
Java™

Wiederholung

- So sieht ein „leeres“ Java-Programm aus

```
public class Programmname {

    public static void main (String[] args) {

        // Hier stehen die Anweisungen

    }

}
```

Wiederholung

- Anweisungen mit Variablen

  - Deklaration einer Variablen (muss initialisiert werden)
    ```
    Datentyp variablenName;
    ```

  - Deklaration und Initialisierung einer Variablen
    ```
    Datentyp variablenName = Wert;
    ```

  - Zuweisung eines Wertes zu einer Variablen
    ```
    variablenName = Ausdruck;
    ```

Wiederholung

- Anweisungen durch Methodenaufrufe
  - Ausgabe auf der Konsole
    ```
    System.out.println(...);
    ```
  - Benutzereingabe einlesen
    ```
    … = scanner.nextDatentyp();
    ```
  - Mathematische Funktionen
    ```
    Math.sqrt(…), Math.sin(…), Math.random()
    ```

Kontrollstrukturen

Kontrollstrukturen

- Sequenz
  - Eine Anweisung wird nach der anderen ausgeführt
- Selektion
  - Anweisung wird in Abhängigkeit einer Bedingung ausgeführt
- Iteration
  - Anweisung wird mehrfach wiederholt ausgeführt

Sequenz

- Reihe von Anweisungen mit Semikolon trennen, eine je Zeile
  - ```
    Anweisung1;
    Anweisung2;
    ...
    ```
- Beispiel
  - Lies eine Variable x vom Benutzer ein
    Gib „Hallo“ aus
    Gib das Doppelte der Variablen x aus

  - ```
    double x = scanner.nextDouble();
    System.out.println("Hallo");
    System.out.println(2*x);
    ```

Selektion

- If-Verzweigung, Bedingte Anweisungen mit Tab einrücken!!!
  - ```
    if (Bedingung) {
        Anweisungen
    }
    ```
- Beispiel (Sequenz und Selektion)
  - Lies das Alter des Benutzers ein
    Wenn das Alter mindestens 18 ist
        gib aus, dass die Volljährigkeit erreicht ist
    Gib „Ende“ aus

  - ```
    int age = scanner.nextInt();
    if (age >= 18) {
        System.out.println("Volljährigkeit erreicht");
    }
    System.out.println("Ende");
    ```

Selektion

- If-else-Verzweigung,
  - ```
    if (Bedingung) {
       Anweisungen
    }
    else {
       Anweisungen
    }
    ```
- Beispiel
  - ```
    int age = scanner.nextInt();
    if (age >= 18) {
       System.out.println("Zugriff erlaubt");
       System.out.println("Filme, Alkohol, etc.");
    }
    else {
       System.out.println("Zugriff verweigert");
    }
    System.out.println("Ende");
    ```

Iteration

- While-Schleife, Wiederholte Anweisungen einrücken!!!
  - ```
    while (Bedingung) {
        Anweisungen
    }
    ```
- Beispiel (Sequenz und Iteration)
  - Lies das Alter des Benutzers ein
    Solange das Alter unter 18 ist
        gib aus, dass noch ein Jahr gewartet werden muss
        erhöhe das Alter um 1

  - ```
    int age = scanner.nextInt();
    while (age < 18) {
        System.out.println("Du musst noch ein Jahr warten");
        ++age;
    }
    ```

Iteration

- Do-While-Schleife, Wiederholte Anweisungen einrücken!!!
  - ```
    do {
       Anweisungen
    } while (Bedingung);
    ```
- Unterschied
  - Anweisungen werden auf jeden Fall einmal ausgeführt
  - Erst am Ende wird die Bedingung geprüft
- Beispiel
  - ```
    int dice;
    do {
       dice = (int)(1+Math.random()*6);
       System.out.println("Du hast eine"+dice+"gewürfelt");
    } while (dice != 6);
    System.out.println("Jetzt kam endlich eine 6.");
    ```

Iteration

- For-Schleife, Wiederholte Anweisungen einrücken!!!
  - ```
    for (AnweisungVorher;Bedingung;AnweisungNachWdh) {
       Anweisungen
    }
    ```
- Entspricht folgender While-Schleife
  - ```
    AnweisungVorher;
    while (Bedingung) {
       Anweisungen
       AnweisungNachWdh;
    }
    ```
- Beispiel
  - ```
    for (int age = scanner.nextInt(); age < 18; ++age) {
       System.out.println("Du musst noch ein Jahr warten");
    }
    ```
    ```

Diagramme

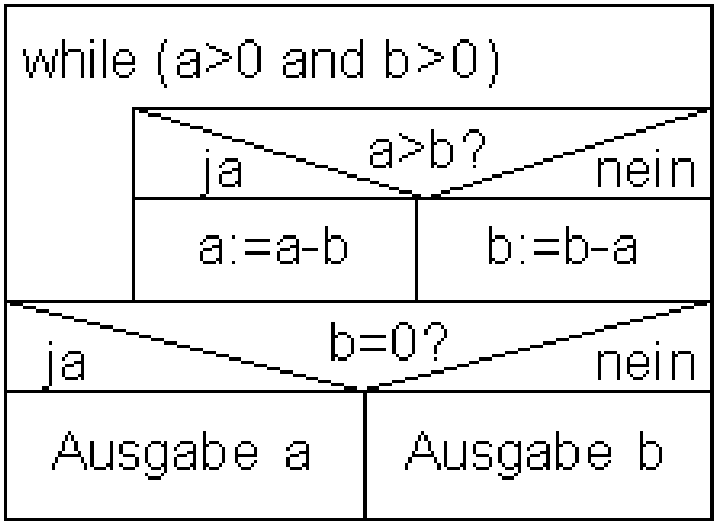
- Java-Programm

```
public class GGT {
   public static void main (String[] args) {
      int a = 123;
      int b = 1452;
      while (a > 0 && b > 0) {
         if (a > b) {
            a = a - b;
         }
         else {
            b = b - a;
         }
      }
      if (b == 0) {
         System.out.println(a);
      }
      else {
         System.out.println(b);
      }
```

Diagramme

- Nassi-Shneidermann-Diagramm (Struktogramm)

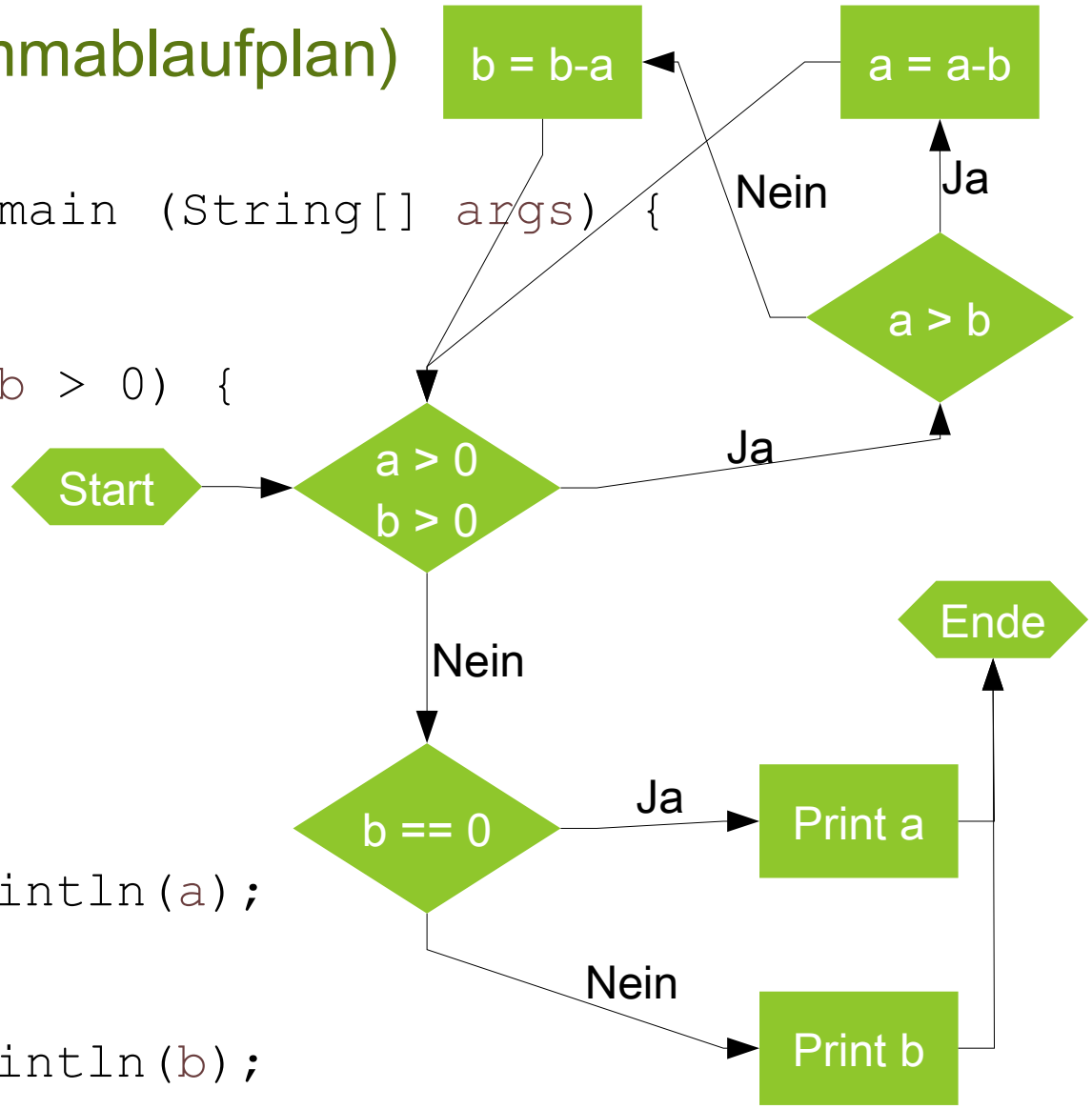
```
public class GGT {
   public static void main (String[] args) {
      int a = 123;
      int b = 1452;
      while (a > 0 && b > 0) {
         if (a > b) {
            a = a - b;
         }
         else {
            b = b - a;
         }
      }
      if (b == 0) {
         System.out.println(a);
      }
      else {
         System.out.println(b);
      }
```



Diagramme

- Ablaufdiagramm (Programmablaufplan)

```
public class GGT {
    public static void main (String[] args) {
        int a = 123;
        int b = 1452;
        while (a > 0 && b > 0) {
            if (a > b) {
                a = a - b;
            }
            else {
                b = b - a;
            }
        }
        if (b == 0) {
            System.out.println(a);
        }
        else {
            System.out.println(b);
        }
```



Blöcke

- Block beginnt `{`
- Block endet `}`
- Gültigkeit von Variablen
  - Variablen, die in einem Block deklariert werden, sind nach dem Ende des Blocks nicht mehr gültig
- Einrücken
  - Blöcke werden eingerückt, damit man auf den ersten Blick sieht, wo wiederholt oder verzweigt wird

Abkürzung

- Ist innerhalb der geschweiften Klammern lediglich eine Anweisung, so kann man die Klammern auch weggelassen
- Beispiel while

```
while (a > 0) {
    --a;
}

while (a > 0)
    --a;
```

- Beispiel if

```
if (a > 0) {
    System.out.println(a);
}

if (a > 0)
    System.out.println(a);
```

Selektion

Java-Programm

- Grundgerüst
  - ```
    public static void main (String[] args) {
        Anweisungen
    }
    ```
- Anweisungen können beliebige Anweisungen sein
  - Sequenzielle Anweisungen
    - Methoden aufrufen (z.B. `System.out.println()`)
    - Variablen deklarieren
    - Variablen Werte zuweisen
  - Selektionen (Verzweigungen)
  - Schleifen (Wiederholungen)
    ```

Selektion

- If-Verzweigung
  - ```
    if (Bedingung) {
       Anweisungen
    }
    ```
- Schlüsselwort `if`
- Bedingung muss vom Datentyp `boolean` sein
- Anweisungen können wieder beliebige Anweisungen sein
  - Stehen innerhalb der geschweiften Klammern
  - Werden nur dann ausgeführt, wenn der Ausdruck in den runden Klammern `true` ist
    ```

Selektion

- If-else-Verzweigung
  - ```
    if (Bedingung) {
       Anweisungen
    }
    else {
       Anweisungen
    }
    ```
- Schlüsselwort `if` und `else`
- Bedingung muss vom Datentyp `boolean` sein
- Anweisungen innerhalb der if-Klammern werden ausgeführt, wenn die Bedingung `true` ist
- Anweisungen innerhalb der else-Klammern werden ausgeführt, wenn die Bedingung `false` ist
    ```

Selektion

- Bedingungsoperator
  - `Bedingung ? AusdruckTrue : AusdruckFalse`
- Bedingung muss vom Datentyp `boolean` sein
- Beide Ausdrücke müssen vom gleichen Datentyp sein
- Das Ergebnis entspricht dem Datentyp des Ausdrucks
- Der Ausdruck vor dem Doppelpunkt wird genommen, wenn die Bedingung `true` ist
- Der Ausdruck hinter dem Doppelpunkt wird genommen, wenn die Bedingung `false` ist

Selektion

- Switch-Anweisung
  - ```
    switch (Variable) {
        case Wert:
            Anweisungen
        break;
        default:
            Anweisungen
        break;
    }
    ```
- Schlüsselwort `switch`, `case`, `default` und `break`
- Variable muss primitiven Datentyp haben oder String
- Je nach Wert der Variablen werden die Anweisungen im entsprechenden case ausgeführt
- Falls keiner der angegebenen Werte übereinstimmt, default
    ```

Switch-Anweisung

- Beispiel

```
int month = scanner.nextInt();

switch (month) {
   case 1: case 3: case 5: case 7: case 8:
   case 10: case 12:
      System.out.println("Der Monat hat 31 Tage.");
   break;
   case 2:
      System.out.println("Der Monat hat 28/29 Tage.");
   break;
   case 4: case 6: case 9: case 11:
      System.out.println("Der Monat hat 30 Tage.");
   break;
   default:
      System.out.println("Das ist kein Monat.");
   break;
}
```

Übung

- Was gibt dieses Programm aus?

```
public class Test {
   public static void main (String[] args) {

      int a = 10;
      int b = 5;
      int c = 0;

      if (a > b) {
         c = a > b ? a : b;
         if (a > c) {
            c += 2*a;
         }
      }

      System.out.println(c);
   }
}
```

Iteration

Java-Programm

- Grundgerüst
  - ```
    public static void main (String[] args) {
        Anweisungen
    }
    ```
- Anweisungen können beliebige Anweisungen sein
  - Sequenzielle Anweisungen
    - Methoden aufrufen (z.B. `System.out.println()`)
    - Variablen deklarieren
    - Variablen Werte zuweisen
  - Selektionen (Verzweigungen)
  - Schleifen (Wiederholungen)
    ```

Iteration

- While-Schleife
  - `while (Bedingung) {`
    `Anweisungen`
    `}`
- Schlüsselwort `while`
- Bedingung muss vom Datentyp `boolean` sein
- Anweisungen können wieder beliebige Anweisungen sein
  - Stehen innerhalb der geschweiften Klammern
  - Werden ausgeführt, solange der Ausdruck in den runden Klammern bei Überprüfung `true` ist
  - Eine Überprüfung findet nicht zwischendurch statt

Iteration

- Do-While-Schleife
  - ```
    do {
       Anweisungen
    } while (Bedingung);
    ```
- Schlüsselwort `do` und `while`
- Bedingung muss vom Datentyp `boolean` sein
- Anweisungen können wieder beliebige Anweisungen sein
  - Stehen innerhalb der geschweiften Klammern
  - Werden auf jeden Fall einmal ausgeführt, danach nur wenn der Ausdruck in den runden Klammern bei Überprüfung `true` ist
  - Eine Überprüfung findet nicht zwischendurch statt
    ```

Iteration

- For-Schleife
  - ```
    for (AnweisungVorher;Bedingung;AnweisungNachWdh) {
       Anweisungen
    }
    ```
- Schlüsselwort `for`
- Bedingung muss vom Datentyp `boolean` sein
- Anweisungen innerhalb der geschweiften Klammern werden wiederholt, wenn die Bedingung `true` ist
- Eine Anweisung (`AnweisungVorher`) wird einmal zu Beginn der Schleife ausgeführt
- Eine Anweisung (`AnweisungNachWdh`) wird nach jedem Schleifendurchlauf ausgeführt
    ```

For oder While -Schleife?

- For-Schleife
  - Anzahl der Wiederholungen steht zu Beginn der Schleife fest
  - Variable wird hochgezählt, um Anzahl der Durchläufe zu zählen
- While-Schleife
  - Abbruchbedingung ist keine Grenze
  - Keine Schleifenvariable zum Mitzählen der Durchläufe wird genutzt
- Do-While-Schleife
  - Wenn die Anweisungen auf jeden Fall einmal durchlaufen werden
  - Ansonsten wie while-Schleife

For oder While -Schleife?

- For-Schleife
  - ```
    for (Anweisung1; Ausdruck; Anweisung2) {
        Anweisungen
    }
    ```
- While-Schleife
  - ```
    Anweisung1;
    while (Ausdruck) {
        Anweisungen
        Anweisung2;
    }
    ```

Endlosschleife

- Was macht dieses Programm?
  - ```
    public class Endlosschleife {
        public static void main (String[] args) {

            int i = 1;
            while (true) {
                System.out.println(i+". Durchlauf");
                ++i;
            }

        }
    }
    ```
    ```

Endlosschleife

- Anweisungen in Schleife wiederholen sich immer wieder
- Kann nicht vom Compiler erkannt werden
- Programmabbruch nötig
  - In Eclipse

- Vermeidung von Endlosschleifen durch Zähler
  - ```
    int counter = 1;
    while (counter < 1000000 && FehlerhafteBedingung) {
        ++counter;
        FehlerhafteAnweisungen;
    }
    ```
    ```

Break und Continue

- Schlüsselwort `break`
  - Springt zur schließenden Klammer der Schleife `}`
    und beendet die Schleife
- Schlüsselwort `continue`
  - Springt zur schließenden Klammer der Schleife `}`
    und prüft die Bedingung, falls `true`, neuer Schleifendurchlauf
- Vermeidung von Endlosschleifen durch Zähler
  - ```
    int counter = 1;
    while (FehlerhafteBedingung) {
        ++counter;
        if (counter > 1000000) {
            break;
        }
        FehlerhafteAnweisungen;
    }
    ```
    ```

Übung

- Was gibt dieses Programm aus?

```
public class Test {
   public static void main (String[] args) {

      int a = 0;
      int b = 2;
      int c = 4;

      while (a < c) {
         --c;
         if (b < a + c) {
            ++a;
         }
         b += 2*a;
      }

      System.out.println(b);
   }
}
```

Übung

- Was gibt dieses Programm aus?

```
public class Test {
   public static void main (String[] args) {
      int x = 1;

      if (12 < 3 || 13 < 4) {
         x = 10;
      }
      else {
         boolean b = true;
         while (b) {
            x *= 2;
            b = x%5 == 1;
         }
      }

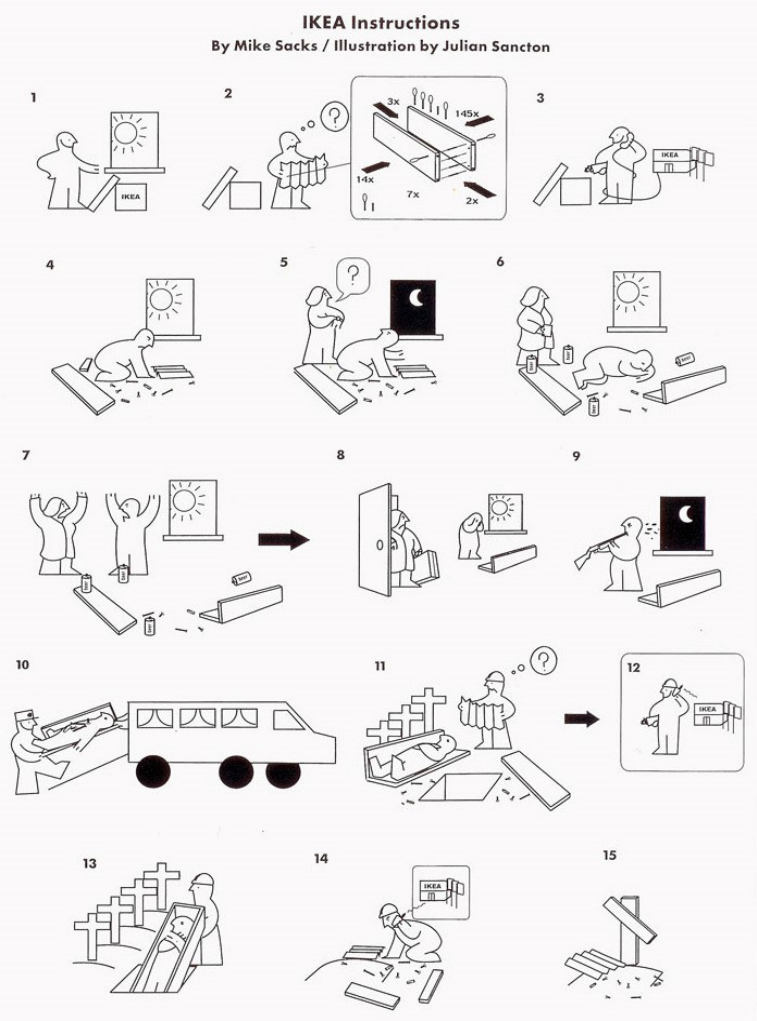
      System.out.println(x);
   }
}
```

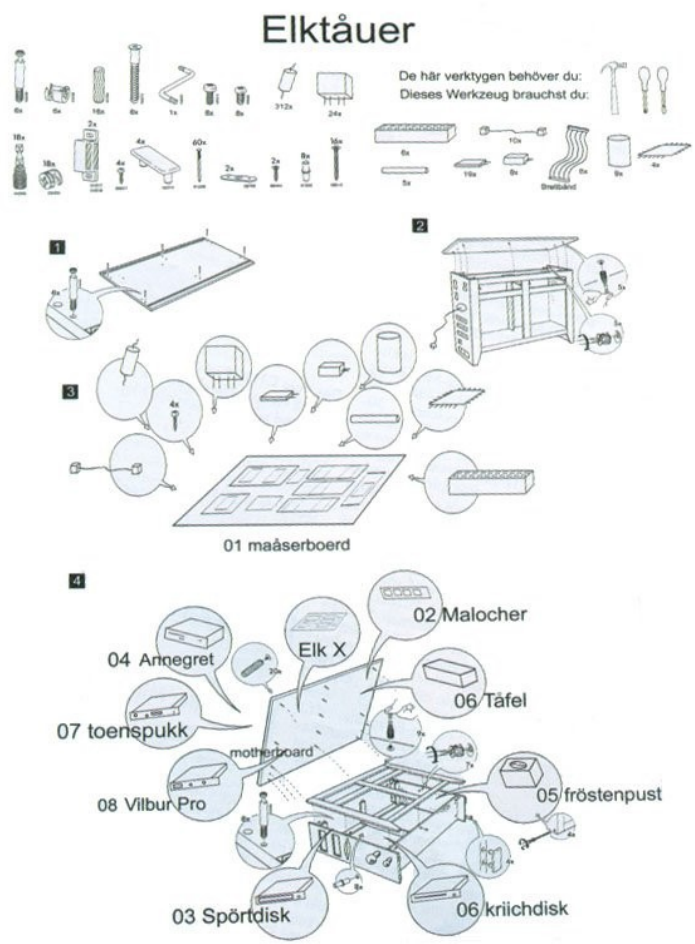
Algorithmus

Algorithmus

- Geregelte Prozeduren zur Lösung definierter Probleme
- Verarbeitungsvorschrift zur systematischen Lösung eines Problems in endlich vielen Schritten
- Genau definierte Handlungsvorschrift zur Lösung eines Problems oder einer bestimmten Art von Problemen in endlich vielen Schritten

Algorithmus





Forderungen an Algorithmen

- Determiniertheit
  - Algorithmus liefert bei jeder Ausführung mit den gleichen Anfangsbedingungen immer das gleiche Ergebnis
- Determinismus
  - Zu jedem Zeitpunkt ist der nächste Handlungsschritt eindeutig definiert

Forderungen an Algorithmen

- Statische Finitheit
  - Die Beschreibung des Algorithmus besitzt eine endliche Länge (Quellcode ist nicht unendlich lang)
- Dynamische Finitheit
  - Der Algorithmus darf nur endlich viele Resourcen benötigen (Speicherplatz, Prozessorleistung, …)
- Terminiertheit
  - Der Algorithmus muss nach endlicher Zeit anhalten (keine Endlosschleife)

Forderungen an Algorithmen

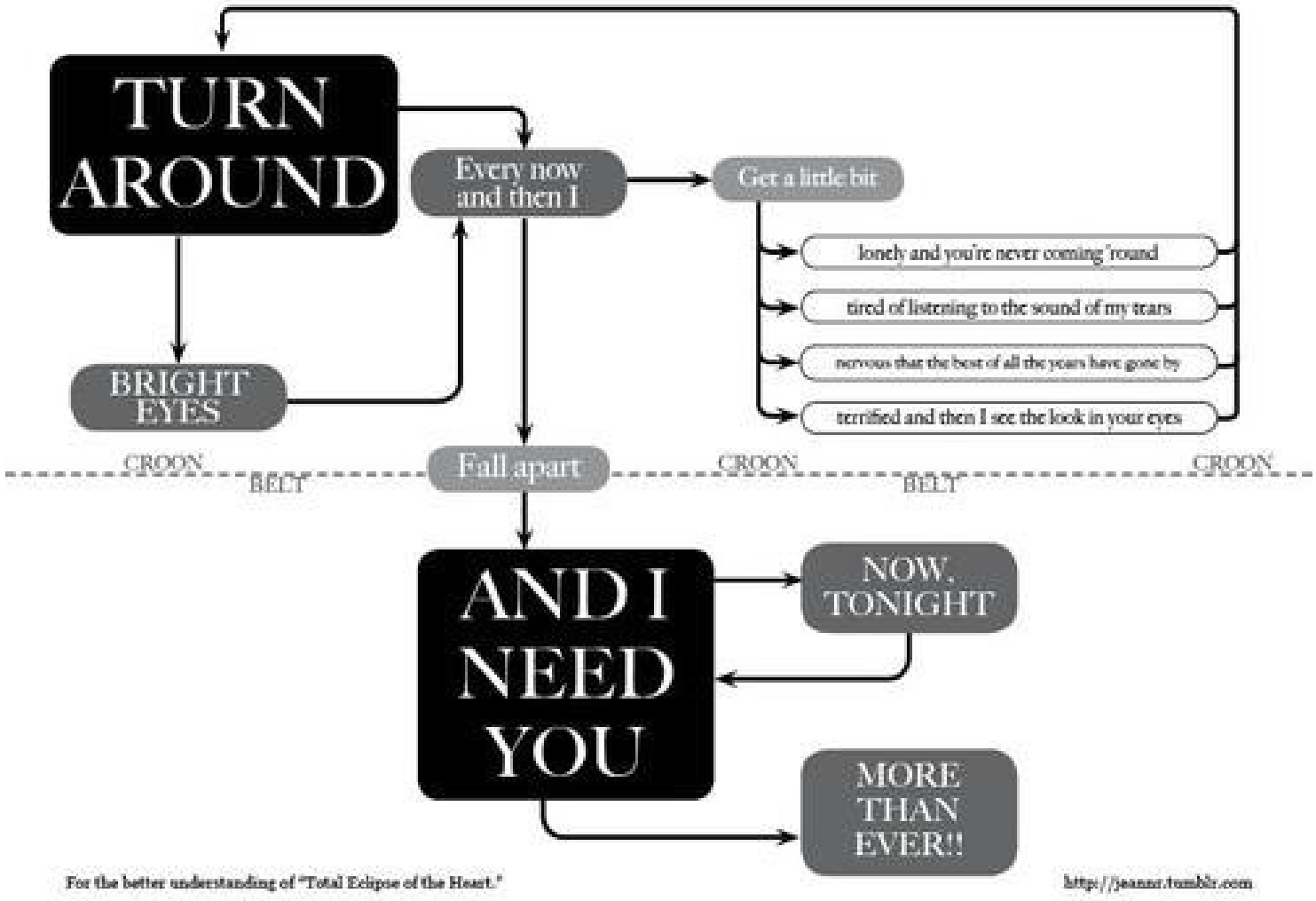
- Genauigkeit
  - Reduktion von Auslöschungen
- Effizienz (Komplexität)
  - Der Algorithmus sollte möglichst ressourcenschonend arbeiten (wenig Speicherverbrauch, wenig Zeit)

Algorithmen Beispiele

- Euklidischer Algorithmus
- Sieb des Eratosthenes
- Gaußsche Osterformel
- Such- und Sortieralgorithmen
- Kryptographische Algorithmen (Ver- und Entschlüsselung)
- Kompression
- Graphentheoretische Algorithmen
  - Kürzeste-Wege
  - Problem des Handlungsreisenden

…

Algorithmen



Algorithmen

- Elementare Anweisungen
  - Teile x durch 2
  - Erhöhe y um 1
- Strukturierte Anweisungen
  - Verzweigung
  - Wiederholung

Hausaufgaben

2.5 Bedingte Anweisungen oder Fallunterscheidungen

2.6 Schleifen