Check und Prepare

Lösungsvorschläge

TECHNISCHE UNIVERSITÄT DARMSTADT

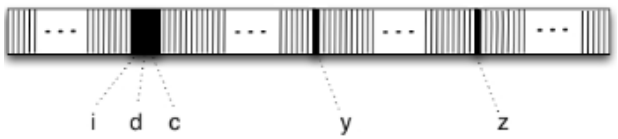
Autoren: Nhan Huynh und Darya Nikitina
Fachbereich: Informatik
Übungsblatt: 02
Version: 20. November 2021
Semesterübergreifend

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:




```java
public class X {
  int i;
  double d;
  char c;
  ...
}
```

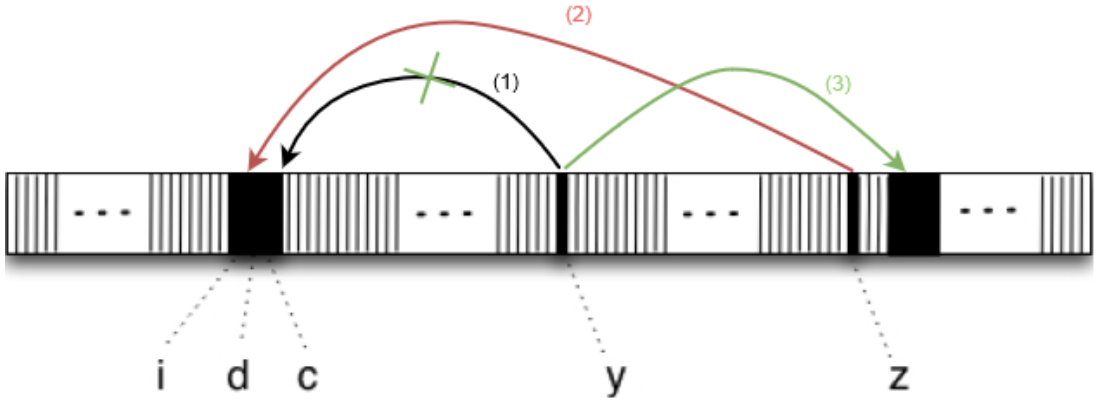
Zeichnen Sie die Referenzpfeile nach den folgenden Aufrufen ein (ergänzen Sie auch die neuen Reservierungen des Speicherplatzes, wenn nötig):

```java
X y = new X();
X z = y;
y = new X();
```

Lösungsvorschlag:

Eine Referenz ist ein Zeiger auf eine Speicheradresse, an der ein Objekt gespeichert ist.

Im folgenden sieht man die eingezeichneten Referenzpfeile:



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.

Lösungsvorschlag:

Zuweisen	**Kopieren**
Zuweisungen werden mit dem Zuweisungsoperator "=" durchgeführt. Bei Referenztypen, also nicht primitiven Datentypen, wird die Adresse des Objekts zugewiesen, d.h. wenn wir zwei Objekte `a` und `b` vom Typ Object haben und `a = b` zuweisen, dann wird die Adresse des Objekts von `b` in `a` gespeichert. Beim Zuweisen wird also die Referenz auf dasselbe Objekt gesetzt.	Beim Kopieren eines Objekts wird jedes Attribut des Objekts in ein neues Objekt kopiert, sodass ein wertgleiches Objekt entsteht. Das neue Objekt weist aber nicht mehr Objektgleichheit zu dem anderen Objekt auf wie beim Zuweisen. Also beim Kopieren bleibt es bei zwei verschiedenen Objekten, aber ihre Attribute haben identische Werte.

Synonym zu betrachten ist Zuweisen und Kopieren bei primitiven Datentypen, da es keine Aufteilung in Referenz und Objekt gibt. Durch die Zuweisung wird der tatsächliche Wert von primitiven Datentypen selbst in den neuen Speicherplatz kopiert.

Information: Ein primitiver Typ ist von der Sprache vordefiniert und wird durch ein reserviertes Schlüsselwort gekennzeichnet. Sie können jeweils nur einen Wert speichern und, wie erwähnt, ist Zuweisung und Kopieren bei ihnen Synonym.

Im Vergleich dazu speichert ein Referenztyp die Adresse des Objektes und nicht den im Objekt gespeicherten Wert.

```
1 public class Car {
2
3   int maxSpeed;
4   double fuel;
5
6 }
```

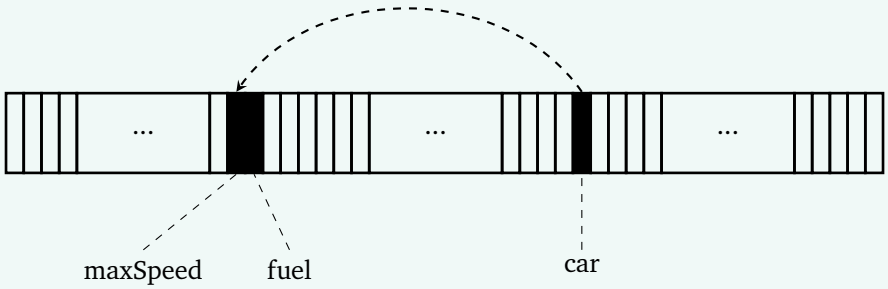


Abbildung 1: Abstrakte Visualisierung des Speicherplatzes eines Car-Objekts - `Car car = new Car();`

In der Abbildung 1 kann man sehen, dass die primitiven Datentypen `maxSpeed` und `fuel` direkt zugewiesen werden und `car` hingegen auf die Speicheradresse des Objekts referenziert.

Mehr Informationen zu primitiven Datentypen finden Sie unter folgendem Verweis:

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Übungsblatt 02: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

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 es können beliebig viele neue Einträge einem Array hinzugefügt werden.

(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 es insbesondere nicht möglich, Roboterobjekte in einem Array zu speichern.

Schreiben Sie die nötigen Codezeilen (auf Papier), 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 `while`- und danach eine `for`-Schleife.

Lösungsvorschlag:

(1) Inkorrekt, denn ein Array kann auch Subtypen des eigentlichen Typs enthalten.

- Beispielarray von Typ `Number` und die davon abgeleiteten Klassen `Integer` und `Double`.

```
1 Integer i = 1; // Integer extends Number
2 Double d = 2.0 // Double extends Number
3 Number[] numbers = {i, d};
```

(2) Inkorrekt, denn ein Array ist statisch, d.h. es hat eine feste Größe.

(3) Wahr. Ein Array repräsentiert einen festen Block von reservierten Speicher eines gleichen Komponententyps (Datentyp). Die Adresse im Speicher bezieht sich bei einem Array auf die Speicheradresse des ersten Elements (Basisadresse). Die nachfolgenden Elemente können von der Basisadresse addiert mit einem Offset angesprochen werden. Dieser Offset entspricht die Größe des Komponententyps. Bspw. um das dritte Element eines Arrays anzusprechen:

$$\text{Basisadresse} + 2 \cdot \text{Offset}$$

Das erste Element befindet sich unter der Basisadresse und das zweite Element unter Basiadresse + Offset.

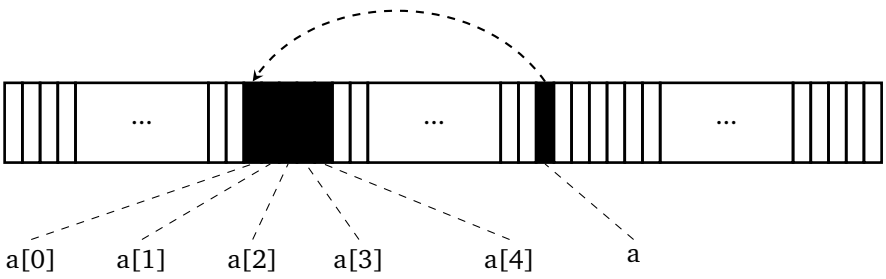


Abbildung 2: Abstrakte Visualisierung des Speicherplatzes eines Arrays `a`

(4) Inkorrekt, denn es ist noch die Größe des Arrays (`length`) abgespeichert, die man berücksichtigen muss.

- Beispiel:

```
int[] array = new int[20];
// size = 20
int size = array.length
```

(5) Inkorrekt, denn es können auch Referenztypen in einem Array gespeichert werden.

- Beispielarray von Typ `Robot`:

```
Robot[] robots = new Robot[3];
robots[0] = new Robot(0, 0, UP, 0);
robots[1] = new Robot(1, 1, UP, 1);
robots[2] = new Robot(2, 2, UP, 2);
```

Die Codezeilen:

```
int[] a = new int[42];

for (int i = 0; i < a.length; i++) {
  a[i] = 2 * i + 1;
}
```

```
int[] a = new int[42];

int i = 0;
while (i < a.length) {
  a[i] = 2 * i + 1;
  i++;
}
```

Übungsblatt 02: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

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 Coins. 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 in jeder zweiten Runde einen Extraschritt. Betrachten Sie den folgenden Codeausschnitt, der die Situation implementieren möchte:

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

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

Lösungsvorschlag:

Probleme:

1. `rabbit` und `turtle` verweisen auf dasselbe Objekt.
2. Die if-Anweisung ermöglicht es nur, dass der `rabbit` in der zweiten Runde einen weiteren Schritt ausführen kann statt in jeder zweiten Runde.

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

Übungsblatt 02: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V5 Roboter miteinander vergleichen

Schreiben Sie eine Methode `int robotsEqual(Robot a, Robot b)`. Diese bekommt zwei Roboter übergeben und soll `2` zurückgeben, wenn die Attribute `x`, `y`, `direction` und `numberOfCoins` bei beiden Robotern die gleichen Werte haben. `1` soll zurückgegeben werden wenn sich beide Roboter nur auf demselben Feld befinden, andernfalls gibt die Methode `0` zurück.

Lösungsvorschlag:

```
1  /**
2   * Checks, whether two robots are equal. The two robots are equal, if they have the same
3   * coordinates, the same direction and an equal amount of coins.
4   *
5   * @param a the first robot
6   * @param b the second robot
7   * @return {@code true} only if the robots a and b are equal
8   */
9  int robotEqual(Robot a, Robot b) {
10   // Check if the coordinates match
11   if (a.getX() == b.getX() && a.getY() == b.getY()) {
12     // Check if the directions and amount of coins match
13     if (a.getDirection() == b.getDirection() &&
14         a.getNumberOfCoins() == b.getNumberOfCoins()) {
15       return 2;
16     } else {
17       return 1;
18     }
19   }
20   return 0;
21 }
```

Übungsblatt 02: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V6 Primitive Datentypen

Schreiben Sie eine Methode `char smallestPDT(long n)`. Diese bekommt eine ganze Zahl übergeben und soll den primitiven Datentyp zurückgeben, der den wenigsten Speicherplatz verbraucht, aber immer noch die übergebene Zahl speichern kann. Geben sie `"l"` für den Datentyp `long` zurück, `" i"` für `integer` usw.

Schreiben Sie nun eine Methode `char[] smallestPDTs(long[] a)`. Diese bekommt ein Array von ganzen Zahlen übergeben und soll ein Array zurückgeben, dass die Methode `smallestPDT(long n)`, in der gleichen Reihenfolge wie in `a`, auf jede Zahl in `a` anwendet.

Lösungsvorschlag:

```java
/**
 * Returns the primitive type where the number fits in the range of values.
 *
 * @param n the number to check
 * @return the smallest primitive type, where the number fits in the range of values
 */
char smallestPDT(long n) {
  // Cast the value to another primitive type and check if its value is the same as before
  // the cast. If the value is not the same, then the value is outside the value range of
  // the tested primitive type

  if (n == (byte) n) { return 'b'; }
  else if (n == (short) n) { return 's'; }
  else if (n == (int) n) { return 'i'; }
  return 'l';
}

/**
 * Returns the primitive type for each number in the specified array, where it fits
 * in the range of values.
 *
 * @param a the array containing numbers to check
 * @return the primitive type for each number in the specified array, where it fits
 * in the range of values
 */
char[] smallestPDTs(long[] a) {
  char[] result = new char[a.length];
  for (int i = 0; i < result.length; i++) {
    result[i] = smallestPDT(a[i]);
  }
  return result;
}
```

Information: Mithilfe des folgenden Verweises finden Sie mehr Informationen zu primitiven Datentypen:

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Übungsblatt 02: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V7 Aufsummieren

Schreiben Sie eine Methode `void sumUp(int[] a)`, die ein Array `a` von Typ `int` erhält.

An Index $i \in \{0, \ldots,$ a.length-i$\}$ in `a` soll nun der Wert a$[0]+\cdots+$a[i] geschrieben werden. Dabei bezeichnen a$[0]+\cdots+$a[i] die Werte in `a` unmittelbar vor dem Aufruf der Methode.

Übergeben wir der Funktion das folgende Array `a` $= [3, 4, 1, 9, -5, 4]$, so wird das Array folgendermaßen modifiziert:

$$\begin{aligned}&\to [3, 3+4, 3+4+1, 3+4+1+9, 3+4+1+9+(-5), 3+4+1+9+(-5)+4]\\&\to [3, 7, 8, 17, 12, 16]\end{aligned}$$

Lösungsvorschlag:

```
1  /**
2   * Adds all previous values of the array to index i.
3   *
4   * @param a array of integers
5   */
6  void sumUp(int[] a) {
7    for (int i = 1; i < a.length; i++) {
8      a[i] += a[i - 1];
9    }
10 }
```

Information: Alternativ kann man die Zuweisung und Addition in Zeile 8 auch folgendermaßen darstellen:

```
1          a[i] = a[i] + a[i - 1];
```

Übungsblatt 02: Lösungsvorschläge von Nhan Huynh und Darya Nikitina

V8 Liste von Positionen ★★★

In dieser Aufgabe werden wir ein wenig kreativ und zeichnen die Initialen der FOP mit dem FOP-Bot. Dazu müssen wir zunächst sicherstellen, dass unser Roboter eine beliebige gegebene Position automatisiert ansteuern kann.

Um uns die Arbeit zu vereinfachen, verwenden wir die Klasse `Point` der Java-Standardbibliothek, deren Instanzen Punkte im zweidimensionalen Raum repräsentieren. Ein `Point`-Objekt kann mittels des Konstruktors `Point(int x, int y)` erzeugt werden. Die Abfrage der Werte ist dann über die Objekt-Attribute `x` und `y` möglich.

V8.1 Setzen der Blickrichtung

Als Erstes soll die `public`-Methode `void setDirection(Robot robot, Direction direction)` implementiert werden: Diese bekommt ein `Robot`-Objekt sowie eine `Direction`-Konstante übergeben. Nach Aufruf der Methode soll der Roboter in die gewünschte Richtung blicken.

Lösungsvorschlag:

```java
/**
 * Lets the given {@link Robot} look in a desired {@link Direction}.
 *
 * @param robot     the robot to turn
 * @param direction the desired direction
 */
public static void setDirection(Robot robot, Direction direction) {
  while (robot.getDirection() != direction) {
    robot.turnLeft();
  }
}
```

V8.2 Bewegen zu einer Position

Implementieren Sie nun die `public`-Methode `void moveToPoint(Robot robot, Point point)`: Mit dem Aufruf der Methode soll der gegebenen Roboter mittels der soeben von Ihnen implementierten Methoden `setDirection` und der Ihnen bereits bekannten Methode `move` an die gegebene Position bewegen werden. Sie können davon ausgehen, dass sich auf dem Weg zu dieser Position keine Hindernisse befinden. Dabei ist nicht wichtig, dass der Roboter den kürzesten Weg findet.

Lösungsvorschlag:

```java
/**
 * Lets a given {@link Robot} move to a given {@link java.awt.Point}.
 *
 * @param robot the {@link Robot} to move
 * @param point the targeted point
 */
public static void moveToPoint(Robot robot, Point point) {
  // Move the robot to the given coordinate
  while (robot.getX() != point.getX() || robot.getY() != point.getY()) {
    // Move right
    if (robot.getX() < point.getX()) {
      setDirection(robot, RIGHT);
      robot.move();
    }
    // Move left
    else if (robot.getX() > point.getX()) {
      setDirection(robot, LEFT);
      robot.move();
    }
    // Move up
    else if (robot.getY() < point.getY()) {
      setDirection(robot, UP);
      robot.move();
    }
    // Move down
    else {
      setDirection(robot, DOWN);
      robot.move();
    }
  }
}
```

V8.3 Coin Patterns

Nun implementieren Sie die `public`-Methode `void putCoins(Robot robot, Point[] points)`: Der gegebene Roboter soll an jeder der im gegeben Array enthaltenen Position eine Münze ablegen, sich anschließend in die Mitte der Welt bewegen und nach oben blicken.

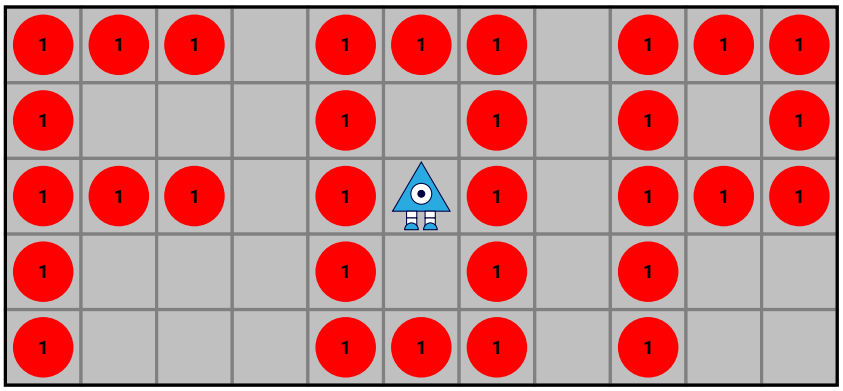


Abbildung 3: Gewünschtes Ergebnis

Verbindliche Anforderung: Die einzelnen Positionen des `Point`-Arrays müssen in einer einzigen `for`-Schleife abgelaufen werden. Die Objektmethode `putCoin` der Klasse `Robot` darf nur innerhalb dieser `for`-Schleife aufgerufen werden.

Lösungsvorschlag:

```java
/**
 * Lays a {@link Coin} on every given {@link java.awt.Point} and then goes to the center
 * of the {@link World}.
 *
 * @param robot  the {@link Robot} protagonist
 * @param points an array of points containing the coordinates of the pattern
 */
public static void putCoins(Robot robot, Point[] points) {
  for (Point point : points) {
    moveToPoint(robot, point);
    robot.putCoin();
  }
  goToPoint(r, new Point(5, 2));
  setDirection(r, UP);
}
```