SoSe 2014
Prof. Dr. Margarita Esponda
ALP2
8. Übungsblatt (Abgabe am 18. Juni)

Ziel: Auseinandersetzung mit Objektorientierten Konzepten wie Vererbung, Schnittstellen, Konstruktoren, Instanz-Variablen und -Methoden.

1. Aufgabe (10 Punkte)

In der Vorlesung wurden die Klassen **Position**, **Walker** und **WalkerSpace** erläutert, die zusammen eine Reihe verrückte Läufer (**Walker**-Objekte) simulieren, die sich in 4 Himmelsrichtungen (Nord, Süd, Ost und West) innerhalb eines quadratischen Bereiches (**WalkerSpace**-Objekt) bewegen können.

Die Läufer markieren jede besuchte Position (**Position**-Objekt) mit ihrer eigenen Farbe und die Anzahl der bis jetzt gelaufenen Schritte. Die markierten Positionen dürfen nicht wieder betreten werden.

Nachdem sich die Läufer eine Weile bewegt haben, landen alle irgendwann in einer Sackgasse und die Simulation wird dann gestoppt.

a) Betrachten Sie die drei erwähnten Klassen und versuchen Sie, diese zuerst zu verstehen.

b) Definieren Sie dann eine **IWalkerSpace**-Klasse, als Unterklasse der **WalkerSpace** Klasse, die die Himmelsrichtungen, die die Läufer benutzen, auf Nordost, Nordwest, Südost und Südwest erweitert. Sie müssen dafür Instanz-Variablen und -Methoden der **WalkerSpace** Klasse in Ihre Unterklasse überschreiben und zusätzliche Methoden programmieren.

Zum Testen müssen Sie in der **Main**-Klasse ein **IWalkerSpace**-Objekt anstatt eine **WalkerSpace**-Objekt produzieren, damit das Programm Ihre neue Unterklasse verwendet. Keine weitere Klasse müssen verändert werden.

Die Klassen **WalkerPanel**, **WalkerWindow** und **WalkersAnim** sind nur zum Testen. Mit Hilfe dieser Klassen wird in einem Fenster der Verlauf visualisiert. Diese Klassen werden in der Vorlesung kurz erläutert.

c) Definieren Sie in Ihrer **IWalkerSpace**-Klasse eine Methode, die die durchschnittliche Weglänge der Läufer berechnet, nachdem alle Läufer stehen geblieben sind.

2. Aufgabe (24 Punkte)

In dieser Aufgabe sollen mindestens drei **Shape**-Klassen definiert werden, die folgende zwei Schnittstellen implementieren:

```
public interface  Shape {
    public void draw(Graphics g);
    public Point getCenter();
    public double getRadius();
```

```
        public Color  getColor();
        public void setShapesWorld( ShapesWorld theWorld );
        public void  userClicked ( double  atX, double  atY );
        public void userTyped( char key );
        public void moveTo( double x,  double y );
    }

    public interface  Animation {
        public static int sleep_time = 30;
        public void  play();
    }
```

Eine genaue Beschreibung der Semantik der einzelnen abstrakten Methoden befindet sich in den Dateien der Java-Interfaces und Java-Klassen, die für diese Aufgaben vorgegeben sind. Siehe unsere Website.

Es soll eine bereits vorhandene **World**-Klasse verwendet werden, die als Behälter innerhalb eines Fensters für die Visualisierung der **Shape**-Objekte zur Verfügung gestellt wird. In dem World-Objekt können sich die **Shape**-Objekte bewegen.

Innerhalb der Implementierung der **Shape**-Klassen kann die Referenz des **World**-Objekts verwendet werden, um mit anderen **Shape**-Objekten zu interagieren. Die Referenz zum **World**-Objekt wird in der **setWorld**-Methode des Interfaces als Argument übergeben und jedes Shape-Objekt soll diese lokal speichern. Die **setWorld**-Methode wird für jedes neue Shape-Objekt aufgerufen.

Folgende Methoden können mit einer Referenz des **World**-Objektes benutzt werden:

```
    public interface World {
        public double  getMin_X();
        public double  getMin_Y();
        public double  getMax_X();
        public double  getMax_Y();
        public  Shape getClosestShape (Shape currentShape);
        public void  addShape (Shape aNewShape);
        public void  removeShape (Shape shapeToBeRemoved);
    }
```

Eine einfache Shape-Implementierung wird zur Verfügung gestellt.

Das Programm muss mit der **ShapeWorld_Main**-Klasse gestartet werden und die implementierten **Shape**-Klassennamen müssen als Argumente der main-Methode angegeben werden. Wenn man mit Eclipse arbeitet, dann müssen die Argumente in der entsprechenden Eclipse-Ausführungskonfiguration eingegeben werden.

a) (5 Punkte) Programmieren Sie eine Klasse **Alien** für Aliens, die sich in alle Richtungen bewegen können, ohne das Fenster zu verlassen. Die **Alien**-Objekte sollen ein von Ihnen konzipierte originale Form haben.

b) (8 Punkte) Programmieren Sie eine Klasse **Panik**. Objekte von der **Panik**-Klasse versuchen andere Objekte zu vermeiden. Wenn sie keine Möglichkeit finden, sich von anderen Objekte zu entfernen, fangen sie an zu zittern, nach einer Weile explodieren sie und produzieren viele **PanikStuck**-Objekte, die nach unten fallen.

c) (6 Punkte) Programmieren Sie eine Klasse **Wrapper**. Die Objekte der **Wrapper**-Klasse sind durchsichtig und verpacken und begleiten für den Rest ihres Lebens die Objekte, die sie als erstes berühren. **Wrapper**-Objekte sollen, wenn diese noch niemand gefangen haben, die **ShapesWorld**-Fläche nicht verlassen.

d) (5 Punkte) Programmieren Sie eine **TimeWrapper** Klasse als Unterklasse der **Wrapper** Klasse, in dem eine zusätzlichen Zeiteigenschaft definiert wird. Bei den **TimeWrapper**-Objekten platzt die Verpackung nach einer Weile, die ursprünglichen Objekte werden befreit, und die **TimeWrapper**-Objekte produzieren mehrere **MiniWrapper**-Objekte. Die **MiniWraper**-Objekte werden mit der Zeit größer und ab einer bestimmten Größe verhalten sie sich wie **Wrapper**-Objekte.

e) (8 Bonuspunkte) Programmieren Sie 1 zusätzliche Klasse von **Shape**-Objekten mit einem Verhalten Ihrer Wahl.

Verwenden Sie dabei sinnvolle Instanzvariablen, die den Zustand Ihrer **Shape**-Objekte zwischenspeichern.

Definieren Sie zusätzliche Instanzmethoden, um das Verhalten Ihrer Shape-Objekte möglichst gut strukturiert zu programmieren.

Versuchen Sie möglichst viele Methoden der **World**-Klasse zu verwenden. Z.B. die **userClicked**-Methode, um ein interessantes Verhalten zu produzieren.

Die Shape-Objekte sollen mit viel Fantasie miteinander interagieren.

Wichtige Hinweise:

1) Verwenden Sie selbsterklärende Namen von Variablen und Methoden.

2) Für die Name aller Bezeichner müssen Sie die Java-Konventionen verwenden.

3) Verwenden Sie vorgegebene Klassen und Methodennamen.

4) Methoden sollten klein gehalten werden, sodass auf den ersten Blick ersichtlich ist, was diese Methode leistet.

5) Methoden sollten möglichst wenige Argumente haben.

6) Methoden sollten entweder den Zustand der Eingabeargumente ändern oder einen Rückgabewert liefern.

7) Verwenden Sie geeignete Hilfsvariablen und definieren Sie sinnvolle Hilfsmethoden in Ihren Klassendefinitionen.

8) Zahlen sollten durch Konstanten ersetzt werden.

9) Löschen Sie alle Programmzeilen und Variablen, die nicht verwendet werden.