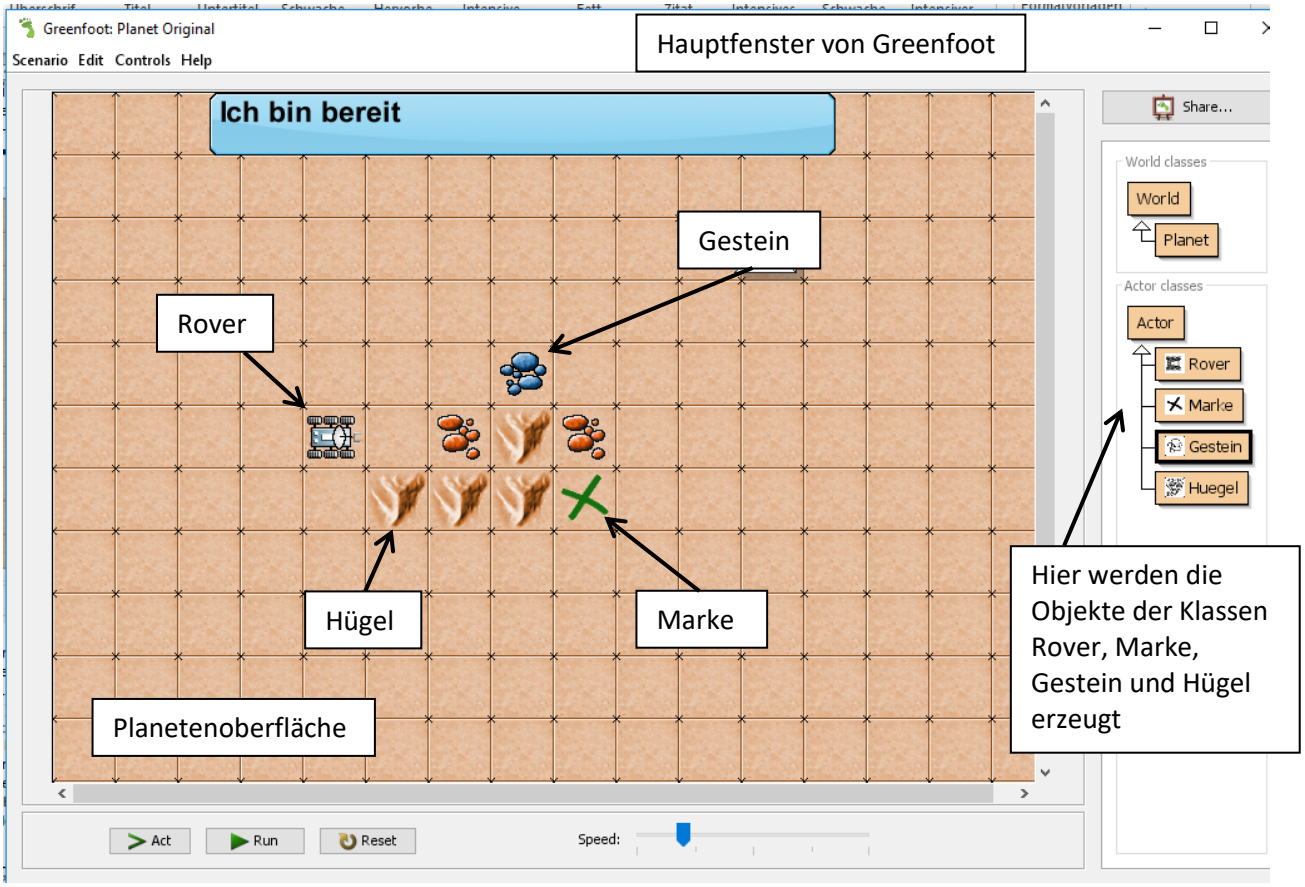
Projekt 1.1: Wie man eine Planetenumgebung mit Greenfoot erkundet

Java ist eine „objektorientierte Sprache“. Was das genau bedeutet, werden wir innerhalb unseres ersten Projektes lernen: Starte das Programm **Greenfoot** mit dem grünen Icon vom Desktop. Du siehst eine Planetenumgebung, die von einem Rover erkundet werden soll. Zunächst ist die Umgebung leer, gestalte die Umgebung so, wie sie in der Grafik zu sehen ist:



Den Rover bewegen

Aufgabe 1: Als erstes plaziere den Rover an der angegebenen Stelle. Klicke dazu auf der rechten Seite mit der rechten Maustaste auf den „Rover“ und wähle den Befehl „new Rover()“ („neuer Rover“). Bewege die Maus auf das richtige Feld und klicke nochmals, so wird der Rover platziert.
Klicke als nächstes (wiede rechte Maustaste) auf „Huegel“, wähle wieder „new Huegel()“. Wenn du die Hochstelltaste gedrückt lässt kannst du mehrmals hintereinander einen Huegel platzieren, wir benötigen vier „Huegel“. Hügel stellen für den Rover ein Hindernis dar, über einen Hügel kann er nicht fahren. Aufgabe des Rovers ist es, Gesteinsproben auf den Wasssergehalt zu überprüfen. Platziere dazu die drei Gesteinsproben wie oben angegeben, die Farbe ist übrigens zufällig. Setze als Letztes eine Marke. Damit deine Planetenumgabung nicht verloren geht, kannst du diese speichern. Klicke dazu auf ein leeres Feld in der „Planetenoberfläche“ mit der rechten Maustaste und wähle den Befehl „Save the world“.

Aufgabe 2: Der Rover geht auf Entdeckungsreise. Klicke mit der rechten Maustaste auf den Rover. Du bekommst angezeigt, was der Rover alles kann. Genauere Angaben zu den einzelnen Befehlen findest du im Buch auf S. 25. Bewege den Rover um das Gebirge herum und untersuche, wenn du auf den entsprechenden Feldern bist die Gesteinsproben. Die Aufgabe ist erfüllt, wenn du die grüne Marke erreicht hast. Du wirst folgende Methoden benötigen: fahre(), drehe(„Richtung“), analysiereGestein().
Bei Drehe musst du in einem fenster angeben in welche Richtung du drehen willst. Die Richtung muß in Anführungszeichen gesetzt werden, z. B. „rechts“. **Achte immer auf Groß- und Kleinschreibung.**

Aufgabe 3: Löse im Buch S. 29 Nr. 8 und 9. Alle Lösungen der Arbeitsaufträge gehören ins Heft.

Was eine Klasse und was ein Objekt ist

Klassen und Objekte

In dem bisherigen Beispiel hast du bereits 4 **Klassen** kennengelernt. Die Klassen Rover, Gestein, Huegel und Marke. Jede die Klassen stellt bestimmte Eigenschaften und Methoden bereit, die ein Objekt dieser Klasse haben kann. Auf der ersten Planetenoberfläche (s. Bild auf S. 1) gibt es ein Objekt der Klasse Rover, vier Objekte der Klasse Huegel, drei Objekte der Klasse Gestein und ein Objekt der Klasse Marke. Die **Objekte** können sich durch ihre Position und Farbe unterscheiden, sehen ansonsten aber immer gleich aus. Ein Objekt der Klassse Rover kann zudem bestimmte **Methoden** ausführen: Er kann ein Feld weiter fahren, sich drehen, eine Gesteinsprobe analysieren usw. Was dabei passiert, wurde in der Klasse Rover von einem Programmierer einmal genau festgelegt. Und jetzt kommt das **Besondere**: Von einer Klasse kann man beliebig viele Objekte erzeugen, dies geht mit dem Befehl „new“. Erzeuge einen zweiten Rover. Du kannst beide Rover fahren lassen. Es fährt jeweils der, auf den du mit der rechten Maustaste gedrückt hast.

Aufgabe 4:

a) Schreibe im Heft mindestens drei Eigenschaften auf, mit der sich verschiedene Objekte der Klasse Rover unterscheiden können.
b) Schreibe im Heft mindestens drei Methoden auf, die ein Objekt der Klasse Rover ausführen kann.
c) Mit welchem Befehl wird eines neues Objekt einer Klasse erzeugt?
d) Welche Methoden kann man durch Rechtsklick auf die Klassen Hügel und Gestein ausführen? Probiere sie jeweils einmal aus? Schreibe deine Antwort ebenfalls ins Heft.
e) Welche Dinge muss der Programmierer in einer Klasse bestimmen, damit anschließend Objekte der Klasse erzeugt werden können. Schreibe deine Antwort ebenfalls ins Heft.

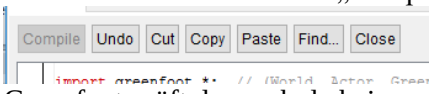
Wie man den Rover programmiert

Bisher ist das Erkunden der Planetenoberfläche ziemlich mühsam, da man jeden einzelnen Schritt mit der Maus und eventuell mit der Tastatur steuern muss – Rover sollen automatisch einen Planeten erkunden, doch dazu muss man sie programmieren: Klicke auf der rechten Seite unterhalb „Actor Classes“ mit der rechten Maustaste auf den Rover und wähle den Befehl „Open editor“. Alternativ kannst du auch einen Doppelklick auf den Eintrag „Rover“ ausführen. Im Buch auf S. 30 und 31 ist genau beschrieben, wie man eine eigene Methode programmiert.

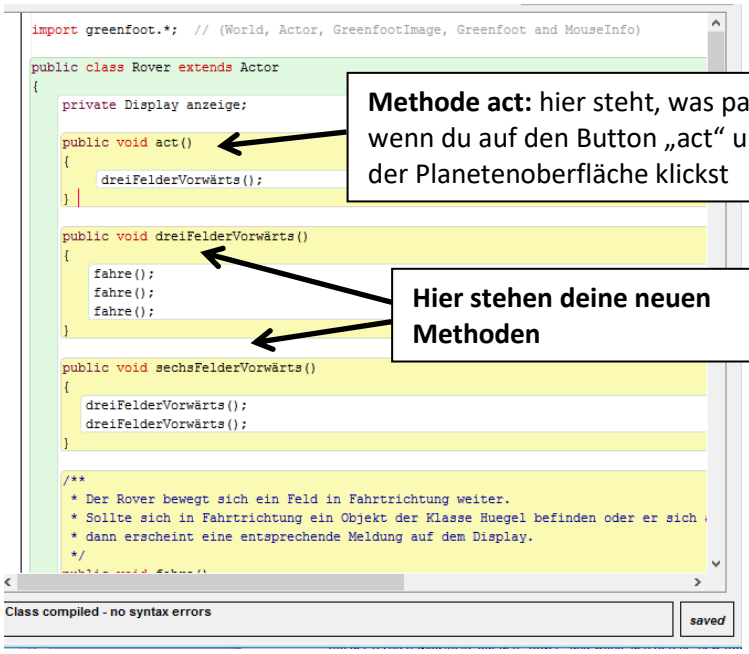
Aufgabe 5: Programmiere die beiden Methoden dreiFelderVorwärts() und sechsFelderVorwärts():

Eine Methode programmieren

Das rechte Bild hilft dir zu sehen, wo genau du die einzelnen Zeilen hinschreiben musst.
Wenn du den Text fertig geschrieben hast, klicke in der Zeile oberhalb deines Textfensters auf den Befehl „Compile“.



Greenfoot prüft dann, ob du beim Programmeiren keinen Fehler gemacht hast. Wenn du dies geschafft hast, steht unterhalb des Textfensters die Zeile „Class compiles – no syntax errors“. Du kannst dein Programm dann ausführen, indem du unterhalb der Planetenfläche auf den Knopf „Act“ klickst. Voraussetzung ist, dass in der Methode „act“ steht, was ausgeführt werden soll. Mit dem Knopf „Run“ wird dein Programm immer wieder hintereinander ausgeführt. Mit dem Button „Reset“ kannst du dein Programm unterbrechen.



Tipp: Achte auf Groß- und Kleinschreibung, Denke an alle Klammern „{ , } , (,)“ und alle Semikolons „;“.

Fehler werden rot unterstrichen, wenn man einmal die Zeile mit dem Fehler verlassen hat. Bewegt man die Maus auf das unterstrichene Wort, erscheint in einem kleinen Fenster die genaue Fehlermeldung.

Probiere aus, was passiert, wenn du in der Methode act() den Befehl dreiFelderVorwärts() durch sechsFelderVorwärts() ersetzt und anschließend den act-Button betätigst.

Aufgabe 6: Am Ende dieser Aufgabe sollst du einige Fragen zum **Aufbau einer Methode** in Java bzw. Greenfoot beantworten, dazu zunächst einige Informationen:

Access Modifier

1.) Jede Methode beginnt bisher mit dem Wort „public“ (zu Deutsch öffentlich). Dieses Wort bedeutet, dass die folgende Methode nicht nur innerhalb der Klasse Rover, sondern auch in anderen Klassen, die zum selben Projekt (hier Planetenerkundeung) gehören, benutzt werden kann. Wenn der Programmierer möchte, dass nur er die Methode in der Klasse Rover benutzen darf, verwendet er stattdessen das Schlüsselwort „private“. Das Schlüsselwort, mit dem ein Methodenaufruf beginnt, nennt man auch „**Access Modifier**“.

2.) Es folgt das Schlüsselwort „void“ (zu Deutsch leer). Das bedeutet, dass die Methode zwar bestimmte Befehle nacheinander ausführt, am Ende aber kein Ergebnis zurück liefern soll. Das ist z. B. bei der Methode gesteinVorhanden anders. Wenn du im Code Editor weiter hinunterscrollst, findest du hier folgende Zeilen:

Rückxgabewert

```
public boolean gesteinVorhanden()
{
    if(getOneIntersectingObject(Gestein.class)!=null)
    {
        nachricht("Gestein gefunden!");
        return true;
    }

    return false;
}
```

Statt „void“ ist das Wort „boolean“ angegeben. Boolean ist ein Variablentyp der nur die Werte wahr oder falsch (true or false) annehmen kann. Steht der Rover auf einem Feld, das ein Gestein enthält, sendet er die Nachricht „true“ zurück (es gibt ein Gestein), ansonsten meldet er die Nachricht „false“ (kein Gestein). Ein solches Ergebnis nennt man in Java auch **Rückgabewert.**

3.) Es folgt der Name der Methode und zwei runde Klammern „()“:

```
public void drehe(String richtung)
{
    if(richtung=="rechts")
    {
        setRotation(getRotation()+90);
    }
    else if (richtung=="links")
    {
        setRotation(getRotation()-90);
    }
    else
    {
        nachricht("Befehl nicht korrekt!");
    }
}
```

Innerhalb der Methode drehe(String Richtung) siehst du, dass man einer Metode einen oder mehrere **Parameter** in runden Klammern mitgeben kann. Wenn du den Befehl drehe („links“) sendest, gibst du die Richtung an, in der sich der Rover drehen soll. String ist ein weiterer Daten-typ, mit dem man beliebige Zeichenketten, z. B. Wörter, Sätze oder Texte übergeben kann. Der Text muss in Anführungszeichen stehen. Alle Parameter werden immer zwischen zwei runden Klammern „()“ geschrieben. Fehlt ein Parameter, stehen die beiden Klammern direkt hintereinander.

Methodennamen

Noch etwas zu Methodennamen: Methodennamen müssen immer mit einem Buchstaben beginnen. Ab der zweiten Stelle dürfen sie auch Zahlen und den Untestrich („_“) enthalten. Veboten sind Leerzeichen und Son-derzeichen. Vielleicht ist dir schon aufgefallen, dass bisher alle Methodennamen klein beginnen und weitere auftretende Wörter immer groß geschrieben werden (also z. B. „analysiereGestein“). Dies ist eine **Konvention**, an die sich alle Java-Programmierer – also auch du – halten müssen. Sie dient der besseren Lesbarkeit. Klassennamen werden immer vorne Großgeschrieben (z. B. Rover), so kann man Methoden und Klassen-namen alleine am Schriftbild unterscheiden. Die erste Zeile einer Methode (also z. B. „public void drehe (String richtung)“ nennt man auch **Methodendeklaration**.

4.) Alle Befehle, die die Methode ausführen soll, werden nun aufgelistet. Die Befehle werden immer in der Reihenfolge ausgeführt, in der sie im Programmcode erscheinen. Vor dem ersten Befehl muss eine öffnende geschweifte Klammer stehen „{„ , nach dem letzten Befehl muss eine schließende geschweifte Klammer stehen „}“. Die geschweiften Klammern umschließen somit alle Befehle, die ausgeführt werden sollen.

5.) Jeder Befehl wird mit einem Semikolon abgeschlossen.

6.) Beliebig viele Methoden kann man nacheinander programmieren. Die Reihenfolge der Methoden spielt keine Rolle. Wenn man eine Methode programmiert hat, kann man die Methode innerhalb einer anderen Methode aufrufen, indem man einfach ihren Namen benutzt. So wurde in deinem ersten Programm z. B. die Methode dreiFelderVorwärts() innerhalb der Methode sechsFeldervorwärts aufgerufen.

Erkläre nun im Heft folgende Begriffe, indem du jeweils dazu ein weiteres Beispiel aus der Klasse Rover aufschreibst: a) Access Modifier b) Rückgabewert c) Parameter d) Konvention für Methodennamen.

Aufgabe 7: Löse die Aufgaben 1, 2 und 3a) im Buch auf S. 33. Das obere Bild ist falsch: Das Getein muss ein Feld tiefer liegen. Schreibe die neuen Methoden direkt unter die Methoden, die du bisher programmiert hast. Gib ihnen sinnvolle Namen, wie fünfFelderErkunden() oder fünfMalfünfFelderErkunden(). Zum Testen musst du die Methode act() jeweils anpassen, indem du die Methode hineinschreibst, die du testen möchtest.

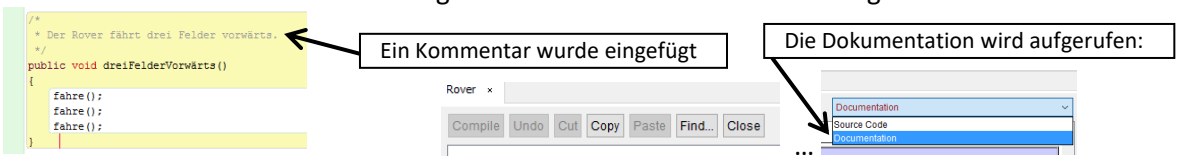
Programmierfehler finden

Das Schwierigste beim Programmieren ist das **Auffinden und Beseitigen von Fehlern**. Wenn du den Lehrer fragen möchtest, warum ein Programm nicht funktioniert, wird der Lehrer dir ab jetzt nur noch helfen, wenn du vorher die folgenden Fehlerquellen ausgeschlossen hast:

1.) Beginnt die Methodendeklaration mit einem Accessmodifier und einem Rückgabewert?
2.) Sind nach dem Methodennamen zwei runde Klammern gesetzt „()“?
3.) Steht vor dem ersten Befehl ein öffnende geschweifte Klammer „{“ und nach dem letzten Befehl eine schließende geschweifte Klammer „}“?
4.) Steht nach jedem Befehl ein Semikolon? 5.) Stehen hinter jedem Methodenaufruf runde Klammern?

Kommentare einfügen

Aufgabe 8: a) Je mehr Methoden du schreibst, desto schwieriger wird es hinterher, den Überblick zu behalten, wozu welche Methode gut war. Füge deshalb vor jeder deiner bisher geschriebenen Methoden einen sinnvollen **Kommentar** ein. Wie das geht hats du bereits auf S. 31 unten gelesen:

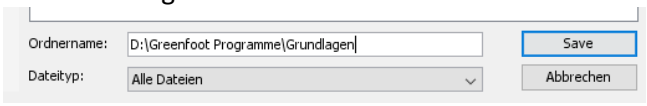


Klicke oberhalb des Codefensters auf der rechten Seite auf die Combobox, in der „Source Code“ eingestellt ist und welche auf „Documentation“. Du kannst jetzt deine Kommentare in der alphabetisch sortierten Liste aller Methoden wiederfinden.

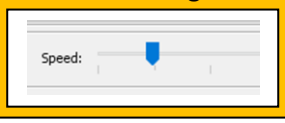
b) Eine weitere Vereinbarung hilf sehr, bei längeren Methoden den Überblick zu behalten:
Befehle werden innerhalb einer Methode eingerückt: im Beispiel siehst du, dass jede Zeile um vier Leerzeichen eingerückt wurde. Sorge dafür, dass in allen Methoden, die du bisher geschrieben hast, das **Einrücken** zu sehen ist.

Ab jetzt wird eine Methode nur noch vom Lehrer abgehakt, wenn Kommentar und Einrückung vorhanden sind.

Aufgabe 9: Damit dein Programm-Code nicht zu lang wird, speichern wir es am Ende jeder Woche, bevor ein neues Kapitel beginnt unter einem neuen Namen ab. Wechsle ins Hauptfenster von Greenfoot, indem die Planetenwelt dargestellt wird und wähle im Menü den Befehl „Scenario“ / „Save as“. Ergänze den Eintrag unter „Oednername“: Hier ist das Verzeichnis angegeben, indem dein bisheriges Programm abgelegt ist. Ergänze ein Unterverzeichnis „Grundlagen“, indem der Programmcode abgespeichert wird, den du diese Woche erzeugt hast.



Übrigens: Du kannst die Geschwindigkeit einstellen, mit der der Rover fährt:



Projekt 1.2: Die While-Schleife

Aufgabe 1: Speichere dein Programm erneut ab unter dem neuen Unterverzeichnis „while-Schleife“. Lösche dann im Programm deine bisherigen Methoden. **Achtung: Lösche nur die Methoden, die du selbst geschrieben hast. Wenn du andere Methoden löschst, wird der Rover nicht mehr fahren!** Deine Greenfoot-Umgebung ist damit vorbereitet, um die nächsten Aufgaben aufzunehmen.

While-Schleifen

Aufgabe 2: Lies im Buch S. 38 – 41 über die while-Schleife. Erzeuge die Planetenlandschaft, wie sie im Bild S. 41 Mitte zu sehen ist. Gib die neue Methode umfahreRechteck() ein, wie sie auf S. 41 oben dargestellt ist und teste dein Programm. Beantworte folgende Fragen im Heft:

a) Wie oft wird die äußere Schleife while (!markeVorhanden()) durchlaufen?
b) Wie oft wird die innere Schleife while (huegelvorhanden(„rechts“)) durchlaufen?
c) Welche Bedeutung hat das Rufzeichen vor markeVorhanden()?
d) Bei while-Schleifen musst du auf die richtige Klammersetzung achten. Beschreibe im Heft: Was steht immer in runden Klammern, was steht in geschweiften Klammern? ***Fehler im Buch: Die geschweifte Klammer nach dem ersten Befehl fahre() ist falsch herum gesetzt!***

Aufgabe 3: a) Im Bild auf S. 41 Unten soll der Rover ein unbekanntes Gebiet umrunden und dabei alle Gesteinsproben nehmen, die auf seinem Weg liegen. Löse dazu S. 41f. Nr. 2 im Heft. Der umgangssprachliche Algorithmus soll so formuliert sein, wie auf S. 41 Oben Links. Der Algorithmus soll nicht implementiert werden!
b) Die Fragen zu S. 42 Nr. 4 im Buch kannst du sehr kurz beantworten.

Aufgabe 4: Auf S. 42 Nr. 5 ist ein kleiner Fehler: Der Rover muss auf dem ersten Gestein stehen, nicht ein Feld davor. Es reicht, wenn du die Methode für a) zunächst programmierst und anschließend die gleiche Methode für b) ergänzt. Über den Reset-Button kannst du die Planetenumgebung wiederherstellen, vorher speichern!

Die Methoden in Nr. 5a) und 5b) sind sehr ähnlich. So kannst du beliebig lange Code-Segmente kopieren: Markiere den Teil, den du kopieren willst, mit der Maus und klicke die Tastenkombination „Strg“ + „C“ zum Kopieren. Welche anschließend an die Stelle, an der du den Code einfügen möchtest, und tippe die Tastenkombination „Strg“ + „V“. Alternativ kannst du in Greenfoot auch mit der rechten Maustaste ein Kontextmenü öffnen und mit den beiden Befehlen „Copy“ und „Paste“ arbeiten.

Zusatzaufgabe: a) Finde zu S. 43 Nr. 7a) die drei Fehler! b) Löse S. 44 Nr. 8.

Projekt 1.3: Die for-Schleife (Zählschleifen)

Beginne wieder ein neues Programm im Unterverzeichnis „for-Schleife“

For-Schleifen

Aufgabe 1: a) Lies im Buch S. 44 – S. 46 Oben über die Bedeutung einer for-Schleife (Zählschleife). Ein Hinweis dazu: Die Anweisung „i++“ in der ersten Java-For-Schleife auf S. 45 Mitte ist eine Abkürzung und bedeutet, das die Variable i um eins erhöht werden soll. Ausführlich müsste dieser Befehl heißen i = i + 1.
b) Löse die Aufgaben S. 46f. Nr. 1a), Nr. 2a) und b), Nr. 3 a) - d) und Nr. 4a) und b) vollständig. Nr. 1 gehört ins Heft. Bei Nr. 2 ist nur eine Methode notwendig, bei Nr.2b) soll die Methode aus a) einfach ergänzt werden. Bei Nr. 3b) und c) ist keine Implementierung vorgesehen, hier soll nur die Änderung in der for-Schleife im Heft beschrieben werden. Bei Nr. 3d) soll das Programm von Nr. 3a) nur ergänzt werden.

Weil in den nächsten Kapiteln viele kleine Methoden geschrieben werden sollen, nennen wir die einzelnen Methoden ab jetzt nach den Seitenzahlen und Nummern im Buch. Die erste Methode, die ihr selbst schreiben sollt (S. 46 Nr. 3a) bekommt also den Methodennamen „public void Seite46Nummer3a“. **Achtung: Leerzeichen dürfen dabei nicht verwendet werden.**

Zu den Aufgaben einige Tipps:
Nr. 3a): Es ist unbekannt, nach vie vielen Feldern die erste Gesteinsprobe kommt. Alle Gesteinsproben liegen dann aber direkt hintereinander. Überlege, wann du eine while- bzw. eine for-Schleife benötigst.
Nr. 3d): Setze eine Marke bevor du losfährst, um den Ursprungspunkt wiederzufinden.

Zusatzaufgabe: Löse Nr. 4 c) und d). Bei Nr. 4c) darfst du selbst bestimmen, aus wie vielen Seiten die Spirale bestehen soll.

Projekt 1.4: Der if-Befehl (Bedingte Anweisungen)

Beginne wieder ein neues Programm im Unterverzeichnis „If-Befehl“.

If-Befehl

Aufgabe 1: a) Lies im Buch S. 47 Mitte – S. 48 über die Bedeutung des if-Befehls (Bedingte Anweisung).
b) Löse im Buch S. 49f. Nr. 1, Nr. 2a,b), Nr. 3 und Nr. 4. Zu den Aufgaben einige Tipps:
Nr. 3c: Bevor du losfährst, solltest du eine Marke setzen, damit du später erkennst, wenn du wieder zurück bist. ***Achtung: Der setzeMarke()-Befehl benötigt anschließend einen fahre()-Befehl, bevor du die Marke mit markeVorhanden() abfragen kannst.***
Nr. 4: Unter der Startposition des Rovers sollte die erste Marke liegen.

Zusatzaufgabe: Löse im Buch S. 51 Nr. 5. Implementiere dazu zunächst eine Methode umfahreHuegel(), die einen Hügel umfährt. Du kannst bei b) und c) die bisherige Version stückweise ergänzen.

Projekt 1.5: Logische Operatoren (Und, Oder, Nicht)

Beginne wieder ein neues Programm im Unterverzeichnis „Logische Befehle“.

Logische Operatoren: &&, ||, !

Aufgabe 1: a) Lies im Buch S. 53 Mitte – S. 57 Oben über die Bedeutung der logischen Operatoren Und, Oder, Nicht. Zwei Hinweise dazu:
1.) Das OderZeichen „||“ erhälst du über die Tastenkombination Alt Gr + <.
2.) Durch zusätzliche Klammern kannst du die Reihenfolge, in denen logische Befehle miteinander verknüpft werden, verändern. In der Zeile

```
((!huegelVorhanden("links") || !huegelVorhanden("rechts")) && gesteinVorhanden()
```

wird z. B. zuerst die Oder-Verknüpfung ausgeführt, dann folgt die Und-Verknüpfung. Ganz zu Anfang werden jedoch die beiden Nicht-Verknüpfungen behandelt, da sie in der Reihenfolge vorrang haben.

b) Löse im Buch S. 57f. Nr. 1 bis Nr. 2. Verwende zu den Aufgaben eine **Kopie** der Seite 58: Die Lösung der Nr. 2 wird einfacher, wenn man auf der Kopie markiert, welche Gesteine vom Rover entfernt wurden.

Zusatzaufgabe: Löse im Buch S. 59 Nr. 3.

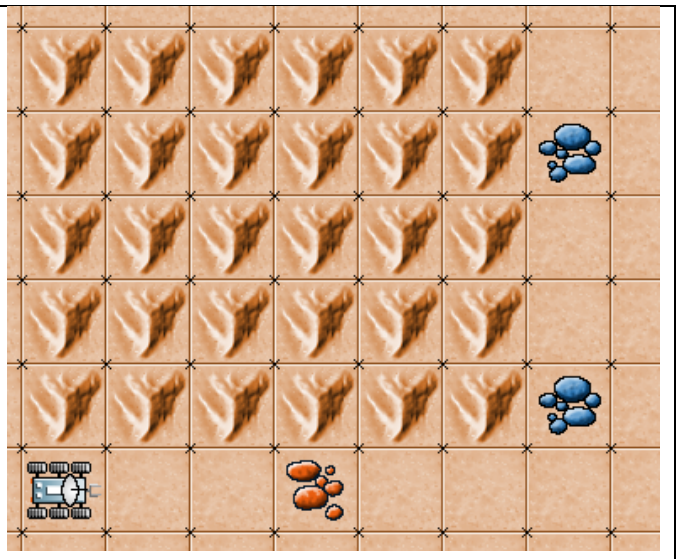
Projekt 1.6: Variablen

Aufgabe 1: Der Roboter soll beginnen, gegen den Uhrzeigersinn um einen rechteckigen Berg herumzufahren. Auf dem Weg soll er drei Gesteinsproben einsammeln und sofort danach auf dem gleichen Weg zu seinem Ausgangspunkt zurückkehren. Er soll während seiner Reise keine Marken hinterlassen. Der Rover weiß außerdem vorher nicht, wo die Gesteinsproben liegen.

Variablen

Diese Aufgabe lässt sich nur mit Hilfe zweier **Variablen** lösen. Variablen sind dabei Platzhalter für Zahlen. Aus der Mathematik kennst du die Variablen x und y, in Java darfst du folgende Namen für Variablen benutzen: Der Name muss mit einem Buchstaben beginnen, weitere Zeichen dürfen Buchstaben, Ziffern oder der Unterstrich „_“ sein. Außerdem vereinbaren wird entsprechend der Java-Konvention, dass Variablen immer mit einem kleinen Buchstaben beginnen und weitere eigenständige Wörter immer groß geschrieben werden.

Wir benötigen hier die Variablen „schrittZahl“ und „gesteinZahl“. Die Variable schrittZahl zählt, wie viele Schritte der Roboter gemacht hat. Mit dieser Variablen können wir den Rover wieder zum Ausgangspunkt zurückbringen. Die Variable gesteinZahl zählt, wie viele Gesteine der Rover bereits untersucht hat. Enthält diese Variable den Wert 3, fährt der Rover wieder zurück.



Datentypen

1.) Wir beginnen unsere Methode mit der **Deklaration** der Variablen. Dies bedeutet, dass wir Greenfoot mitteilen, welche Variablen wir benutzen wollen und welchen Datentyp sie haben sollen:

```
public void VariablenNr1()
{
    int schrittZahl = 0;
    int gesteinZahl = 0;
... }
```

Die wichtigsten Datentypen:
int: ganze Zahlen (Zahlen ohne Komma)
von etwa –2 Milliarden bis 2 Miliarden
double: Kommazahlen (größerer Datenbereich als GTR)
string: Folge von beliebeigen Zeichen (Text)
boolean: true oder false (wahl oder falsch)

Da wir nur Schritte oder Gesteine zählen wollen, wählen wir den Datentyp Integer (int).
Die Zeile „int schrittZahl = 0;“ bedeutet, dass eine Variable vom Datentyp Integer angelegt wird und zu Beginn den Wert 0 erhält. Wenn Variablen innerhalb einer Methode deklariert werden, kann man diese Variablen auch nur innerhalb dieser Methode benutzen. Solche Variablen nennt man **lokale Variablen**. Deklariert man die Variabeln vor der ersten Methode also bei uns nach der Zeile „class Rover extends Actor“ darf man die Variablen in der gesamten Klasse benutzen, solche Variablen nennt man **Attribute**.

Variablenwerte vergleichen

2.) Den Wert einer Variablen kann man verändern. Auf der linken Seite des Gleichheitszeichens steht immer, welche Variable verändert werden soll, auf der rechten Seite steht, welchen Wert die Variable annehmen soll:

schrittZahl = 10;	Der Wert der Variablen erhält den Wert 10.
schrittZahl = SchrittZahl + 2;	Der Wert der Variablen wird um den Wert 2 erhöht.
schrittZahl++;	Dies ist eine Kurzform für schrittZahl = SchrittZahl + 1;
schrittZahl--;	Dies ist eine Kurzform für schrittZahl = SchrittZahl - 1;

3.) Den Wert einer Variabeln kann man abfragen, um Bedingungen für eine Schleife oder Bedingungen für einen if-Befehl festzulegen:

while (gesteinZahl < 3) …	Die while-Schleife wird so lange durchlaufen, bis der Wert der Variablen gesteinZahl nicht mehr kleiner als 3 ist

if (gesteinZahl == 3) …

Achte auf den Operator „==“ (gleich)

Der if-Befehl wird nur ausgeführt, wenn der Wert der Variabeln gesteinZahl gleich 3 ist. **Achtung:** Der Operator „=“ aus der Mathematik hat in Java die Form „==“, dies ist für Anfänger eine häufige Fehlerquelle. Ansonsten dürfen die Operatoren „==“, „!=“ (für ungleich), „<=“ (für ≤), „>=“(für≥),, „<“ und „>“ benutzt werden.

Implementiere nun die gestellte Aufgabe. **Tipp:** Du wirst zweimal eine while-Schleife benötigen, einmal für das Suchen der Gesteine und einmal für den Rückweg. Überlege wie du die beiden Bedingungen für die while-Schleifen mit den Variablen schrittZahl und gesteinZahl formulieren musst und wann du die Werte der beiden Variablen verändern musst.

Aufgabe 2: Die Planetenoberfläche bleibt gleich. Der Rover soll einen rechteckigen Berg jetzt vollständig umrunden ohne umzudrehen. Beim Start soll er zunächst die X- und Y-Koordinate seiner Startposition im Display anzeigen. Wenn er ein Gestein gefunden hat, soll er die Zahl der bisher untersuchten Gesteine anzeigen, ansonsten seine aktuelle Position nennen. Kommt er an seine Startposition zurück, soll er die Zahl der untersuchten Gesteine aufschreiben.

Mit dem folgenden Befehl kann der Rover im Display etwas anzeigen:

```
nachricht ("Gesteine bisher untersucht: " + gesteinZahl);
```

Texte werden dabei in Anführungszeichen gesetzt, Variabelnwerte werden aufgeschrieben, indem der name der Variablen genannt wird. Mehrere Texte oder Variablenwerte werden nacheinander angezeigt, wenn man die Bestandteile mit dem „+“-Zeichen verknüpft.

Die X-Koordinate und Y-Koordiante der aktuellen Position des Rovers bekommt man über die Methoden getX() und getY (), sie liefern jeweils eine Integer-Zahl.

Tipps: Um die Startposition während der gesamten Fahrt zu speichern, benötigst du zwei Variabeln, z. B. startPositionX und startPositionY. Mit einer while-Schleife kannst du den Rover so lange fahren lassen, bis er wieder an seine alte Position zurückkehrt. Überlege wie man die Bedingung formulieren muss. Damit er nicht sofort mit seiner Runde aufhört, kann man zusätzlich in die Bedingung aufnehmen, dass er mindestens ein Gestein untersucht haben muss.

Zusatzaufgabe: Überlege, wie man die Methode verändern muss, damit der Rover auch eine Runde dreht, wenn kein einziges Gestein vorhanden ist.

Code drucken in Greenfoot

So druckst du den Code der Methoden eines Kapitels aus:
Greenfoot hat eine eigene Druckfunktion. Der Nachteil ist jedoch dabei, dass man nicht den Druckbereich eingrenzen kann. Es würden also immer alle Methoden ausgedruckt, auch die, die du gar nicht geschrieben hast. Besser ist es, wenn du die Methoden, die du ausdrucken möchtest, zunächst markierst, diese in die Zwischenablage kopierst (Strg + C), danach Word öffnest und den kopierten Text einfügst (Strg + V).

Was du in diesem Kapitel gelernt hast:
1.) Folgende Elemente solltest du jetzt programmieren können, die wir im folgenden Kapitel benutzen, wenn wir direkt in Java programmieren:

While-Schleifen:	while (gesteinZahl < 3) {fahre();}
For-Schleifen:	for (int i = 1; i <= 5; i++) {fahre();}
If-Befehle:	if (!huegelVorhanden("links")){drehe ("links");}
Logische Verknüpfungen:	if (!huegelVorhanden("links") || (!huegelVorhanden("rechts")) …
Vewenden von Variablen:	int schrittZahl = 0; schrittZahl++;

2.) Außerdem solltest du folgende Fachbegriffe kennen - dies wird genauso wichtig wie Punkt 1:

Klasse:	z. B. Rover, Gestein, Huegel; jede Klasse legt Eigenschaften und Methoden fest
Objekt:	jedes Objekt, das du mit einem new-Befehl angelegt hast, gehört zu einer Klasse
Eigenschaften:	mehrere Objekte einer Klasse haben ein ähnliches Aussehen, können sich aber durch verschiedene Eigenschaften voneinander unterscheiden, dies waren z. B. die Position, beim Gestein auch die Farbe und der Wassergehalt
Methoden:	du hast viele Methoden programmiert, die alle Objekte einer Klasse ausführen können