Klausur der LV Multimedia-Engineering II im WS 09/10

Name:

Matr. Nr.:

Punkte: 9,5/10 Note 1.0

Datum: 23.01.2010

Dies ist mein 3. Prüfungsversuch zu dieser Lehrveranstaltung: Ja ☐ Nein ☒

Achtung, bearbeiten Sie die Klausur keinesfalls mit Bleistiften oder unter Verwendung roter Farbe. Geben Sie auf jeder Seite Ihren Namen und Ihre Matrikel-Nr. an. Es werden nur leicht lesbare Klausuren bewertet.

---

1. Frage zu ActionScript 3:

   a) Was bedeutet das Schlüsselwort *dynamic*?

   b) Gibt es eine Beziehung zwischen der Punkt- und der []-Notation bei Zugriff auf Instanzvariablen?

2. Frage zu Flex und MXML:

   a) Implementieren Sie eine minimale Flex-Applikation, die einen Button innerhalb eines Panels enthält.

   b) Wenn der Button angeklickt wird, soll im Panel ein zweiter Button erscheinen.

3. Frage zu Design Pattern:

   Das *State Design Pattern* demonstriert, wie sich mit Mitteln der objektorientierten Sprachen Fallunterscheidungen mit *if, else, switch, case* vermeiden lassen.

   Erläutern Sie anhand eines Code Snippets, wie das möglich ist.

23.01.10

Klausur Multi Media Engineering II

1.

a) dynamic bedeutet, dass eine Klasse dynamisch ist und man ihren Instanzen Methoden oder Instanzvariablen hinzufügen kann.

```
package {

    public dynamic class Position {

        var x:int;
        var y:int;

        public function Position (x:int, y:int) {

            this.x = x;
            this.y = y;
        }

    }

}
```

Test:

```
var pos: Position = new Position(1,2);

pos.z = 3;      // dynamische Variable

pos.neueMethode = function(): void{ // dynamische Methode

    pos.z = 5;

}
```

b) Bei assoziativen Arrays gilt hier:

```
arr.west == arr["west"];
```

1,5 / 2

```
<Application ...>
  <mx:Panel id="panel" x="0" y="0">
    <mx:Button click="neuerButton()" x="10"
               label="Drück mich" y="10" />
  </mx:Panel>
  <mx:SCRIPT>
    [CDATA]
      import mx.controls.Button;
      public function neuerButton():void{
        var butt:Button = new Button();
        butt.x = 10;
        butt.y = 100;
        butt.label = "Du hast gedrückt!";
        panel.addChild(butt);
        butt.addEventListener(MouseEvent.MOUSE_CLICK,
                              theWas());
      }
      public function theWas(event:MouseEvent):void{
        trace event.toString;
      }
    [/CDATA]
  </mx:SCRIPT>
</Application>
```

Natürlich muss dann noch MouseEvent importiert werden.

4/4

3.

```
public Interface IState {

    function stopToPlay();
    function playToStop();

}
```

Das Interface enthält für alle Zustandsübergänge Methodenschnittstellen

Dieses Interface muss von allen Zustandsklassen implementiert werden. So bald ein neuer Zustandsübergang hinzukommt, muss es dem Interface hinzugefügt werden. Wenn ein neuer Zustand benötigt wird, muss man eine neue Zustandsklasse, die das Interface implementiert, erzeugen.

```
public class Stop implements IState {
    public function stopToPlay():void {
        // starte Player
    }

    public function playToStop():void {
        // gib aus, dass der Player bereits gestoppt ist
    }

}
```

Wir brauchen auch einen Play-Zustand:

```
public class Play implements IState {
    public function stopToPlay():void {
        // gib aus, dass der Player bereits läuft
    }

    public function playToStop():void {
        // stoppe Player
    }
```

In der Main-Class braucht man von jeden Zustand eine Variable und einen aktuellen State, dem dann der jeweils aktuelle State übergeben wird.

```
public class Main {

    var stop : Stop = new Stop ();
    var play : Play = new Play ();
    var state : IState;

    public function setActualState (state : IState) : void {

        this.state = state;

    }

    public function getActualState () : IState {

        return state;

    }
```

So muss man sich jeweils nur den aktuelle State ausgeben lassen und ruft eine Methode aus der Schnittstelle auf (diese Methoden muss ja jede Zustandsklasse komplett implementieren)

```
state = getActualState ();
state.playToStop ();
```

Wenn state nun vom Typ Play ist wird der Player gestoppt und wenn vom Typ Stop wird ausgegeben dass er bereits gestoppt ist.

Da beide Klassen die Methoden haben muss man nicht fragen von welchen Typ state nun ist. Primus! 4/4