Methode, griech. der Weg zu etwas, planmäßiges Verfahren
- Brockhaus

Parameterübergabemechanismen für den Methodenaufruf

Martin Wirsing

in Zusammenarbeit mit
Matthias Hölzl, Piotr Kosiuczenko, Dirk Pattinson

04/03

Ziele

- Verstehen des Parameterübergabebegriffs von Java
- Verstehen der Unterschiede zwischen
  Call-by-Value und Call-by-Reference

Call-by-Value-Parameterübergabe

Call-by-Value Sei gegeben

Methodendeklaration: `void m (D x1, int x2) {Rumpf}`

Aufruf: `o.m(a1, a2);`

formale Parameter

aktuelle Parameter

1. **Schritt:** Berechne die R-Werte o_1, v_1, v_2 der aktuellen Parameter o, a_1, a_2 und weise diese Werte dem impliziten Parameter `this` und den formalen Parametern x_1, x_2 zu, die als lokale Variablen des Rumpfs verwendet werden.
2. **Schritt:** Werte den Rumpf von `m` aus.
3. **Schritt:** Bei Beendigung der Auswertung des Rumpfs werden die lokalen Variablen **`this`**, x_1, x_2 gelöscht (durch Zurücksetzung des „Top-Zeigers" des Laufzeitkellerspeichers).

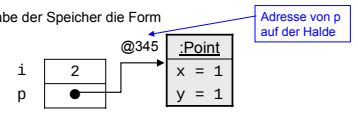
Call-by-Value-Parameterübergabe

Beispiel:

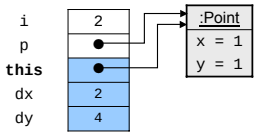
```
int i = 2;
Point p = new Point(1,1);      // (1)
p.move(i, 2+2);                // (2)
```


Call-by-Value-Parameterübergabe: Beispiel

Zum Zeitpunkt (1) habe der Speicher die Form

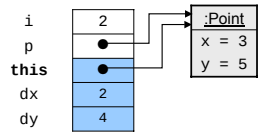


Beim Aufruf werden 3 lokale Variablen angelegt, die Werte von p (im Bsp. @345), dx und dy berechnet und die Initialisierungen
`this` `= @345; dx = 2; dy = 4;` ausgeführt:

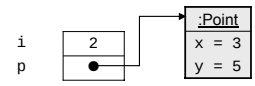


Call-by-Value-Parameterübergabe: Beispiel

Dann wird der Rumpf **`this`**`.x = `**`this`**`.x + dx `**`this`**`.y = `**`this`**`.y + dy;` ausgeführt:



Zum Zeitpunkt (2) sind die lokalen Variablen **`this`**, `dx` und `dy` des Blocks wieder gelöscht:



Call-by-Value mit Objektparameter

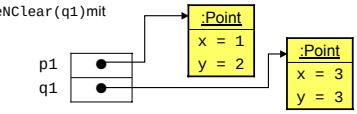
Beispiel:

Erweitere die Klasse Point um die Methode `moveNClear`, die das aktuelle Objekt um die Koordinaten von p verschiebt und dann p auf den Ursprung setzt.

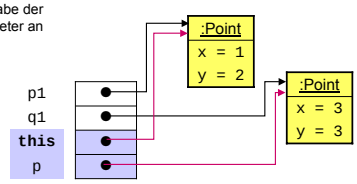
```
public void moveNClear (Point p)
{     int dx = p.getX();
      int dy = p.getY();
      this.move(dx, dy);
      p.move(-dx, -dy);
}
```


Call-by-Value mit Objektparameter

Aufruf `p1.moveNClear(q1)`mit

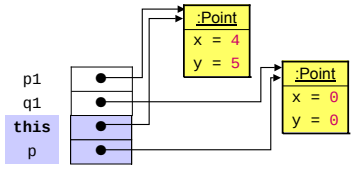


Schritt 1: Übergabe der aktuellen Parameter an **`this`**, p



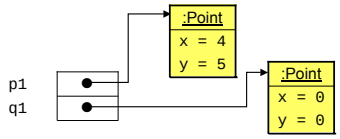
Call-by-Value mit Objektparameter

Schritt 2: Ausführung des Rumpfs ergibt



Call-by-Value mit Objektparameter

Schritt 3: Löschen der lokalen Variablen **`this`**, p



Call-by-Value-Parameterübergabe

Folgerung

- Da bei Call-by-Value nur die (R-)Werte übergeben werden, ändern sich die Werte der aktuellen Parameter **nicht**.
- Es können aber die Werte der Instanzvariablen eines aktuellen Parameters (vom Objekttyp) verändert werden.

Call-by-Value und Call-by-Reference

In anderen Programmiersprachen (wie C++, Ada, C,...) gibt es neben Call-by-Value auch den Parameterübergabemechanismus Call-by-Reference (Adressübergabe).

Gegeben sei eine Methodendeklaration

```
type m(T *x) {body} // nicht in Java!
```

und ein Aufruf

```
o.m(p);
```

Referenz auf einen Wert von T

Call-by-Reference

Schritt 1: Übergabe des R-Werts von **o** an **this** und des **L-Werts von p an x**

Schritt 2: Ausführung von `body`. Änderungen von `x` werden unter der Adresse von `p` gespeichert, d.h. direkt am aktuellen Parameter ausgeführt.

Schritt 3: Am Ende werden **this und x gelöscht.**

⇒ Änderung des R-Werts von `p` möglich

Call-by-Reference

Beispiel: choose

„Wähle Punkt mit der größeren y-Koordinate“

Call-by-Value für this und q1.

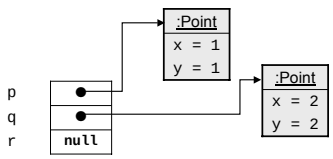
```
public void choose(Point q1, Point *higherPoint)
{     if (this.getY() >= q1.getY())
            *higherPoint = this;
      else
            *higherPoint = q1; //(2)
}
public static void main(String[] args)
{     Point p = new Point(1,1);
      Point q = new Point(2,2);
      Point r;                    // (1)
      p.choose(q, r);             // (3)
      ...
}
```

Bei der Ausführung werden Änderungen von higherPoint unter der Adresse des aktuellen Parameters gespeichert.

Call-by-Reference: higherPoint wird als Referenz (auf einen Zeiger auf Point) übergeben.

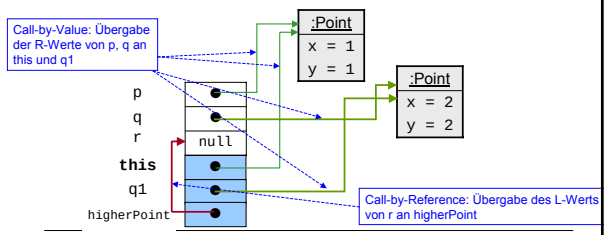
Call-by-Reference

Im Zeitpunkt (1) gilt folgender Speicherzustand:



Call-by-Reference

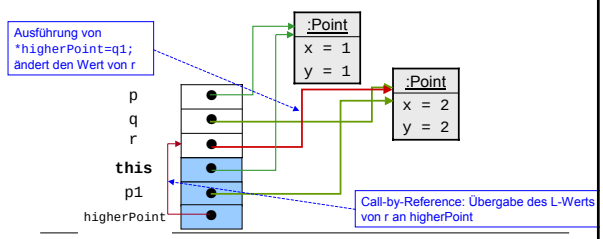
Bei der Parameterübergabe wird für den Call-by-Reference-Parameter higherPoint der L-Wert (die Adresse) des aktuellen Parameters r übergeben:



Call-by-Reference

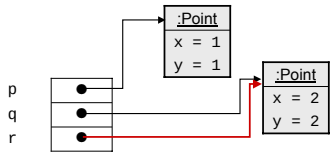
Während der Ausführung des Rumpfs (Zeitpunkt 2) wird bei der Zuweisung an higherPoint die Änderung an der Adresse des aktuellen Parameters r durchgeführt:

:



Call-by-Reference

Im Zeitpunkt (3) werden die lokalen Variablen gelöscht und man erhält man wegen Call-by-Reference eine Änderung des aktuellen Parameters: `r` ist nicht mehr `null`, sondern zeigt auf ein anderes Objekt.



- `r` zeigt auf den Punkt mit der größeren y-Koordinate!

Call-by-Value (Java)

Beispiel: Das (fast) gleiche Programm in Java mit Call-by-Value für `higherPoint`

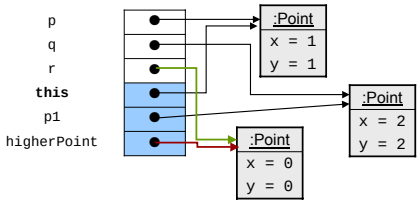
```
public void choose(Point p, Point higherPoint)
{       if (this.getY() >= p1.getY())
                higherPoint = this;
        else
                higherPoint = p1;       //(2)
}

public static void main(String[] args)
{       Point p = new Point(1,1);
        Point q = new Point(2,2);
        Point r = new Point();          // (1)
        p.choose(q, r);                 // (3)
        ...
}
```

Call-by-Value

Initialisierung von r mit Standardkonstruktor

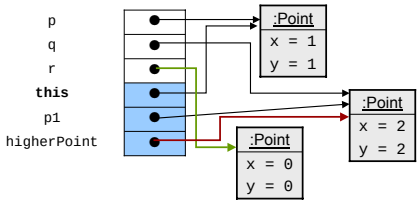
Call-by-Value (Java)

Zu Beginn der Ausführung des Rumpfs von `p.choose(q, r)`:



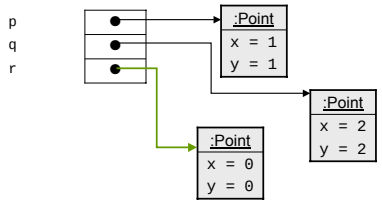
Call-by-Value (Java)

Zeitpunkt (2): Fallunterscheidung bewirkt eine Änderung der lokalen Variablen `higherPoint`.



Call-by-Value (Java)

Zum Zeitpunkt (3) haben sich also bei Call-by-Value die aktuellen Parameter **nicht** geändert:



Call-by-Value (Java)

Um den gleichen Effekt wie bei Call-by-Reference zu erzielen, führt man in Java ein Ergebnis ein:

```
public Point chooseJava(Point q1)
        {
                if (this.getY() >= q1.getY())
                        return this;
                else
                        return q1;
        }

        public static void main(String[] args)
        {
                Point p = new Point(1,1);
                Point q = new Point(2,2);
                Point r = p.chooseJava(q);        // (3)
        }
```

Zum Zeitpunkt (2) ergibt sich jetzt der gewünschte Speicherzustand, bei dem `r` auf das gleiche Objekt zeigt, wie `q`.

Zusammenfassung

- Eine Methode berechnet ihr Resultat abhängig vom Zustand des aktuellen Objekts und der aktuellen expliziten Parameter.
- Der Parameterübergabemechanismus von Java ist Call-by-Value. Dabei werden die Werte der aktuellen Parameter an die formalen Parameter übergeben. Die Werte der aktuellen Parameter werden durch Call-by-Value nicht verändert.
- Bei Call-by-Reference (wie in C, C++, Modula möglich) können die Werte der aktuellen Parameter verändert werden, da ihre Adressen (die L-Werte) übergeben werden. Java hat kein Call-by-Reference; es lassen sich aber durch Call-by-Value bei Objekten ähnliche Effekte und Speicherplatzersparnis erzielen.