolling
 * @version 1.0 (December 2002)
 */
public class Player {
    private Room currentRoom;

    /**
     * Constructor for objects of class Player
     */
    public Player() {
        currentRoom = null;
    }

    /**
     * Return the current room for this player.
     */
    public Room getCurrentRoom() {
        return currentRoom;
    }

    /**
     * Set the current room for this player.
     */
    public void setCurrentRoom(Room room) {
        currentRoom = room;
    }

    /**
     * Try to walk in a given direction. If there is a
doors
     * this will change the player's location.
     */
    public void walk(String direction) {
        // Try to leave current room.
        Room nextRoom = currentRoom.getExit(direction);
        if (nextRoom == null) {
            System.out.println("There is no door!");
        } else {
            setCurrentRoom(nextRoom);
            System.out.println(nextRoom.getLongDescription());
        }
    }
}

import java.util.Set;
import java.util.HashMap;
import java.util.Iterator;

/*
 * Class Room - a room in an adventure game.
 * This class is the main class of the "World of Zuul"
 * application.
 * "World of Zuul" is a very simple, text based
 * adventure game.
 * A "Room" represents one location in the scenery of
 * the game. It is
 * connected to other rooms via exits. For each
 * existing exit, the room
 * stores a reference to the neighboring room.
 */
@Author Michael Kolling and David J. Barnes
@Version 1.0 (February 2002)*/
class Room {
    private String description;
    private HashMap exits; // stores exits of this room.

    /**
     * Create a room described "description". Initially, it has no exits.
     * "description" is something like "in a kitchen" or "in an open court
     * yard".
     */
    public Room(String description) {
        this.description = description;
        exits = new HashMap();
    }

    /**
     * Define an exit from this room.
     */
    public void setExit(String direction, Room neighbor) {
        exits.put(direction, neighbor);
    }

    /**
     * Return the description of the room (the one that was defined in the
     * constructor).
     */
    public String getShortDescription() {
        return description;
    }

    /**
     * Return a long description of this room, in the form:
     * You are in the kitchen.
     * Exits: north west
     */
    public String getLongDescription() {
        return "You are " + description + ";\n" + getExitString();
    }

    /**
     * Return a string describing the room's exits, for example
     * "Exits: north west".
     */
    private String getExitString() {
        String returnString = "Exits:
        for(Iterator iter = keys.iterator(); iter.hasNext(); )
            returnString += " " + iter.next();
        return returnString;
    }

    /**
     * Return the room that is reached if we go from this room in direction
     * "direction". If there is no room in that direction, return null.
     */
    public Room getExit(String direction) {
        return (Room)exits.get(direction);
    }
}