TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Software Entwicklung 1

Annette Bieniusa / Arnd Poetzsch-Heffter

AG Softech
FB Informatik
TU Kaiserslautern

Fallstudie: Lauftagebuch

Erstellen einer Klasse

1. Studiere die Problembeschreibung. Identifiziere die darin beschriebenen Objekte und ihre Attribute und Methoden.
2. Erstelle entsprechende Klassendiagramme.
3. Übersetze die Klassendiagramm in eine Klassendefinition. Füge einen Kommentar hinzu, der den Zweck der Klasse erklärt.
4. Repräsentiere einige Beispiele durch Objekte. Erstelle Objekte und stelle fest, ob sie Beispielobjekten entsprechen.

Aufgabe

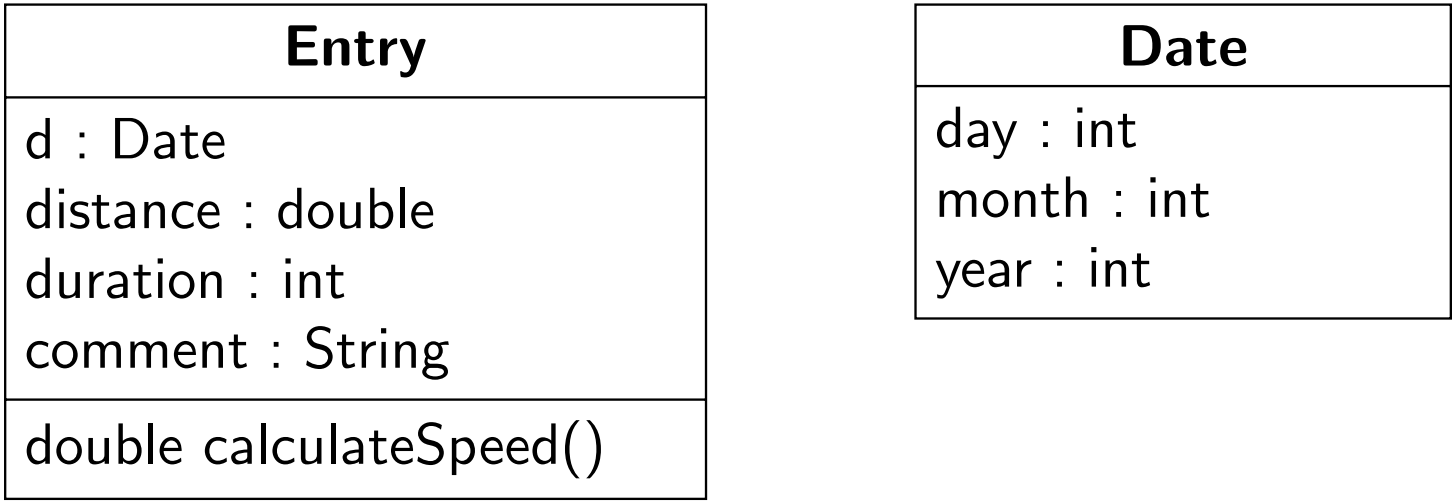
Entwickle ein Programm, das ein persönliches Lauftagebuch führt. Es enthält einen Eintrag pro Lauf. Einträge können hinzugefügt und entfernt werden. Außerdem soll der Eintrag an einer bestimmten Position ermittelt werden.

Ein Eintrag besteht aus dem Datum, der zurückgelegten Entfernung, der Dauer des Laufs und einem Kommentar zum Zustand des Läufers nach dem Lauf. Für einen Eintrag kann man die Durchschnittsgeschwindigkeit ermitteln.

Ein Datum besteht aus Tag, Monat und Jahr.

- Substantive liefern Hinweise auf Klassen oder Attribute
- Verben liefern Hinweise für Methoden

Klassendiagramm



Eine Klasse kann durch ein **Klassendiagramm** spezifiziert werden.

- Klassendiagramme dienen hauptsächlich der Datenmodellierung.
- Sie sind im UML (Unified Modeling Language) Standard definiert.
- Sie enthalten den *Name* der Klasse, *Attribute* mit Typ und *Methoden* mit Signaturen.

Implementierung: Eintrag

```
// Eintrag in einem Lauftagebuch
class Entry {
    Date d;
    double distance; // in km
    int duration;    // in min
    String comment
    Entry(Date d, double distance, int duration,
          String comment) {
        this.d = d;
        this.distance = distance;
        this.duration = duration;
        this.comment = comment;
    }
}
```

Implementierung: Datum

```
// Datum mit Tag, Monat, Jahr
class Date {
    int day;
    int month;
    int year;

    Date(int day, int month, int year) {
      this.day = day;
      this.month = month;
      this.year = year;
    }
}
```

Beispielobjekte

- Beispieleinträge
  - am 5. Juni 2015, 8.5 km in 27 Minuten, gut
  - am 6. Juni 2015, 4.5 km in 24 Minuten, müde
  - am 23. Juni 2015, 42.2 km in 150 Minuten, erschöpft
- . . . als Objekte in einem Ausdruck

```
new Entry (new Date (5,6,2015), 8.5, 27, "gut")
new Entry (new Date (6,6,2015), 4.5, 24, "muede")
new Entry (new Date (23,6,2015), 42.2, 150, "k.o.")
```

- . . . in zwei Schritten mit Hilfsdefinition

```
Date d1 = new Date (5,6,2015);
Entry e1 = new Entry (d1, 8.5, 27, "gut");
```

Implementierung: Geschwindigkeit

```
class Entry {
    Date d;
    double distance; // in km
    int duration;    // in min
    String comment;
    ...
    double calculateSpeed() {
        double h = duration / 60.0;
        return distance / h;
    }
}
```

Darstellung als String I

```
class Date {
    int day;
    int month;
    int year;
    ...
    public String toString() {
        return day + "." + month + "." + year;
    }
}
```

- Alle Referenztypen in Java haben eine Methode `toString()`.
- Man kann eigene String-Repräsentationen definieren. Diese müssen als `public` deklariert werden (siehe Abschnitt “Vererbung”).
- Die String-Repräsentation enthält üblicherweise Informationen zu den Attributwerten.

Darstellung als String II

- `toString()` wird (automatisch) aufgerufen, wenn das Objekt in einem Kontext verwendet wird, das einen String erwartet.

```
Date d = new Date (5,6,2015);
StdOut.println(d.toString());
StdOut.println(d); //alternativ
```

Delegation

```
class Entry {
  Date d;
  double distance; // in km
  int duration;    // in min
  String comment;
  ...
  public String toString() {
        return d.toString() + ": " + distance + " km in "
            + duration + " min; " + comment;
  }
}
```

- Die Implementierung von `toString()` in `Entry` verwendet die Implementierung dieser Methode in `Date`.
- Dieses Muster ist typisch für Klassen, deren Objekte (u.a.) aus Objekten anderer Klassen zusammengesetzt sind (sogenannte **Aggregate** oder **Kompositionen**).

Objekte mit Referenzen zu Objekten der gleichen Klasse

Entwickle ein Programm, das ein Lauftagebuch unter einem Namen führt. Es enthält einen Eintrag pro Lauf. Einträge können hinzugefügt und entfernt werden. Desweiteren soll der Eintrag an einer bestimmten Position ermittelt werden.

Ein Eintrag besteht aus dem Datum, der zurückgelegten Entfernung, der Dauer des Laufs und einem Kommentar zum Zustand des Läufers nach dem Lauf. Für einen Eintrag kann man ausserdem die Durchschnittsgeschwindigkeit ermitteln.

- Bereits erledigt: Klasse `Entry` für einzelne Einträge
- Noch zu tun: Ein Tagebuch enthält eine **beliebige Anzahl** von `Entry`-Objekten

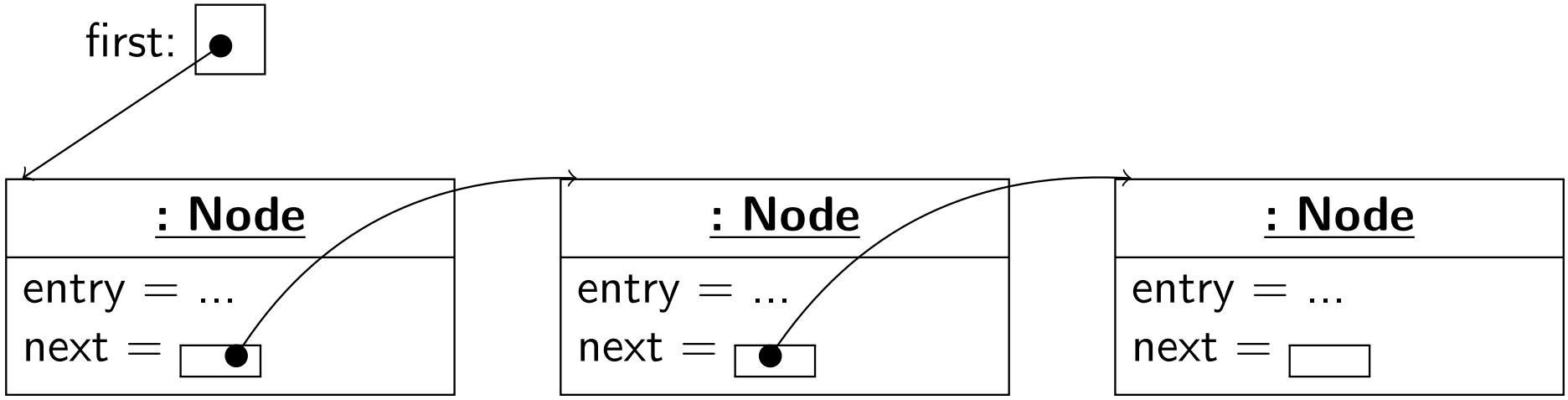
$\Rightarrow$ Idee: Repräsentiere das Tagebuch durch eine **Liste** von Einträgen

Einfachverkettete Listen

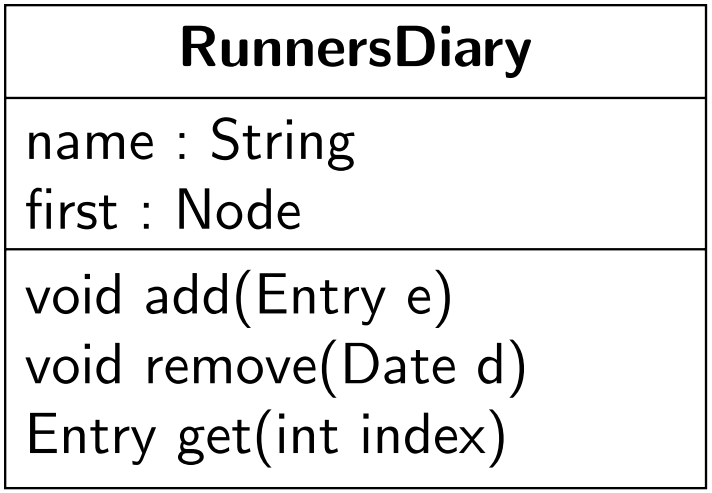
Bei einfachverketteten Listen wird für jedes Listenelement ein Objekt mit zwei Attributen angelegt:

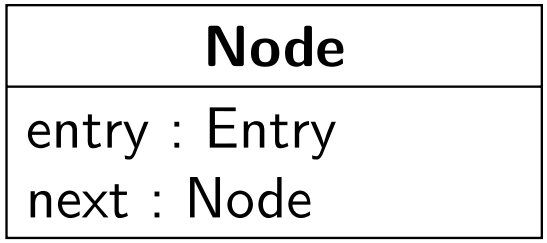
- zum Speichern des Elements
- zum Speichern der Referenz auf den Rest der Liste.

Die Liste erhält also folgende Repräsentation:



Klassendiagramm: Lauftagebuch





Implementierung: Lauftagebuch

```
class RunnersDiary {
    String name;
    Node first;
    RunnersDiary(String name) {
        this.name = name;
    }
    // Fuegt einen neuen Eintrag hinzu
    void add(Entry e) {
      // TODO
    }
    // Entfernt alle Eintraege zu einem Datum
    void remove(Date d) {
      // TODO
    }
    /* Liefert das Element an Position index
       requires 0 <= index < Anzahl der Eintraege
     */
    Entry get(int index) {
      // TODO
    }
    // Darstellung als String
    public String toString() {
      // TODO
    }
}
```

Hinzufügen von neuen Einträgen

```
void add(Entry e) {
  Node newNode = new Node(e,null);
  if (first == null) {
    first = newNode;
  } else {
    Node n = first;
    while (n.next != null) {
      n = n.next;
    }
    n.next = newNode;
  }
}
```

- Erstelle zunächst einen neuen Knoten (ohne Nachfolger!).
- Fallunterscheidung
  - List bisher leer: Füge das Element als erstes Element ein.
  - Sonst: Iteriere zum Ende der Liste und füge das Element dort an.
- *Optimierung*: `RunnersDiary`-Objekte könnten zusätzliche eine Referenz auf das letzte Element der Liste verwalten ($\rightarrow$ Übungen)

Darstellung als String

```
public String toString() {
  String result = "Lauftagebuch von " + name + "\n";
  Node n = first;
  while (n != null) {
    result += n.entry.toString() + "\n";
    n = n.next;
  }
  return result;
}
```

- Iteriere, ausgehend vom ersten Knoten, über die Liste und ermittle die String-Repräsentation der Einträge des jeweiligen Knotens.
- Hinweis: Um Strings zusammenzubauen, verwendet man in der Regel `StringBuilder` (siehe Übungen).

Element an Position

```
/* Liefert das Element an Position index
   requires 0 <= index < Anzahl der Eintraege
 */
Entry get(int index) {
  Node n = first;
  for (int i = 0; i < index; i++) {
    n = n.next;
  }
  return n.entry;
}
```

- Die Indizes sind, analog zu Arrays, ab 0 nummeriert.
- Zunächst iterieren wir über die Listeneinträge bis zum gewünschten Knoten. Danach wird der Eintrag, der in diesem Knoten abgelegt ist, zurückgeliefert.

Beispielobjekte

```
RunnersDiary diary = new RunnersDiary("Hugo");

diary.add(new Entry(new Date(2,3,2015),5,28,"frisch"));
diary.add(new Entry(new Date(5,7,2015),8.2,40,"k.o."));
diary.add(new Entry(new Date(8,9,2015),10.4,54,"muede"));

StdOut.println(diary);

StdOut.println(diary.get(0));
StdOut.println(diary.get(2));
```

Zusammenfassung

- Fallbeispiel: Lauftagebuch
- Komposition von Klassen
- String-Repräsentationen von Objekten mittels `toString()`
- Klassische Datenstruktur: Einfachverkettete Listen