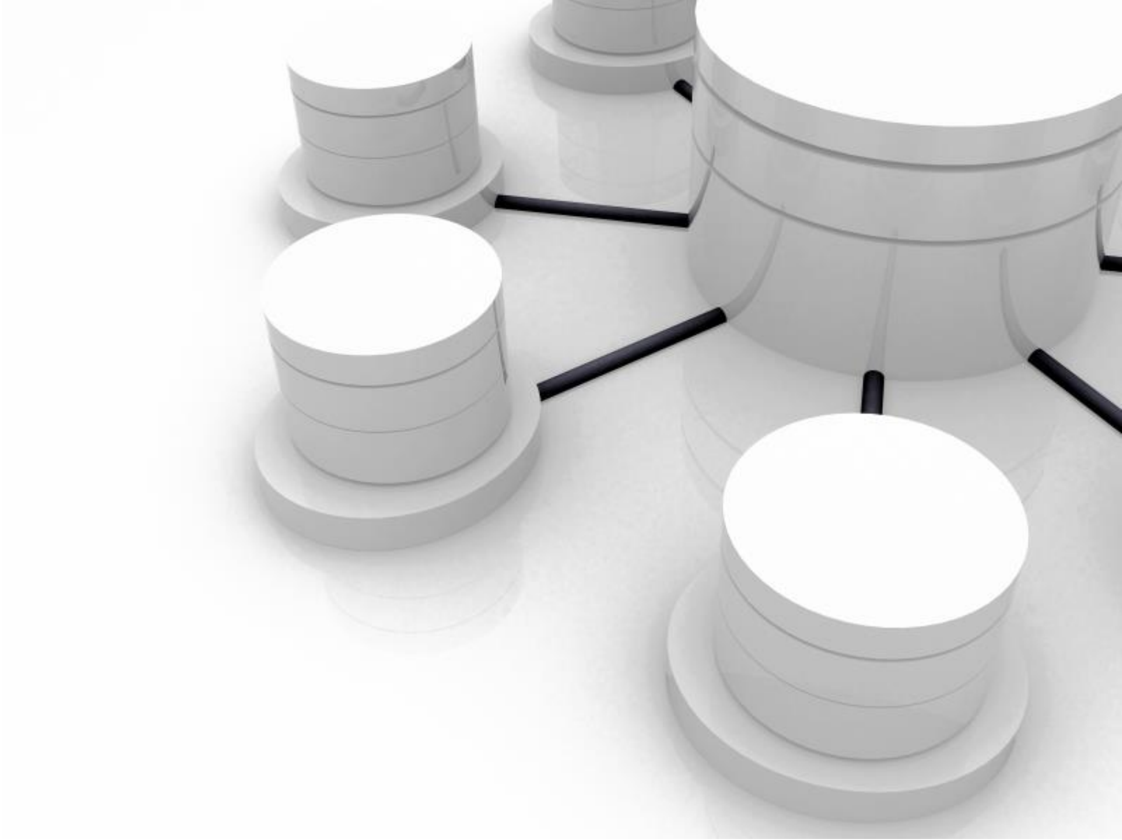
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich



Informatik 2

Exercise Sheet 10

Procedure

Toolbox

Repetition of the lecture content for the exercises

Übungsserien

Partial solution of the current exercise on the black board

Exam preparation

With a focus on content relevant for the exam

Presence hours

this week

in HIL F15.4

Goal of today's exercise

Topics

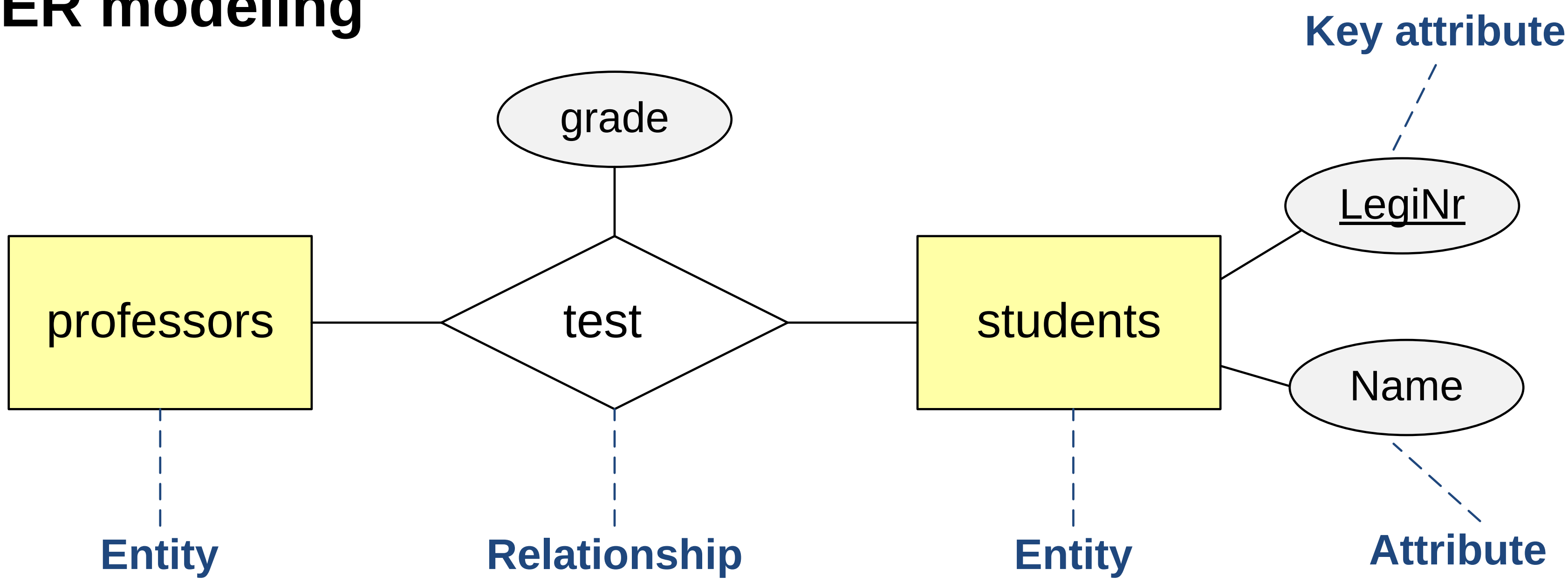
- Understand how to model data for a data base
- Be able to draw an Entity-Relationship (ER) diagram
- Functionalities

Important for the Exam

- Be able to draw an Entity-Relationship diagram for a given scenario
- Correct usage of functionalities

Toolbox for current exercise

ER modeling

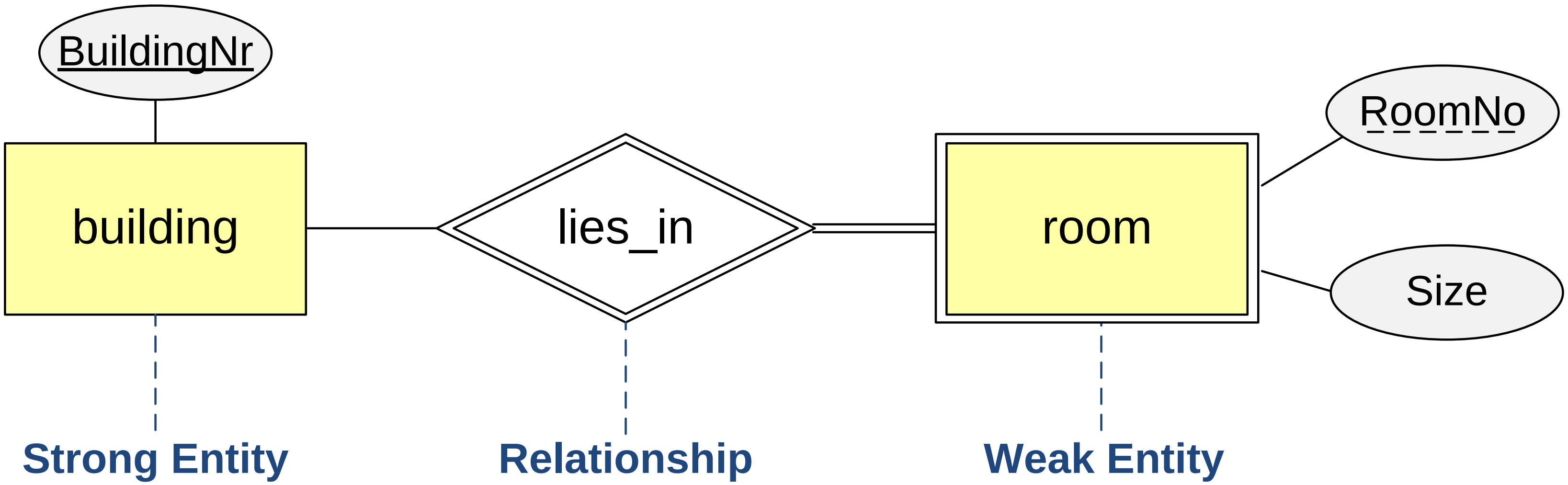


What „knows“ ER?

- Entities (Gegenstände)
- Relationships
- Attributes und Rolls

Toolbox for current exercise (2)

Weak entities



Keep in Mind

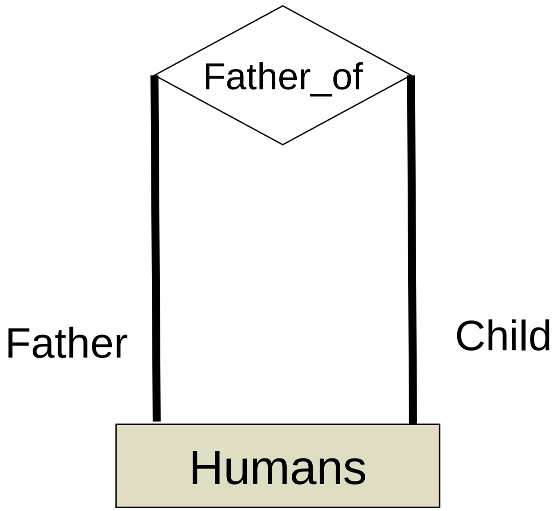
- **Weak Entities** cannot live without **strong entities** and can only be identified uniquely together with strong entities

Functionalities

- For better understanding of 1:1 oder 1:N functionality do always write down (partial) functions
- Direction of the function always "to 1"

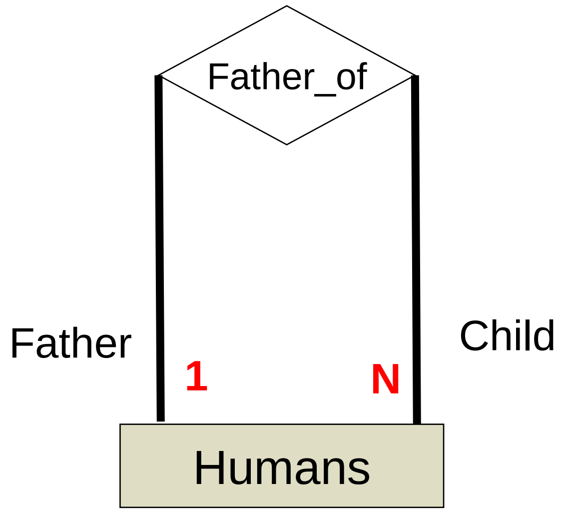
Functionalities

- Which Functionalities?



Functionalities

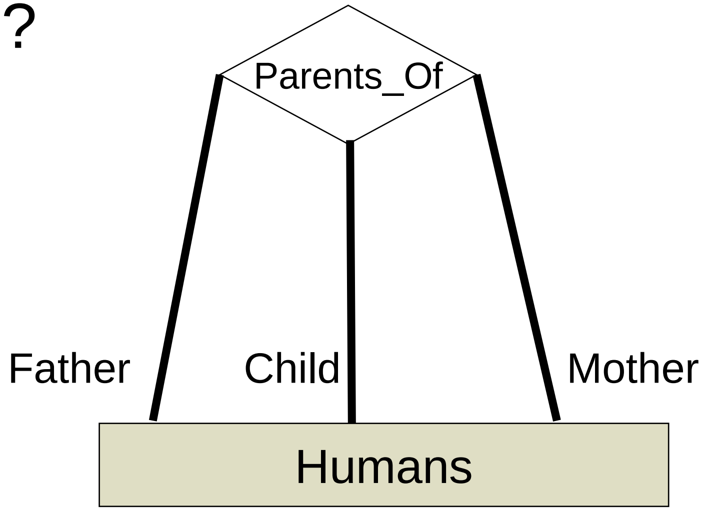
- Which Functionalities?



Father_of: $\text{Child: Humans} \rightarrow \text{Father: Humans}$

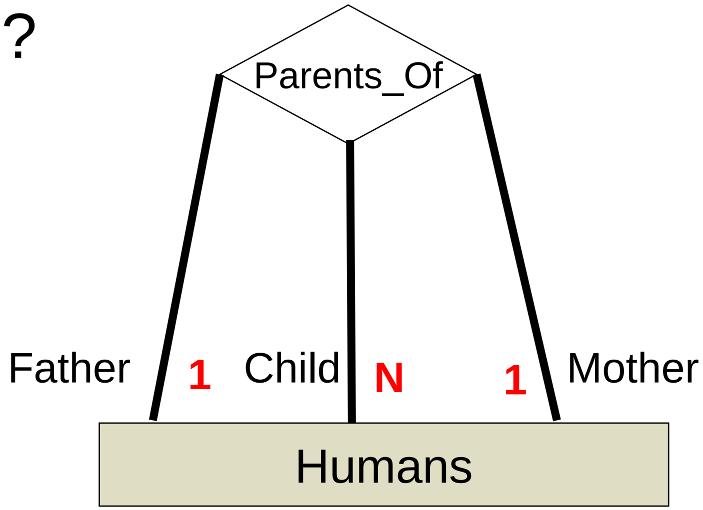
Functionalities

- Which Functionalities?
- Good Model?



Functionalities

- Which Functionalities?
- Good Model?



Parents_Of:

Child: Humans × Father: Humans → Mother: Humans
Child: Humans × Mother: Humans → Father: Humans

Problem:

(Anna, Bernhard) ↦ Clara
(Anna, Doris) ↦ Emil

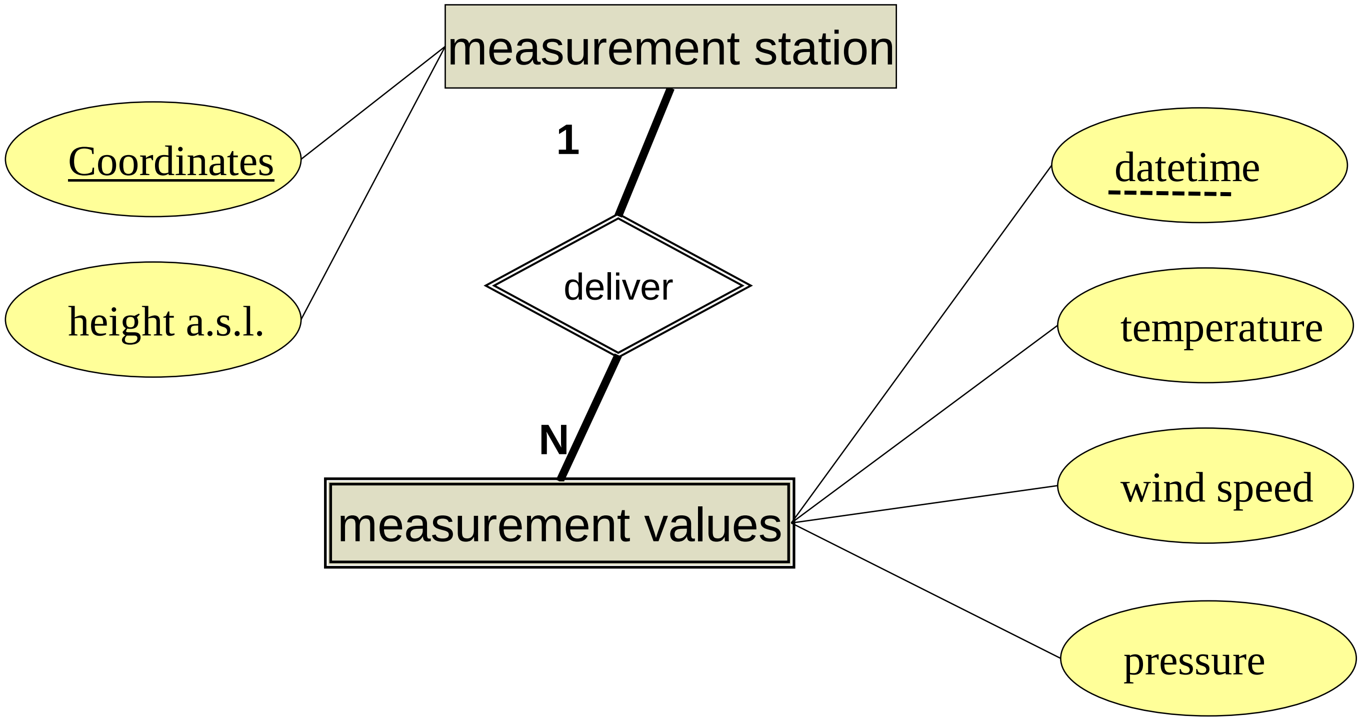
→ Inconsistency possible
(Bernhard, Anna, Clara),
(Emil, Anna, Doris)

Example Measurement Station

Model

- Mesasurement Stations with known location (coordinates and height over sea) deliver measured values temperature, wind and air pressure at given points in time.

Example Measurement Station



Current exercise

ETH zürich

Discussion Assignment 9

Felix Friedrich, Lars Widmer
TA lecture, Informatics II D-BAUG

May 7, 2014

Outline

1. Know How
   - Recursion
   - Example
2. Postdiscussion Assignment 9
   - Binary Tree
   - Extending Highscore
   - Funny Graphics
3. Wrap Up
   - Questions
   - Good Luck

Recursion Repeated

- **Recursion** is when a method **calls itself.** It can be puzzling but in the end it's **simple** and very **powerful.**

Recursion Repeated

- **Recursion** is when a method **calls itself**. It can be puzzling but in the end it's **simple** and very **powerful**.
- It's very **natural** to use recursion in order to **traverse a tree**. Every recursion brings us a level further down on the current branch. So, that's what we choose for the example solution in the following slides.

Recursion Repeated

- **Recursion** is when a method **calls itself**. It can be puzzling but in the end it's **simple** and very **powerful**.
- It's very **natural** to use recursion in order to **traverse a tree**. Every recursion brings us a level further down on the current branch. So, that's what we choose for the example solution in the following slides.
- But first let's have a very quick look at recursion itself.

Recursive Example

```java
private static void callMyself(int n) {
    System.out.print(n+" ");
    if (n > 0) {
        callMyself(n-1);
    }
}

public static void main(String[] args) {
    callMyself(5);
}
```

What's the output? →

Recursive Example

```java
private static void callMyself(int n) {
    System.out.print(n+" ");
    if (n > 0) {
        callMyself(n-1);
    }
}

public static void main(String[] args) {
    callMyself(5);
}
```

What's the output? → 5 4 3 2 1 0

Example Execution, *starting with 2*

```
callMyself(2) {
    System.out.print("2 ");
    if (2 > 0) {
        callMyself(2-1);
        // ...
}   }
```

Example Execution, *starting with 2*

```
callMyself(2) {
    System.out.print("2 ");
    if (2 > 0) {
        callMyself(2-1);
        ==> callMyself(1) {
            System.out.print("1 ");
            if (1 > 0) {
                callMyself(1-1);
                // ...
}   }   }   }
```

Example Execution, *starting with 2*

```
callMyself(2) {
    System.out.print("2 ");
    if (2 > 0) {
        callMyself(2-1);
        ==> callMyself(1) {
            System.out.print("1 ");
            if (1 > 0) {
                callMyself(1-1);
                ==> callMyself(0) {
                    System.out.print("0 ");
                    if (0 > 0) { // false
                        // stop therefore
}    }    }    }    }    }
```

Outline

1. Know How
   - Recursion
   - Example
2. Postdiscussion Assignment 9
   - Binary Tree
   - Extending Highscore
   - Funny Graphics
3. Wrap Up
   - Questions
   - Good Luck

TreeElement, *our data structure*

```java
public class TreeElement
{   public String data;
    public TreeElement lower;
    public TreeElement higher;
    public TreeElement(String str)
    {   data = str;
        lower = null;
        higher = null;
    }
    public String toString()
    {   return data+" ";
    }
}
```

Traverse *in Order* (for toString)

```
private String traverseInOrder(TreeElement iter) {
    if (iter == null) {
        return "";
    }
    return traverseInOrder(iter.lower)
        +iter+traverseInOrder(iter.higher);
}

public String toString() {
    return traverseInOrder(root); // start from top
}
```

Traverse *in Order* Explained

```
private String traverseInOrder(TreeElement iter) {
    if (iter == null) {
        return "";
    }
    return traverseInOrder(iter.lower)
       +iter+traverseInOrder(iter.higher);
}
```

- Each call we do two recursions, one for the `lower` branch, one for the `higher`.

Traverse *in Order* Explained

```
private String traverseInOrder(TreeElement iter) {
    if (iter == null) {
        return "";
    }
    return traverseInOrder(iter.lower)
       +iter+traverseInOrder(iter.higher);
}
```

- Each call we do two recursions, one for the `lower` branch, one for the `higher`.
- When we arrive at a `null`-reference, we're done for this branch.

Traverse *in Order* Explained

```
private String traverseInOrder(TreeElement iter) {
    if (iter == null) {
        return "";
    }
    return traverseInOrder(iter.lower)
       +iter+traverseInOrder(iter.higher);
}
```

- Each call we do two recursions, one for the `lower` branch, one for the `higher`.
- When we arrive at a `null`-reference, we're done for this branch.
- To get the right order, we concatinate the result of the `lower` branch before our `data` of the current node and at the end we put the result of the `higher` branch.

Locate the Node where to `Add`

```java
private void followToAdd(TreeElement iter,
                         TreeElement newTE) {

    // compare the two elements:
    int comp = newTE.data.compareTo(iter.data);
    // ...
            followToAdd(iter.higher, newTE);
            // ...
            followToAdd(iter.lower, newTE);
            // ...
}
```

- When traversing the tree in order to `add` a new element, we don't have to traverse the whole tree.

Locate the Node where to `Add`

```java
private void followToAdd(TreeElement iter,
                         TreeElement newTE) {

    // compare the two elements:
    int comp = newTE.data.compareTo(iter.data);
    // ...
            followToAdd(iter.higher, newTE);
            // ...
            followToAdd(iter.lower, newTE);
            // ...
}
```

- When traversing the tree in order to `add` a new element, we don't have to traverse the whole tree.
- At every node we can decide on which further branch the new element belongs. The other branch can be neglected.

Locate the Node where to `Add`

```java
private void followToAdd(TreeElement iter,
                         TreeElement newTE) {

    // compare the two elements:
    int comp = newTE.data.compareTo(iter.data);
    // ...
            followToAdd(iter.higher, newTE);
            // ...
            followToAdd(iter.lower, newTE);
            // ...
}
```

- When traversing the tree in order to `add` a new element, we don't have to traverse the whole tree.
- At every node we can decide on which further branch the new element belongs. The other branch can be neglected.
- When we arrive at a `null`-reference, we found the place to actually place the element.

Locate the Node where to `Add`

```
private void followToAdd(TreeElement iter,
                         TreeElement newTE) {

    // compare the two elements:
    int comp = newTE.data.compareTo(iter.data);

    if (comp == 0) { // Check for duplicates
        return; // found duplicate
    }

    // Carry on if the new element is higher ...
    // Carry on if the new element is lower ...
}
```

Locate the Node where to `Add`

```java
// ...
// Carry on if the new element is higher:
if (comp > 0) { // higher
    if (iter.higher == null) {
        iter.higher = newTE;
    } else { // not free => recursion
        followToAdd(iter.higher, newTE);
    }
} else { // lower
    // ...
```

Locate the Node where to `Add`

```
    // ...
    // Carry on if the new element is lower:
    } else { // lower
        if (iter.lower == null) {
            iter.lower = newTE;
        } else { // not free => recursion
            followToAdd(iter.lower, newTE);
        }
    }
}
```

Add Element

Using the traversing method, adding now is quite simple.

```
public void add(String str) {
    TreeElement newTE = new TreeElement(str);
    if (root == null) {
        root = newTE;
    } else {
        followToAdd(root, newTE);
    }
}
```

Traverse for Maximum Depth

We use a recursive helper method to determine the maximum depth of the tree.

```
public int depth() {
    return traverseForMaxDepth(root, 0);
}
```

Traverse for Maximum Depth

```
private int traverseForMaxDepth
        (TreeElement iter, int level) {
    if (iter == null) {
        return level;
    }
    ++level;
    return Math.max(
       traverseForMaxDepth(iter.lower, level),
       traverseForMaxDepth(iter.higher, level));
}
```

Outline

1. Know How
   - Recursion
   - Example
2. Postdiscussion Assignment 9
   - Binary Tree
   - Extending Highscore
   - Funny Graphics
3. Wrap Up
   - Questions
   - Good Luck

Extending Highscore, Class Structure

```
public class FiledHighscore extends Highscore {

    public FiledHighscore(int s) {
        super(s);
    }

    // ...

}
```

Class `FiledHighscore`, Method `store`

```java
public void store() {
    try {
        BufferedWriter out = new BufferedWriter
            (new FileWriter("highscore.txt"));
        for (Player p : highscore) {
            out.write(p.score);
            out.write(p.name);
            out.write('\n');
        }
        out.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Class `FiledHighscore`, Method `load`

```
public void load() {
    try {
        // open file, read all, close
        // ...
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Method `load`

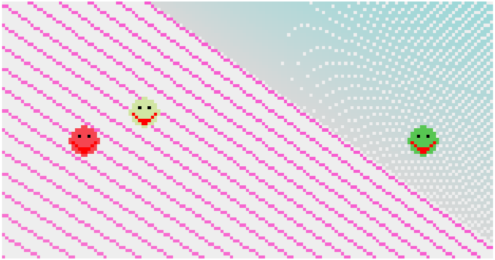
```
// open file, read all, close
BufferedReader in = new BufferedReader(
        new FileReader("highscore.txt"));
highscore.clear(); // empty current list
while (in.ready()) {
    int score = in.read();
    String name = in.readLine();
    Player p = new Player(name,score);
    highscore.add(p);
}
in.close();
```

Outline

1. Know How
   - Recursion
   - Example
2. Postdiscussion Assignment 9
   - Binary Tree
   - Extending Highscore
   - Funny Graphics
3. Wrap Up
   - Questions
   - Good Luck

Bouncing Smileys!

You can find the full project for this example solution on the website.



If the authors agree, we might also publish **cool student projects** on the website.

Outline

1. Know How
   - Recursion
   - Example
2. Postdiscussion Assignment 9
   - Binary Tree
   - Extending Highscore
   - Funny Graphics
3. Wrap Up
   - Questions
   - Good Luck

Feel free . . .

Ladies & Gentlemen!
. . . Please name any

- Questions?
- Feedback?
- Additions?
- Remarks?
- Wishes?
- . . .

All the Best!