1 Java Projekt vorbereiten

Die Erstellung von Projekten und das Importieren von Quelltexten hängt von der verwendeten Entwicklungsumgebung ab. Für Eclipse sehen Sie bitte in den Foliensatz zur Übung nach.

- Erstellen Sie ein Java-Projekt `labor` für alle vorgegebenen Klassen dieser Rechnerübungen.
- Laden Sie sich das ZIP-Archive `labor.zip` aus Ilias herunter. Importieren Sie alle Quelltexte in das Java Projekt. Die Klassen dieses Projekts dürfen nicht geändert werden.
- Erstellen Sie ein weiteres Java-Projekt `labor_loesung` für Ihre Lösungen.
- Konfigurieren Sie dieses Java-Projekt so, dass die Programme aus `labor` mitverwendet werden.
- Laden Sie sich das ZIP-Archive `labor_loesung.zip` aus Ilias herunter. Importieren Sie alle Quelltexte in das Java-Projekt `labor_loesung`.

2 Circle implementieren

Erstellen Sie eine Klasse `Circle` im Paket `de.hska.iwi.ads.solution.interfaces`. Diese Klasse soll von `de.hska.iwi.ads.interfaces.AbstractCircle` erben.

Implementieren Sie folgende Konstruktoren und Methoden. Fügen Sie keine Attribute oder Methoden zu Ihren Klassen hinzu. Beachten Sie auch die zugehörige Javadoc in AbstractCircle:

- `public Circle(Vector middlePoint, double radius)`: Erstellt einen neuen Kreis mit einem Mittelpunkt `middlePoint` und nicht negativem Radius.
- `public Circle(double radius)`: Erstellt einen neuen Kreis um den Null-Punkt und einem nicht negativem Radius. Verwenden Sie diesen Konstruktor bei den nachfolgenden Tests, um Objekte von Circle zu erstellen.
- `public void scale(double factor)`: Vergrößert ($factor > 1$) oder verkleinert ($0 \leq factor < 1$) den Radius um den gegebenen Faktor. Wenn der Faktor negativ ist, dann macht diese Methode nichts.
- `public double area()`: Gibt den Flächeninhalt dieses Kreises zurück ($radius^2 \cdot \Pi$). Sie finden die Kreiszahl Π als Konstante in `Math.PI`.
- `public double getDimension()` : Gibt den Radius dieses Kreises zurück.

3 Circle mit JUnit teilweise testen

Implementieren Sie in der Klasse `de.hska.iwi.ads.solution.interfaces.CircleTest` die vorgegebenen noch nicht programmierten Tests

Führe Sie diese Tests aus. Falls Tests fehlschlagen, suchen sie erst einen Fehler in der fehlgeschlagenen Testmethode, bevor sie den Fehler in Ihrer Circle-Implementierung suchen.

Sie können noch weitere Tests hinzufügen, um Fehler in Ihrer Circle-Implementierung zu finden. Zu jeder öffentlichen Methode und Konstruktor sollte mindestens eine Testmethode implementiert werden.

4 Vector mit kartesischen Koordinaten

`de.hska.iwi.ads.interfaces.Vector` definiert einen zwei-dimensionalen Vektor im euklidischen Raum. Die Länge eines Vektors kann mit einem Faktor skaliert werden. Dies ist in `de.hska.iwi.ads.interfaces.Scalable` definiert. Vector erweitert dieses Interface um zusätzliche Methoden.

Erstellen Sie eine Klasse `de.hska.iwi.ads.solution.interfaces.CartesianVector`, die das Interface Vector implementiert. `CartesianVector` soll einen Vektor mit Hilfe kartesischer x- und y-Koordinaten implementieren.

Die Richtung `getDirection()` ist ein Winkel im Bogenmaß innerhalb des Wertebereichs $[-\Pi..\Pi]$. Die Definition dieses Winkels ist so gewählt, dass er direkt mit `Math.atan2(y,x)` berechnet werden kann.

- Bei Winkelwert 0 zeigt der Vektor in Richtung der positiven x-Achse.
- Bei Winkelwert $\frac{\Pi}{2}$ zeigt der Vektor in Richtung der positiven y-Achse.
- Bei Winkelwert Π zeigt der in Vektor, der in Richtung der negativen x-Achse.
- Bei Winkelwert $-\frac{\Pi}{2}$ zeigt der Vektor in Richtung der negativen y-Achse.

Die Winkelwerte von 0 bis Π beschreiben den Winkel *entgegen* dem Uhrzeigersinn von der positiven x-Achse aus. Die Winkelwerte von 0 bis $-\Pi$ beschreiben den Winkel *im* Uhrzeigersinn von der positiven x-Achse aus.

Erstellen Sie eine Klasse `de.hska.iwi.ads.solution.interfaces.CartesianVectorTest`, die von `de.hska.iwi.ads.interfaces.VectorTest` erbt. Diese Klasse enthält bereits einige JUnit-Tests. Implementieren Sie dort die abstrakte Methode, um einen CartesianVector für zwei Koordinaten zu erzeugen und zurückzugeben. Testen Sie Ihre Vektor-Implementierung mit dieser Klasse. Zur Fehlersuche sollte Sie in CartesianVectorTest noch weitere Tests implementieren.

5 Optional: Vector mit Polar-Koordinaten

Ein Interface kann viele Implementierungen besitzen, die sich gemäß der Interface-Spezifikation in der Javadoc aber gleich verhalten müssen.

Implementieren Sie eine Klasse `PolarVector` auf Basis einer Länge und der Richtung gegeben als Bogenmaß.

Programmteile, die mit CartesianVector identisch sind, sollten zur Vermeidung redundanten Codes in eine gemeinsame abstrakte Klasse verschoben werden, z.B. `AbstractVector`.