Funktionale und objektorientierte Programmierkonzepte

Florian Kadner und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 18/19
v1.3

Übungsblatt 5

Themen:	Referenzsemantik, Klassen, Strings, Arrays
Relevante Folien:	KarelJ, Lex. Bestandteile, OO Abstraktion
Abgabe der Hausübung:	30.11.2018 bis 23:55 Uhr

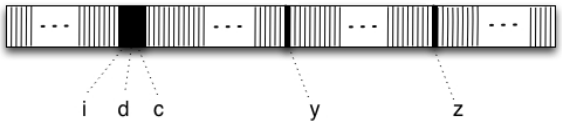
V Vorbereitende Übungen

V1 Referenzen

☆ ☆ ☆

Geben Sie in eigenen Worten wieder, was man unter einer Referenz versteht.

Betrachten Sie außerdem folgendes Schaubild und den Codeausschnitt aus der Vorlesung:




```
1 public class X{
2  int i;
3  double d;
4  char c;
5  ...
6 }
```

Zeichnen Sie die Referenzpfeile nach den folgenden Aufrufen ein (ergänzen Sie auch die neuen Reservierungen des Speicherplatzes, wenn nötig):

```
1 X y = new X();
2 X z = y;
3 y = new X();
```

V2 Zuweisen und Kopieren

☆ ☆ ☆

Erläutern Sie in Ihren eigenen Worten den Unterschied zwischen Zuweisen und Kopieren. In welchen Fällen sind beide Aktionen synonym zu betrachten?

Wie können Sie eine Zuweisung beziehungsweise eine Kopie in Java umsetzen? Nennen Sie jeweils ein Beispiel.

V3 Arrays ★ ☆ ☆

Welche Aussagen zu einem gegebenen Array `a` sind wahr?

(1) Alle Einträge des Arrays müssen vom selben Typ sein.

(2) Ein Array hat keine feste Größe und kann beliebig viele neue Einträge hinzufügen.

(3) Um die Anfangsadresse einer Komponente an Index i zu bekommen, wird i-mal die Größe einer Komponente auf die Anfangsadresse von `a` addiert.

(4) Außer den eigentlichen Komponenten des Arrays enthält das Arrayobjekt nichts weiteres.

(5) Ein Array kann nur primitive Datentypen wie zum Beispiel `int`, `char` oder `double` speichern. Somit ist insbesondere nicht möglich, Roboterobjekte in einem Array zu speichern.

Schreiben Sie die nötigen Codezeilen, um ein Array `a` der Größe 42 vom Typ `int` anzulegen. Füllen Sie danach das Array mithilfe einer Schleife, sodass an der Stelle `a[i]` der Wert `2i+1` steht. Nutzen Sie dabei zuerst eine normale, danach eine verkürzte Schleife.

V4 Wettrennen ★ ☆ ☆

Sie haben einen schnellen Roboter `rabbit` erstellt und wollen ihm nun noch ein langsames Gegenstück `turtle` bauen. Beide starten an einem gemeinsamen Punkt, schauen in die gleiche Richtung und besitzen die gleiche Anzahl an Beepern. Sie wollen nun schauen, wer in 10 Runden mehr Strecke zurücklegen kann. Jeder der beiden Kontrahenten kommt pro Runde genau einen Schritt voran, der schnelle `rabbit` erhält jedoch jede zweite Runde sogar einen Extraschritt. Betrachten Sie den folgenden Codeausschnitt, der die Situation implementieren möchte:

```
1  Robot rabbit = new Robot(1,1,East,0);
2  Robot turtle = rabbit;
3
4  for(int i = 0; i < 10; i++){
5
6   if(i / 2 == 0){
7    rabbit.move();
8   }
9
10  rabbit.move();
11  turtle.move();
12
13 }
```

Führen Sie den Code einmal selbst aus und schauen Sie was passiert! Beheben Sie danach **alle** vorhandenen Fehler in der Implementierung, um die oben beschriebene Situation exakt umzusetzen.

V5 Beeper im Laufen ablegen ★ ☆ ☆

In dieser Aufgabe sollen Sie eine erste eigene Methode implementieren. Als Orientierung betrachten Sie die Methode `public void move(int numberOfSteps)` der Klasse `FastRobot` aus der Vorlesung. Schreiben Sie nun eine neue Funktion

```
public void beeperMove(int numberOfSteps)
```

für den `SymmTurner`-Roboter, den Sie in der Vorlesung kennengelernt haben. Diese soll `numberOfSteps` Schritte nach vorne gehen und dabei jedes mal einen Beeper ablegen. Sollte die geforderte Anzahl an Schritten größer sein als die Anzahl an Beepern soll der Roboter einfach stehen bleiben und sich ausschalten (dafür gibt es bereits die Methode `public void turnOff()`). Sollten mehr Beeper vorhanden sein als geforderte Schritte, soll er an seiner finalen Position alle verbleibenden Beeper ablegen.

V6 Himmelsrichtungsdreher ★ ☆ ☆

In vielen Aufgaben reichen uns die eingeschränkten Methoden eines Roboters der KarelJ-Werke nicht. Daher definieren wir uns neue Roboter, welche die technischen Anforderungen erfüllen. Im Foliensatz zu KarelJ haben Sie bereits Beispiele wie den `SymmTurner` Roboter dazu gesehen. In dieser Aufgabe sollen Sie eine neue Roboterklasse definieren, welche sich in alle beliebigen Himmelsrichtungen drehen kann. Dafür ist folgendes Grundgerüst gegeben:

```
1  public class DirectionTurner extends Robot{
2
3   public DirectionTurner (int street, int avenue,
4                           int beepers, Direction direction){
5          super(street,avenue,beepers,direction);
6          }
7
8   // ..........
9
10 }
```

An der Stelle ..... schreiben wir die neuen Funktionalitäten des Roboters mithilfe von Methoden (vergleiche vorherige Aufgabe). Ergänzen Sie vier neue Methoden namens `public void turnNorth()` etc., damit sich der Roboter gezielt in alle vier Himmelsrichtungen drehen kann.

V7 BeeperMover ★ ☆ ☆

Legen Sie eine neue Roboterklasse `BeeperMover` an. Die `move()`-Methode überladen Sie mit den Anweisungen, die in V5 die Methode `beeperMove` definiert haben. Legen Sie zusätzlich eine neue Funktion `putAllBeepers()` an, welche alle restlichen Beeper aus der Bag ablegt. Nutzen Sie diese Methode, um die `move()`-Methode zu implementieren.

V8 Test auf Gleichheit ★ ☆ ☆

Schreiben Sie eine Methode `int strEqual(String a, String b)`. Diese bekommt zwei Strings übergeben und soll bei gleicher Objektidentität der beiden Strings 2, bei Wertgleichheit 1 und bei Wertungleichheit 0 zurückgeben.

V9 Karel, bist du da? ★ ★ ☆

Schreiben Sie eine Methode `boolean karelAreYouThere(String input)`. Diese bekommt einen String übergeben und soll testen, ob sich ein Substring darin befindet, der mit `"K"` beginnt, mit `"rel"` endet und dazwischen genau ein Zeichen hat, so wie `"Karel"`. Z.B. soll für die Eingaben `"HalloKarel"` und `"34hfK7relase"` `true` zurückgegeben werden.

V10 Ziffern addieren ★ ★ ☆

Schreiben Sie eine Methode `int sumAllDigits(String input)`. Diese soll die Summe aller vorkommenden Ziffern 0-9 im String zusammenaddieren. Sind keine Ziffern im String vorhanden, so soll 0 zurückgegeben werden.

Beispiel: Der Aufruf `sumAllDigits("1DAS0Macht23Spa66ss9")` soll 27 zurückgeben.

Hinweise: Mittels `Character.isDigit(char c)` testen Sie, ob ein gegebener `char` eine Ziffer von 0-9 ist. Mittels `Integer.parseInt(String s)` können Sie einen gegebenen `String`, der aus einer Zahl besteht, zu `int` konvertieren.

V11 TeamRobot ★ ★ ★

In dieser Aufgabe sollen Sie ihre erste eigene Roboterklasse von Grund auf implementieren. Erstellen Sie dazu eine neue Klasse `TeamRobot`, die die Klasse `Robot` erweitert, also von ihr erbt. Der Konstruktor der Klasse `TeamRobot` übernimmt die Parameter des Konstruktors der Oberklasse `Robot` und besitzt zusätzlich die Parameter `int left` und `int right`. Der Parameter `int left` gibt an, wie viele zusätzliche Roboter beim Aufruf des Konstruktors links (also westlich) neben des `TeamRobot`s platziert werden. Der Parameter `int right` ist analog, für die Roboter rechts (also östlich). Der `TeamRobot`, sowie die Roboter links und rechts von ihm bilden ein Team. Die zusätzlichen Roboter werden vom `TeamRobot` im Konstruktor erzeugt. Bekommt der `TeamRobot` einen Befehl, so soll dieser von allen Robotern im Team ausgeführt werden. Die zusätzlichen Roboter selbst sind dabei nicht ansprechbar, dass heißt auf ihnen können keine Methoden aufgerufen werden. Überlegen Sie sich, wie Sie die Roboter des Teams in der `TeamRobot`-Klasse speichern können und wie Sie die Befehle die ein `TeamRobot` erhält, an alle Roboter im Team weiterreichen können. Die Befehle meinen hier die Methoden: `move()`, `turnLeft()`, `pickBeeper()` und `putBeeper()`.

Beispiel: Beim Erstellen eines `TeamRobot`s mit den Parametern `right = 1` und `left = 2` an der Position $(4, 4)$, werden zusätzlich 3 Roboter erstellt, die dem Team angehören, nämlich an den Position $(4, 2)$, $(4, 3)$ (links) und $(4, 5)$ (rechts).

V12 Kopieren ★ ★ ★

Gegeben seien folgende zwei Klassen:

```
1  public class A {
2          public int number;
3          public String text;
4
5          public A(int nr, String txt) {
6                  number = nr;
7                  text = txt;
8          }
9
10         public static A copy(A a) {
11                 // ...
12         }
13 }
14
15 public class B {
16         public String str;
17         public A a;
18
19         public B(String s, A x) {
20                 str = s;
21                 a = x;
22         }
23
24         public static B copy(B b) {
25                 // ...
26         }
27 }
```

Vervollständigen Sie zuerst die Methode `public static A copy(A a)` der Klasse A. Diese soll eine wertgleiche Kopie des übergebenen Objekts `a` erstellen und zurückgeben. Demnach dürfen die selben Attribute der Kopie und des ursprünglichen Objekts `a` nicht auf dasselbe Objekt verweisen.

Nun sollen Sie ebenfalls die Methode `public static B copy(B b)` der Klasse B vervollständigen. Diese soll eine wertgleiche Kopie des übergebenen Objekts `b` erstellen und zurückgeben.

Um zu testen, ob unsere Implementation der beiden Kopiermethode erfolgreich war, erweitern wir nun die Klasse B um eine Methode `public static boolean isCopy(B original, B copy)`. Dieses soll nun prüfen, ob nicht doch die selben Attribute der Kopie `copy` und des ursprünglichen Objekts `original` auf dasselbe Objekt verweist, ist dies der Fall oder sind die selben Attribute nicht wertgleich, soll `false` zurückgegeben werden. Welche Attribute der Klasse A und B müssen Sie überprüfen? Welche Methode und/oder welcher Vergleichsoperator muss dafür verwendet werden? In welchem Fall müssen Sie nur den Vergleichsoperator verwenden und warum?

H Fünfte Hausübung — Gesamt 11 Punkte
RepairBot

Auch ein Roboterleben ist hart. Denn selbst wenn unsere mechanischen Freunde nicht von Krankheit betroffen sind, so gibt es auch in Ihrer Welt eine große Gefahr – Verschleiß. Durch das ganze Herumlaufen, Beeper ablegen und Herumdrehen werden die Bauteile unserer Roboter aus den Karel-Werken ziemlich schnell abgenutzt, und so hat sich die Firmenleitung etwas tolles überlegt: In Zukunft sollen sogenannte Repairbots eingesetzt werden, um unsere kaputten Roboter wieder auf Vordermann zu bringen.

Da Sie unsere besten Angestellten in den Karel-Werken sind, ist es Ihre Aufgabe in dieser fünften Hausübung das Ganze sinnvoll umzusetzen!

Nutzen Sie dazu die von uns bereitgestellte Vorlage und halten Sie sich genau an die Vorgaben in den einzelnen Aufgabenteilen.

Don't panic! Der Aufgabentext mag auf den Blick extrem lang erscheinen, die eigentlichen Aufgaben sind dann aber im Vergleich sehr kurz. Lassen Sie sich davon nicht abschrecken, dies ist eine durchaus übliche Situation in der Informatik.

H1 Klasse Battery — 1 Punkt

Solange unsere Karel-Roboter noch keine ausgereiften Perpetuum mobile sind, sind auch Sie leider auf einen Akku angewiesen. In diesem Aufgabenteil wollen wir eben diesen umsetzen.

Ganz allgemein sprechen wir bei den Komponenten der Roboter von `Parts`. Diese sind bereits in einer eigenen Klasse umgesetzt und besitzen zwei Strings `name` und `condition`, welche den Namen des Bauteils und seinen aktuellen Zustand beschreiben.

Erstellen Sie eine neue Klasse `Battery` im Package `Parts`, die die Klasse `Part` erweitert, also von ihr erbt.

Die Klasse besitzt ein zusätzliches private-Attribut `int level`. Dieses wird benutzt, um den aktuellen Akkustand der Batterie zu speichern.

Der Konstruktor der Klasse `Battery` hat zwei Parameter `String condition` und `int level`. Wird dieser aufgerufen, soll zunächst der Superkonstruktor mit den Parametern `"Battery"` und `condition` aufgerufen werden und anschließend der Wert des Klassenattributs `level` auf den Wert des übergebenen Parameters `level` gesetzt werden.

Implementieren Sie die Methode `public int getLevel()`, die den aktuellen Akkustand der Batterie zurückgibt.

Implementieren Sie die Methode `public void setLevel(int level)`, diese setzt den aktuellen Akkustand der Batterie auf den Wert des übergebenen Parameters. Sollte der übergebene Parameter kleiner als 100 sein, so ist zusätzlich der Zustand der Batterie auf `Part.conditionUsed` zu setzen. Ist der übergebene Parameter kleiner oder gleich 0, so wird der Zustand der Batterie auf `Part.conditionDamaged` gesetzt.

H2 Klasse Bot 5 Punkte

Als nächstes folgen unsere normalen und kurzlebigen Roboter, die wir erstellen wollen. Diese Roboter nennen wir `Bots`, und sie bestehen aus verschiedenen Komponenten, also `Parts`.

Alle Teilaufgaben in Aufgabe H2 werden in der Klasse `Bot` im Package `Robots` umgesetzt.

H2.1 Das Grundgerüst der Bot-Klasse 1 Punkt

Erstellen Sie eine neue Klasse `Bot`, die die Klasse `Robot` erweitert, also von ihr erbt. Die Klasse besitzt zwei `private`-Attribute. Ein Array `parts` vom Typ `Part` und ein `boolean waitingForRepair`.

Der Konstruktor der Klasse `Bot` übernimmt die Parameter des Konstruktors der Oberklasse `Robot` und soll zunächst den Superkonstruktor aufrufen, im Anschluss soll `waitingForRepair` auf `false` gesetzt werden und das Array `parts` folgendermaßen initialisiert werden:

(1) Länge des Arrays = 4

(2) Index 0 des Arrays bekommt ein neues Objekt der Klasse `Battery` zugewiesen. Der Zustand der Batterie ist `Part.conditionNew`, der Akkustand beträgt `100`.

(3) Index 1 des Arrays bekommt ein neues Objekt der Klasse `Part` zugewiesen. Der Name des Roboterbauteils lautet `"Camera"`, der Zustand ist ebenfalls `Part.conditionNew`.

(4) Index 2 des Arrays bekommt ein neues Objekt der Klasse `Part` zugewiesen. Der Name des Roboterbauteils lautet `"Legs"`, der Zustand ist `Part.conditionNew`.

(5) Index 3 des Arrays bekommt ein neues Objekt der Klasse `Part` zugewiesen. Der Name des Roboterbauteils lautet `"Arms"`, der Zustand ist `Part.conditionNew`.

Zum Abschluss dieser Teilaufgabe implementieren Sie noch drei Methoden, um die Bauteile eines Roboters zu setzen oder zurück zu liefern.

Implementieren Sie die Methode `public Part getPart(int index)`. Diese bekommt einen Index übergeben und gibt das Objekt zurück, dass sich im Array `parts` am übergebenen Index befindet.

Implementieren Sie die Methode `public void setPart(int index, Part part)`. Diese bekommt einen Index sowie ein Objekt der Klasse `Part` übergeben und weist dem Array `parts` das übergebene Objekt am übergebenen Index zu.

Implementieren Sie die Methode `public int getPartIndexByName(String name)`. Diese bekommt den Namen eines Roboterbauteils als `String` übergeben und gibt den Index des ersten Roboterbauteils im Array `parts` zurück, dessen Name **wertgleich** mit dem übergebenen Name ist. Ist kein Roboterbauteil mit dem übergebenen Namen im Array `parts` vorhanden, soll -1 zurückgegeben werden.

Verbindliche Anforderung: Sie können nicht davon ausgehen, dass sich an Index 0 des Arrays `parts` die Batterie befindet, an Index 1 die Kamera usw.

H2.2 Lauf Bot lauf! 1 Punkt

Implementieren Sie die Methode `public void faceDirection(Direction dir)`. Diese bekommt eine Himmelsrichtung übergeben und lässt den Roboter in diese blicken. Sollte er bereits in die übergebene Himmelsrichtung blicken, soll nichts passieren (vergleichen Sie dazu auch Aufgabe V6).

Implementieren Sie die Methode `public void randomMove()`. Diese lässt den Roboter zunächst in eine zufällig gewählt Himmelsrichtungen blicken. Nutzen Sie die Klassenmethode `int getRandomNumber(int min, int max)` der mitgelieferten Klasse `MainController`, um eine Zufallszahl zwischen den übergebenen Parametern `min` und `max` zu generieren (beide inklusive). Alle 4 Himmelsrichtungen sollen mit gleicher Wahrscheinlichkeit auftreten. Blickt der Roboter aufgrund der zufällig gesetzten Himmelsrichtung gegen eine Wand, so ist die Methode `turnLeft()` solange aufzurufen, bis dies nicht mehr der Fall ist. Anschließend soll der Roboter einen Schritt in Richtung momentan blickender Himmelsrichtung gehen.

H2.3 Autsch! Der Verschleiß tut weh 1 Punkt

Implementieren Sie die Methode `public Part checkForDamagedParts()`. Diese gibt das erste Roboterbauteil im Array `parts` zurück, dessen Zustand `Part.conditionDamaged` ist. Ist kein Roboterbauteil beschädigt, soll `null` zurückgegeben werden.

Implementieren Sie die Methode `public void wearOutParts()`. Diese benutzen wir im weiteren Verlauf, um die Abnutzung der einzelnen Roboterbauteile zu simulieren. Später werden wir diese Methode nach jedem gegangenen Schritt des Roboters aufrufen. Nach dem Aufruf der Methode, soll der Akkustand der Batterie des Roboters um 1 reduziert werden. Die Zustände aller Teile im Array `parts` sind entweder `Part.conditionUsed` oder `Part.conditionDamaged`. Ob ein Roboterbauteil nach dem Aufruf als beschädigt deklariert ist, wird zufällig nach folgenden Wahrscheinlichkeiten entschieden:

(1) Roboterbauteile mit dem Namen `"Camera"` haben eine Wahrscheinlichkeit von 10%, dass Sie durch einen gegangenen Schritt beschädigt werden.

(2) Roboterbauteile mit dem Namen `"Legs"` haben eine Wahrscheinlichkeit von 22%, dass Sie durch einen gegangenen Schritt beschädigt werden.

(3) Roboterbauteile mit dem Namen `"Arms"` haben eine Wahrscheinlichkeit von 12.5%, dass Sie durch einen gegangenen Schritt beschädigt werden.

Hierbei ist zu beachten, dass pro Aufruf der Methode `wearOutParts()`, der Zustand von nicht mehr als einem Roboterbauteil auf `Part.conditionDamaged` gesetzt wird (die Batterie ist hierbei auch zu betrachten). Nutzen Sie auch hier wieder die Klassenmethode `int getRandomNumber(int min, int max)` der mitgelieferten Klasse `MainController` um eine Zufallszahl zwischen den übergebenen Parametern `min` und `max` zu generieren (beide inklusive).

Verbindliche Anforderungen: Sie können nicht davon ausgehen, dass sich an Index 0 des Arrays `parts` die Batterie befindet, an Index 1 die Kamera usw. Sie dürfen ledig-

lich davon ausgehen, dass das Array `parts` keine mehrfachen Roboterbauteile mit dem wertgleichen Namen enthält.

Es darf nur die oben genannte Methode `getRandomNumber` zur Generierung von Zufallszahlen benutzt werden, andernfalls führt dies zu Punktabzug!

H2.4 Schwirrt aus, kleine Roboter! 2 Punkte

Zum Abschluss der Klasse `Bot` implementieren Sie die Methode `public void doMove()`. Diese bestimmt welche Aktion ein Roboter der Klasse `Bot` in der aktuellen Runde ausführt. In jeder Runde ruft der `MainController` (Klasse, welche bereits von uns gegeben ist) die `doMove`-Methode eines jeden in der Welt befindlichen Roboters auf. Die `doMove`-Methode soll die folgenden Fälle behandeln:

1. Wartet der Roboter nicht auf eine Reparatur und sind alle Roboterbauteile intakt (Zustand: `Part.conditionUsed` oder `Part.conditionNew`), so geht der Roboter einen Schritt in eine zufällige Richtung und die Methode zur Simulation der Abnutzung der Roboterbauteile wird aufgerufen.
2. Wartet der Roboter nicht auf eine Reparatur und ist mindestens 1 Roboterteil defekt (Zustand: `Part.conditionDamaged`), so wird der Roboter als wartend deklariert und eine neue `RepairInstruction` mit dem Roboter und dem defekten Roboterbauteil erstellt um diese dann an den `MainController` zu übergeben. Nutzen Sie dazu die Klassenmethode `void orderRepairInstruction(RepairInstruction rep)` der Klasse `MainController`.
3. Wartet der Roboter noch auf eine Reparatur, wobei alle Roboterbauteile intakt sind, so wird der Roboter als nicht mehr wartend deklariert und die Schritte des obigen Falles (Fall 1.) werden ausgeführt.
4. Wartet der Roboter noch auf eine Reparatur und ist mindestens 1 Roboterbauteil defekt, so ist nichts zu tun.

Zur Erinnerung: Um anzuzeigen, ob ein Roboter auf eine Reparatur wartet, gibt es das Attribut `waitingForRepair`.

H3 Klasse RepairBot 5 Punkte

Kommen wir nun zu unseren neuen, tollen Reparaturroboter, welche unseren normalen Robotern das Leben erleichtern sollen.

Alle Teilaufgaben in Aufgabe H3 werden in der Klasse `RepairBot` im Package `Robots` umgesetzt.

H3.1 Das Grundgerüst der Klasse RepairBot 1 Punkt

Erstellen Sie eine neue Klasse `RepairBot`, die die Klasse `Bot` erweitert, also von ihr erbt. Die Klasse besitzt zwei `private`-Attribute. Ein Array `spareParts` vom Typ `Part` und ein Attribut `currentJob` vom Typ `RepairInstruction`. Der Konstruktor der Klasse `RepairBot` übernimmt die Parameter des Konstruktors der Oberklasse `Bot` und soll zunächst den Superkonstruktor aufrufen, anschließend soll das Array `spareParts` folgendermaßen initialisiert werden:

(1) Länge des Arrays = 20

(2) Indizes 0-4 des Arrays bekommen jeweils ein neues Objekt der Klasse `Battery` zugewiesen. Der Zustand der Batterie ist `Part.conditionNew`, der Akkustand beträgt `100`.

(3) Indizes 5-9 des Arrays bekommen jeweils ein neues Objekt der Klasse `Part` zugewiesen. Der Name des Roboterbauteils lautet `"Camera"`, der Zustand ist ebenfalls `Part.conditionNew`.

(4) Indizes 10-14 des Arrays bekommen jeweils ein neues Objekt der Klasse `Part` zugewiesen. Der Name des Roboterbauteils lautet `"Legs"`, der Zustand ist ebenfalls `Part.conditionNew`.

(5) Indizes 15-19 des Arrays bekommen jeweils ein neues Objekt der Klasse `Part` zugewiesen. Der Name des Roboterbauteils lautet `"Arms"`, der Zustand ist ebenfalls `Part.conditionNew`.

Implementieren Sie die Methode `public int sparePartAvailable(String partName)`. Diese bekommt einen Namen eines Roboterbauteils als String übergeben und gibt den ersten Index des Roboterbauteils im Array `spareParts` zurück, dessen Name **wertgleich** mit dem des übergebenen Namens ist und dessen Zustand entweder `Part.conditionNew` oder `Part.conditionUsed` ist. Sollte kein Roboterbauteil gefunden werden, dass diesen Anforderungen entspricht, so soll -1 zurückgegeben werden.

H3.2 Einmal Service bitte! 2 Punkte

Implementieren Sie die Methode `public void replacePart(Bot r, int sparePartIndex)`. Diese bekommt einen Roboter der Klasse `Bot` sowie einen Index des Arrays `spareParts` übergeben. Die Methode tauscht das erste `Item` im Array `parts` des übergebenen Roboters, dessen Name **wertgleich** ist mit dem Name des `Item` an Index `sparePartIndex` im Array `spareParts`, mit dem `Item` an Index `sparePartIndex` im Array `spareParts`.

Implementieren Sie die Methode `public void getCloserToRobot(Robot r)`. Diese bekommt eine Roboter vom Typ `Robot` übergeben und lässt den `RepairBot` einen Schritt näher zum übergebenen Roboter laufen. Befinden sich beide Roboter auf der gleichen Position, so soll nichts passieren.

Hinweis: Der `RepairBot` soll zunächst die Street des zu reparierenden Roboters erreichen, im Anschluss dann die Avenue.

H3.3 Schwirrt aus, kleine Reparaturroboter! 2 Punkte

Zum Abschluss der Klasse `RepairBot`, implementieren Sie noch zusätzlich die Methode `public void doMove()`. Diese bestimmt welch Aktion ein Roboter der Klasse `RepairBot` in der aktuellen Runde ausführt. In jeder Runde ruft der `MainController` die `doMove`-Methode eines jeden in der Welt befindlichen Roboters auf. Die `doMove`-Methode soll die folgenden Fälle behandeln:

1. Ist dem Roboter keine aktuelle `RepairInstruction` zugewiesen (`currentJob == null`), so ist zu überprüfen ob die Klassenmethode `getNextRepairInstruction()` der Klasse `MainController` eine neue `RepairInstruction`, die vom Roboter auszuführen ist, zurückgibt oder nicht.
   (a) Gibt die Methode `getNextRepairInstruction()` `null` zurück, so geht der Roboter einen Schritt in eine zufällige Richtung.
   (b) Gibt die Methode `getNextRepairInstruction()` eine neue `RepairInstruction` zurück, so wird diese der Variable `currentJob` des Roboters zugewiesen und die unten genannten Schritte zum Ausführen der `RepairInstruction` werden noch in dieser Runde befolgt (Fall 2.).
2. Ist dem Roboter hingegen eine aktuelle `RepairInstruction` zugewiesen, so sind folgende Fälle zu betrachten:
   (a) Besitzt der Roboter das passende Ersatzteil das zur Ausführung der georderten `RepairInstruction` nötig ist, so nähert er sich dem zu reparierenden Roboter in jeder Runde um einen Schritt. In der Runde an dem sich beide Roboter auf der gleichen Position befinden, tauscht der `RepairBot` das defekte Roboterbauteil, des liegengebliebenen Roboters, mit dem noch intakten Ersatzteil aus. Sobald die Roboterbauteile getauscht wurden, wird die aktuell zugewiesene `RepairInstruction` verworfen.
   (b) Besitzt der Roboter kein passendes Ersatzteil, zum Ausführen der georderten `RepairInstruction`, so wird die aktuell zugewiesene `RepairInstruction` direkt verworfen und der Roboter geht einen Schritt in eine zufällige Richtung.

Beachten Sie den folgenden *Hinweis* für alle obigen vier Fälle:

Rufen Sie **nicht** die Methode zur Simulation der Abnutzung der Roboterbauteile auf. Kein Bauteil eines Roboters der Klasse `RepairBot` nutzt sich in dieser Simulation ab.