6

Tik–Tok Victoryyy

Sieg auf ganzer Linie, 6

Florian Sihler ○ KW 49

Präsenzaufgabe

1

Glückselig sind die Studierenden

Präsenzaufgabe

1 **Glückselig sind die Studierenden**

In dieser Aufgabe sollen Sie ihre ersten Schritte mit Objektorientierter Programmierung machen, indem Sie ein einfaches Glücksrad implementieren. Dieses Glücksrad soll eine bestimmte Anzahl an Slots besitzen, welche mit Einträgen gefüllt werden können und bei einer Drehung soll ein zufälliger Eintrag ausgewählt werden.
Erstellen Sie dafür eine Klasse `WheelOfFortune`, welche ein `String` Array namens `slots` und einen Integer `nextSlot` als Instanzvariablen besitzt. Die Klasse soll einen Konstruktor besitzen der einen Integer `numSlots` als Parameter besitzt, welcher die Größe des Arrays bestimmt. Der Konstruktor soll beide Instanzvariablen außerdem geeignet initialisieren. Implementieren Sie dann zusätzlich die folgenden Methoden:

1. Die Methode `public boolean addEntry(String entry)` soll den übergebenen Wert in den nächsten freien Slot schreiben. Stellen Sie dabei sicher, dass Sie nicht mehr Einträge in das Array schreiben als dessen Größe

Präsenzaufgabe

1

Glückselig sind die Studierenden

In dieser Aufgabe sollen Sie ihre ersten Schritte mit Objektorientierter Programmierung machen, indem Sie ein einfaches Glücksrad implementieren. Dieses Glücksrad soll eine bestimmte Anzahl an Slots besitzen, welche mit Einträgen gefüllt werden können und bei einer Drehung soll ein Eintrag ausgewählt werden.

Erstellen Sie dafür eine Klasse `WheelOfFortune`, welche ein `String` Array namens `slots` und einen Integer `nextSlot` als Instanzvariablen besitzt. Die Klasse soll einen Konstruktor besitzen der einen Integer `numSlots` als Parameter besitzt, welcher die Größe des Arrays bestimmt. Der Konstruktor soll beide Instanzvariablen außerdem geeignet initialisieren. Implementieren Sie dann zusätzlich die folgenden Methoden:

1. Die Methode `public boolean addEntry(String entry)` soll den übergebenen Wert in den nächsten freien Slot schreiben. Stellen Sie dabei sicher, dass Sie nicht mehr Einträge in das Array schreiben als dessen Größe

Präsenzaufgabe

1

Glückselig sind die Studierenden

Implementieren Sie ein veränderbares `WheelOfFortune`.

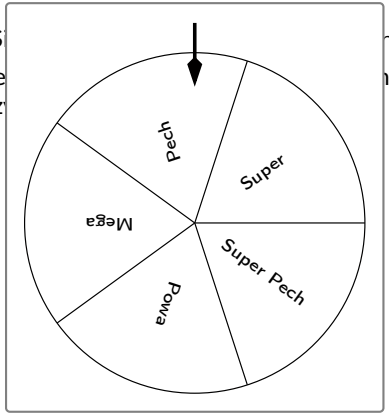
Präsenzaufgabe

1

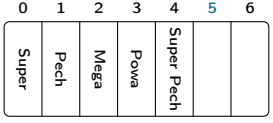
Glückselig sind die Studierenden

Implementieren S…ne.

1. Erstellen Sie …nes ein String Array `slots` fester Größe und eine Instanz…



`nextSlot = 5`



Präsenzaufgabe

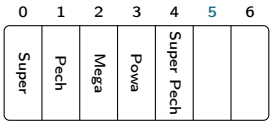
1

Glückselig sind die Studierenden

Implementieren Sie ein veränderbares `WheelOfFortune`.

1. Erstellen Sie eine Klasse `WheelOfFortune` welches ein String Array `slots` fester Größe und eine Instanzvariablen `int nextSlot` besitzt.
2. `public boolean addEntry(String entry)` fügt einen Eintrag hinzu, sofern es die Größe zulässt (der Rückgabewert kennzeichnet den Erfolg).

`nextSlot = 5`



Präsenzaufgabe

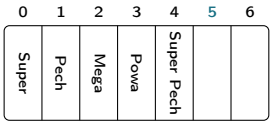
1

Glückselig sind die Studierenden

Implementieren Sie ein veränderbares `WheelOfFortune`.

1. Erstellen Sie eine Klasse `WheelOfFortune` welches ein String Array `slots` fester Größe und eine Instanzvariablen `int nextSlot` besitzt.
2. `public boolean addEntry(String entry)` fügt einen Eintrag hinzu, sofern es die Größe zulässt (der Rückgabewert kennzeichnet den Erfolg).
3. `public boolean removeEntry()` entfernt den letzten hinzugefügten Eintrag wieder (der Rückgabewert kennzeichnet ob ein Eintrag entfernt wurde).

`nextSlot = 5`



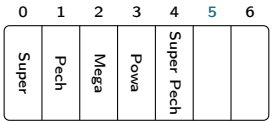
Präsenzaufgabe

1 **Glückselig sind die Studierenden**

Implementieren Sie ein veränderbares `WheelOfFortune`.

1. Erstellen Sie eine Klasse `WheelOfFortune` welches ein String Array `slots` fester Größe und eine Instanzvariablen `int nextSlot` besitzt.
2. `public boolean addEntry(String entry)` fügt einen Eintrag hinzu, sofern es die Größe zulässt (der Rückgabewert kennzeichnet den Erfolg).
3. `public boolean removeEntry()` entfernt den letzten hinzugefügten Eintrag wieder (der Rückgabewert kennzeichnet ob ein Eintrag entfernt wurde).
4. `public int getNumEntries()` soll die Gesamtanzahl der Einträge liefern.

`nextSlot = 5`



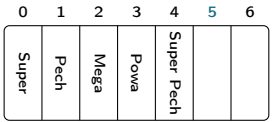
Präsenzaufgabe

1 Glückselig sind die Studierenden

Implementieren Sie ein veränderbares `WheelOfFortune`.

1. Erstellen Sie eine Klasse `WheelOfFortune` welches ein String Array `slots` fester Größe und eine Instanzvariablen `int nextSlot` besitzt.
2. `public boolean addEntry(String entry)` fügt einen Eintrag hinzu, sofern es die Größe zulässt (der Rückgabewert kennzeichnet den Erfolg).
3. `public boolean removeEntry()` entfernt den letzten hinzugefügten Eintrag wieder (der Rückgabewert kennzeichnet ob ein Eintrag entfernt wurde).
4. `public int getNumEntries()` soll die Gesamtanzahl der Einträge liefern.
5. `public String spin()` dreht das Glücksrad und liefert den Eintrag zurück. Gibt es keinen Eintrag wünschen wir einen leeren String.

`nextSlot = 5`



Präsenzaufgabe

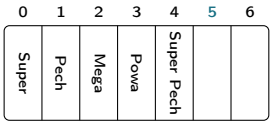
1 **Glückselig sind die Studierenden**

Implementieren Sie ein veränderbares `WheelOfFortune`.

1. Erstellen Sie eine Klasse `WheelOfFortune` welches ein String Array `slots` fester Größe und eine Instanzvariablen `int nextSlot` besitzt.
2. `public boolean addEntry(String entry)` fügt einen Eintrag hinzu, sofern es die Größe zulässt (der Rückgabewert kennzeichnet den Erfolg).
3. `public boolean removeEntry()` entfernt den letzten hinzugefügten Eintrag wieder (der Rückgabewert kennzeichnet ob ein Eintrag entfernt wurde).
4. `public int getNumEntries()` soll die Gesamtanzahl der Einträge liefern.
5. `public String spin()` dreht das Glücksrad und liefert den Eintrag zurück. Gibt es keinen Eintrag wünschen wir einen leeren String.

Erzeugen Sie zudem eine `main`-Methode welche das Rad sinnvoll benutzt.

`nextSlot = 5`



- Eine Basis (WheelOfFortune.java):

- Eine Basis (WheelOfFortune.java):

```
public class WheelOfFortune {









}
```

- Eine Basis (WheelOfFortune.java):

```
public class WheelOfFortune {
    private String[] slots;









}
```

- Eine Basis (WheelOfFortune.java):

```
public class WheelOfFortune {
    private String[] slots;
    private int nextSlot;









}
```

- Eine Basis (WheelOfFortune.java):

```
public class WheelOfFortune {
    private String[] slots;
    private int nextSlot;

    public WheelOfFortune(int numSlots) {




    }
}
```

- Eine Basis (WheelOfFortune.java):

```
public class WheelOfFortune {
    private String[] slots;
    private int nextSlot;

    public WheelOfFortune(int numSlots) {
        nextSlot = 0;




    }
}
```

- Eine Basis (WheelOfFortune.java):

```
public class WheelOfFortune {
    private String[] slots;
    private int nextSlot;

    public WheelOfFortune(int numSlots) {
        nextSlot = 0;
        // keine negative Länge
        slots = new String[Math.max(numSlots, 0)];




    }
}
```

- Eine Basis (WheelOfFortune.java):

```
public class WheelOfFortune {
    private String[] slots;
    private int nextSlot;

    public WheelOfFortune(int numSlots) {
        nextSlot = 0;
        // keine negative Länge
        slots = new String[Math.max(numSlots, 0)];
        // Am Anfang setzen wir alle auf den leeren String
        for(int i = 0; i < slots.length; i++)
            slots[i] = ""; // vs. new String("");
    }
}
```

- Eine Basis (WheelOfFortune.java):

```
public class WheelOfFortune {
    private String[] slots;
    private int nextSlot;

    public WheelOfFortune(int numSlots) {
        nextSlot = 0;
        // keine negative Länge
        slots = new String[Math.max(numSlots, 0)];
        // Am Anfang setzen wir alle auf den leeren String
        for(int i = 0; i < slots.length; i++)
            slots[i] = ""; // vs. new String("");
    }
}
```

Von **Klassen** lassen sich durch den **Konstruktor Objekte** erzeugen. Der **Zustand** eines Objekts ist durch die **Instanzvariablen** definiert. Jedes Objekt hat seinen *eigenen Zustand*.

Präsenzaufgabe - Lösung

Hinzufügen

Präsenzaufgabe - Lösung

Hinzufügen

```
public boolean addEntry(String entry) {




}
```

Präsenzaufgabe - Lösung

Hinzufügen

```
public boolean addEntry(String entry) {
    if(nextSlot == slots.length) // Sind wir voll?
        return false;




}
```

Präsenzaufgabe - Lösung

Hinzufügen

```
public boolean addEntry(String entry) {
    if(nextSlot == slots.length) // Sind wir voll?
        return false;

    slots[nextSlot] = entry;
    nextSlot++;

}
```

Präsenzaufgabe - Lösung

Hinzufügen

```
public boolean addEntry(String entry) {
    if(nextSlot == slots.length) // Sind wir voll?
        return false;

    slots[nextSlot] = entry;
    nextSlot++;
    return true;
}
```

Präsenzaufgabe - Lösung

Entfernen

Präsenzaufgabe - Lösung

Entfernen

```
public boolean removeEntry() {




}
```

Präsenzaufgabe - Lösung

Entfernen

```
public boolean removeEntry() {
    if(nextSlot == 0) // Sind wir leer?
        return false;




}
```

Präsenzaufgabe - Lösung

Entfernen

```
public boolean removeEntry() {
    if(nextSlot == 0) // Sind wir leer?
        return false;

    nextSlot--;
    slots[nextSlot] = "";

}
```

Präsenzaufgabe - Lösung

Entfernen

```
public boolean removeEntry() {
    if(nextSlot == 0) // Sind wir leer?
        return false;

    nextSlot--;
    slots[nextSlot] = "";
    return false;
}
```

Präsenzaufgabe - Lösung

Präsenzaufgabe - Lösung

```
public int getNumEntries() {

}
```

Präsenzaufgabe - Lösung


```java
public int getNumEntries() {
    return nextSlot;
}
```

Präsenzaufgabe - Lösung

Es geht wieder loooos

Präsenzaufgabe - Lösung

Es geht wieder loooos

```
public String spin() {




}
```

Präsenzaufgabe - Lösung

Es geht wieder loooos

```
public String spin() {
    // Sind wir leer?
    if(slots.length == 0)
        return "";



}
```

Präsenzaufgabe - Lösung

Es geht wieder loooos

```
public String spin() {
    // Sind wir leer?
    if(slots.length == 0)
        return "";

    int slot = (int) (Math.random() * nextSlot);

}
```

Präsenzaufgabe - Lösung

Es geht wieder loooos

```
public String spin() {
    // Sind wir leer?
    if(slots.length == 0)
        return "";

    int slot = (int) (Math.random() * nextSlot);
    return slots[slot];
}
```

Präsenzaufgabe - Lösung

Es geht wieder loooos

```java
public String spin() {
    // Sind wir leer?
    if(slots.length == 0)
        return "";

    int slot = (int) (Math.random() * nextSlot);
    return slots[slot];
}
```

(Pseudo-)**Zufall** ist ein spannendes Thema. Anstelle von `Math.random()` können wir uns direkt ein „Random-Objekt“ bauen:

```java
Random rnd = new Random();
rnd.nextInt(15); // {0,...,14}
```

Präsenzaufgabe - Lösung


```
public static void main(String[] args) {
    int numSlots = 6;
    WheelOfFortune wheel = new WheelOfFortune(numSlots);

    for(int i = 0; i < numSlots + 1; i++) {
        String entry = "Eintrag " + (char) ('A' + i);
        if(wheel.addEntry(entry))
            System.out.println("Eintrag " + entry + " hinzugefügt");
        else System.out.println("Glücksrad ist voll");
    }

    String result = wheel.spin();
    System.out.println("Ergebnis einer Drehung: " + result);
    while(wheel.getNumEntries() > 0)
        wheel.removeEntry();
    result = wheel.spin();
    System.out.println("Drehen des leeren Glücksrades ergibt: " + result);
}
```

1	2	3
4	5	6
7	8	9

Player „X“: 5

Player „X“: 5 *Player „O“: 3*


1 2 3
4 5 6
7 8 9
1 2 3
4 X 6
7 8 9
Player „X“: 5
1 2 O
4 X 6
7 8 9
Player „O“: 3
1 X O
4 X 6
7 8 9
Player „X“: 2


1 2 3
4 5 6
7 8 9
→
1 2 3
4 X 6
7 8 9
Player „X“: 5
→
1 2 O
4 X 6
7 8 9
Player „O“: 3
→
1 X O
4 X 6
7 8 9
Player „X“: 2
→
1 X O
4 X 6
7 O 9
Player „O“: 8

Übungsblatt 6 - Aufgabe 1a)

Übungsblatt 6 - Aufgabe 1a)

- We wanta print a board (TicTacToe.java):

Übungsblatt 6 - Aufgabe 1a)

- We wanta print a board (TicTacToe.java):

```
public static void printBoard(char[][] board) {









}
```

Übungsblatt 6 - Aufgabe 1a)

- We wanta print a board (TicTacToe.java):

```
public static void printBoard(char[][] board) {
    for (char[] row : board) {




    }

}
```

Übungsblatt 6 - Aufgabe 1a)

- We wanta print a board (TicTacToe.java):

```
public static void printBoard(char[][] board) {
    for (char[] row : board) {
        System.out.println("+---+---+---+");




    }

}
```

Übungsblatt 6 - Aufgabe 1a)

- We wanta print a board (TicTacToe.java):

```
public static void printBoard(char[][] board) {
    for (char[] row : board) {
        System.out.println("+---+---+---+");
        for (char cell : row)
            System.out.print("| " + (cell != 0 ? cell : ' ') + " ");

    }

}
```

Übungsblatt 6 - Aufgabe 1a)

- We wanta print a board (TicTacToe.java):

```
public static void printBoard(char[][] board) {
    for (char[] row : board) {
        System.out.println("+---+---+---+");
        for (char cell : row)
            System.out.print("| " + (cell != 0 ? cell : ' ') + " ");
        System.out.println("|");
    }

}
```

Übungsblatt 6 - Aufgabe 1a)

- We wanta print a board (TicTacToe.java):

```
public static void printBoard(char[][] board) {
    for (char[] row : board) {
        System.out.println("+---+---+---+");
        for (char cell : row)
            System.out.print("| " + (cell != 0 ? cell : ' ') + " ");
        System.out.println("|");
    }
    System.out.println("+---+---+---+\n");
}
```

Übungsblatt 6 - Aufgabe 1a)

- We wanta print a board (TicTacToe.java):

1	2	3
4	5	6
7	8	9

```
public static void printBoard(char[][] board) {
    for (char[] row : board) {
        System.out.println("+---+---+---+");
        for (char cell : row)
            System.out.print("| " + (cell != 0 ? cell : ' ') + " ");
        System.out.println("|");
    }
    System.out.println("+---+---+---+\n");
}
```

Übungsblatt 6 - Aufgabe 1a)

- We wanta print a board (TicTacToe.java):

```
public static void printBoard(char[][] board) {
    for (char[] row : board) {
        System.out.println("+---+---+---+");
        for (char cell : row)
            System.out.print("| " + (cell != 0 ? cell : ' ') + " ");
        System.out.println("|");
    }
    System.out.println("+---+---+---+\n");
}
```

1	2	3
4	5	6
7	8	9

```
+---+---+---+
|   |   |   |
+---+---+---+
|   |   |   |
+---+---+---+
|   |   |   |
+---+---+---+
```

Übungsblatt 6 - Aufgabe 1b)

Übungsblatt 6 - Aufgabe 1b)

- Request from the best:

Übungsblatt 6 - Aufgabe 1b)

- Request from the best:

```
public static char[][] getMove(int player, char[][] board) {




}
```

Übungsblatt 6 - Aufgabe 1b)

- Request from the best:

```
public static char[][] getMove(int player, char[][] board) {
    System.out.println("Spieler " + player + " ist am Zug:");
    int move = 0;




}
```

Übungsblatt 6 - Aufgabe 1b)

- Request from the best:

```
public static char[][] getMove(int player, char[][] board) {
    System.out.println("Spieler " + player + " ist am Zug:");
    int move = 0;
    do {

    } while

}
```

Übungsblatt 6 - Aufgabe 1b)

- Request from the best:

```
public static char[][] getMove(int player, char[][] board) {
    System.out.println("Spieler " + player + " ist am Zug:");
    int move = 0;
    do {
        move = scanner.nextInt();
    } while

}
```

Übungsblatt 6 - Aufgabe 1b)

- Request from the best:

```
public static char[][] getMove(int player, char[][] board) {
    System.out.println("Spieler " + player + " ist am Zug:");
    int move = 0;
    do {
        move = scanner.nextInt();
    } while (!isMoveValid(board, move));

}
```

Wir versuchen **lesbaren Code** zu schreiben. Anstelle einfach `isMoveValid` und `makeMove` zu integrieren, lagern wir den Code in Subroutinen aus, welche dessen Semantik als Namen tragen. In späteren Veranstaltungen ist dies im Rahmen der **Testbarkeit** und **Lesbarkeit** von besonderem Interesse.

Übungsblatt 6 - Aufgabe 1b)

- Request from the best:

```
public static char[][] getMove(int player, char[][] board) {
    System.out.println("Spieler " + player + " ist am Zug:");
    int move = 0;
    do {
        move = scanner.nextInt();
    } while (!isMoveValid(board, move));
    return makeMove(board, move, player);
}
```

Wir versuchen **lesbaren Code** zu schreiben. Anstelle einfach `isMoveValid` und `makeMove` zu integrieren, lagern wir den Code in Subroutinen aus, welche dessen Semantik als Namen tragen. In späteren Veranstaltungen ist dies im Rahmen der **Testbarkeit** und **Lesbarkeit** von besonderem Interesse.

Übungsblatt 6 - Aufgabe 1b)

Übungsblatt 6 - Aufgabe 1b)

- Check it and regret it:

Übungsblatt 6 - Aufgabe 1b)

- Check it and regret it:

```
private static boolean isMoveValid(char[][] board, int move) {


}
```

Übungsblatt 6 - Aufgabe 1b)

- Check it and regret it:

```
private static boolean isMoveValid(char[][] board, int move) {
    return move >= 1 && move <= 9

}
```

Übungsblatt 6 - Aufgabe 1b)

- Check it and regret it:

```
private static boolean isMoveValid(char[][] board, int move) {
    return move >= 1 && move <= 9
        && board[getY(move)][getX(move)] == 0;
}
```

Übungsblatt 6 - Aufgabe 1b)

- Check it and regret it:

```
private static boolean isMoveValid(char[][] board, int move) {
    return move >= 1 && move <= 9
        && board[getY(move)][getX(move)] == 0;
}

public static int getX(int cellNumber) {

}
```

Übungsblatt 6 - Aufgabe 1b)

- Check it and regret it:

```
private static boolean isMoveValid(char[][] board, int move) {
    return move >= 1 && move <= 9
        && board[getY(move)][getX(move)] == 0;
}

public static int getX(int cellNumber) {
    return (cellNumber - 1) % 3;
}
```

Übungsblatt 6 - Aufgabe 1b)

- Check it and regret it:

```
private static boolean isMoveValid(char[][] board, int move) {
    return move >= 1 && move <= 9
        && board[getY(move)][getX(move)] == 0;
}

public static int getX(int cellNumber) {
    return (cellNumber - 1) % 3;
}

public static int getY(int cellNumber) {
    return (cellNumber - 1) / 3;
}
```

1	2	3
4	5	6
7	8	9

Übungsblatt 6 - Aufgabe 1b)

Übungsblatt 6 - Aufgabe 1b)

- Execute Order 66:

Übungsblatt 6 - Aufgabe 1b)

- Execute Order 66:

```
private static char[][] makeMove(char[][] board, int cellNumber, int player) {



}
```

- Wo wir schon bei Star Wars™sind…

Übungsblatt 6 - Aufgabe 1b)

- Execute Order 66:

```
private static char[][] makeMove(char[][] board, int cellNumber, int player) {
    char[][] newBoard = board.clone(); // odeeeer?


}
```

- Wo wir schon bei Star Wars™sind…
- Have you ever heard the tragedy of Darth Clone**a**able the wise?

Übungsblatt 6 - Aufgabe 1b)

- Execute Order 66:

```
private static char[][] makeMove(char[][] board, int cellNumber, int player) {
    char[][] newBoard = board.clone(); // odeeeer?
    newBoard[getY(cellNumber)][getX(cellNumber)] = getPlayerSymbol(player);

}
```

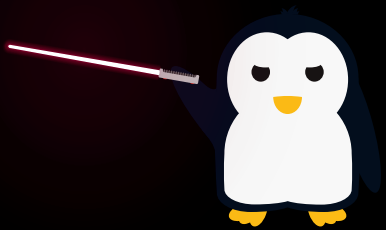
- Wo wir schon bei Star Wars™sind…
- Have you ever heard the tragedy of Darth Cloneable the wise? Well, then prepare yourself for… something at least.

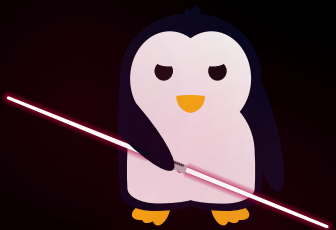
Übungsblatt 6 - Aufgabe 1b)

- Execute Order 66:

```
private static char[][] makeMove(char[][] board, int cellNumber, int player) {
    char[][] newBoard = board.clone(); // odeeeer?
    newBoard[getY(cellNumber)][getX(cellNumber)] = getPlayerSymbol(player);
    return newBoard;
}
```

- Wo wir schon bei Star Wars™sind…
- Have you ever heard the tragedy of Darth Clone**a**able the wise? Well, then prepare yourself for… something at least.






clone


clone

Präsenzaufgabe - Exkurs: Object

Präsenzaufgabe - Exkurs: Object

- `Object` ist die Klasse, welche die grundlegenden Eigenschaften für jede andere Klassen zur Verfügung stellt.

Präsenzaufgabe - Exkurs: Object

- `Object` ist die Klasse, welche die grundlegenden Eigenschaften für jede andere Klassen zur Verfügung stellt.
- Wir werden sie später im Rahmen von Vererbung genauer kennen lernen.
  Bis da hin: `toString` stammt beispielsweise dort her.

Präsenzaufgabe - Exkurs: Object

- `Object` ist die Klasse, welche die grundlegenden Eigenschaften für jede andere Klassen zur Verfügung stellt.
- Wir werden sie später im Rahmen von Vererbung genauer kennen lernen.
  Bis da hin: `toString` stammt beispielsweise dort her.
- Diese Klasse `Object` liefert auch die Methode `clone`, die auf jedem Objekt aufgerufen werden kann

Präsenzaufgabe - Exkurs: Object

- `Object` ist die Klasse, welche die grundlegenden Eigenschaften für jede andere Klassen zur Verfügung stellt.
- Wir werden sie später im Rahmen von Vererbung genauer kennen lernen.
  Bis da hin: `toString` stammt beispielsweise dort her.
- Diese Klasse `Object` liefert auch die Methode `clone`, die auf jedem Objekt aufgerufen werden kann…

Präsenzaufgabe - Exkurs: Object

- `Object` ist die Klasse, welche die grundlegenden Eigenschaften für jede andere Klassen zur Verfügung stellt.
- Wir werden sie später im Rahmen von Vererbung genauer kennen lernen.
  Bis da hin: `toString` stammt beispielsweise dort her.
- Diese Klasse `Object` liefert auch die Methode `clone`, die auf jedem Objekt aufgerufen werden kann… …

Präsenzaufgabe - Exkurs: Object

- `Object` ist die Klasse, welche die grundlegenden Eigenschaften für jede andere Klassen zur Verfügung stellt.
- Wir werden sie später im Rahmen von Vererbung genauer kennen lernen.
  Bis da hin: `toString` stammt beispielsweise dort her.
- Diese Klasse `Object` liefert auch die Methode `clone`, die auf jedem Objekt aufgerufen werden kann… … …

Präsenzaufgabe - Exkurs: Object

- `Object` ist die Klasse, welche die grundlegenden Eigenschaften für jede andere Klassen zur Verfügung stellt.
- Wir werden sie später im Rahmen von Vererbung genauer kennen lernen.
  Bis da hin: `toString` stammt beispielsweise dort her.
- Diese Klasse `Object` liefert auch die Methode `clone`, die auf jedem Objekt aufgerufen werden kann… … …
- `clone` muss aber von jeder Klasse selbst und individuell implementiert werden, wenn es denn unterstützt werden soll.

Präsenzaufgabe - Exkurs: Kopien

Präsenzaufgabe - Exkurs: Kopien

- Da die Variable eines Objekts in Java nur eine Referenz auf das eigentliche Objekt enthält, erzeugen wir durch den folgenden Code *keine* Kopie:

Präsenzaufgabe - Exkurs: Kopien

- Da die Variable eines Objekts in Java nur eine Referenz auf das eigentliche Objekt enthält, erzeugen wir durch den folgenden Code *keine* Kopie:

```
Scanner a = new Scanner(System.in);
Scanner b = a;
```

Präsenzaufgabe - Exkurs: Kopien

- Da die Variable eines Objekts in Java nur eine Referenz auf das eigentliche Objekt enthält, erzeugen wir durch den folgenden Code *keine* Kopie:

```
Scanner a = new Scanner(System.in);
Scanner b = a;
```

- Oft stellen sich auch Fragen wie:

Präsenzaufgabe - Exkurs: Kopien

- Da die Variable eines Objekts in Java nur eine Referenz auf das eigentliche Objekt enthält, erzeugen wir durch den folgenden Code *keine* Kopie:

```
Scanner a = new Scanner(System.in);
Scanner b = a;
```

- Oft stellen sich auch Fragen wie:
  - Was soll kopiert werden?

Präsenzaufgabe - Exkurs: Kopien

- Da die Variable eines Objekts in Java nur eine Referenz auf das eigentliche Objekt enthält, erzeugen wir durch den folgenden Code *keine* Kopie:

```
Scanner a = new Scanner(System.in);
Scanner b = a;
```

- Oft stellen sich auch Fragen wie:
  - Was soll kopiert werden?
  - Funktioniert eine Kopie überhaupt?

Präsenzaufgabe - Exkurs: Kopien

- Da die Variable eines Objekts in Java nur eine Referenz auf das eigentliche Objekt enthält, erzeugen wir durch den folgenden Code *keine* Kopie:

```
Scanner a = new Scanner(System.in);
Scanner b = a;
```

- Oft stellen sich auch Fragen wie:
  - Was soll kopiert werden?
  - Funktioniert eine Kopie überhaupt?
  - Was ist, wenn das zu kopierende Objekt selbst wieder (zum Beispiel in den Attributen) Referenzen auf andere Objekte enthält?

Präsenzaufgabe - Exkurs: Kopien

- Da die Variable eines Objekts in Java nur eine Referenz auf das eigentliche Objekt enthält, erzeugen wir durch den folgenden Code *keine* Kopie:

```
Scanner a = new Scanner(System.in);
Scanner b = a;
```

- Oft stellen sich auch Fragen wie:
  - Was soll kopiert werden?
  - Funktioniert eine Kopie überhaupt?
  - Was ist, wenn das zu kopierende Objekt selbst wieder (zum Beispiel in den Attributen) Referenzen auf andere Objekte enthält?
  - Was, wenn diese Referenzen zirkulär sind?

Präsenzaufgabe - Exkurs: Kopien

Präsenzaufgabe - Exkurs: Kopien

- Deswegen unterscheiden wir zwei Arten von Kopien: *shallow* und *deep*.

  (Genau genommen gibt es noch viel mehr, wie zum Beispiel *lazy*, aber das soll uns hier nicht weiter stören.)

Präsenzaufgabe - Exkurs: Kopien

- Deswegen unterscheiden wir zwei Arten von Kopien: *shallow* und *deep*.

  (Genau genommen gibt es noch viel mehr, wie zum Beispiel *lazy*, aber das soll uns hier nicht weiter stören.)
- Eine *shallow copy* kopiert „so wenig wie möglich“. Oder auch nur „die erste Hierarchieebene.“

Präsenzaufgabe - Exkurs: Kopien

- Deswegen unterscheiden wir zwei Arten von Kopien: *shallow* und *deep*.

  (Genau genommen gibt es noch viel mehr, wie zum Beispiel *lazy*, aber das soll uns hier nicht weiter stören.)
- Eine *shallow copy* kopiert „so wenig wie möglich“. Oder auch nur „die erste Hierarchieebene.“
- Eine *deep copy* kopiert das komplette Objekt, sowie alle Objekt-Referenzen die dieses Objekt wieder besitzt und so weiter.

Präsenzaufgabe - Exkurs: Kopien

- Deswegen unterscheiden wir zwei Arten von Kopien: *shallow* und *deep*.
  (Genau genommen gibt es noch viel mehr, wie zum Beispiel *lazy*, aber das soll uns hier nicht weiter stören.)
- Eine *shallow copy* kopiert „so wenig wie möglich“. Oder auch nur „die erste Hierarchieebene.“
- Eine *deep copy* kopiert das komplette Objekt, sowie alle Objekt-Referenzen die dieses Objekt wieder besitzt und so weiter.
- Während shallow copies meist von Hand geschrieben werden,

Präsenzaufgabe - Exkurs: Kopien

- Deswegen unterscheiden wir zwei Arten von Kopien: *shallow* und *deep*.

  (Genau genommen gibt es noch viel mehr, wie zum Beispiel *lazy*, aber das soll uns hier nicht weiter stören.)
- Eine *shallow copy* kopiert „so wenig wie möglich“. Oder auch nur „die erste Hierarchieebene.“
- Eine *deep copy* kopiert das komplette Objekt, sowie alle Objekt-Referenzen die dieses Objekt wieder besitzt und so weiter.
- Während shallow copies meist von Hand geschrieben werden, wird eine deep copy meistens durch (De-)Serialisierung gelöst.

Präsenzaufgabe - Exkurs: Kopien

- Deswegen unterscheiden wir zwei Arten von Kopien: *shallow* und *deep*.

  (Genau genommen gibt es noch viel mehr, wie zum Beispiel *lazy*, aber das soll uns hier nicht weiter stören.)
- Eine *shallow copy* kopiert „so wenig wie möglich“. Oder auch nur „die erste Hierarchieebene.“
- Eine *deep copy* kopiert das komplette Objekt, sowie alle Objekt-Referenzen die dieses Objekt wieder besitzt und so weiter.
- Während shallow copies meist von Hand geschrieben werden, wird eine deep copy meistens durch (De-)Serialisierung gelöst.

  (Sonst müssten bei einer deep copy auch alle vom Objekt verwendeten Ressourcen wieder deep-copyable sein.

Präsenzaufgabe - Exkurs: Kopien

- Deswegen unterscheiden wir zwei Arten von Kopien: *shallow* und *deep*.

  (Genau genommen gibt es noch viel mehr, wie zum Beispiel *lazy*, aber das soll uns hier nicht weiter stören.)
- Eine *shallow copy* kopiert „so wenig wie möglich“. Oder auch nur „die erste Hierarchieebene.“
- Eine *deep copy* kopiert das komplette Objekt, sowie alle Objekt-Referenzen die dieses Objekt wieder besitzt und so weiter.
- Während shallow copies meist von Hand geschrieben werden, wird eine deep copy meistens durch (De-)Serialisierung gelöst.

  (Sonst müssten bei einer deep copy auch alle vom Objekt verwendeten Ressourcen wieder deep-copyable sein. Das Verfahren hat allerdings ebenfalls Nachteile. Stichwort: `transient`.)

Präsenzaufgabe - Exkurs: Kopien, Beispiel

Präsenzaufgabe - Exkurs: Kopien, Beispiel

- Betrachten wir die folgende Klasse (`Y` und `Z` seien ebenfalls gegeben):

Präsenzaufgabe - Exkurs: Kopien, Beispiel

- Betrachten wir die folgende Klasse (`Y` und `Z` seien ebenfalls gegeben):

```
class X {
    int a;
    Y y;
    Z z;
}
```

Präsenzaufgabe - Exkurs: Kopien, Beispiel

- Betrachten wir die folgende Klasse (`Y` und `Z` seien ebenfalls gegeben):

```
class X {
    int a;
    Y y;
    Z z;
}
```

- Eine *shallow copy* würde ein neues `X`-Objekt erzeugen und die Attribute durch „=“ zuweisen.

Präsenzaufgabe - Exkurs: Kopien, Beispiel

- Betrachten wir die folgende Klasse (`Y` und `Z` seien ebenfalls gegeben):

```
class X {
    int a;
    Y y;
    Z z;
}
```

- Eine *shallow copy* würde ein neues `X`-Objekt erzeugen und die Attribute durch „=“ zuweisen. Damit wird `a` kopiert, `y` und `z` referenzieren aber (je) dasselbe Objekt.

Präsenzaufgabe - Exkurs: Kopien, Beispiel

- Betrachten wir die folgende Klasse (`Y` und `Z` seien ebenfalls gegeben):

```
class X {
    int a;
    Y y;
    Z z;
}
```

- Eine *shallow copy* würde ein neues `X`-Objekt erzeugen und die Attribute durch „=“ zuweisen. Damit wird `a` kopiert, `y` und `z` referenzieren aber (je) dasselbe Objekt.
- Eine *deep copy* würde rekursiv auch neue Objekte von `Y` und `Z` erzeugen.

Präsenzaufgabe - Exkurs: Kopien, Beispiel

- Betrachten wir die folgende Klasse (`Y` und `Z` seien ebenfalls gegeben):

```
class X {
    int a;
    Y y;
    Z z;
}
```

- Eine *shallow copy* würde ein neues `X`-Objekt erzeugen und die Attribute durch „=“ zuweisen. Damit wird `a` kopiert, `y` und `z` referenzieren aber (je) dasselbe Objekt.
- Eine *deep copy* würde rekursiv auch neue Objekte von `Y` und `Z` erzeugen.

  (Quizfrage: was passiert oder besser, was kann alles passieren, wenn `Y` wieder eine Referenz auf `X` enthält?)

Präsenzaufgabe - Exkurs: Kopien, Beispiel

- Betrachten wir die folgende Klasse (`Y` und `Z` seien ebenfalls gegeben):

```
class X {
    int a;
    Y y;
    Z z;
}
```

- Eine *shallow copy* würde ein neues `X`-Objekt erzeugen und die Attribute durch „=“ zuweisen. Damit wird `a` kopiert, `y` und `z` referenzieren aber (je) dasselbe Objekt.
- Eine *deep copy* würde rekursiv auch neue Objekte von `Y` und `Z` erzeugen.

  (Quizfrage: was passiert oder besser, was kann alles passieren, wenn `Y` wieder eine Referenz auf `X` enthält?)
- Und was davon macht jetzt `Object::clone()`?

Präsenzaufgabe - Exkurs: Kopien, Beispiel

- Betrachten wir die folgende Klasse (`Y` und `Z` seien ebenfalls gegeben):

```
class X {
    int a;
    Y y;
    Z z;
}
```

- Eine *shallow copy* würde ein neues `X`-Objekt erzeugen und die Attribute durch „=“ zuweisen. Damit wird `a` kopiert, `y` und `z` referenzieren aber (je) dasselbe Objekt.
- Eine *deep copy* würde rekursiv auch neue Objekte von `Y` und `Z` erzeugen.

  (Quizfrage: was passiert oder besser, was kann alles passieren, wenn `Y` wieder eine Referenz auf `X` enthält?)
- Und was davon macht jetzt `Object::clone()`? Spoiler: Nobody knows.

Präsenzaufgabe - Exkurs: Cloneable

Präsenzaufgabe - Exkurs: Cloneable

- Das Interface Cloneable *fordert nichts* von einer gegebene Klasse.

Präsenzaufgabe - Exkurs: Cloneable

- Das Interface Cloneable *fordert nichts* von einer gegebene Klasse.
- Es ist ein sogenanntes „Marker“ interface.

Präsenzaufgabe - Exkurs: Cloneable

- Das Interface Cloneable *fordert nichts* von einer gegebene Klasse.
- Es ist ein sogenanntes „Marker“ interface. Wer es implementiert, sorgt dafür, dass gewisse Methoden sich einem gewissen Vertrag unterwerfen.

Präsenzaufgabe - Exkurs: Cloneable

- Das Interface Cloneable *fordert nichts* von einer gegebene Klasse.
- Es ist ein sogenanntes „Marker“ interface. Wer es implementiert, sorgt dafür, dass gewisse Methoden sich einem gewissen Vertrag unterwerfen.

  (Diese Erklärung ist an sich nachträglich entstanden. Die komplette Geschichte um das `Cloneable`-Interface ist eine einzige Tragödie.)

Präsenzaufgabe - Exkurs: Cloneable

- Das Interface Cloneable *fordert nichts* von einer gegebene Klasse.
- Es ist ein sogenanntes „Marker“ interface. Wer es implementiert, sorgt dafür, dass gewisse Methoden sich einem gewissen Vertrag unterwerfen.

  (Diese Erklärung ist an sich nachträglich entstanden. Die komplette Geschichte um das `Cloneable`-Interface ist eine einzige Tragödie.)
- Aus dem Java-Contract:

Präsenzaufgabe - Exkurs: Cloneable

- Das Interface Cloneable *fordert nichts* von einer gegebene Klasse.
- Es ist ein sogenanntes „Marker“ interface. Wer es implementiert, sorgt dafür, dass gewisse Methoden sich einem gewissen Vertrag unterwerfen.

  (Diese Erklärung ist an sich nachträglich entstanden. Die komplette Geschichte um das `Cloneable`-Interface ist eine einzige Tragödie.)
- Aus dem Java-Contract:

```
/*
 * Note that this interface does not contain the clone method.
 * Therefore, it is not possible to clone an object merely by virtue of the
 * fact that it implements this interface.
```

Präsenzaufgabe - Exkurs: Cloneable

- Das Interface Cloneable *fordert nichts* von einer gegebene Klasse.
- Es ist ein sogenanntes „Marker“ interface. Wer es implementiert, sorgt dafür, dass gewisse Methoden sich einem gewissen Vertrag unterwerfen.

  (Diese Erklärung ist an sich nachträglich entstanden. Die komplette Geschichte um das `Cloneable`-Interface ist eine einzige Tragödie.)
- Aus dem Java-Contract:

```
/*
 * Note that this interface does not contain the clone method.
 * Therefore, it is not possible to clone an object merely by virtue of the
 * fact that it implements this interface.  Even if the clone method is invoked
 * reflectively, there is no guarantee that it will succeed.
 */
```

Präsenzaufgabe - Exkurs: Cloneable, Schadensbericht

Präsenzaufgabe - Exkurs: Cloneable, Schadensbericht

- Das Interface widerspricht der sonstigen Schreibweise.

Präsenzaufgabe - Exkurs: Cloneable, Schadensbericht

- Das Interface widerspricht der sonstigen Schreibweise. (Clonable vs. Cloneable.)

Präsenzaufgabe - Exkurs: Cloneable, Schadensbericht

- Das Interface widerspricht der sonstigen Schreibweise. (Clonable vs. Cloneable.)
- Es fordert keine Methode und kann theoretisch von jeder Klasse implementiert werden.

Präsenzaufgabe - Exkurs: Cloneable, Schadensbericht

- Das Interface widerspricht der sonstigen Schreibweise. (Clonable vs. Cloneable.)
- Es fordert keine Methode und kann theoretisch von jeder Klasse implementiert werden.
- Es liefert keine weiteren Informationen über die Kopie.

Präsenzaufgabe - Exkurs: Cloneable, Schadensbericht

- Das Interface widerspricht der sonstigen Schreibweise. (Clonable vs. Cloneable.)
- Es fordert keine Methode und kann theoretisch von jeder Klasse implementiert werden.
- Es liefert keine weiteren Informationen über die Kopie.
- Es stellt keine Anforderungen an die Java-Syntax (Sichtbarkeit), die nicht gleichermaßen geprüft werden können.

Präsenzaufgabe - Exkurs: Cloneable, Schadensbericht

- Das Interface widerspricht der sonstigen Schreibweise. (Clonable vs. Cloneable.)
- Es fordert keine Methode und kann theoretisch von jeder Klasse implementiert werden.
- Es liefert keine weiteren Informationen über die Kopie.
- Es stellt keine Anforderungen an die Java-Syntax (Sichtbarkeit), die nicht gleichermaßen geprüft werden können.
- Der Rückgabewert der `Object::clone`-Methode ist stets `Object`, wir brauchen also einen expliziten Cast.

Präsenzaufgabe - Exkurs: Cloneable, Schadensbericht

- Das Interface widerspricht der sonstigen Schreibweise. (Clonable vs. Cloneable.)
- Es fordert keine Methode und kann theoretisch von jeder Klasse implementiert werden.
- Es liefert keine weiteren Informationen über die Kopie.
- Es stellt keine Anforderungen an die Java-Syntax (Sichtbarkeit), die nicht gleichermaßen geprüft werden können.
- Der Rückgabewert der `Object::clone`-Methode ist stets `Object`, wir brauchen also einen expliziten Cast. (Wie bei `(Date) this.birthday.clone()`.)

Präsenzaufgabe - Exkurs: Cloneable, Schadensbericht

- Das Interface widerspricht der sonstigen Schreibweise. (Clonable vs. Cloneable.)
- Es fordert keine Methode und kann theoretisch von jeder Klasse implementiert werden.
- Es liefert keine weiteren Informationen über die Kopie.
- Es stellt keine Anforderungen an die Java-Syntax (Sichtbarkeit), die nicht gleichermaßen geprüft werden können.
- Der Rückgabewert der `Object::clone`-Methode ist stets `Object`, wir brauchen also einen expliziten Cast. (Wie bei `(Date) this.birthday.clone()`.)
- …

Präsenzaufgabe - Exkurs: Cloneable, Schadensbericht

- Das Interface widerspricht der sonstigen Schreibweise. (Clonable vs. Cloneable.)
- Es fordert keine Methode und kann theoretisch von jeder Klasse implementiert werden.
- Es liefert keine weiteren Informationen über die Kopie.
- Es stellt keine Anforderungen an die Java-Syntax (Sichtbarkeit), die nicht gleichermaßen geprüft werden können.
- Der Rückgabewert der `Object::clone`-Methode ist stets `Object`, wir brauchen also einen expliziten Cast. (Wie bei `(Date) this.birthday.clone()`.)
- … …

Präsenzaufgabe - Exkurs: Cloneable, Schadensbericht

- Das Interface widerspricht der sonstigen Schreibweise. (Clonable vs. Cloneable.)
- Es fordert keine Methode und kann theoretisch von jeder Klasse implementiert werden.
- Es liefert keine weiteren Informationen über die Kopie.
- Es stellt keine Anforderungen an die Java-Syntax (Sichtbarkeit), die nicht gleichermaßen geprüft werden können.
- Der Rückgabewert der `Object::clone`-Methode ist stets `Object`, wir brauchen also einen expliziten Cast. (Wie bei `(Date) this.birthday.clone()`.)
- … … …

Präsenzaufgabe - Exkurs: Long story short

Präsenzaufgabe - Exkurs: Long story short

- Manche Java Klassen implementieren `Cloneable` korrekt.

Präsenzaufgabe - Exkurs: Long story short

- Manche Java Klassen implementieren `Cloneable` korrekt. (Wie `Date`.)

Präsenzaufgabe - Exkurs: Long story short

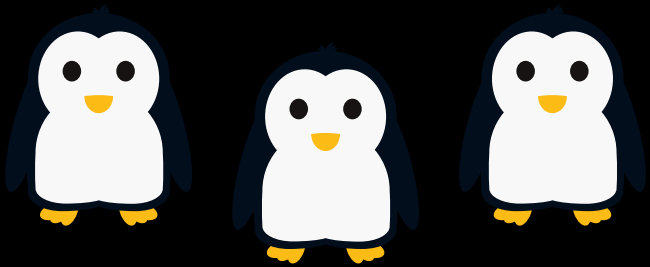
- Manche Java Klassen implementieren `Cloneable` korrekt. (Wie `Date`.)
- Ansonsten haben sich andere Prinzipien durchgesetzt. So beispielsweise Copy-Konstruktoren.

Präsenzaufgabe - Exkurs: Long story short

- Manche Java Klassen implementieren `Cloneable` korrekt. (Wie `Date`.)
- Ansonsten haben sich andere Prinzipien durchgesetzt. So beispielsweise Copy-Konstruktoren.
- Diesen werden noch später im Semester begegnen.

Präsenzaufgabe - Exkurs: Long story short

- Manche Java Klassen implementieren `Cloneable` korrekt. (Wie `Date`.)
- Ansonsten haben sich andere Prinzipien durchgesetzt. So beispielsweise Copy-Konstruktoren.
- Diesen werden noch später im Semester begegnen.
- Also kehren wir uns lieber von der dunklen Seite ab und zur Aufgabe zurück…



Übungsblatt 6 - Aufgabe 1b) Fortsetzung

Übungsblatt 6 - Aufgabe 1b) Fortsetzung

- Outstanding Deep-Copy:

Übungsblatt 6 - Aufgabe 1b) Fortsetzung

- Outstanding Deep-Copy:

```
char[][] newBoard = new char[3][3];
```

Übungsblatt 6 - Aufgabe 1b) Fortsetzung

- Outstanding Deep-Copy:

```
char[][] newBoard = new char[3][3];
for (int i = 0; i < 3; i++) {



}
```

Übungsblatt 6 - Aufgabe 1b) Fortsetzung

- Outstanding Deep-Copy:

```
char[][] newBoard = new char[3][3];
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {

    }
}
```

Übungsblatt 6 - Aufgabe 1b) Fortsetzung

- Outstanding Deep-Copy:

```
char[][] newBoard = new char[3][3];
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        newBoard[i][j] = board[i][j];
    }
}
```

Übungsblatt 6 - Aufgabe 1b) Fortsetzung

- Outstanding Deep-Copy:

```
char[][] newBoard = new char[3][3];
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        newBoard[i][j] = board[i][j];
    }
}
```

- Wer `clone` doch benutzen „muss“:

Übungsblatt 6 - Aufgabe 1b) Fortsetzung

- Outstanding Deep-Copy:

```
char[][] newBoard = new char[3][3];
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        newBoard[i][j] = board[i][j];
    }
}
```

- Wer `clone` doch benutzen „muss“:

```
for(int i = 0; i < 3; i++) {

}
```

Übungsblatt 6 - Aufgabe 1b) Fortsetzung

- Outstanding Deep-Copy:

```
char[][] newBoard = new char[3][3];
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        newBoard[i][j] = board[i][j];
    }
}
```

- Wer `clone` doch benutzen „muss“:

```
for(int i = 0; i < 3; i++) {
    newBoard[i] = board[i].clone();
}
```

Übungsblatt 6 - Aufgabe 1c)

Übungsblatt 6 - Aufgabe 1c)

- Outstanding Move:

Übungsblatt 6 - Aufgabe 1c)

- Outstanding Move:

```
public static boolean winningMove(int player, char[][] board) {



}
```

Übungsblatt 6 - Aufgabe 1c)

- Outstanding Move:

```
public static boolean winningMove(int player, char[][] board) {
    char c = getPlayerSymbol(player);


}
```

Übungsblatt 6 - Aufgabe 1c)

- Outstanding Move:

```
public static boolean winningMove(int player, char[][] board) {
    char c = getPlayerSymbol(player);
    return winsHorizontal(board, c) || winsVertical(board, c)
        || winsDiagonal(board, c);
}
```

Übungsblatt 6 - Aufgabe 1c)

- Outstanding Move:

```
public static boolean winningMove(int player, char[][] board) {
    char c = getPlayerSymbol(player);
    return winsHorizontal(board, c) || winsVertical(board, c)
        || winsDiagonal(board, c);
}
```

- Für das Spielersymbol:

Übungsblatt 6 - Aufgabe 1c)

- Outstanding Move:

```
public static boolean winningMove(int player, char[][] board) {
    char c = getPlayerSymbol(player);
    return winsHorizontal(board, c) || winsVertical(board, c)
        || winsDiagonal(board, c);
}
```

- Für das Spielersymbol:

```
public static char getPlayerSymbol(int player) {


}
```

Übungsblatt 6 - Aufgabe 1c)

- Outstanding Move:

```
public static boolean winningMove(int player, char[][] board) {
    char c = getPlayerSymbol(player);
    return winsHorizontal(board, c) || winsVertical(board, c)
        || winsDiagonal(board, c);
}
```

- Für das Spielersymbol:

```
public static char getPlayerSymbol(int player) {
    return (player == 1) ? 'x' : 'o';
}
```

Übungsblatt 6 - Aufgabe 1c)

- Outstanding Move:

```
public static boolean winningMove(int player, char[][] board) {
    char c = getPlayerSymbol(player);
    return winsHorizontal(board, c) || winsVertical(board, c)
        || winsDiagonal(board, c);
}
```

- Für das Spielersymbol:

```
public static char getPlayerSymbol(int player) {
    return (player == 1) ? 'x' : 'o';
}
```

Hier könnte man auch eine Enumeration verwenden.

Übungsblatt 6 - Aufgabe 1c)

Übungsblatt 6 - Aufgabe 1c)

- Win me Baby:

Übungsblatt 6 - Aufgabe 1c)

- Win me Baby:

```
private static boolean winsHorizontal(char[][] board, char c) {



}
```

Übungsblatt 6 - Aufgabe 1c)

- Win me Baby:

```
private static boolean winsHorizontal(char[][] board, char c) {
  return ((board[0][0] == c) && (board[0][1] == c) && (board[0][2] == c)) ||
       ((board[1][0] == c) && (board[1][1] == c) && (board[1][2] == c)) ||
       ((board[2][0] == c) && (board[2][1] == c) && (board[2][2] == c));
}
```

Übungsblatt 6 - Aufgabe 1c)

- Win me Baby:

```
private static boolean winsHorizontal(char[][] board, char c) {
  return ((board[0][0] == c) && (board[0][1] == c) && (board[0][2] == c)) ||
       ((board[1][0] == c) && (board[1][1] == c) && (board[1][2] == c)) ||
       ((board[2][0] == c) && (board[2][1] == c) && (board[2][2] == c));
}

private static boolean winsVertical(char[][] board, char c) {



}
```

Übungsblatt 6 - Aufgabe 1c)

- Win me Baby:

```
private static boolean winsHorizontal(char[][] board, char c) {
  return ((board[0][0] == c) && (board[0][1] == c) && (board[0][2] == c)) ||
       ((board[1][0] == c) && (board[1][1] == c) && (board[1][2] == c)) ||
       ((board[2][0] == c) && (board[2][1] == c) && (board[2][2] == c));
}

private static boolean winsVertical(char[][] board, char c) {
  return ((board[0][0] == c) && (board[1][0] == c) && (board[2][0] == c)) ||
       ((board[0][1] == c) && (board[1][1] == c) && (board[2][1] == c)) ||
       ((board[0][2] == c) && (board[1][2] == c) && (board[2][2] == c));
}
```

Übungsblatt 6 - Aufgabe 1c)

- Win me Baby:

```
private static boolean winsHorizontal(char[][] board, char c) {
  return ((board[0][0] == c) && (board[0][1] == c) && (board[0][2] == c)) ||
       ((board[1][0] == c) && (board[1][1] == c) && (board[1][2] == c)) ||
       ((board[2][0] == c) && (board[2][1] == c) && (board[2][2] == c));
}

private static boolean winsVertical(char[][] board, char c) {
  return ((board[0][0] == c) && (board[1][0] == c) && (board[2][0] == c)) ||
       ((board[0][1] == c) && (board[1][1] == c) && (board[2][1] == c)) ||
       ((board[0][2] == c) && (board[1][2] == c) && (board[2][2] == c));
}

private static boolean winsDiagonal(char[][] board, char c) {


}
```

Übungsblatt 6 - Aufgabe 1c)

- Win me Baby:

```
private static boolean winsHorizontal(char[][] board, char c) {
  return ((board[0][0] == c) && (board[0][1] == c) && (board[0][2] == c)) ||
       ((board[1][0] == c) && (board[1][1] == c) && (board[1][2] == c)) ||
       ((board[2][0] == c) && (board[2][1] == c) && (board[2][2] == c));
}

private static boolean winsVertical(char[][] board, char c) {
  return ((board[0][0] == c) && (board[1][0] == c) && (board[2][0] == c)) ||
       ((board[0][1] == c) && (board[1][1] == c) && (board[2][1] == c)) ||
       ((board[0][2] == c) && (board[1][2] == c) && (board[2][2] == c));
}

private static boolean winsDiagonal(char[][] board, char c) {
  return ((board[0][0] == c) && (board[1][1] == c) && (board[2][2] == c)) ||
       ((board[2][0] == c) && (board[1][1] == c) && (board[0][2] == c));
}
```

Übungsblatt 6 - Aufgabe 1d)

Übungsblatt 6 - Aufgabe 1d)

- The main:

Übungsblatt 6 - Aufgabe 1d)

- The main:

```
int player = 1; // Start player 1
char[][] board = emptyTicTacToeBoard();
```

Übungsblatt 6 - Aufgabe 1d)

- The main:

```
int player = 1; // Start player 1
char[][] board = emptyTicTacToeBoard();

// Game loop
for (int i = 0; i < 9; i++) {









}
```

Übungsblatt 6 - Aufgabe 1d)

- The main:

```
int player = 1; // Start player 1
char[][] board = emptyTicTacToeBoard();

// Game loop
for (int i = 0; i < 9; i++) {
    board = getMove(player, board);









}
```

Übungsblatt 6 - Aufgabe 1d)

- The main:

```
int player = 1; // Start player 1
char[][] board = emptyTicTacToeBoard();

// Game loop
for (int i = 0; i < 9; i++) {
    board = getMove(player, board);
    printBoard(board);









}
```

Übungsblatt 6 - Aufgabe 1d)

- The main:

```
int player = 1; // Start player 1
char[][] board = emptyTicTacToeBoard();

// Game loop
for (int i = 0; i < 9; i++) {
    board = getMove(player, board);
    printBoard(board);
    if (winningMove(player, board)) {


    }

}
```

Übungsblatt 6 - Aufgabe 1d)

- The main:

```
int player = 1; // Start player 1
char[][] board = emptyTicTacToeBoard();

// Game loop
for (int i = 0; i < 9; i++) {
    board = getMove(player, board);
    printBoard(board);
    if (winningMove(player, board)) {
        System.out.println("Spieler " + player + " hat gewonnen!");
        return;
    }

}
```

Übungsblatt 6 - Aufgabe 1d)

- The main:

```
int player = 1; // Start player 1
char[][] board = emptyTicTacToeBoard();

// Game loop
for (int i = 0; i < 9; i++) {
    board = getMove(player, board);
    printBoard(board);
    if (winningMove(player, board)) {
        System.out.println("Spieler " + player + " hat gewonnen!");
        return;
    }
    player = otherPlayer(player);
}
```

Übungsblatt 6 - Aufgabe 1d)

- The main:

```
int player = 1; // Start player 1
char[][] board = emptyTicTacToeBoard();

// Game loop
for (int i = 0; i < 9; i++) {
    board = getMove(player, board);
    printBoard(board);
    if (winningMove(player, board)) {
        System.out.println("Spieler " + player + " hat gewonnen!");
        return;
    }
    player = otherPlayer(player);
}
System.out.println("Unentschieden");
```

Übungsblatt 6 - Aufgabe 1d)

- The main:

```
int player = 1; // Start player 1
char[][] board = emptyTicTacToeBoard();

// Game loop
for (int i = 0; i < 9; i++) {
    board = getMove(player, board);
    printBoard(board);
    if (winningMove(player, board)) {
        System.out.println("Spieler " + player + " hat gewonnen!");
        return;
    }
    player = otherPlayer(player);
}
System.out.println("Unentschieden");
scanner.close(); // wichtig
```

Übungsblatt 6 - Aufgabe 1d)

- The main:

```
int player = 1; // Start player 1
char[][] board = emptyTicTacToeBoard();

// Game loop
for (int i = 0; i < 9; i++) {
    board = getMove(player, board);
    printBoard(board);
    if (winningMove(player, board)) {
        System.out.println("Spieler " + player + " hat gewonnen!");
        return;
    }
    player = otherPlayer(player);
}
System.out.println("Unentschieden");
scanner.close(); // wichtig
```

Scanner::close ist eure **Lebensversicherung!** Schließt eine Ressource die ihr anfordert **immer** und **ohne Ausnahme** idealerweise hält die selbe Abstraktionsebene Anforderung und Abstoßung. Später machen das auto-closeables angenehmer.

Übungsblatt 6 - Aufgabe 1d)

Übungsblatt 6 - Aufgabe 1d)

- Initialize the board:

Übungsblatt 6 - Aufgabe 1d)

- Initialize the board:

```
public static char[][] emptyTicTacToeBoard() {




}
```

Übungsblatt 6 - Aufgabe 1d)

- Initialize the board:

```
public static char[][] emptyTicTacToeBoard() {
    char[][] board = new char[3][3];









}
```

Übungsblatt 6 - Aufgabe 1d)

- Initialize the board:

```
public static char[][] emptyTicTacToeBoard() {
    char[][] board = new char[3][3];
    for (int i = 0; i < 3; i++) {



    }

}
```

Übungsblatt 6 - Aufgabe 1d)

- Initialize the board:

```
public static char[][] emptyTicTacToeBoard() {
    char[][] board = new char[3][3];
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {

        }
    }

}
```

Übungsblatt 6 - Aufgabe 1d)

- Initialize the board:

```
public static char[][] emptyTicTacToeBoard() {
    char[][] board = new char[3][3];
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            board[i][j] = 0;
        }
    }

}
```

Übungsblatt 6 - Aufgabe 1d)

- Initialize the board:

```
public static char[][] emptyTicTacToeBoard() {
    char[][] board = new char[3][3];
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            board[i][j] = 0;
        }
    }
    return board;
}
```

Übungsblatt 6 - Aufgabe 1d)

- Initialize the board:

```
public static char[][] emptyTicTacToeBoard() {
    char[][] board = new char[3][3];
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            board[i][j] = 0;
        }
    }
    return board;
}
```

Ist eine solche Initialisierung **notwendig**? Nein. Java weist Variablen „Default-Werte“ zu („die 0“). Zahlen werden 0, booleans *false*, chars kriegen das Zeichen mit ASCII 0, ...

Dennoch, „explicit is better than implicit“. Andere Sprachen machen das andes, was das Lesen erschwert. Zudem zwingt explizit dazu, sich über den „Standardwert“ Gedanken zu machen.

Übungsblatt 6 - Aufgabe 1d)

Übungsblatt 6 - Aufgabe 1d)

- Toggle den Spieler in dir:

Übungsblatt 6 - Aufgabe 1d)

- Toggle den Spieler in dir:

```
public static int otherPlayer(int player) {

}
```

Übungsblatt 6 - Aufgabe 1d)

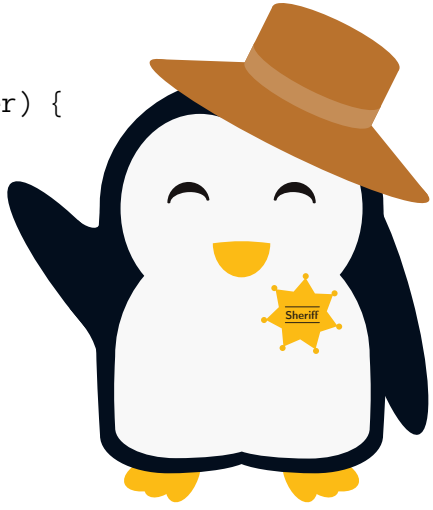
- Toggle den Spieler in dir:

```
public static int otherPlayer(int player) {
    return player == 1 ? 2 : 1;
}
```

Übungsblatt 6 - Aufgabe 1d)

- Toggle den Spieler in dir:

```
public static int otherPlayer(int player) {
    return player == 1 ? 2 : 1;
}
```



What if i don't have friends?

What if i don't have friends?

How about AI?

What if i don't have friends?

How about AI?

Maybe on a later day (die ganze LaTeX-KI umsonst).

Evaluation – Vorlesung

Evaluation – Vorlesung

https://t1p.de/n7mhl