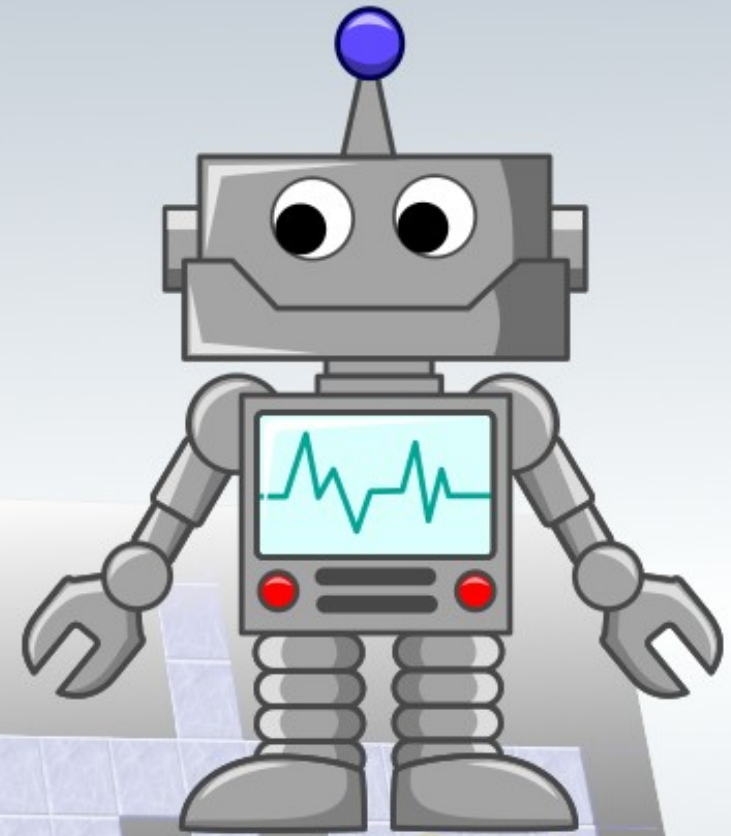


# Einführung in die Programmierung

## Methoden





Unter einem Algorithmus verstehen wir eine eindeutige Tätigkeitsbeschreibung, die zum Ziel führt.

Dabei dürfen nur die zur Verfügung stehenden Befehle und Anfragen verwendet werden.

*(nach Wirth und Hromkovic (ETH Zürich))*



## Zur Verfügung stehende **Befehle** und **Anfragen**

```
void einsVor()  
void dreheLinks()  
void benutzeAkku()  
void benutze(String name)
```

```
boolean istVorneFrei()  
boolean istAufGegenstand()  
boolean istInventarLeer()  
int getAnzahl(String name)
```

The screenshot shows the class hierarchy for the Rescue-Robot class. The class is inherited from Actor and Roboter. The method list is divided into two sections: inherited methods and methods defined in the class. The methods are color-coded to match the text in the previous blocks: yellow for commands and orange for queries.

```
void ablegen(String name)  
void aufnehmen()  
void benutze(String name)  
void dreheLinks() [ redefined in AB1 ]  
void dreheRechts() [ redefined in AB1 ]  
void dreheUm()  
void einsVor() [ redefined in AB1 ]  
void gehilfeEinsetzen(Roboter r)  
int getAnzahl()  
int getAnzahl(String name)  
int getEnergie()  
boolean hatGegenstand(String name)  
boolean istAufGegenstand(String name)  
boolean istAufGegenstand()  
boolean istEnergieLeer()  
boolean istInventarLeer()  
boolean istVorne(String name)  
boolean istVorneFrei()  
boolean istWandLinks()  
boolean istWandRechts()  
boolean istWandVorne()  
void melde(String text, boolean istWichtig)  
void setAnzahlVonGegenstand(String name, int anzahl)  
void setLocation(int x, int y)  
void setRotation(int x)  
void verbraucheEnergie(int verlust)  
void warne(String text, Actor actor)
```

Roboterszenario: Bildquellen der Objekte: siehe Bildnachweise.html



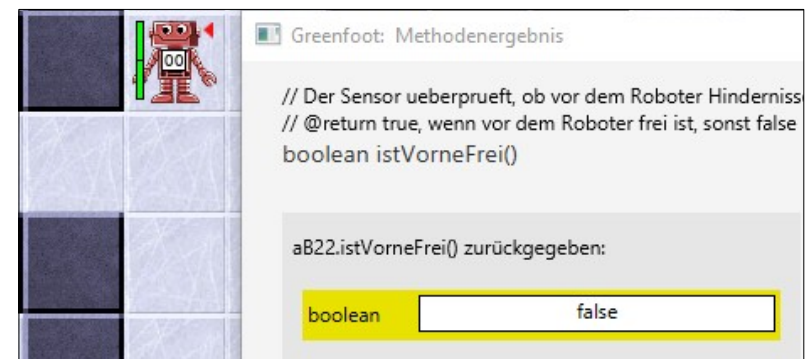
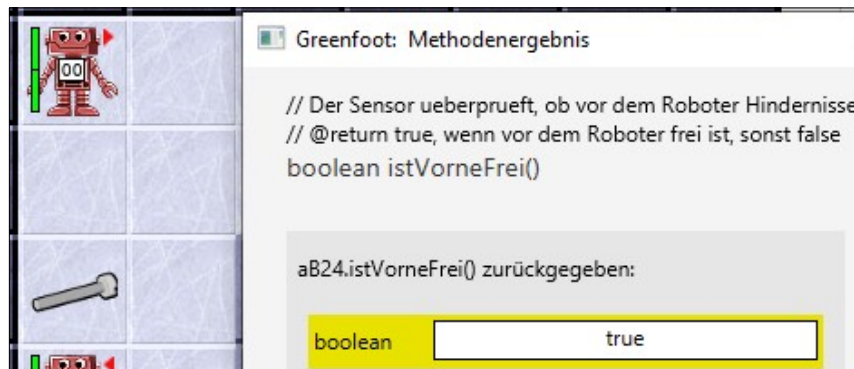
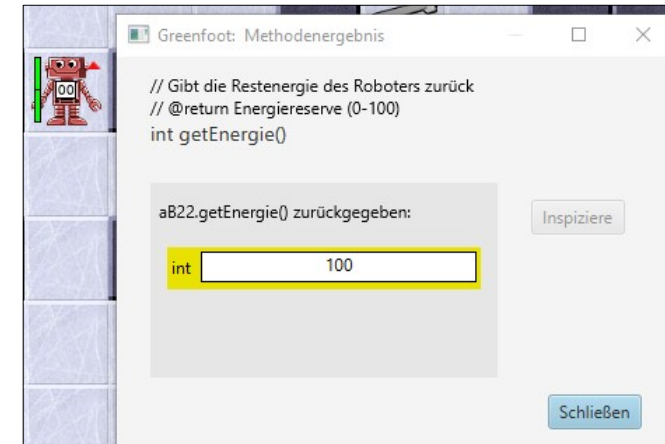
**Fähigkeiten** eines Objekts der Klasse Roboter, bei denen **geantwortet** wird, sind z.B.:

`int getEnergie()`

Int: Antwort ist eine Zahl

`boolean istVorneFrei()`

Boolean: Antwort ist true oder false



Roboterszenario: Bildquellen der Objekte: siehe Bildnachweise.html



## Programmablauf

```
69  /*#
70  * Aufgabe 7:
71  */
72  public boolean istFassRechts() {
73      dreheRechts();
74      if (istVorne("Fass")) {
75          dreheLinks();
76          return true;
77      }else {
78          dreheLinks();
79          return false;
80      }
81  }
82
83  public int gibLaengeFassReihe() {
84      int laenge;
85      laenge = 0;
86      while(istFassRechts()) {
87          laenge++;
88          einsVor();
89      }
90      return laenge;
91  }
```

true  
false

1

Greenfoot: Method...

int gibLaengeFassReihe()

ab8.gibLaengeFassReihe()  
returned:

int

Inspect  
Get

Close



Greenfoot: Debugger

Options

Call Sequence  
AB8.gibLaengeFassR

Static variables

Instance variables  
private int zaehler1 = 0  
private double wert = 0.0  
private int energie = 100  
private ArrayList<Gegenstand> inventar = <object reference>

Local variables

Stop Halt  
Step  
Step Into  
Continue  
Terminate

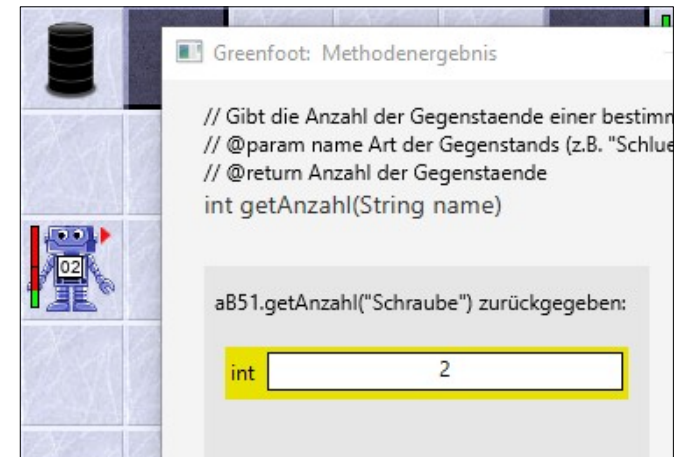
Roboterszenario: Bildquellen der Objekte: siehe Bildnachweise.html



Manche Methoden benötigen Zusatzinformationen, um korrekt arbeiten zu können:

```
int getAnzahl(String name)
```

String name: Ein Text ist erforderlich  
z.B. `getAnzahl ("Schraube");`



```
void setLocation(int x, int y)
```

int x, int y: Zwei ganze Zahlen als Koordinaten erforderlich  
z.B. `setLocation(5,2);`

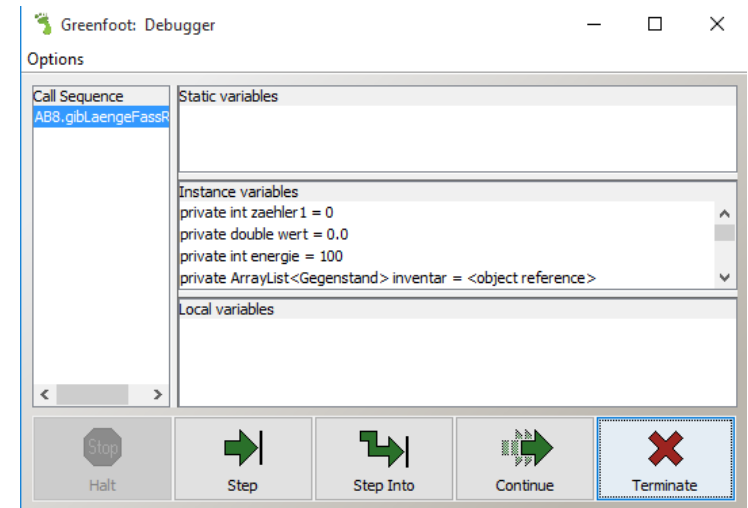


## Programmablauf

```
/*#
 * Aufgabe 6: Hartes Training
 */
public void gehe3Schritte() {
    geheSchritte(3);
}

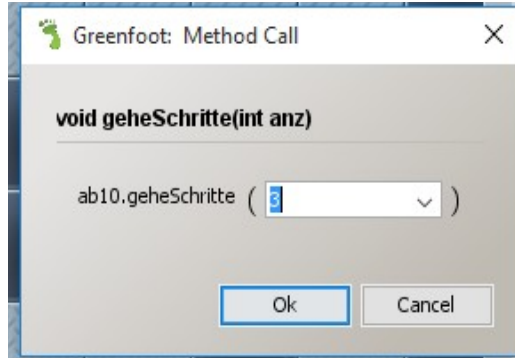
public void geheSchritte(int anz) {
    for (int j=0; j<anz; j++) {
        einsVor();
    }
}
```

Diagram illustrating the program flow. A call to `gehe3Schritte()` is shown, which calls `geheSchritte(3)`. The parameter `anz` is passed as `3`. The `geheSchritte` method then executes a loop for `j=0` to `j<anz` (3), calling `einsVor()` for each iteration.

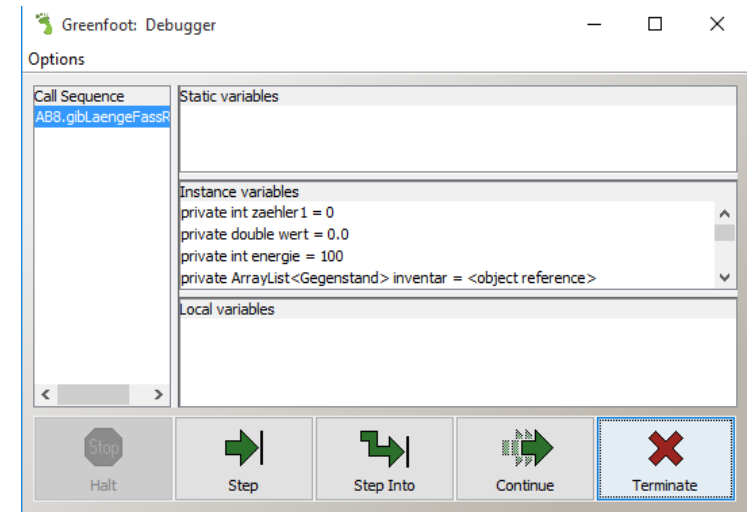




## Programmablauf



anz = 3



```
/**
 * Aufgabe 6: Hartes Training
 */
public void gehe3Schritte() {
    geheSchritte(3);
}

public void geheSchritte(int anz) {
    for (int j=0; j<anz; j++) {
        einsVor();
    }
}
```



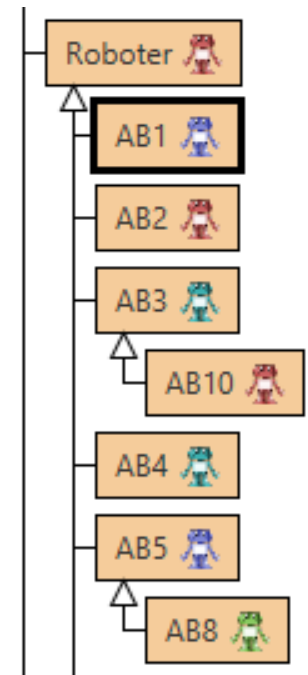


## Unterklassen erweitern bestehende Klassen.

Ein Objekt der Klasse „AB1“ hat die gleichen Eigenschaften und Fähigkeiten wie ein Objekt der Klasse „Roboter“ (A1 erbt von Roboter).

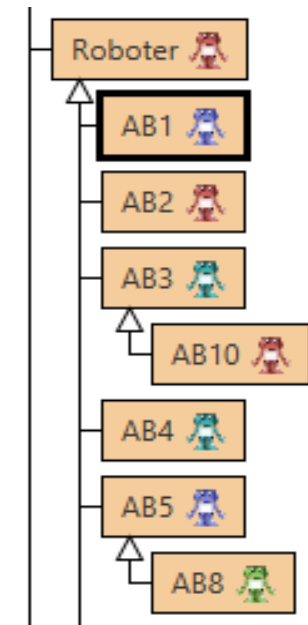
Ein Objekt der Klasse „AB1“ kann die Eigenschaften und Fähigkeiten von „Roboter“ verändern.

Ein Objekt der Klasse „AB1“ kann zusätzlich eigene Eigenschaften und Fähigkeiten haben.





<i>inherited from Actor</i>	
<i>inherited from Roboter</i>	
void akkuAufnehmen()	void ablegen(String name)
void benutzeAkku()	void aufnehmen()
void dreheLinks()	void benutze(String name)
void dreheRechts()	void dreheLinks() [ redefined in AB1 ]
void einsVor()	void dreheRechts() [ redefined in AB1 ]
int gibAnzahlSchrauben()	void dreheUm()
boolean istAufSchraube()	void einsVor() [ redefined in AB1 ]
boolean istVorratLeer()	void gehilfeEinsetzen(Roboter r)
void schraubeAblegen()	int getAnzahl()
void schraubeAufnehmen()	int getAnzahl(String name)
<i>Inspect</i>	int getEnergie()
<i>Remove</i>	boolean hatGegenstand(String name)
	boolean istAufGegenstand(String name)
	boolean istAufGegenstand()
	boolean istEnergieLeer()



Zusätzlich **erbt** er alle Methoden von Roboter.

Der AB1-Roboter hat z.B. die neuen Fähigkeiten schraubeAblegen() und schraubeAufnehmen()

Roboterszenario: Bildquellen der Objekte: siehe Bildnachweise.html



## Eigene Methoden

Objekte können jederzeit mit neuen Fähigkeiten ausgestattet werden, indem der Klasse eine neue Methode hinzugefügt wird.

Diese Methoden können dann in anderen Methoden verwendet werden.

Komplexe Aufgaben sollten in einfachere Methoden aufgeteilt werden.

