

```
/**
 * 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. It also evaluates and
 * executes the commands that the parser returns.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 2006.03.30
 */

public class Game
{
    private Parser parser;
    private Room currentRoom;

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

    /**
     * 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.setExits(null, theatre, lab, pub);
        theatre.setExits(null, null, null, outside);
        pub.setExits(null, outside, null, null);
        lab.setExits(outside, office, null, null);
        office.setExits(null, null, null, lab);

        currentRoom = outside; // start game 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();
        finished = processCommand(command);
    }
    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("World of Zuul is a new, incredibly boring adventure gam
e.");
    System.out.println("Type 'help' if you need help.");
    System.out.println();
    System.out.println("You are " + currentRoom.getDescription());
    System.out.print("Exits: ");
    if(currentRoom.northExit != null)
        System.out.print("north ");
    if(currentRoom.eastExit != null)
        System.out.print("east ");
    if(currentRoom.southExit != null)
        System.out.print("south ");
    if(currentRoom.westExit != null)
        System.out.print("west ");
    System.out.println();
}

/**
 * Given a command, process (that is: execute) the command.
 * @param command The command to be processed.
 * @return true If the command ends the game, false otherwise.
 */
private boolean processCommand(Command command)
{
    boolean wantToQuit = false;

    if(command.isUnknown()) {
        System.out.println("I don't know what you mean...");
        return false;
    }

    String commandWord = command.getCommandWord();
    if (commandWord.equals("help"))
        printHelp();
    else if (commandWord.equals("go"))
        goRoom(command);
}
```

```
        else if (commandWord.equals("quit"))
            wantToQuit = quit(command);

        return wantToQuit;
    }

    // implementations of user commands:

    /**
     * Print out some help information.
     * Here we print some stupid, cryptic message and a list of the
     * command words.
     */
    private void printHelp()
    {
        System.out.println("You are lost. You are alone. You wander");
        System.out.println("around at the university.");
        System.out.println();
        System.out.println("Your command words are:");
        System.out.println("    go quit help");
    }

    /**
     * Try to go to one direction. If there is an exit, enter
     * the new room, otherwise print an error message.
     */
    private void goRoom(Command command)
    {
        if(!command.hasSecondWord()) {
            // if there is no second word, we don't know where to go...
            System.out.println("Go where?");
            return;
        }

        String direction = command.getSecondWord();

        // Try to leave current room.
        Room nextRoom = null;
        if(direction.equals("north")) {
            nextRoom = currentRoom.northExit;
        }
        if(direction.equals("east")) {
            nextRoom = currentRoom.eastExit;
        }
        if(direction.equals("south")) {
            nextRoom = currentRoom.southExit;
        }
        if(direction.equals("west")) {
            nextRoom = currentRoom.westExit;
        }

        if (nextRoom == null) {
            System.out.println("There is no door!");
        }
        else {
            currentRoom = nextRoom;
            System.out.println("You are " + currentRoom.getDescription());
            System.out.print("Exits: ");
        }
    }
}
```

```
        if(currentRoom.northExit != null)
            System.out.print("north ");
        if(currentRoom.eastExit != null)
            System.out.print("east ");
        if(currentRoom.southExit != null)
            System.out.print("south ");
        if(currentRoom.westExit != null)
            System.out.print("west ");
        System.out.println();
    }
}

/**
 * "Quit" was entered. Check the rest of the command to see
 * whether we really quit the game.
 * @return true, if this command quits the game, false otherwise.
 */
private boolean quit(Command command)
{
    if(command.hasSecondWord()) {
        System.out.println("Quit what?");
        return false;
    }
    else {
        return true; // signal that we want to quit
    }
}
}
```

```
/**
 * Class Room - a room in an adventure game.
 *
 * This class is part 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. The exits are labelled north,
 * east, south, west. For each direction, the room stores a reference
 * to the neighboring room, or null if there is no exit in that direction.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 2006.03.30
 */
public class Room
{
    public String description;
    public Room northExit;
    public Room southExit;
    public Room eastExit;
    public Room westExit;

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

    /**
     * Define the exits of this room. Every direction either leads
     * to another room or is null (no exit there).
     * @param north The north exit.
     * @param east The east exit.
     * @param south The south exit.
     * @param west The west exit.
     */
    public void setExits(Room north, Room east, Room south, Room west)
    {
        if(north != null)
            northExit = north;
        if(east != null)
            eastExit = east;
        if(south != null)
            southExit = south;
        if(west != null)
            westExit = west;
    }

    /**
     * @return The description of the room.
     */
    public String getDescription()
    {
```

```
        return description;  
    }  
}
```

```
import java.util.Scanner;
import java.util.StringTokenizer;

/**
 * This class is part 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 2006.03.30
 */
public class Parser
{
    private CommandWords commands; // holds all valid command words
    private Scanner reader;        // source of command input

    /**
     * Create a parser to read from the terminal window.
     */
    public Parser()
    {
        commands = new CommandWords();
        reader = new Scanner(System.in);
    }

    /**
     * @return The next command from the user.
     */
    public Command getCommand()
    {
        String inputLine; // will hold the full input line
        String word1 = null;
        String word2 = null;

        System.out.print("> "); // print prompt

        inputLine = reader.nextLine();

        // Find up to two words on the line.
        Scanner tokenizer = new Scanner(inputLine);
        if(tokenizer.hasNext()) {
            word1 = tokenizer.next(); // get first word
            if(tokenizer.hasNext()) {
                word2 = tokenizer.next(); // get second word
                // note: we just ignore the rest of the input line.
            }
        }

        // Now check whether this word is known. If so, create a command
        // with it. If not, create a "null" command (for unknown command).
    }
}
```

```
        if(commands.isCommand(word1)) {
            return new Command(word1, word2);
        }
        else {
            return new Command(null, word2);
        }
    }
}
```

```
/**
 * This class is part of the "World of Zuul" application.
 * "World of Zuul" is a very simple, text based adventure game.
 *
 * This class holds information about a command that was issued by the user.
 * A command currently consists of two strings: a command word and a second
 * word (for example, if the command was "take map", then the two strings
 * obviously are "take" and "map").
 *
 * The way this is used is: Commands are already checked for being valid
 * command words. If the user entered an invalid command (a word that is not
 * known) then the command word is <null>.
 *
 * If the command had only one word, then the second word is <null>.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 2006.03.30
 */
public class Command
{
    private String commandWord;
    private String secondWord;

    /**
     * Create a command object. First and second word must be supplied, but
     * either one (or both) can be null.
     * @param firstWord The first word of the command. Null if the command
     *                 was not recognised.
     * @param secondWord The second word of the command.
     */
    public Command(String firstWord, String secondWord)
    {
        commandWord = firstWord;
        this.secondWord = secondWord;
    }

    /**
     * Return the command word (the first word) of this command. If the
     * command was not understood, the result is null.
     * @return The command word.
     */
    public String getCommandWord()
    {
        return commandWord;
    }

    /**
     * @return The second word of this command. Returns null if there was no
     * second word.
     */
    public String getSecondWord()
    {
        return secondWord;
    }

    /**
     * @return true if this command was not understood.
     */
}
```

```
    */
    public boolean isUnknown()
    {
        return (commandWord == null);
    }

    /**
     * @return true if the command has a second word.
     */
    public boolean hasSecondWord()
    {
        return (secondWord != null);
    }
}
```

```
/**
 * This class is part of the "World of Zuul" application.
 * "World of Zuul" is a very simple, text based adventure game.
 *
 * This class holds an enumeration 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 2006.03.30
 */

public class CommandWords
{
    // a constant array that holds all valid command words
    private static final String[] validCommands = {
        "go", "quit", "help"
    };

    /**
     * Constructor - initialise the command words.
     */
    public CommandWords()
    {
        // nothing to do at the moment...
    }

    /**
     * Check whether a given String is a valid command word.
     * @return true if a given string is a valid command,
     * false if it isn't.
     */
    public boolean isCommand(String aString)
    {
        for(int i = 0; i < validCommands.length; i++) {
            if(validCommands[i].equals(aString))
                return true;
        }
        // if we get here, the string was not found in the commands
        return false;
    }
}
```