Programmiermethodik 1
Programmiertechnik

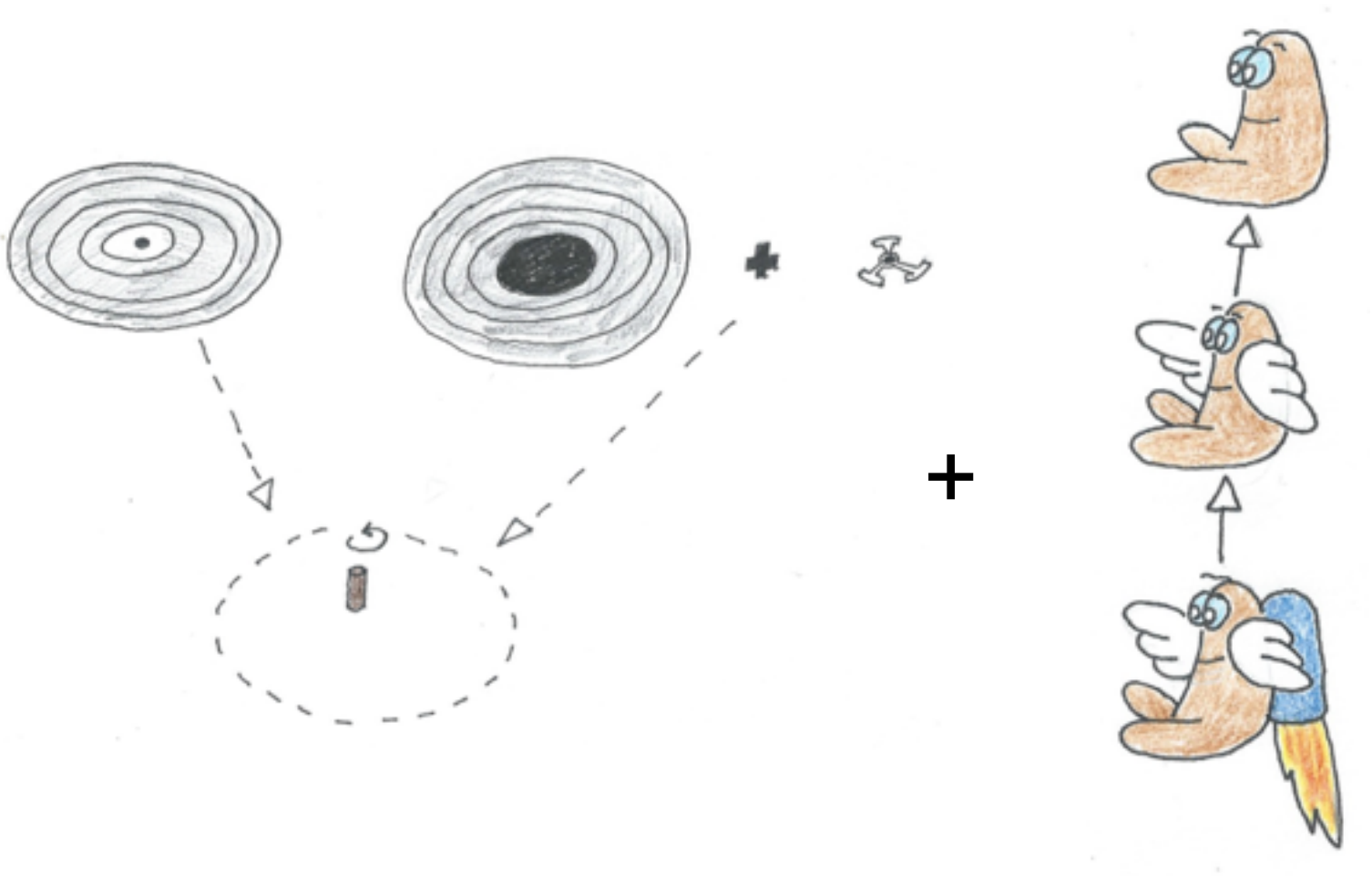
Abstrakte Klassen

Wiederholung

- Vererbung
- Methoden
- Konstruktor und Objektvariablen

Ausblick

Worum gehts?



Agenda

- Abstrakte Basisklassen

Abstrakte Basisklassen

Abstrakte Basisklassen

- Bisher:
  - Interfaces
    - ausschließlich Methodenköpfe, keine Methodenrümpfe, keine Konstruktoren, keine Objektvariablen
  - Konkrete Basisklassen
    - vollständig mit Methodenrümpfen, Konstruktoren, Objektvariablen
- Mittelweg: Abstrakte Basisklassen (engl. abstract base classes)
- Definition mit Modifier
  - abstract class ...
- Methoden in einer abstrakten Basisklasse sind wahlweise …
  - konkret: mit Rumpf (wie in konkreten Klassen), oder
  - abstrakt: nur Signatur (wie bei Interfaces), statt Rumpf nur „;“

Beispiel: Abstrakter Zähler

```
public abstract class AbstrakterZaehler {
   protected int wert = 0;

   public int getWert() {
      return wert;
   }

   public void reset() {
      wert = 0;
   }

   public abstract void erhoehen();
}
```

Ableiten einer abstrakten Basisklasse

- eine abstrakte Basisklasse ...
  - ist unvollständig, wie ein Interface
  - dient lediglich zum Ableiten
  - kann nicht eigenständig instanziiert werden
  - nur über Objekte abgeleiteter Klassen
- abgeleitete Klassen müssen die fehlenden (abstrakten) Methoden der abstrakten Basisklasse implementieren
  - Wenn nicht oder nicht vollzählig implementiert:
  - abgeleitete Klasse ist selbst abstrakte Basisklasse, muss noch weiter abgeleitet werden

Ableiten einer abstrakten Basisklasse

- Beispiel: KonkreterZaehler abgeleitet von AbstrakterZaehler:

```
public class KonkreterZaehler extends AbstrakterZaehler {
   public void erhoehen(){
      wert++;
   }
   ...
```

Vorteile einer abstrakten Basisklasse

	Interface	**Abstrakte Basisklasse**
Signaturen	nur public	ohne Einschränkung
Methoden	ohne Einschränkung	ohne Einschränkung
Objektvariablen	keine	ohne Einschränkung
Klassenvariablen	nur public static final	ohne Einschränkung
Konstruktoren	keine	für abgeleitete Klassen (super), oft protected
Ableitung	von Interfaces	ohne Einschränkung

Einfache und mehrfache Vererbung

- abstrakte Basisklassen mit ausschließlich abstrakten Methoden = "rein abstrakte Basisklasse"
  - engl. pure abstract base class
  - konzeptionell ähnlich zu Interfaces, aber kein Ersatz für Interfaces!
- eine Klasse kann …
  - von einer direkten Basisklasse erben (konkret, abstrakt oder rein abstrakt)
  - beliebig viele Interfaces implementieren
- in Java wird nur einfache Vererbung unterstützt, keine mehrfache Vererbung
  - nach extends darf maximal eine Basisklasse angegeben werden
  - nach implements aber mehrere Interfaces

Blockieren der Vererbung

- in seltenen Fällen sinnvoll: aktives Verhindern der Ableitung
- Modifier final der ganzen Klasse erlaubt keine abgeleiteten Klassen mehr
    - public final class FinalLastWords
    - Populäres Beispiel: Klasse String
    - public class SuperString extends String // Fehler!
- feinere Dosierung
    - Modifier final verhindert Redefinition einer einzelnen Methode

```
public class Bruch {
   public final Bruch mult(Bruch r)
      ...
   }
```

- final-Klasse beendet Folge von Ableitungen
- final-Methode beendet Folge von Redefinitionen

Dynamische Typbestimmung: Motivation

- in manchen Fällen muss zur Laufzeit der konkrete Typ (die Klasse) eines Objekts ermittelt werden
- Beispiel (Code in irgendeiner Anwendung):

```
public void sichereWiederherstellung(Zaehler zaehler) {
   if (zaehler instanceof SpeicherZaehler) {
      // nur bei SpeicherZaehler
      ((SpeicherZaehler) zaehler).speichern();
   }
   zaehler.reset(); // alle Zähler-Typen
}
```

- Probleme:
  - zaehler.speichern() nicht möglich für regulären Zaehler

Typprüfung mit instanceof

- zweistelliger Operator instanceof prüft, ob das Objekt x vom Typ T ist:
  - x instanceof T
- Ergebnis:
  - true: x ist kompatibel zu T (Instanz der Klasse T oder einer abgeleiteten Klasse oder Implementierung des Interfaces T)
  - false ansonsten
- instanceof testet den
  - dynamischen Typ: Laufzeittyp
  - nicht den statischen Typ: gemäß Definition, Sicht des Compilers

Lösung durch Typecast

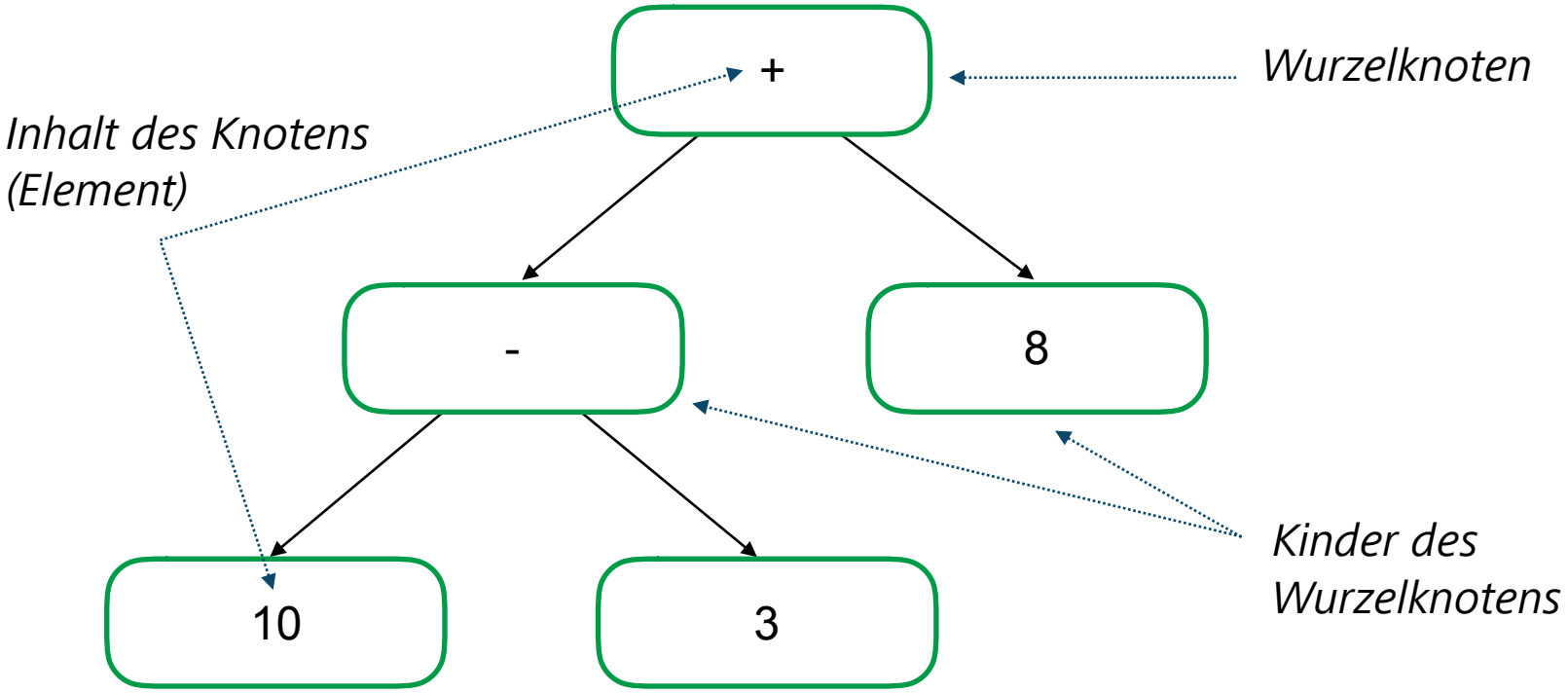
```
public void sichereWiederherstellung(Zaehler zaehler) {
   if (zaehler instanceof SpeicherZaehler) {
      // nur bei SpeicherZaehler
      ((SpeicherZaehler) zaehler).speichern();
   }
   zaehler.reset(); // alle Zähler-Typen
}
```

- Typecast unschön, aber harmlos: Vorangegangener Test stellt Zieltyp sicher
- Klammern nötig wegen Operatorenvorrang
  - Methodenaufruf bindet stärker als Typecast
  - (SpeicherZaehler) zaehler.speichern(); ✗
  - ((SpeicherZaehler) zaehler).speichern(); ✔

Übung: RollenspielCharakter

- Schreiben Sie eine abstrakte Klasse RollenspielCharakter. Jeder RollenspielCharakter hat einen Namen, der im Konstruktor gesetzt wird. Jede RollenspielCharakter kann außerdem kämpfen, allerdings kämpfen die unterschiedlichen Charaktere sehr unterschiedlich (keine Implementierung).
- Schreiben Sie eine Klasse Elf (ist auch ein RollenspielCharakter). Beim Kämpfen schießt ein Elf einen Pfeil.
- Ein Elf kann außerdem einen Zauberspruch sagen.
- Schreiben Sie ein Code-Snippet, bei der für einen gegebenen RollenspielCharakter, die Kampf-Methode aufgerufen wird. Falls es sich bei dem Charakter um einen Elf handelt, wird außerdem ein wenig gezaubert.

Übung: Baum



Größe des Baumes =
Anzahl Kinder + Kindeskinder des Wurzelknotens +1 =
5

Übung: Baum

- Ein Baum besteht aus Knoten.
- Es gibt einen Knoten am oberen Ende der Hierarchie: Wurzelknoten
- jeder Knoten …
    - beinhaltet ein Element.
    - kann Kindknoten beinhalten (maximale Anzahl im Konstruktor gesetzt)
    - kann die Größe des Teilbaumes berechnen von dem er der Wurzelknoten ist: getGroesse()
        - Hinweis für den Wurzelknoten liefert diese Methode die Anzahl der Knoten im Baum

Übung: Baum - Aufgabe 1

- Überlegen Sie sich eine Architektur zur Repräsentation eines Baumes mit Elementen, die Zeichenketten beinhalten
- Erstellen Sie ein Klassendiagramm mit allen benötigten Klassen und Interfaces

Übung: Baum - Aufgabe 2

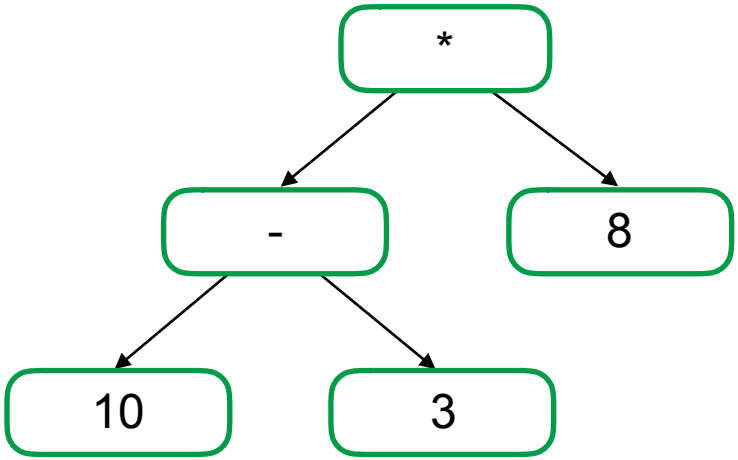
- Schreiben Sie die abstrakte Klasse Element und die Klasse Knoten
- Schreiben Sie Testcode, um einen einfachen Baum, der Zeichenketten repräsentiert, zu erstellen
    - dazu benötigen Sie natürlich eine Implementierung von Element für Zeichenketten

Übung: Baum - Aufgabe 3

- Schreiben Sie eine Klasse Traversierung mit Methode ausgeben(), die als Argument einen Knoten bekommt.
- In der Methode wird das Element des Knotens ausgegeben und es wird (rekursiv) die Methode ausgeben() für die Kindknoten aufgerufen.

Übung: Baum - Aufgabe 4

- Implementieren Sie die notwendige Funktionalität, um einen Baum aus Arithmetischen Operationen zu repräsentieren und auszuwerten.
- Wir betrachten nur die binären Operationen +, -, *, /
- Sie benötigen dazu einen neuen Elementtyp und eine neue Traversierungs-Methode.



= (10 - 3) * 8 = 7 * 8 = 56

Zusammenfassung

- Abstrakte Basisklassen