Vorgabe: Rotkäppchen hat sich im Wald verirrt und spaziert ziemlich ziellos durch die Landschaft. Ein Wolf ist natürlich auch noch im Wald und soll sich vorerst auch ziemlich ziellos bewegen.
Lege ein neues Projekt *Rotkaeppchen_Und_Der_Wolf* an. Lade von
`http://www.gierhardt.de/informatik/blueJSuM`
die Dateien `MeinProgramm.java`, `Wolf.java` und `Rotkaeppchen.java` herunter und verschiebe sie in das Fenster des neuen BlueJ-Projektes.

1. Der Wolf kann schon den Abstand bestimmen, aber diese Berechnung hat im vorgelegten Programm noch keine Konsequenzen. Modifiziere den Quelltext der Klasse *Wolf* so, dass der Wolf dem Rotkäppchen hinterher läuft.

2. Spendiere dem Rotkäppchen auch eine Methode zur Berechnung des Abstandes zum Wolf und eine Methode zum „Kennenlernen“ des Wolfes. Dann sollte Rotkäppchen in der Lage sein, dem Wolf wegzulaufen.

3. Analysiere die Quelltexte und erstelle für die Klassen *MeinProgramm*, *Rotkaeppchen* und *Wolf* jeweils ein Klassendiagramm.

4. Erstelle für das Programm ein Beziehungsdiagramm mit allen *hat-* *kennt-* und *ist-*Beziehungen.

5. Die Konstruktoren für *Wolf* und *Rotkaeppchen* sind fast identisch. Formuliere die Konstruktormethoden so um, dass sich Identität ergibt.

6. Sind jetzt noch Unterschiede in den Klassen *Wolf* und *Rotkaeppchen* vorhanden? Welche sind das?

7. Die beiden Klassen *Wolf* und *Rotkäppchen* sind bis auf gewisse Unterschiede in den Methoden *bewege* und *abstand* und die unterschiedlichen Methoden *lerneKennen* gleich.

	Rotkaeppchen
-	zAbstand: double
-	zGeschwindigkeit: double
+	Rotkaeppchen(double, int, Bildsch.)
+	lerneKennen(Wolf):void
+	bewege():void
+	hPosition():double
+	vPosition():double
-	amLinkenRand():boolean
-	amRechtenRand():boolean
-	amOberenRand():boolean
-	amUnterenRand():boolean
-	abstand():double
+	gibFrei():void

	Wolf
-	zAbstand: double
-	zGeschwindigkeit: double
+	Wolf(double, int, Bildsch.)
+	lerneKennen(Rotkaeppchen):void
+	bewege():void
+	hPosition():double
+	vPosition():double
-	amLinkenRand():boolean
-	amRechtenRand():boolean
-	amOberenRand():boolean
-	amUnterenRand():boolean
-	abstand():double
+	gibFrei():void

Durch das doppelte Schreiben von vielen Methoden ergibt sich ein enormer Schreibaufwand. Dieser ließe sich enorm reduzieren, wenn die für beide Klassen identischen

Methoden in einer **Oberklasse** wie z.B. *Waldbewohner* nur einmal zu formulieren wären. Die speziellen Methoden könnten dann in von der Oberklasse abgeleiteten *Unterklassen Wolf* und *Rotkaeppchen* formuliert werden.

Man schreibt also eine neue Klasse mit allen identischen Methoden wie folgt:

	Waldbewohner
-	zAbstand: double
-	zGeschwindigkeit: double
+	Waldbewohner(double, int, Bildsch.)
+	lerneKennen(Waldbewohner):void
+	hPosition():double
+	vPosition():double
-	amLinkenRand():boolean
-	amRechtenRand():boolean
-	amOberenRand():boolean
-	amUnterenRand():boolean
-	abstand():double
+	gibFrei():void

Die vorhandenen Klassen sehen dann wie folgt aus:

	Rotkaeppchen
+	Rotkaeppchen(double, int, Bildsch.)
+	bewege():void

	Wolf
+	Wolf(double, int, Bildsch.)
+	bewege():void

Man schreibt z.B. für die von *Waldbewohner* **abgeleitete** Klassen *Wolf* in Java

```
class Wolf extends Waldbewohner
```

Die abgeleiteten Klassen **erben** alle Attribute und Methoden der Oberklasse. Man spricht von **Vererbung** als wichtigem Prinzip der objektorientierten Programmierung. Die Unterklassen lassen sich je nach Bedürfnis mit weiteren Methoden erweitern.

Im Hauptprogramm werden nur Objekte der Klasse *Wolf* und *Rotkaeppchen*, aber keine Objekte direkt von der Klasse *Waldbewohner* erzeugt. Deshalb nennt man *Waldbewohner* eine **abstrakte Klasse**. Bezeichner von abstrakten Klassen werden im Klassendiagramm kursiv gesetzt.

Für die abgeleiteten Klassen gilt die *ist*-Beziehung. Man zeichnet z.B. von der Klasse *Wolf* einen Pfeil zur Oberklasse *Waldbewohner*, dessen Pfeilspitze ein unausgefülltes Dreieck darstellt.

(a) Formuliere die Quelltexte für die drei Klassen wie angegeben.

(b) Stelle die Klassen mit allen *hat*- *kennt*- und *ist*-Beziehungen in einem Diagramm dar.

8. Ergänzende Aufgabe: Das Programm soll beendet sein, wenn der Wolf dem Rotkäppchen lebensgefährlich nahe kommt.