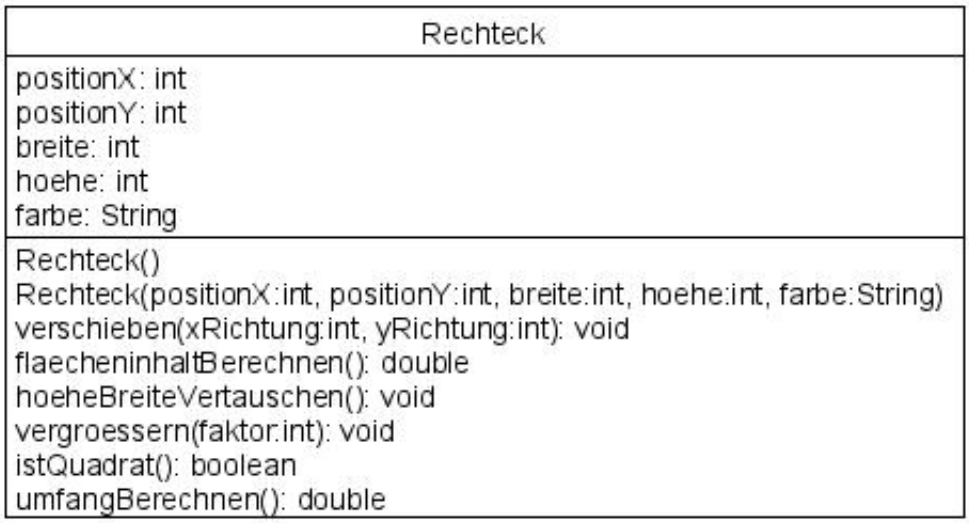
4 Implementierung von Methoden

Nachdem Sie sich nun ausführlich mit Attributen beschäftigt haben, geht es nun um die Implementierung von Methoden. Die Klassenkarte der Klasse `Rechteck` wurde dafür um die zu implementierenden Methoden erweitert:



Eine Methode wird dabei nach folgendem Muster implementiert:

```
Rückgabedatentyp Methodenname(Übergabeparameter mit Datentyp) //Methodenkopf
{
  //Funktionalität der Methode: Methodenrumpf
}
```

Eine Implementierung der Methode `verschieben` könnte dabei wie folgt aussehen:

Für Methoden, die einen Wert zurückgeben, muss das Schlüsselwort **`return`** verwendet werden. Dies wäre beispielsweise bei der Implementierung der Methode `flaecheninhaltBerechnen` notwendig:

Wichtig ist, dass die `return`-Zeile die letzte Anweisung in der Methode ist.

Aufgaben:

1. Implementieren Sie alle restlichen Methoden aus der obigen Klassenkarte in Ihrer bisherigen Klasse `Rechteck`. Die Methode `vergroessern` soll dabei das Rechteck um den übergebenen Faktor vergrößern.
   Hinweis: Mit dem Operator == können in Java zwei Werte eines primitiven Datentyps vergleichen werden.

   - `vergroessern(faktor: int): void`
   - `umfangBerechnen(): double`
   - `istQuadrat(): boolean`
   - `hoeheBreiteVertauschen(): void`

2. Erstellen sie zum Testen Ihrer Implementierung eine neue Klasse `MethodenTest` mit einer `main`-Methode. Erstellen Sie dort ein neues blaues Rechteck `rechteck1` an der Position (100|200) mit einer Länge von 100 und einer Breite von 50.
   Überprüfen Sie mithilfe von Konsolenausgaben, ob die Methoden `istQuadrat`, `flaecheninhaltBerechnen` und `umfangBerechnen` beim Objekt `rechteck1` folgende Werte zurückgeben:

   ```
   false
   5000.0
   300.0
   ```

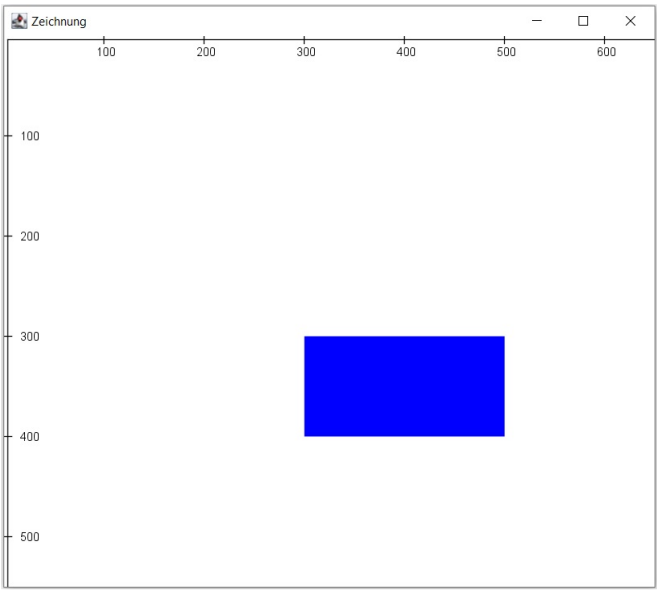
   Erstellen Sie anschließend in dieser Klasse ein neues Objekt `leinwand` der Klasse `Leinwand`. Die restlichen Methoden, die die Attributwerte eines Objekts verändern, können Sie nach folgendem Muster testen:

   ```
   leinwand.zeichne(rechteck1); // Zeichnen des Rechtecks vor Methodenaufruf
   rechteck1.verschieben(100,100); // Ausführen der Methode
   leinwand.warte(2000); // Warte 2 Sekunden
   leinwand.zeichne(rechteck1); // Zeichnen des Rechtecks nach der Veränderung

   rechteck1.vergroessern(2);
   leinwand.warte(2000);
   leinwand.zeichne(rechteck1);

   rechteck1.hoeheBreiteVertauschen();
   leinwand.warte(2000);
   leinwand.zeichne(rechteck1);
   ```

   Danach sollte Ihr Rechteck auf der Leinwand wie folgt aussehen:



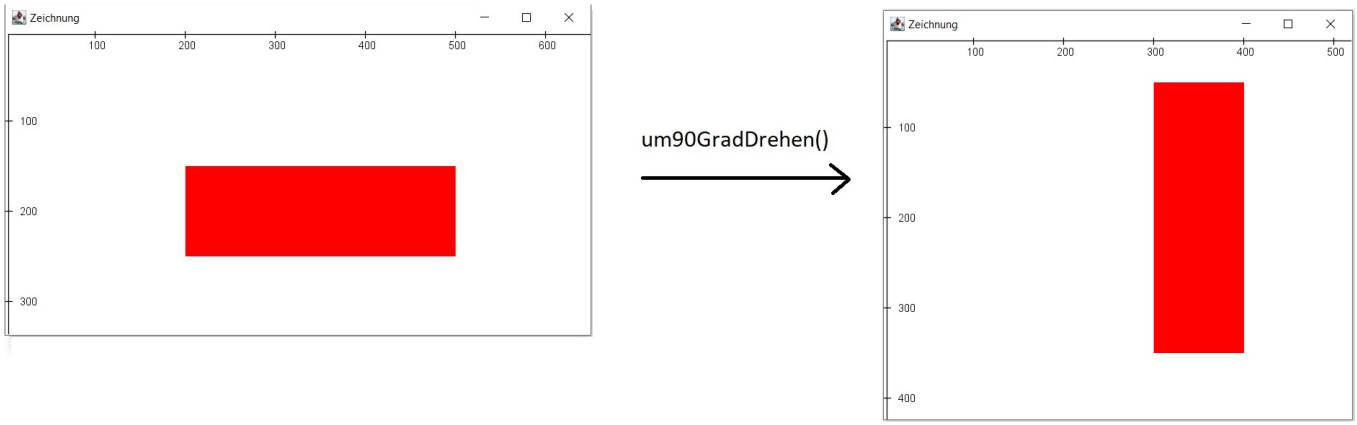
3. Implementieren Sie in der Klasse `Kreis` ebenfalls folgende Methoden:

   - `verschieben(xRichtung: int, yRichtung: int): void`
   - `vergroessern(zusaetzlicherRadius: int): void`
   - `flaecheninhaltBerechnen(): double`
   - `umfangBerechnen(): double`

   Testen Sie Ihre Implementierung selbstständig analog zur vorherigen Aufgabe. Bei einem Kreis mit einem Radius von 100 sollte für den Umfang ca. 628,32 und für den Flächeninhalt ca. 31415,93 herauskommen.

4. **Für Experten**

   (a) Implementieren Sie eine Methode `void um90GradDrehen()`, welche das Rechteck um 90 Grad um den Mittelpunkt des Rechtecks dreht:



   Testen Sie anschließend Ihre Implementierung, indem Sie ein Rechteck vier-mal um 90 Grad drehen lassen und überprüfen, ob sich das Rechteck wieder im Ausgangszustand befindet.

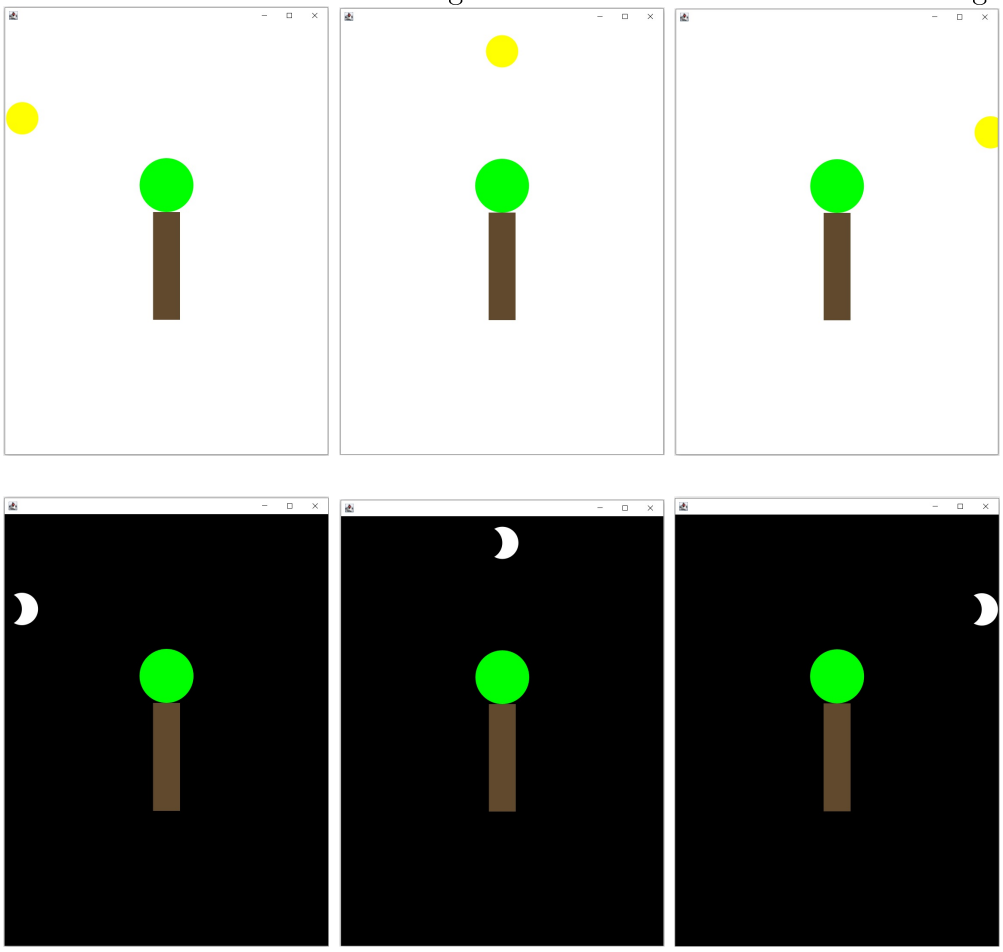
   (b) Animationen

      i. Erstellen Sie eine neue Klasse `EinfacheAnimation` mit einer `main`–Methode und versuchen Sie eine einfache Animation nach dem unten beschriebenen Verfahren zu erzeugen. Verändern Sie anschließend mal die Werte für die Verschiebung oder den Parameter der Methode `warten` und beobachten Sie, wie sich die Animation dabei verändert:

      Mithilfe einer Endlosschleife lassen sich nun auch einfache Animationen umsetzen. Hierfür muss beispielsweise ein Kreis mit einem entsprechenden Objekt der Klasse `Leinwand` gezeichnet werden, anschließend der Kreis minimal verschoben werden (beispielsweise 5 Einheiten in x- bzw. y-Richtung) und bevor der Kreis erneut gezeichnet wird, die Methode `warte` mit einem Wert von beispielsweise 30 auf dem entsprechenden Objekt der Klasse `Leinwand` aufgerufen wird. Durch die Wiederholung dieser Anweisungen entsteht eine einfache Animation.

ii. **Anspruchsvoll:**
Passen Sie Ihre Animation dahingehend an, dass der Kreis an den Rändern der Leinwand abprallt. Die Breite bzw. Höhe der Leinwand können mit den Methoden `getLeinwandBreite()` bzw. `getLeinwandHoehe()` ausgelesen werden. Die Übergabeparameter für die Methode `verschieben` müssen in entsprechenden Variablen gespeichert werden. Überlegen Sie sich, wie sie beispielsweise bei einem Kreis mithilfe des Mittelpunkts und des Radius eine Kollision an den Rändern erkennen. Bei einer Kollision muss das Vorzeichen der entsprechenden Variable für die Verschiebung in x- oder y-Richtung, abhängig davon bei welchem Rand es zur Kollision kommt, umgedreht werden.
Achten Sie auch darauf, dass beim Verändern des Leinwandfensters während der Animation, der Kreis nicht verschwindet.
Erstellen Sie eine analoge Animation mit einem Rechteck.

iii. **Sehr anspruchsvoll:**
Passen Sie Ihre erste Zeichnung (Baum mit Sonne) so an, dass die Sonne aufgeht, sich im Bild auf einem Kreis bewegt und anschließend untergeht. Nach dem Sonnenuntergang soll es entsprechend Nacht werden und anstelle der Sonne soll sich ein sichelförmiger Mond im Bild auf einem Kreis bewegen.



5 Zugriffsmodifikatoren bzw. Sichtbarkeiten

Aufgabe:

1. Erstellen Sie in einer Testklasse `SichtbarkeitenTest` ein Objekt `kreis1` der Klasse `Kreis` und setzen Sie den Radius auf -1.
   Lassen Sie das Objekt `kreis1` mit der Methode `zeichne` eines Objekts der Klasse `Leinwand` zeichnen. Was fällt Ihnen auf?

Um obige Situationen zu vermeiden, müssen die meisten Attribute vor unkontrollierten Zugriff geschützt werden. Dies geschieht, indem Sie vor den Attributen das Schlüsselwort **private** schreiben:

```
public class Rechteck{
   private int positionX, positionY, breite, hoehe;
   private String farbe;
}
```

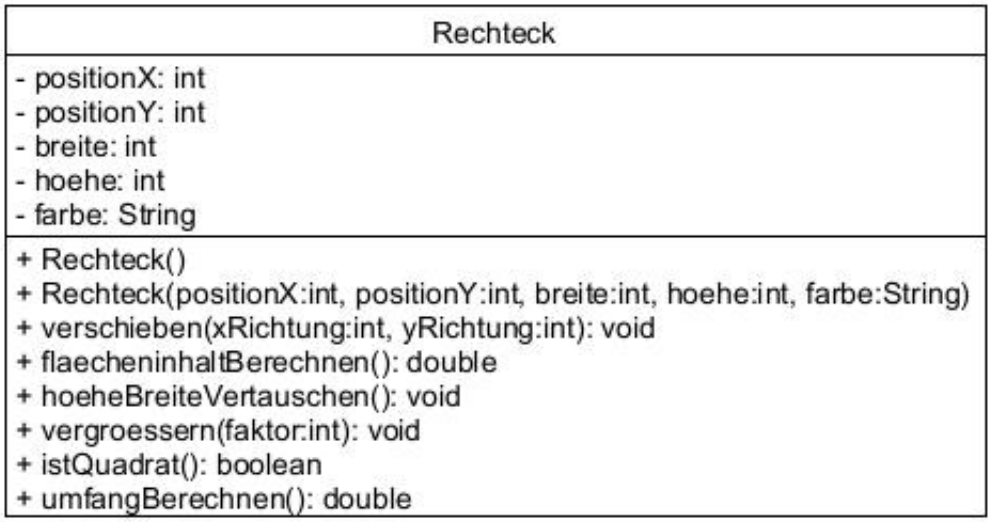
Dadurch ändert sich die so genannte **Sichtbarkeit** der Attribute. Auf `private`–Attribute kann nicht von andren Klassen mit dem Punktoperator direkt zugegriffen werden bzw. sie sind für andere Klassen nicht sichtbar.
`Private`–Sichtbarkeit wird in UML mit einem – gekennzeichnet:

Wird in Java keine Sichtbarkeit explizit angegeben, so wird die so genannte Paketsichtbarkeit verwendet. In folgender Tabelle finden Sie eine Übersicht über die verschiedenen Sichtbarkeiten in Java.

Sichtbarkeit	Symbol (UML)	Sichtbar in eigener Klasse	Sichtbar für Klassen im gleichen Paket	Sichtbar für Klassen in anderem Paket
`public`	+	Ja	Ja	Ja
`private`	-	Ja	Nein	Nein
paketsichtbar	~	Ja	Ja	Nein

Die weitere Sichtbarkeit `protected` wird zu einem späteren Zeitpunkt behandelt.

Diese Sichtbarkeiten gelten auch für Methoden. Die bisherigen Methoden der Klasse `Rechteck` sollen alle anderen Klassen (unabhängig davon in welchem Paket sie liegen) nutzen können. So müssen die Methoden die Sichtbarkeit `public` besitzen:



Um nun auf `private`–Attribute immer noch zugreifen zu können, benötigt man so genannte **set-** bzw. **get-**Methoden in der Klasse `Rechteck`:

```
public void setPositionX(int positionX){


}

public int getPositionX(){


}
```

Mit diesen Methoden kann für jedes Attribut individuell der lesende bzw. schreibende Zugriff geregelt werden.

Aufgaben:

1. Ändern Sie die Sichtbarkeit der Attribute der Klassen `Rechteck` und `Kreis` auf `private`. In der Testklasse `SichtbarkeitenTest` sollte nun ein Compiler-Fehler auftreten. Erklären Sie kurz warum.

2. Legen Sie für jedes Attribut in der Klasse `Rechteck` und `Kreis` eine `set-` bzw. `get-`Methode an. Da dies bei vielen Attributen sehr viel (stupide) Arbeit bedeuten würde, kann Eclipse diese Methode automatisch generieren:
   Menüreiter `Source` → „`Generate Getters and Setters`“
   Hinweis: Sollten Sie die Methoden händisch anlegen, achten Sie darauf, dass die `get-` bzw. `set-` Methoden wie nach obigen Muster benannt werden: set/get[Attributsname mit groß geschriebenem Anfangsbuchstaben]

3. Passen Sie nun Ihre Klassen `Rechteck` und `Kreis` so an, dass keine negativen Werte mehr für `hoehe`, `breite` und `radius` möglich sind. Sollte man versuchen, diesen Attributen einen negativen Wert zuzuweisen, so soll eine entsprechende Ausgabe auf der Konsole erscheinen.
   (Hinweis: `if`)