Kapitel 06: Prozeduren

Software Entwicklung 1

Annette Bieniusa, Mathias Weber, Peter Zeller

In diesem Kapitel befassen wir uns mit dem wichtigsten Abstraktionsmechanismus der imperativen Programmierung: Prozeduren. Wir behandeln dabei sowohl die Definition als auch die Verwendung von Prozeduren. Außerdem werden wir den Unterschied zwischen Haupt- und Seiteneffekte von Prozeduren kennenlernen. Diese Unterscheidung wird im weiteren Verlauf der Vorlesung u.a. beim Thema Testen wichtig sein. Als typische Beispiele für den Einsatz von Prozeduren in Java behandeln wir die Bibliothek für Characters und erstellen selbst eine Bibliothek zur statistischen Datenauswertung.

Lernziele dieses Kapitels:

- Prozeduren als statische Methoden Java zu deklarieren und an geeigneter Stelle aufzurufen.
- Die Semantik eines Prozeduraufrufs zu beschreiben.
- Overloading in Prozeduren mit Hilfe von Beispielen zu erklären.
- Haupt- und Seiteneffekte von Prozeduren zu ermitteln.
- Gültigkeitsbereiche von Variablen benennen zu können.

1 Strukturierung durch Prozeduren

Eine ***Prozedur*** ist eine Abstraktion einer Anweisung (bzw. einer Folge von Anweisungen). Sie gibt der Anweisung einen Namen und legt fest, was die Parameter der Anweisung sind.
Dies ermöglicht:

- *Wiederverwendung:* Wir können die gleiche Berechnung bzw. Funktionalität an verschiedenen Stellen und in unterschiedlichen Kontexten verwenden ohne den Code kopieren zu müssen.

- *Schnittstellenbildung und Information Hiding:* Wir verbergen die Details der Implementierung und interagieren nur über eine festgelegte Schnittstelle. Dies erlaubt es, ein Problem in einfachere Teilprobleme aufzuteilen und diese getrennt von einander zu behandeln. Außerdem kann die konkrete unterliegende Implementierung ausgetauscht werden, solange die Schnittstelle und die tatsächliche Funktionalität nicht verändert wird.

Prozeduren können Ergebnisse liefern. Darüber hinaus ermöglichen Prozeduren die rekursive Ausführung von Anweisungen (vgl. Kapitel 08 “Terminierung”).

Beispiel Schauen wir uns zunächst ein einfaches Beispiel mit folgender Aufgabenstellung an:

> Berechne den Absolutbetrag einer ganzen Zahl, gespeichert in Variable i, und schreibe ihn in die Variable x.

Wir können die Aufgabe mit folgendem Algorithmus umsetzen:

1. Wenn i größer oder gleich 0, liefere i als Ergebnis; andernfalls $-i$.
2. Weise das Ergebnis an x zu.

```
public class Absolutbetrag {
  public static void main(String[] args){
    int i;
    int x;
    i = StdIn.readInt();
    int result;
    if (i >= 0) {
      result = i;
    } else {
      result = -i;
    }
    x = result;
    System.out.println(x);
  }
}
```

Abstraktion bzgl. `i`, `result` wird zum Ergebnis.

Um diese Berechnung auch an anderer Stelle verfügbar zu machen, extrahieren wir die Berechnung des Absolutbetrags in eine Prozedur namens `abs`. Diese erhält einen ***formalen Parameter*** `n` für die Zahl, desssen Absolutbetrag ermittelt werden soll. Der Absolutbetrag wird als Ergebnis zurückgeliefert.

```
1  public class Absolutbetrag {
2    // Berechnung des Absolutbetrags
3    public static int abs(int n) {
4      if (n >= 0) {
5        return n;
6      } else {
7        return -n;
8      }
9    }
10
11   public static void main(String[] args){
12     int i;
13     int x;
14     i = StdIn.readInt();
15     // Aufruf der statischen Methode
16     x = abs(i);
```

```
17      StdOut.println(x);
18    }
19  }
```

In der `main`-Methode wird die zusammengesetzte Anweisung durch einen Prozeduraufruf ersetzt. Dabei wird als ***aktueller Parameter*** der Wert der Variablen `i` übergeben.

1.1 Deklaration von Prozeduren in Java

In Java werden Prozeduren als statische Methode implementiert. Jede `.java`-Datei kann eine beliebige[1] Anzahl von statischen Methoden definieren. Eine Prozedur gehört immer zu einer Klasse, muss also zwischen "`class ... {`" und der entsprechenden schließenden Klammer definiert werden. Die Reihenfolge der Methoden in der Klasse ist dabei beliebig.

Im Java-Sprachgebrauch gibt es keine Prozeduren, sondern nur verschiedene Arten von Methoden. Wir verwenden den Begriff *Prozedur* hier austauschbar mit dem Begriff der *statischen Methode*, da dieses Sprachkonstrukt in Java zur Implementierung von Prozeduren verwendet wird.

Syntax in Java:

ProzedurDeklaration →
  `public static` TypOderVoid ≪ *Bezeichner* ≫ `(`FormaleParameter`)`
    Anweisungsblock

TypOderVoid → Typ | `void`

FormaleParameter →
    FormalParamListe
  | ϵ

FormalParamListe →
    FormalParam `,` FormalParamListe
  | FormalParam

FormalParam →
    Typ ≪ *Bezeichner* ≫

Semantik Die Ausführung einer Prozedur wird beendet, wenn

- alle Statements der Prozedur ausgeführt wurden; oder
- eine Rückgabeanweisung ausgeführt wird; oder

[1] Genau genommen ist die Anzahl der Prozeduren und Methoden durch die Java Virtual Machine auf maximal 65535 beschränkt (siehe Kapitel 4.11 "Limitations of the Java Virtual Machine" in der Java Virtual Machine Specification)

- eine Exception ausgelöst wird (siehe späteres Kapitel),

je nachdem, welcher Fall zuerst eintritt.
Die Semantik ist durch den Aufrufmechanismus und den Anweisungsblock, den sogenannten ***Prozedurrumpf*** bestimmt. Wir diskutieren diese in Abschnitt 1.2 und 1.3.

1.2 Rückgabeanweisung

Der Typ des Rückgabewerts einer Prozedur wird in der Prozedurdeklaration festgelegt. Im Prozedurrumpf kann die Rückgabeanweisung verwendet werden, um den Rückgabewert zurückzuliefern.

Syntax in Java:

```
Anweisung  →  ...
    | return ;
    | return Ausdruck ;
```

Semantik:

- Das einfache `return` ist nur in Prozeduren **ohne** Rückgabewert erlaubt.
  Es beendet die Ausführung der Prozedur und setzt die Ausführung an der Aufrufstelle fort.
  Prozeduren ohne Rückgabe haben als Rückgabetyp `void`.
- Die Anweisung `return` mit Ausdruck ist nur in Prozeduren **mit** Rückgabewert erlaubt. Zunächst wird der Ausdruck ausgewertet. Dann wird die Ausführung der Prozedur beendet. Der Wert des Ausdrucks wird als Ergebnis zurückgeliefert und die Ausführung an der Aufrufstelle fortgesetzt. Der Typ des Rückgabewertes muss dem deklarierten Rückgabetyp entsprechen.
  Prozeduren, die Ergebnisse liefern, heißen ***Funktionsprozeduren***.

Eine Methode kann immer nur (max.) einen Wert zurückliefern, auch wenn sie mehrere `return`-Anweisungen enthält.
Falls der Rückgabetyp einer Prozedur nicht `void` ist, prüft der Compiler, dass es am Ende jedes Pfades durch den Code eine `return`-Anweisung gibt. Falls nicht, meldet der Compiler den Fehler `missing return statement`.
Der Code nach einer `return`-Anweisung gilt als nicht erreichbar. Deshalb meldet der Compiler den Fehler `unreachable statement`, falls nach einem `return` weitere Anweisungen folgen.

1.3 Prozeduraufruf (engl. procedure call)

Die Verwendung einer Prozedur erfolgt über den Prozeduraufruf. Dazu benötigt man den Name der Prozedur sowie die aktuellen Parameter.

Syntax in Java:

MethodenAufruf → ≪ *Bezeichner* ≫ (AktuelleParameterListe)

wobei

AktuelleParameterListe → ε
| AusdruckListe

AusdruckListe → Ausdruck
| Ausdruck , AusdruckListe

Semantik:

1. Werte die Ausdrücke der aktuellen Parameter von links nach rechts aus.
2. Übergebe die Parameter an den aktuellen Prozeduraufruf.
3. Führe die Anweisungen der Prozedur aus.
4. Liefere den Rückgabewert, wenn vorhanden.

Im Detail funktioniert die Parameterübergabe folgendermaßen: Beim Prozeduraufruf wird eine neue Prozedurinkarnation erzeugt (siehe Abschnitt 4.1). Für jeden Prozedurparameter wird dabei ein Speicherplatz angelegt und mit dem Wert des entsprechenden aktuellen Parameters initialisiert.

Beispiel: Maximum von drei Zahlen Wir schreiben den Code zur Maximumsberechnung von drei Integern so um, dass die statische Methode `int max(int a, int b)` verwendet wird.

```
1  public class MaxDrei {
2    //Berechnet das Maximum von zwei int-Werten
3    public static int max(int a, int b) {
4      if (a > b) {
5        return a;
6      } else {
7        return b;
8      }
9    }
10
11   public static void main(String[] args) {
12     int x = StdIn.readInt();  // Initialisieren von x
13     int y = StdIn.readInt();  // Initialisieren von y
14     int z = StdIn.readInt();  // Initialisieren von z
15
16     // Berechnung des Maximums von x,y,z erfolgen!
17     int maximum = max(max(x,y), z);
18
19     StdOut.println(maximum);
20   }
21 }
```

1.4 Überladen von Methoden

Methoden können den gleichen Namen haben, wenn sie über die Anzahl oder Typen der Parameter unterschieden werden können. Dies ist eine weitere Art des Überladens (engl. *overloading*), das wir bereits bei Operatoren kennengelernt haben.
Die beiden folgenden Prozeduren berechnen den Absolutwert einer ganzen Zahl bzw. einer Gleitkommazahl.

```
public static int abs(int x) {
  if (x >= 0) {
    return x;
  } else {
    return -x;
  }
}

public static double abs(double x) {
  if (x >= 0.0) {
    return x;
  } else {
    return -x;
  }
}
```

Frage 1: Warum wird die folgende Variante nicht vom Java-Compiler akzeptiert?

```
public static int abs(int x) {
  if (x >= 0) {
    return x;
  } else if (x < 0) {
    return -x;
  }
}
```

1.5 Effekte

Die Ausführung von Prozeduren verändert den Speicherzustand und bewirkt Ein- und Ausgaben. Zusammenfassend sprechen wir von den ***Effekten*** der Ausführung einer Prozedur. Wir unterscheiden dabei:

- ***Haupteffekte***: Liefern von Ergebnissen über den Rückgabewert (in anderen Sprachen auch über das Verändern von Variablenparametern)
- ***Seiteneffekte***: Alle anderen Effekte, zum Beispiel: Ein- und Ausgabe, Verändern globaler Variablen, Verändern von Einträgen eines übergebenen Arrays

Als Beispiel betrachten wir die folgende Prozedur:

```
// Lese Zahlen von der Eingabe ein und addiere sie,
// bis eine Null eingegeben wird.
// Gebe die Summe der eingegebenen Zahlen aus
// und liefere die Anzahl der Eingaben zurueck.
public static int summiereEingaben() {
  int anz = 0;
  int summe = 0;
  int eingabe = 0;
  do {
    StdOut.print("Summand (0 beendet):");
    eingabe = StdIn.readInt();
    summe   = summe + eingabe;
    anz++;
  } while( eingabe != 0 );

  StdOut.println( "Summe: " + summe );
  return anz-1;
}
```

Anwendung der Prozedur `summiereEingaben`:

```
public static void main( String[] args ) {
  boolean b = true;
  while(b) {
    int anzahl;
    anzahl = summiereEingaben();
    StdOut.println("Es wurden " + anzahl + " Zahlen summiert");

    StdtOut.println("Beenden mit j/n:");
    String abbruch = StdIn.readString();
    if(abbruch.equals("j")) {
      b = false;
    }
  }
}
```

Frage 2: Was sind die Haupteffekte und die Seiteneffekte der Prozedur `summiereEingabe()`?

2 Bibliotheken

Sammlungen von Prozeduren (bzw. statischen Methoden), die gedacht sind in verschiedenen Programmen eingesetzt zu werden, nennen wir ***Bibliotheken*** (engl. *libraries*). Dabei werden Prozeduren, die ähnliche Aufgabengebiete abdecken, in jeweils eigene Bibliotheken zusammengefasst. Beispiele hierfür sind die bereits vorgestellten und verwendeten Bibliotheken `StdOut, StdIn, Math`.
Bibliotheken definieren dabei eine Programmierschnittstelle (API, *Applications Programming Interface*). Um eine Prozedur einer Bibliothek verwenden zu können, benötigen Programmierer

- die Signaturen der Prozeduren

- Beschreibung bzw. Spezifikation des Verhaltens der Prozedur

Die Dokumentation der Standardbibliotheken von Java stellt diese Information zur Verfügung (siehe z.B. die Bibliothek zu Character `https://docs.oracle.com/javase/7/docs/api/java/lang/Character.html`).

2.1 Aufruf von Bibliotheks-Prozeduren

Wir haben bereits Aufrufe wie `StdOut.println(s)` in unseren Beispielen verwendet. Hier bezeichnet `StdOut` die Klasse, in der `println` als statische Methode deklariert ist. Im Allgemeinen können Prozeduraufrufe sowohl als Ausdruck als auch als Anweisung verwendet werden. Sie haben die folgende Syntax:

Anweisung → MethodenAufruf ;

Ausdruck → MethodenAufruf

MethodenAufruf →
    ≪ *Bezeichner* ≫ (AktuelleParameterListe)
  | Ausdruck . ≪ *Bezeichner* ≫ (AktuelleParameterListe)

Als Ausdruck vor dem Punkt betrachten wir momentan nur Klassennamen, um auf statische Methoden in anderen Klassen zugreifen zu können. Im Kapitel zur objektorientierten Programmierung werden wir weitere Ausdrücke kennenlernen, die eine ähnliche Syntax verwenden.

Beispiel: Die `Character`-Bibliothek Die `Character`-Bibliothek von Java stellt eine Reihe von statischen Methoden zum Umgang mit Charactern zur Verfügung, unter anderem folgende:

`public class Character`	
`boolean isLetter(char ch)`	Testet, ob das Zeichen ein Buchstabe ist
`boolean isDigit(char ch)`	Testet, ob das Zeichen eine Ziffer ist
`boolean isUpperCase(char ch)`	Testet, ob das Zeichen ein Großbuchstabe ist
`String toString(char ch)`	Liefert ein String-Objekt, das aus einem Character besteht

Diese Prozeduren können folgendermaßen verwendet werden:

```
1  public class CharacterExample {
2    public static void main(String[] args) {
3      char c = StdIn.readChar();
4
5      System.out.println("Buchstabe? " + Character.isLetter(c));
6      System.out.println("Großbuchstabe? " + Character.isUpperCase(c));
7      System.out.println("Ziffer? " + Character.isDigit(c));
8    }
9  }
```

2.2 Die `Statistics`-Bibliothek

Wir wollen im Folgenden eine eigene Bibliothek `Statistics` zur Analyse von Datensammlungen entwickeln. Diese soll folgende API implementieren:

`public class Statistics`	
`double max(double[] a)`	Berechnet das Maximum der Arraykomponenten
`double min(double[] a)`	Berechnet das Minimum der Arraykomponenten
`double mean(double[] a)`	Berechnet den Mittelwert
`double var(double[] a)`	Berechnet die Varianz
`double stddev(double[] a)`	Berechnet die Standardabweichung
`double median(double[] a)`	Berechnet den Zentralwert
`void sort(double[] a)`	Sortiert die Einträge des Arrays
`double[] readArray()`	Liest ein Array von `double`-Werten ein
`void printArray(double[] a)`	Gibt ein Array von `double`-Werten aus

Zur Implementierung dieser Bibliothek speichern wir in einer Datei `Statistics.java` eine Klasse `Statistics`, in der wir die jeweiligen Prozedurdeklarationen einfügen.

```
public class Staticstics {
 // hier werden die Prozedurdeklarationen eingefuegt
}
```

Für die Implementierung der Bibliotheksprozeduren hier behandeln wir nur zunächst nur Arrays, die mindestens ein Element enthalten (d.h. `a.length > 0`).
Arrays können auch Länge 0 haben! Diesen Fall werden wir in Kapitel 07 “Spezifikation und Testen” näher betrachten.

Prozedur zur Berechnung des Maximums Folgende Prozedur ermittelt das maximale Element des Arrays `a`:

```
// Berechnet das Maximum der Arrayeintraege
public static double max(double[] a) {
  double max = a[0];
  for (int i = 1; i < a.length; i++) {
    if (a[i] > max) {
      max = a[i];
    }
  }
  return max;
}
```

Prozedur zur Berechnung des Mittelwerts Die mathematische Definition des Mittelwertes von n Messwerten $a_0, \ldots, a_{n-1}$ ist gegeben durch:

$$\bar{a} = \frac{1}{n} \sum_{i=0}^{n-1} a_i$$

```
// Berechnet den Mittelwert der Arrayeintraege
public static double mean(double[] a) {
  double sum = 0.0;
  for (int i = 0; i < a.length; i++) {
    sum = sum + a[i];
  }
  return sum/a.length;
}
```

Prozedur zur Berechnung der Varianz Die mathematische Definition der korrigierten Stichprobenvarianz von n Messwerten $a_0, \ldots, a_{n-1}$ ist gegeben durch folgende Formel:

$$s^2 = \frac{1}{n-1} \sum_{i=0}^{n-1} (a_i - \bar{a})^2$$

```
// Berechnet die korrigierte Stichprobenvarianz der Arrayeintraege
public static double var(double[] a) {
  double m = mean(a);
  double sum = 0.0;
  for (int i = 0; i < a.length; i++) {
    sum = sum + (a[i] - m) * (a[i] - m);
  }
  return sum / (a.length - 1);
}
```

Prozedur zum Einlesen von Werten Um die Daten im Code als Array verfügbar zu machen, müssen sie zunächst eingelesen werden. Hier eine Prozedur zum Einlesen von `double`-Werten:

```
// Einlesen des Arrays: Als erster Wert wird die
// Grosse des Arrays eingelesen, dann die einzelnen Eintraege.
public static double[] readArray() {
    int n = StdIn.readInt();
    double[] a = new double[n];

    for(int i = 0; i < n; i++){
      a[i] = StdIn.readDouble();
    }
    return a;
  }
```

Prozedur zur Berechnung des Median Der Median kann auf folgende Weise bestimmt werden:

1) Alle Werte werden (aufsteigend) geordnet.

2a) Wenn die Anzahl der Werte ungerade ist, ist die mittlere Wert der Median.

2b) Wenn die Anzahl der Werte gerade ist, definieren wir den Median als arithmetisches Mittel der beiden mittleren Werte.

$$\tilde{a} = \begin{cases} a_{n/2} & \text{falls } n \text{ ungerade ist} \\ \frac{1}{2}(a_{n/2} + a_{n/2-1}) & \text{falls } n \text{ gerade ist} \end{cases}$$

Frage 3: Wie sieht eine Implementierung der entsprechenden Prozedur `double median(double[] a)` aus?
Hinweis: Verwenden Sie dazu die Methode `void sort(double[] a)` der `Statistics`-Bibliothek!

Hinweis: Die Implementierung von `void sort(double[] a)` betrachten wir in der Vorlesungseinheit zu "Suchen und Sortieren" im Detail.

3 Gültigkeitsbereich von Variablenbezeichnern

Der Gültigkeitsbereich (engl. *scope*) eines Variablenbezeichnern sind die Anweisungen, die auf diesen Variablenbezeichner Bezug nehmen können.
In Java gilt:

- Für *Parameter* von statische Methoden entspricht der Gültigkeitsbereich dem Methodenrumpf.
- Für *Variablen*, die in einem Anweisungsblock einer statischen Methode deklariert werden, entspricht der Gültigkeitsbereich den Anweisungen von der Deklaration bis zum Ende dieses Blocks.
- Für Variablen, die im `forInit`-Teil einer `for`-Schleife deklariert werden, ist der Gültigkeitsbereich die `for`-Schleife selbst.
- In Anweisungsblöcken, die *nicht* ineinander verschachtelt sind, können Variablen mit identischen Namen definiert werden (vgl. die Zählvariablen in den for-Schleifen aus dem Kapitel 05 zu Arrays).

Werden Variablen außerhalb ihres Gültigkeitsbereichs verwenden, kann der Compiler Fehler liefern wie `cannot find symbol`.

Frage 4: Was ist der Gültigkeitsbereich der Variablenbezeichner `versuche`, `cash`, `t` in folgendem Code?

```
1  public class Wettspiel {
2    public static void main (String[] args) {
3      int einsatz  = Integer.parseInt(args[0]);
4      int ziel     = Integer.parseInt(args[1]);
5      int versuche = Integer.parseInt(args[2]);
6      int gewinne  = 0;
7
8      for (int t = 0; t < versuche; t++) {
```

```
 9         int cash = einsatz;
10         while (cash > 0 && cash < ziel) {
11           if (Math.random() < 0.5) {
12             cash++;
13           } else {
14             cash--;
15           }
16           if (cash == ziel) {
17             gewinne++;
18           }
19         }
20       }
21       System.out.println(100 * gewinne/versuche + "% Erfolg");
22     }
23   }
```

4 Globale und lokale Variablen

Jede Variablendeklaration definiert eine ***Programmvariable***.
In vielen Programmiersprachen unterscheidet man zwei Arten von Variablen:

- ***Globale Variablen*** sind Variablen, deren Deklaration außerhalb von Prozeduren erfolgt.
- ***(Prozedur-)Lokale Variablen*** werden innerhalb einer Prozedur deklariert.

Jeder Programmvariablen entsprechen zur Ausführungszeit/Laufzeit des Programms eine oder mehrere Speichervariablen:

- Jeder globalen Variablen ist genau eine Speichervariable zugeordnet.
- Ist v eine lokale Variable zur Prozedur p, dann gibt es zu jeder Inkarnation von p eine Speichervariable für v.
- (Weitere Variablentypen behandeln wir später.)

Die ***Lebensdauer*** einer Speichervariable ist der Teil des Ablaufs, in dem sie für die Ausführung bereit steht. Die Lebensdauer globaler Variablen erstreckt sich über den gesamten Ablauf des Programms. Variablen zu lokalen Deklarationen leben solange wie die zugehörige Prozedurinkarnation bzw. über die Dauer der Ausführung des Blocks, in der sie deklariert wurden.
In Java kann man globale Variablen als statische Variablen implementieren. Wir behandeln dies im Detail in einem späteren Kapitel.

Globale Variablen können an beliebigen Stellen in einem Programm modifiziert werden. Daher ist es oft schwierig nachzuvollziehen, wie und wann sich der Zustand einer globalen Variable ändert. Es ist guter Programmierstil, globale Variablen nur in wohlbegründeten Ausnahmefällen zu verwenden.

4.1 Prozedurinkarnation

Beim Aufruf einer Prozedur p wird eine neue ***Inkarnation*** von p erzeugt. Dabei wird:

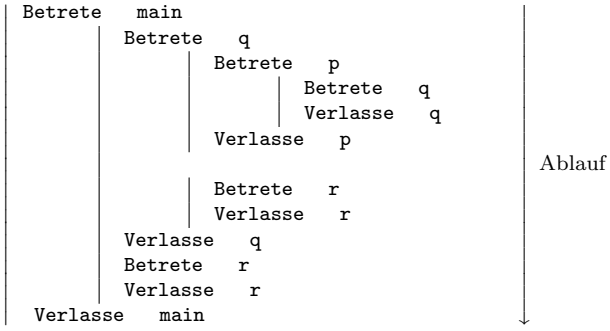
- Speicherplatz für die Parameter und lokalen Variablen angelegt;
- die Anweisung gemerkt, an der nach Ausführung der Prozedurinkarnation die Ausführung des Programms fortzusetzen ist.

Die ***Lebensdauer*** einer Prozedurinkarnation beginnt mit deren Erzeugung und endet mit der Ausführung des Rumpfes zu dieser Inkarnation. Sie erstreckt sich über einen Teil des Programmablaufs.
Die Lebensdauern von Inkarnationen der gleichen Prozedur können sich überlappen.
Deshalb muss es ggf. mehrere Kopien/Instanzen der gleichen lokalen Variablen geben.

Beispiel Das folgende Programm hat mehrere verschachtelte Aufrufe von Prozeduren.

```
1  // Veranschaulichung des Konzepts Aufrufbaum
2  public class Aufrufbaum {
3    public static void main( String[] args ) {
4      System.out.println("Betrete main");
5      q(0);
6      r();
7      System.out.println("Verlasse main");
8    }
9
10   public static void p() {
11     System.out.println("Betrete p");
12     q(5);
13     System.out.println("Verlasse p");
14   }
15
16   public static void q(int i) {
17     System.out.println("Betrete q");
18     if (i == 0) {
19       p();
20       r();
21     }
22     System.out.println("Verlasse q");
23   }
24
25   public static void r() {
26     System.out.println("Betrete r");
27     System.out.println("Verlasse r");
28   }
29 }
```

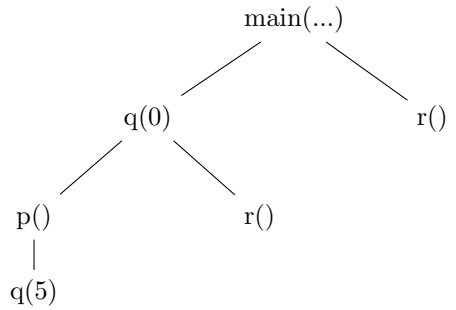
In der Visualisierung der Ausführung entspricht jeder Balken einer Prozedurinkarnation. Es gibt mehrere Inkarnationen der gleichen Prozedur.



4.2 Prozeduraufrufbaum

Der ***Prozeduraufrufbaum*** beschreibt die Prozeduraufrufstruktur in einem Programm- oder Prozedurablauf.
Seine Baumknoten sind mit Prozedurinkarnationen markiert, so dass jede Inkarnation p_i genau die Inkarnationen als Kinder hat, die von p_i aus erzeugt wurden und zwar in der Reihenfolge des Ablaufs.

Beispiel Der Prozeduraufrufbaum für das obige Beispiel hat folgende Struktur:



4.3 Lebensdauer von Arrays

Variablen speichern Werte, d.h. elementare Daten von primitiven Datentypen oder Referenzen auf Objekte (z.B. Arrays, Strings). Wie bereits erwähnt erstreckt sich die Lebensdauer von lokalen Variablen dabei über die zugehörige Prozedurinkarnation bzw. die Dauer der Ausführung des Blocks, in der sie deklariert wurden. Was ist jedoch mit der Lebensdauer der referenzierten Objekten wie Arrays?
Arrays können durch Referenzvariablen referenziert werden. Die Lebensdauer eines Arrays hängt daher nicht von der Lebensdauer seiner Referenzvariablen ab, sondern beginnt mit der Erzeugung des Arrays und endet mit der Terminierung des Programms.

Beispiel Durch den Aufruf der Prozedur `erzeugeArray` wird ein Array erzeugt und initialisiert. Die Referenz auf das Array wird an den Aufrufer zurückgegeben. Über diese Referenz kann später auf das Array zugegriffen werden.

```
static int[] erzeugeArray (int n) {
  if (n < 0) {
        n = 0;
  }
  return new int[size];
}

static void verwendeArray() {
  int[] a = erzeugeArray(78);
  // ...
  a[5] = 87;
}
```

5 Zusammenfassung: Statische und dynamische Aspekte der Programmierung

In diesem Kapitel haben wir Prozeduren als Abstraktionsmittel der imperativen Programmierung kennengelernt. Wir haben gesehen, dass einer Prozedurdeklaration im Programmcode mehrere Prozedurinkarnationen zur Laufzeit entsprechen können. Eine solche Unterscheidung in statische und dynamische Aspekte der Programmierung finden sich an verschiedenen Stellen.
Statisch bezeichnet man in der Programmierung alle Aspekte, die sich auf das Programm beziehen und die man aus ihm ersehen kann, ohne es auszuführen. Statische Aspekte sind unabhängig von Eingaben, wie beispielsweise

- Programmvariablen,
- Prozedurdeklarationen,
- Anweisungen und Ausdrücke.

Dynamisch bezeichnet man alle Aspekte, die sich auf die Ausführungszeit beziehen. Dazu gehören unter anderem:

- Speichervariablen,
- Prozedurinkarnationen,
- Ablauf, Ausführung,
- Aufrufbäume,
- Lebensdauer (von Prozedurinkarnationen, usw.).

Hinweise zu den Fragen

Hinweise zu Frage 1: Der Java-Compiler betrachtet Bedingungen nur dann, wenn sie konstant sind. Deshalb ist für den Compiler auch der Pfad möglich, in dem beide Bedingungen zu `false` auswerten und für diesen Fall fehlt die `return`-Anweisung.

Hinweise zu Frage 2: Der Haupteffekt von `summiereEingabe()` ist das Ergebnis des Aufsummierens der Eingaben. Seiteneffekte sind die Ausgabe(n) von `StdOut.print("Summand (0 beendet):");` sowie `StdOut.prinln("Summe :" + summe)` und das Einlesen der Zahlen mittels `StdIn.readInt()`.

Hinweise zu Frage 3: Eine mögliche Implementierung ist die folgende:

```
// Berechnet den Median der Arrayeintraege
public static double median(double[] a){
  double[] copy = new double[a.length];
  for(int i =0; i < a.length; i++) {
    copy[i] = a[i];
  }

  sort(copy);
  if (copy.length % 2 == 0) {
    return (copy[copy.length/2 - 1] + copy[copy.length/2]) /
 2.0;
  } else {
    return (copy[copy.length/2]);
  }
}
```

- Beim Aufruf von `void sort(double[] a)` wird als Argumentvariable `a` die **Referenz** auf ein Array übergeben.
- Dieses referenzierte Array wird durch den Aufruf der Methode `void sort(double[] a)` verändert.
- Die Methode liefert daher weder ein neues Array noch eine neue Referenz zurück (im Gegensatz zu `double[] readArray()`).
- Um diesen (häufig unerwünschten) Effekt zu verhindern, kann man eine Kopie des Arrays erzeugen, die dann der Methode übergeben und dort modifiziert wird, während das ursprüngliche Array unverändert bleibt.

Hinweise zu Frage 4:

- Der Gültigkeitsbereich von `versuche` entspricht den Anweisungen in Zeile 5-21.
- Der Gültigkeitsbereich von `cash` entspricht den Anweisungen in Zeile 9-20.
- Der Gültigkeitsbereich von `t` entspricht den Anweisungen in Zeile 8-20.