Kapitel 7.1 bis 7.4

1. zu bearbeitende Aufgaben: 7.1, 7.2, 7.4 (Bauen Sie zwei zusätzliche Räume ein.) und 7.5

7.1:

a) What does this application do?

„World of Zuul“: Ein einfaches Text basiertes Spiel. Man kann verschiedene Räume betreten und verlassen.

b) What commands does the game accept?

help, quit, go [richtung]

c) What does each command do?

help: zeigt die möglichen Befehle

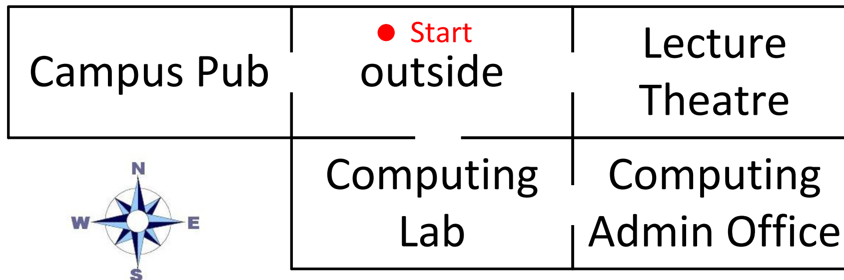
quit: beendet das Spiel

go [Richtung]: Falls in diese Richtung eine Tür ist wird durch diese gegangen und der nächste Raum betreten. Mögliche Richtungen sind: north, east, south, west

d) How many rooms are in the scenario?

5

e) Draw a map of the existing rooms.



7.2:

Game: *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.

Parser: *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.

Command: *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").*

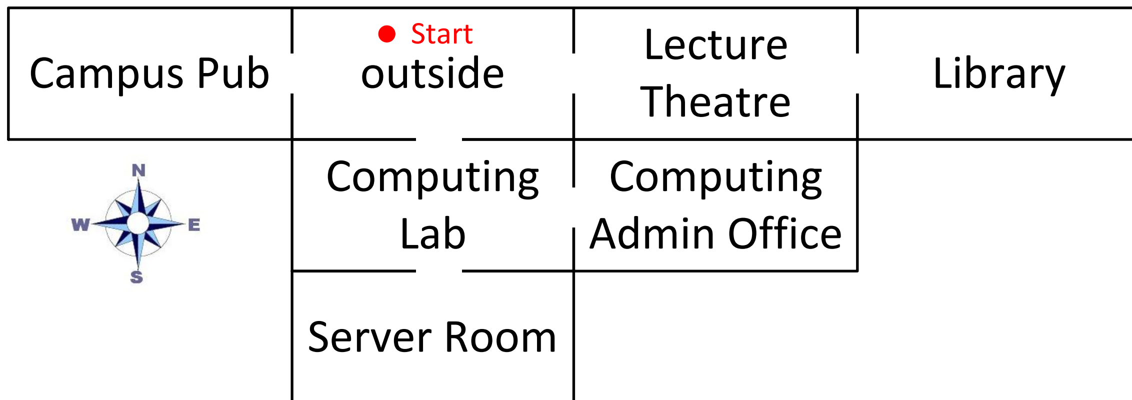
The way this is used is: Commands are already checked for being valid command words. If the user entered an invalid command (a word that is not known) then the command word is <null>.

If the command had only one word, then the second word is <null>.

CommandWords: *This class holds an enumeration of all command words known to the game. It is used to recognise commands as they are typed in.*

Room: *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.*

7.4: (Bauen Sie zwei zusätzliche Räume ein.)




```
    private void createRooms()
    {
        Room outside, theatre, pub, lab, office, server, library;

        // 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");
        server = new Room("in the server room");
        library = new Room("in the library");


        // initialise room exits
        outside.setExits(null, theatre, lab, pub);
        theatre.setExits(null, library, null, outside);
        pub.setExits(null, outside, null, null);
        lab.setExits(outside, office, server, null);
        office.setExits(null, null, null, lab);
        server.setExits(lab, null, null, null);
        library.setExits(null, null, null, theatre);

        currentRoom = outside;  // start game outside
    }
```

```
You are outside the main entrance of the university
Exits: east south west
> go south
You are in a computing lab
Exits: north east south
> go south
You are in the server room
Exits: north
> go north
You are in a computing lab
Exits: north east south
> go north
You are outside the main entrance of the university
Exits: east south west
> go east
You are in a lecture theatre
Exits: east west
> go east
You are in the library
Exits: west
```

7.5:

```
/**
 * Print out location info
 */
private void printLocationInfo()
{
    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();
}


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 game.");
    System.out.println("Type 'help' if you need help.");
    System.out.println();

    printLocationInfo();
}


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;

        printLocationInfo();
    }
}
```

2. Was versteht man unter Kopplung? Wie soll die Kopplung sein?
   Unter Kopplung versteht man die Abhängigkeiten von Klassen untereinander. Es sollte möglichst versucht werden eine „lose Kopplung" zu haben, d.h. möglichst wenig Abhängigkeiten von einer einzelnen Klassen zu einer anderen. Dadurch ist es möglich einzelne Klassen anzupassen, erweitern ohne das alle anderen Klassen ebenfalls abgeändert werden müssen.

3. Was versteht man unter Kohäsion? Wie soll die Kohäsion sein?
   In der objektorientierten Programmierung beschreibt Kohäsion, wie gut eine Programmeinheit eine logische Aufgabe oder Einheit abbildet. In einem System mit starker Kohäsion ist jede Programmeinheit (eine Methode, eine Klasse oder ein Modul) verantwortlich für genau eine wohldefinierte Aufgabe oder Einheit.
   http://de.wikipedia.org/wiki/Koh%C3%A4sion_%28Informatik%29

4. Was ist die Problematik bei Code-Duplikation?
   Bei Änderungen müssen diese an mehreren Orten durchgeführt werden. Ggf. gehen auch einzelne Stellen dadurch vergessen. Der Code ist unübersichtlich und die Wartung nur sehr schwer oder gar nicht möglich.

5. Von was ist Code-Duplikation häufig ein Symptom (schlechter Kopplung oder schlechter Kohäsion)?
   Schlechter Kohäsion
   Mehrere Methoden erfüllen die gleiche Aufgabe / lösen das gleiche Problem.
   ⇨ *Neue Methode erstellen für diese Aufgabe und an den anderen Stellen diese Methode aufrufen.*

Kapitel 7.5 bis 7.11

6. zu bearbeitende Aufgaben: 7.6 bis 7.8, 7.11, 7.14, 7.16 und 7.17

7.6:

```
public class Room
{
    private String description;
    private Room northExit;
    private Room southExit;
    private Room eastExit;
    private Room westExit;

    [...]

    /**
     * @return The exit of the room.
     */
    public Room getExit(String direction)
    {
        if(direction.equals("north"))
            return northExit;
        if(direction.equals("east"))
            return eastExit;
        if(direction.equals("south"))
            return southExit;
        if(direction.equals("west"))
            return westExit;
        return null;
    }
}

public class Game
{

    [...]

    /**
     * Print out location info
     */
    private void printLocationInfo()
    {
        System.out.println("You are " + currentRoom.getDescription());
        System.out.print("Exits: ");
        if(currentRoom.getExit("north") != null) {
            System.out.print("north ");
        }
        if(currentRoom.getExit("east") != null) {
            System.out.print("east ");
        }
        if(currentRoom.getExit("south") != null) {
            System.out.print("south ");
        }
        if(currentRoom.getExit("west") != null) {
            System.out.print("west ");
        }
        System.out.println();
    }
}
```

7.7:

```
public class Room
{
    [...]

    /**
     * Return a string describing the room’s exits,
     * for example, "Exits: north west".
     */
    public String getExitString()
    {
        String res;
        res = "Exits: ";
        if(northExit != null) {
            res += "north ";
        }
        if(southExit != null) {
            res += "east ";
        }
        if(eastExit != null) {
            res += "south ";
        }
        if(westExit != null) {
            res += "west ";
        }
        return res;
    }
}


public class Game
{
    [...]

    /**
     * Print out location info
     */
    private void printLocationInfo()
    {
        System.out.println("You are " + currentRoom.getDescription());
        System.out.println(currentRoom.getExitString());
    }
}
```

7.8:

```
import java.util.HashMap;

public class Room
{
    private String description;
    private HashMap exits;

    /**
     * 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;
        exits = new HashMap<String, Room>();
    }

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

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

```
    /**
     * 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); //Cast to Room
    }

    [...]
}

public class Game
{
    [...]

    /**
     * Create all the rooms and link their exits together.
     */
    private void createRooms()
    {
        Room outside, theatre, pub, lab, office, server, library;

        // 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");
        server = new Room("in the server room");
        library = new Room("in the library");

        // initialise room exits
        outside.setExits("east", theatre);
        outside.setExits("south", lab);
        outside.setExits("west", pub);
        theatre.setExits("east", library);
        theatre.setExits("west", outside);
        pub.setExits("east", outside);
        lab.setExits("north", outside);
        lab.setExits("east", office);
        lab.setExits("south", server);
        office.setExits("west", lab);
        server.setExits("north", lab);
        library.setExits("west", theatre);

        currentRoom = outside;  // start game outside
    }
}
```

7.11:

```
public class Game
{
    [...]

    /**
     * Create all the rooms and link their exits together.
     */
    private void createRooms()
    {
        Room outside, theatre, pub, lab, office, server, library, cellar;

        // create the rooms
        [...]
        cellar = new Room("in the cellar");

        // initialise room exits
        [...]
        office.setExits("down", cellar);
        cellar.setExits("up", office);
        [...]
    }

    /**
     * Print out location info
     */
    private void printLocationInfo()
    {
        System.out.println(currentRoom.getLongDescription());
    }
}


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

public class Room
{
        [...]

    /**
     * Return a string describing the room's exits,
     * for example, "Exits: north west".
     */
    public String getExitString()
    {
        String res = "Exits:";
        Set<String> keys = exits.keySet();
        for(String exit : keys) {
            res += " " + exit;
        }
        return res;
    }

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

7.14:

```
public class CommandWords
{
    [...]

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


public class Game
{
    [...]

    /**
     * Look arround the room and print out the exits
     */
    private void look()
    {
        System.out.println(currentRoom.getLongDescription());
    }

    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 look quit help");
    }

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

        return wantToQuit;
    }
}
```

7.16:

```
public class CommandWords
{
    [...]

    /*
     * Print all valid commands to System.out.
     */
    public void showAll()
    {
        for(String command : validCommands) {
            System.out.print(command + " ");
        }
        System.out.println();
    }
}


public class Parser
{
    [...]

    /**
     * Print out a list of valid command words.
     */
    public void showCommands()
    {
        commands.showAll();
    }
}


public class Game
{
    [...]

    /**
     * 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:");
        parser.showCommands();
    }
}
```

7.17:

Nein.

Die verfügbaren Commands werden anhand dem Array validCommands angezeigt.

7. Die Klasse Room verwaltet neu benachbarte Räume mit Hilfe einer `HashMap`. Allerdings mussten dazu an etlichen Stellen Anpassungen gemacht werden! Ist dies ein Indiz für zu starke oder für zu lose Kopplung?
   Dies ist ein Indiz für zu starke Kopplung.

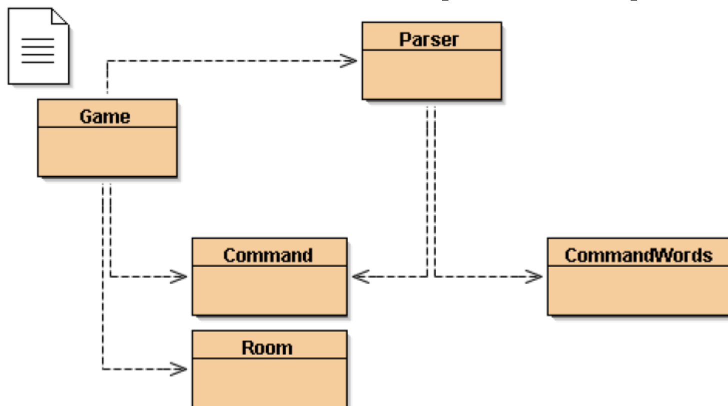
8. Was ist Information Hiding bzw. Datenkapselung und was bewirkt dieses grundlegende Prinzip?
   Information Hiding basiert auf dem Grundsatz, dass Informationen über ein Objekt nur über definierte Schnittstellen (getter / setter methoden) zur Verfügung stehen.

   Datenkapselung ist die Technik, das Prinzip heisst Information Hiding.

   Ein Entwickler muss sich nicht um das „WIE“ kümmern, er weiss von einer Klasse nur das „WAS“ und kann die ganze Klasse verwenden.

9. Was mein das Konzept "localizing change"?
Bei einem Code-Change sollten die Einflüsse möglichst klein sein. Es sollte so lokal wie möglich bleiben.
Dadurch verbessert sich die Wartbarkeit vom Code, sowie erleichtert es die Zusammenarbeit mit anderen Entwicklern an einer gemeinsamen Software.

10. Inwiefern unterscheiden sich explizite und implizite Kopplung?



Explizite Kopplung: Game ist „direkt" (explizit) mit Command gekoppelt.
Implizite Kopplung: Game ist „indirekt" (implizit) mit CommandWords via Parser gekoppelt.
Bei der Impliziten Kopplung besteht keine direkte Abhängigkeit, jedoch via weiteren Klassen die dazwischen sind bestehen trotzdem Abhängigkeiten, Kopplungen.

11. Machen kohäsive Methoden auch Sinn?
Ja, eine Methode sollte genau für eine Aufgabe verantwortlich sein. (Kohäsion: innerer Zusammenhalt)

12. Welches sind die beiden Hauptvorteile von starker Kohäsion?
- *1 Klasse repräsentiert 1 logische Einheit (Entität).*
- *1 Methode repräsentiert 1 zusammenhängende Aufgabe.*
- *Jede Klasse handhabt ihre Daten möglichst selber. (Responsibility-Driven Design)*
- *Es liegt möglichst wenig Code-Duplikation vor.*
- *Potenzielle Änderungen haben*

Kapitel 7.12 und 7.14

13. zu bearbeitende Aufgabe: 7.27
7.27:
Es sollten positive und negative Tests durchgeführt werden:
- *testen das alle Räume richtig miteinander verbunden sind*
- *testen das alle Befehle richtig ausgeführt werden, inkl. „go" ohne zweiten Parameter*

14. Was ist der Auslöser für Refactoring?
Das Anpassen des Design / Code an neue Anforderungen und Gegebenheiten – wenn Änderungen am Code durchgeführt werden.

15. Beschreiben Sie die Vorgehensweise bzw. die Schritte beim Refactoring.
0. Falls der bestehende Code noch nicht getestet wurde (!): Testspezifikation schreiben und den Code testen.
*1. Code re-designen **ohne** funktionale Änderungen und Code erneut testen (analog 0.).*
*2. Code erweitern **mit** funktionalen Änderungen und Code erneut entsprechend testen.*

16. Wann ist eine Methode zu lang?
Relativ
- *wenn sie mehr als eine Aufgabe erledigt*
- *wenn Code Fragmente in eigene Methoden ausgelagert werden könnte*
- *wenn der Methodenname zu lang wird (z.B. ab drei Wörter)*

17. Wann ist eine Klasse zu komplex?
Relativ
- *wenn die Klasse nicht mehr mit einfachen, wenigen Worte beschrieben werden kann*
- *wenn der Autor es selber nicht mehr weiss*

Handout ALG4

18. Illustrieren Sie den Unterschied zwischen stabilem und instabilem Sortieren an einem einfachen Beispiel.
Bei einem stabilen Sortierverfahren wird die Reihenfolge der Datensätze bei gleichem Sortierschlüssel beibehalten. Bei instabilen Sortieralgorithmen können diese ändern.

Beispiele für stabile Sortierverfahren:

Binary Tree Sort, Bubblesort, Insertionsort, Mergesort

Stabiles Sortierverfahren nach Zahlen:

```
1 Anton        1 Anton
4 Karl         1 Paul
3 Otto         3 Otto
5 Bernd   →    3 Herbert
3 Herbert      4 Karl
8 Alfred       5 Bernd
1 Paul         8 Alfred
```

Beispiele für instabile Sortierverfahren:

Quicksort, Shellsort, Selectionsort

Instabiles Sortierverfahren nach Zahlen:

```
1 Anton        1 Paul         1 Anton        1 Paul         1 Anton
4 Karl         1 Anton        1 Paul         1 Anton        1 Paul
3 Otto         3 Otto         3 Herbert      3 Herbert      3 Otto
5 Bernd   →    3 Herbert oder 3 Otto    oder 3 Otto    oder 3 Herbert
3 Herbert      4 Karl         4 Karl         4 Karl         4 Karl
8 Alfred       5 Bernd        5 Bernd        5 Bernd        5 Bernd
1 Paul         8 Alfred       8 Alfred       8 Alfred       8 Alfred
```

Bei instabiler Sortierung kann Paul vor Anton oder Herbert vor Otto stehen.

http://de.wikipedia.org/wiki/Stabiles_Sortierverfahren

19. Wie gross ist der Aufwand bei einfachen/direkten Sortieralgorithmen?
$O(n^2)$

20. Wie gross ist der Aufwand bei höheren Sortieralgorithmen?
$O(n * \log(n))$

21. Benennen Sie 3 einfache/direkte Sortieralgorithmen.
Bubblesort, Insertionsort, Mergesort

22. Welche der behandelten Sortieralgorithmen arbeiten instabil?
Shellsort

23. Beim Analysieren von Sortieralgorithmen ist wichtig zu wissen, wie man folgende Summe einfach berechnen kann:

$$1 + 2 + 3 + \cdots + n = \sum_{i=1}^{n} i = ?$$

Mit der Gausssche Summenformel:

$$\sum_{i=1}^{n} i \quad => \quad \frac{\boldsymbol{n\,(n+1)}}{\boldsymbol{2}}$$

http://de.wikipedia.org/wiki/Gau%C3%9Fsche_Summenformel

Optional: Kapitel 7.13 (Seite 233 bis 237)

24. Inwiefern unterscheidet sich ein Klassentyp (vgl. Keyword `class`) von einem Aufzählungstyp (vgl. Keyword `enum`)?

enum ist +/- gleich wie eine class, es können nicht nur Aufzählungs-Objekte erstellt werden, sondern deren auch Methoden etc. zuweisen. Bei enum ist einfach im Voraus hard-coded die Anzahl Objekte festgelegt.

enum ist eine „Liste", Aufzählungs-Objekt (Enumeration) von einem anderen Datentyp.

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}

public enum Obst {APFEL, KIRSCHE, PFLAUME}
```

25. Jeder Aufzählungstyp kennt eine Methode `values()`. Was liefert diese Methode als Rückgabewert?

Liefert ein Array mit allen Punkten dieser Enumeration zurück.