Karlsruher Institut für Technologie

Vorlesung Softwaretechnik I
Übung 5

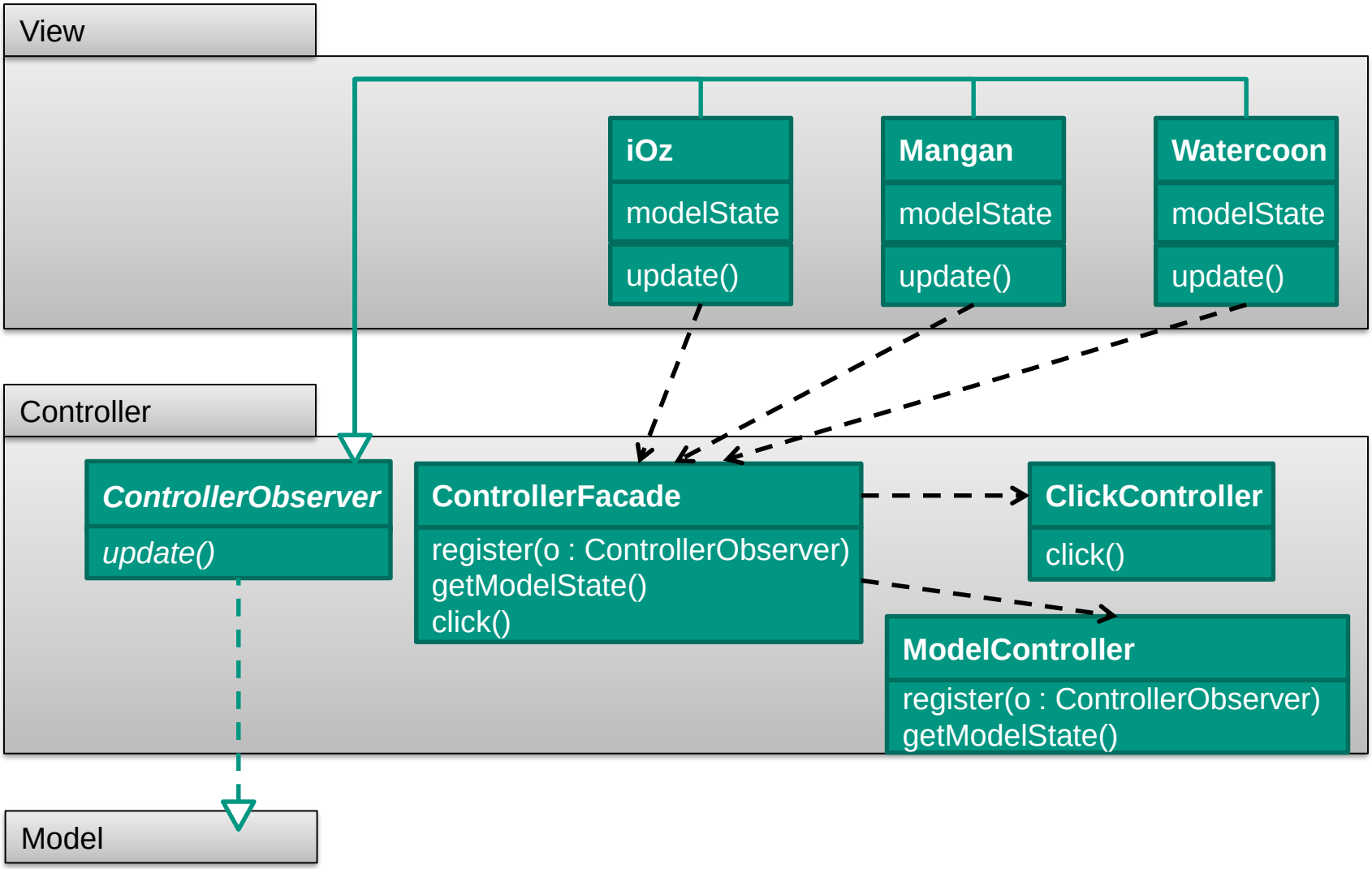
SWT I – Sommersemester 2019
Walter F. Tichy, Sebastian Weigelt, Tobias Hey

IPD Tichy, Fakultät für Informatik

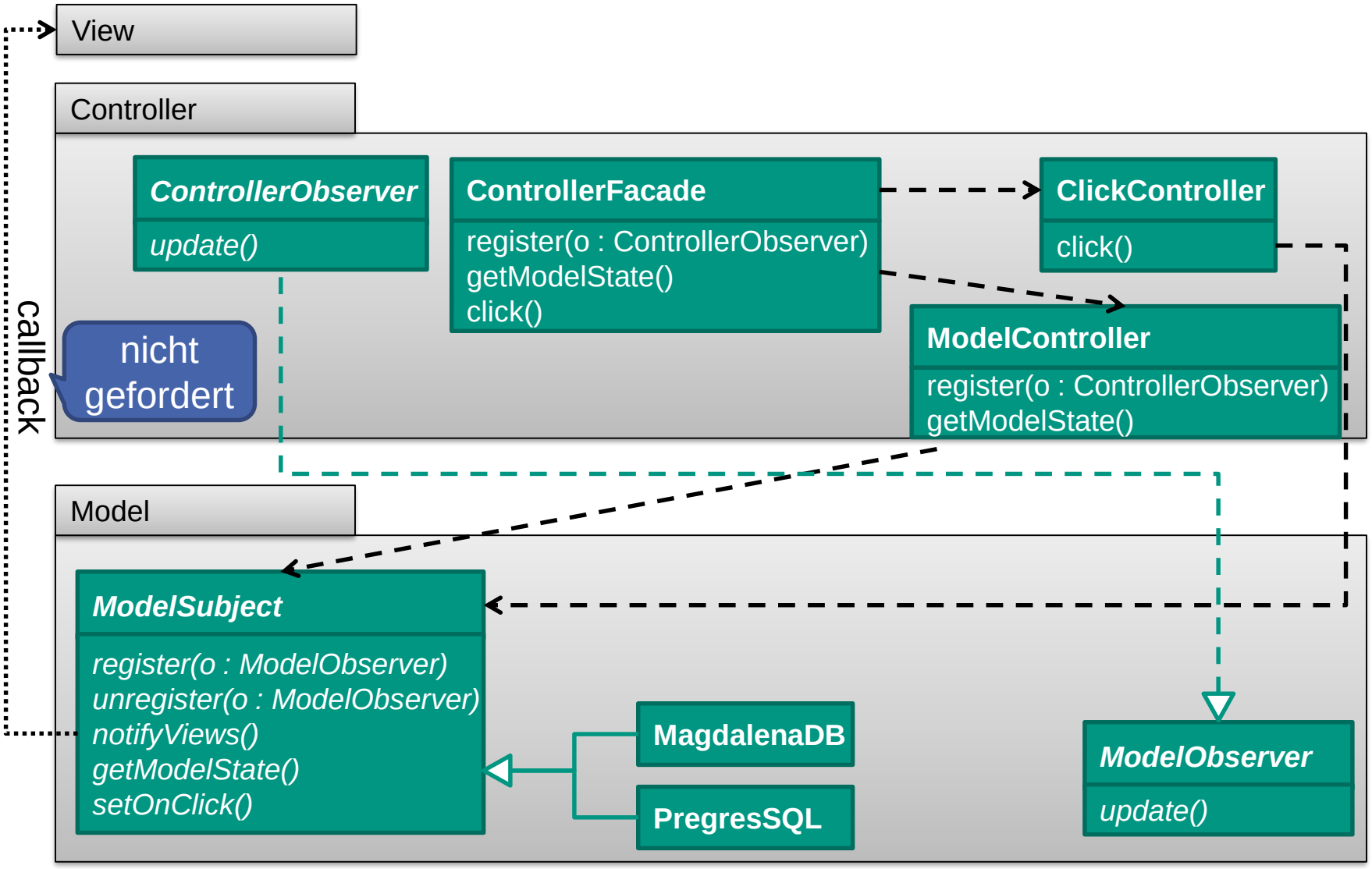
Architekturstile

AUFGABE 1

Aufgabe 1
Architekturmuster – Architektur



Aufgabe 1
Architekturmuster – Architektur



Aufgabe 1
Architekturmuster – Entwurfsmuster

- Beobachter

- Warum?
  - Neuer Zustand des Modells muss zu den Präsentationens-Komponenten gelangen

- Wie?
  - Präsentationen als Beobachter
  - Anmeldung bei Modell über Kontroll-Komponente
  - Benachrichtigung mittel Rückrufs (*callback*)

Aufgabe 1
Architekturmuster – Entwurfsmuster

- Delegierer (in der Vorlesung nicht vorgestellt)

engl. *delegate* oder *delegation*

- Warum?
  - Registrierung und das Holen des Modell-Zustands müssen durch die intransparente Kontrollschicht zum Modell delegiert werden
- Wie?
  - ClickController delegiert Methodenaufrufe an entsprechende Modell-Methoden
  - alle Fassadenmethoden delegieren ebenfalls
  - Fassaden-Muster somit eine Spezialisierung des Delegierer-Musters
- Je nach Lösung weitere EM möglich!

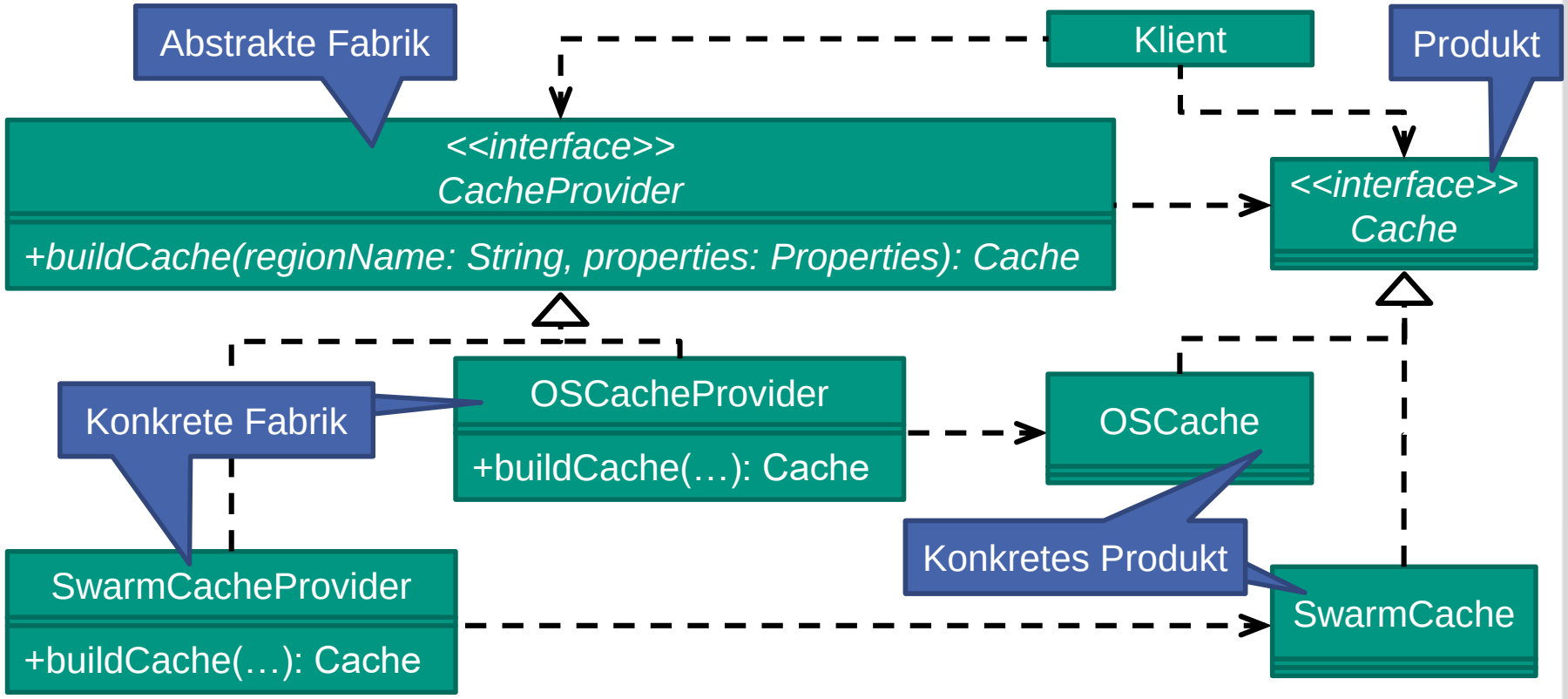
Jäger der verlorenen Entwurfsmusters

AUFGABE 2

Aufgabe 2: Jäger des verlorenen Entwurfsmusters

Abstrakte Fabrik

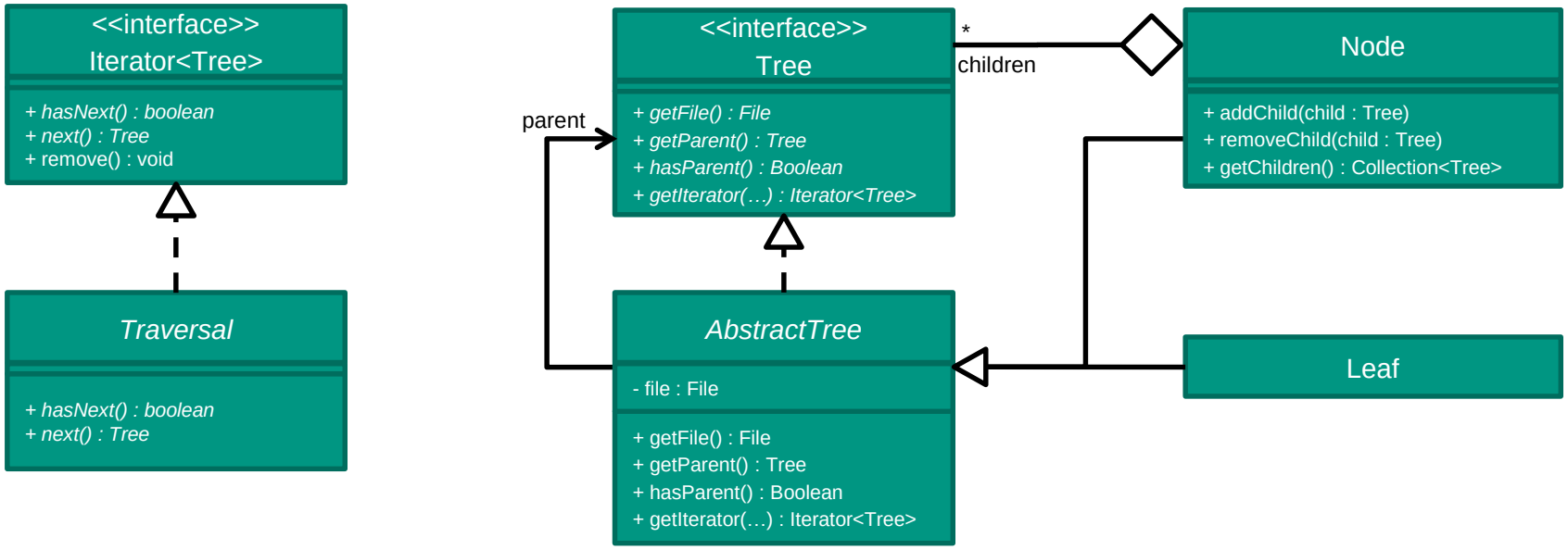
Bietet eine Schnittstelle zum Erzeugen von Familien verwandter oder voneinander abhängiger Objekte, ohne ihre konkreten Klassen zu benennen.



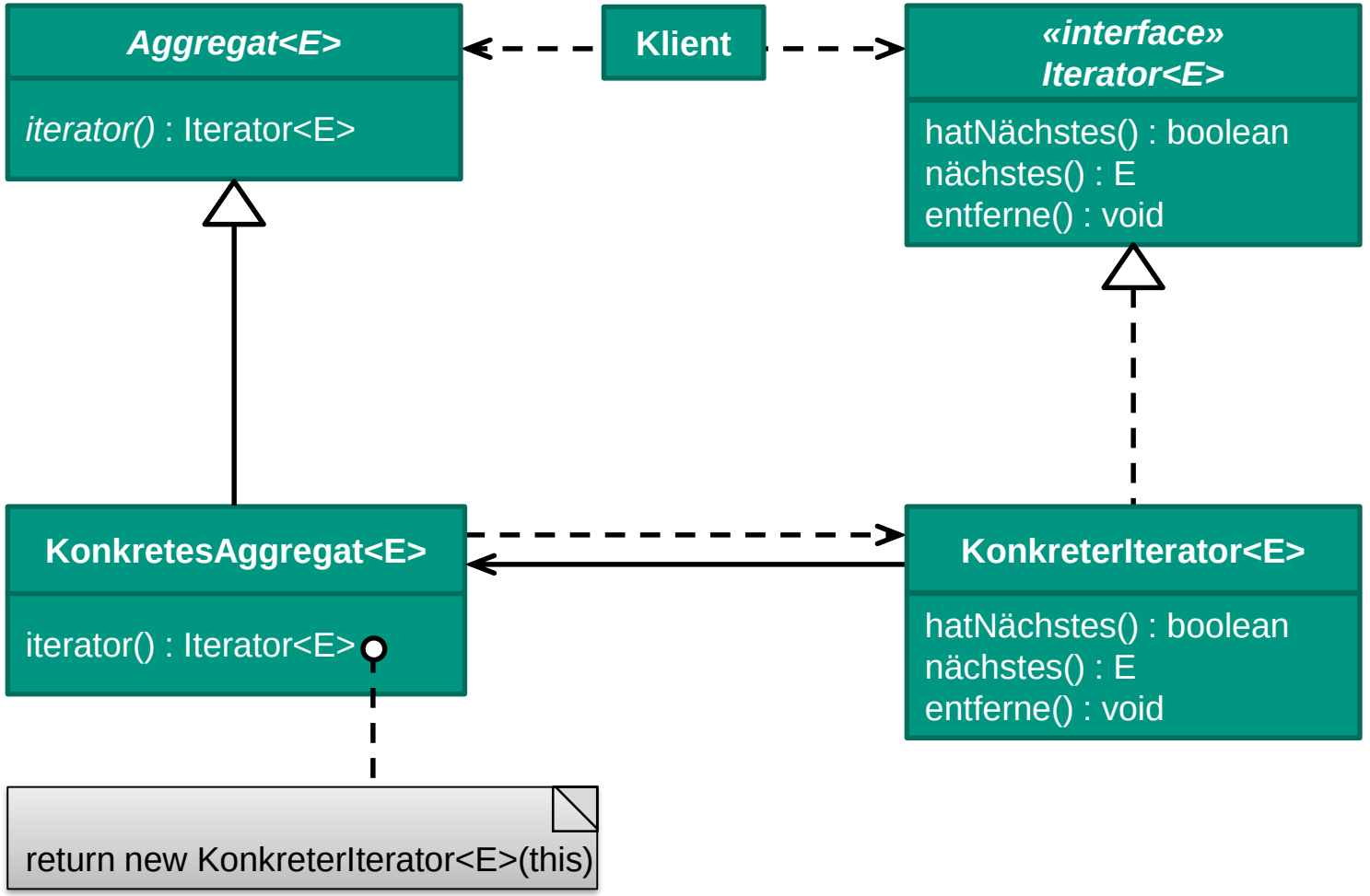
Entwurfsmuster umsetzen

AUFGABE 3

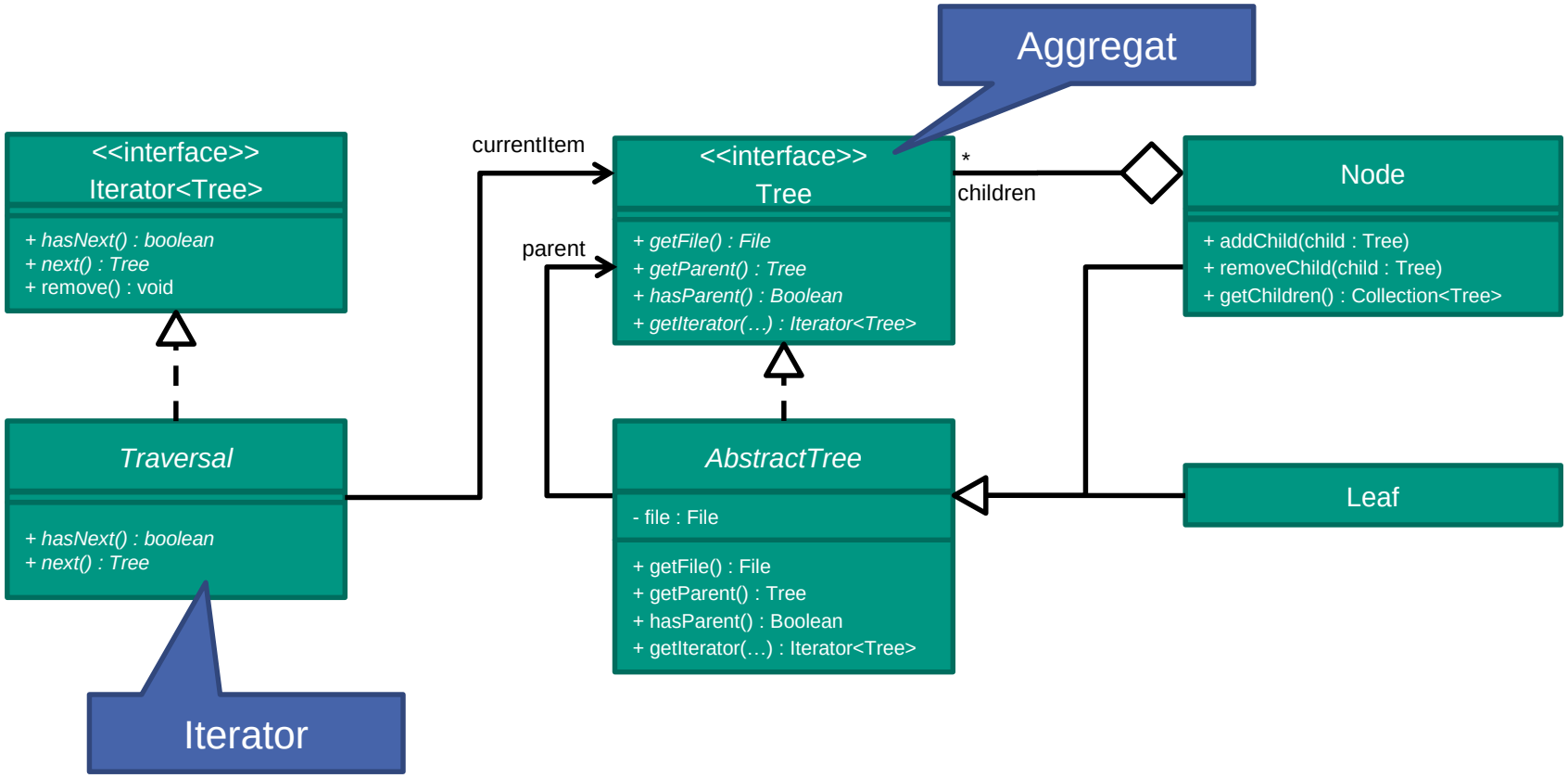
Aufgabe 3: Entwurfsmuster umsetzen



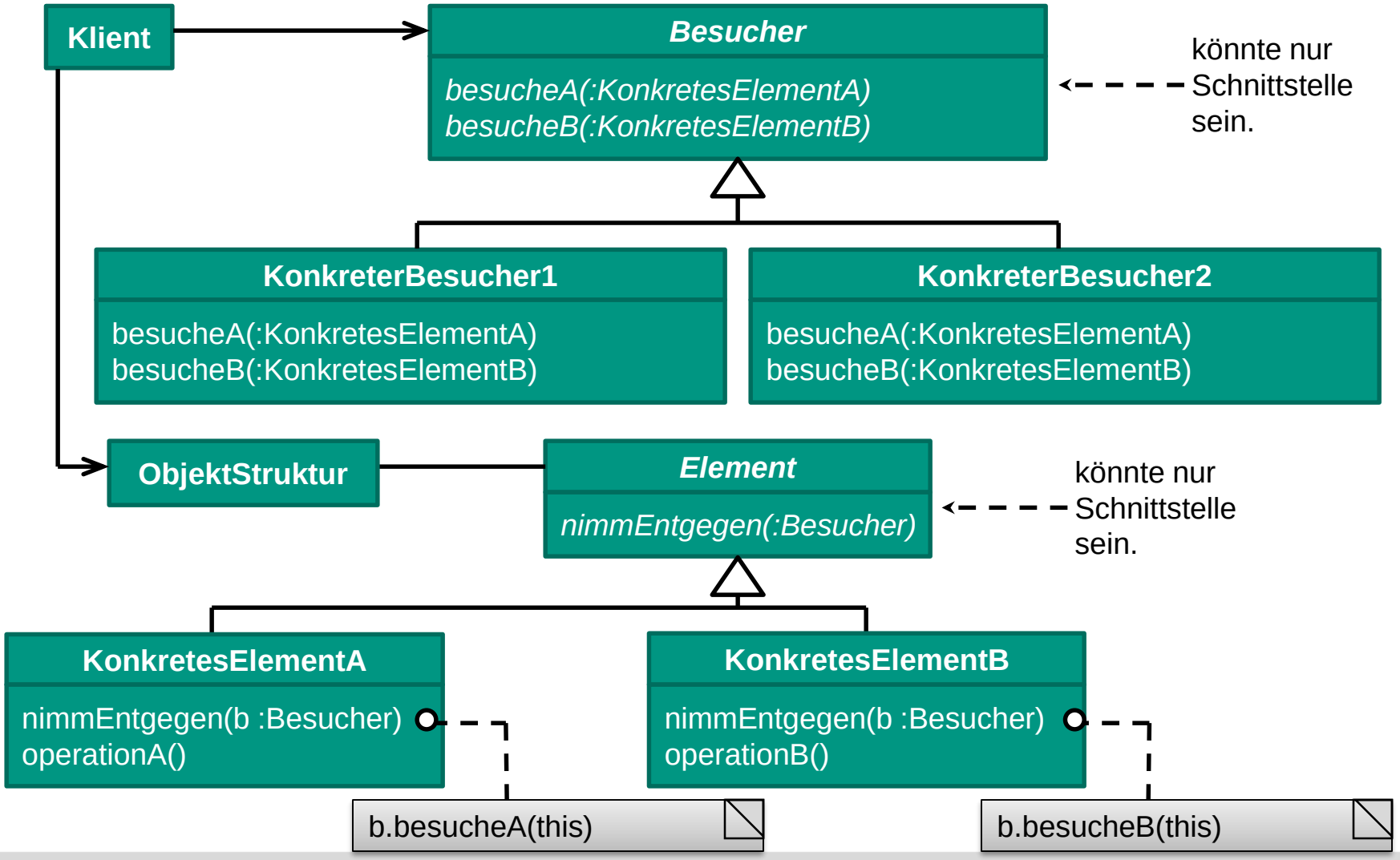
Erinnerung: Iterator Struktur



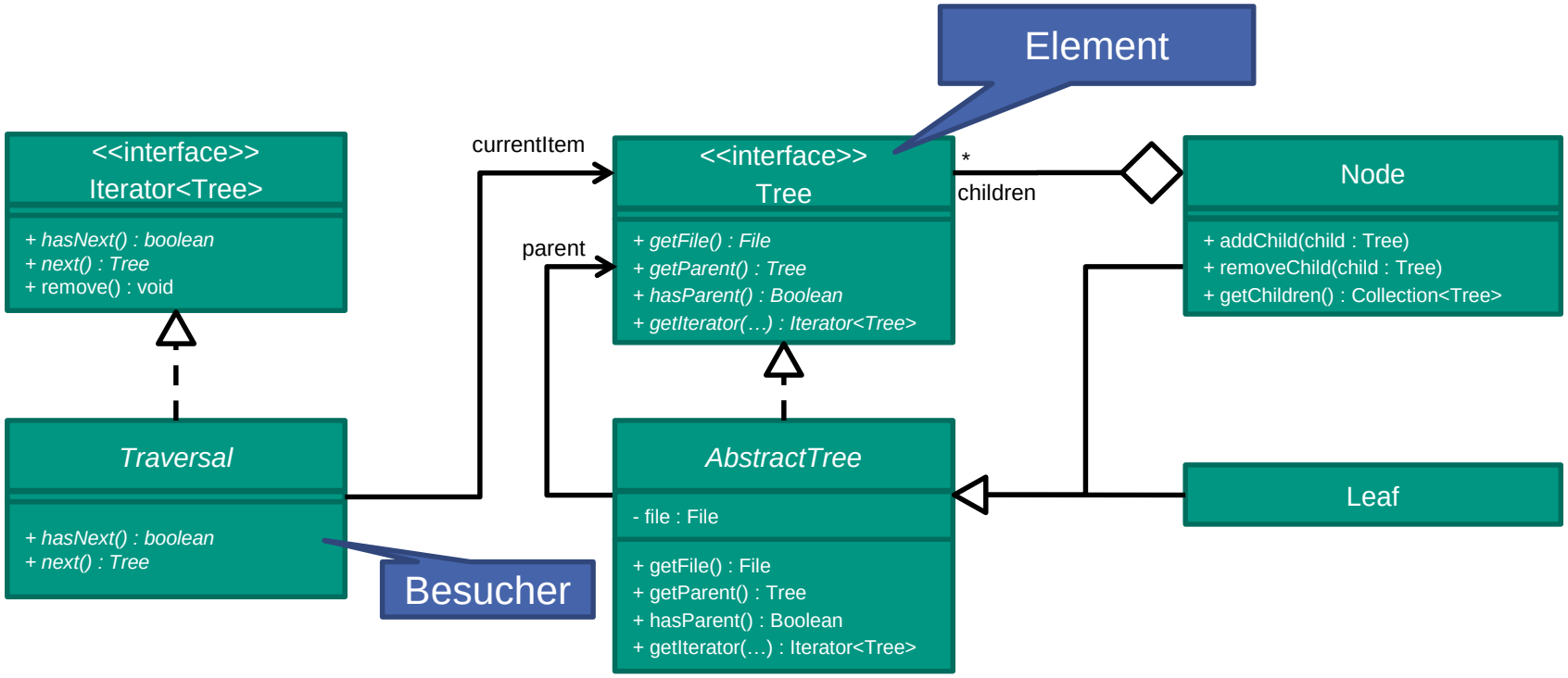
Aufgabe 3: Entwurfsmuster umsetzen



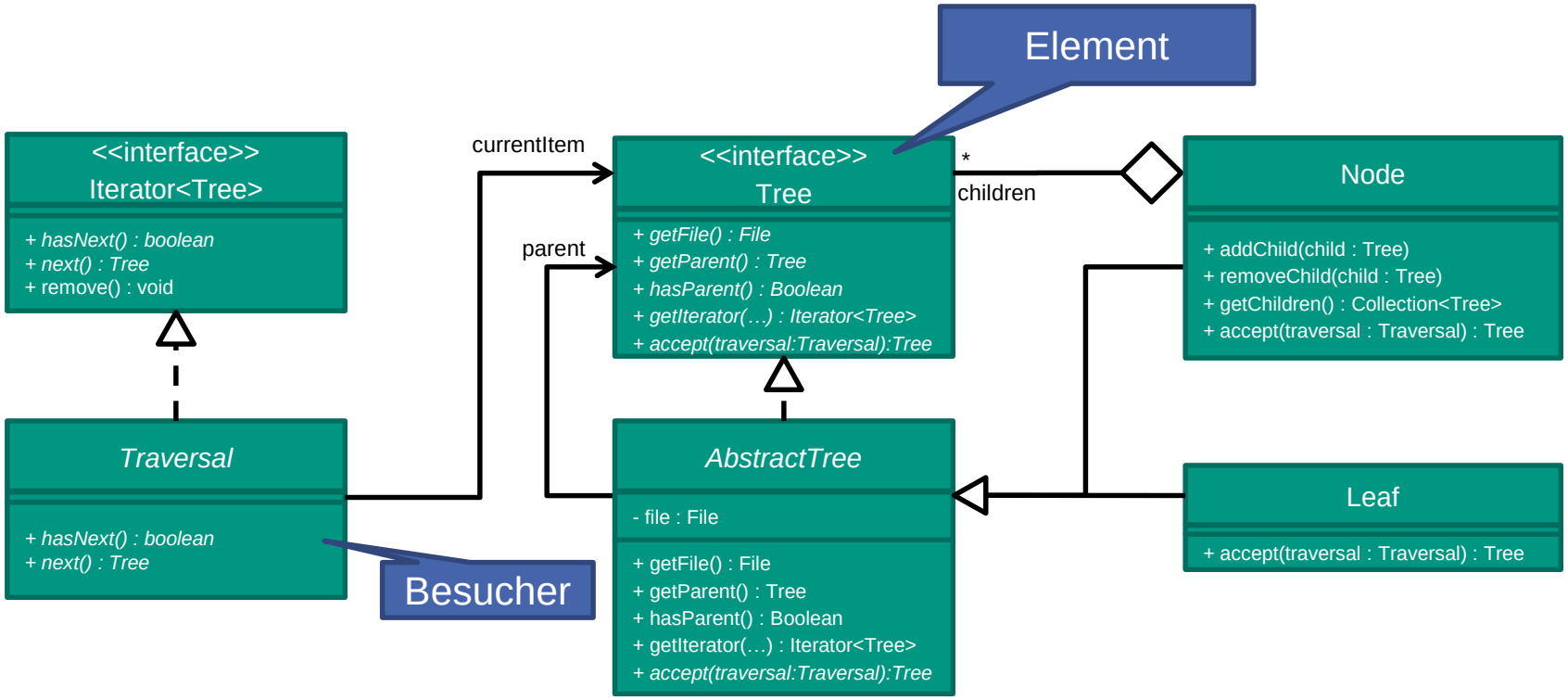
Erinnerung: Besucher Struktur



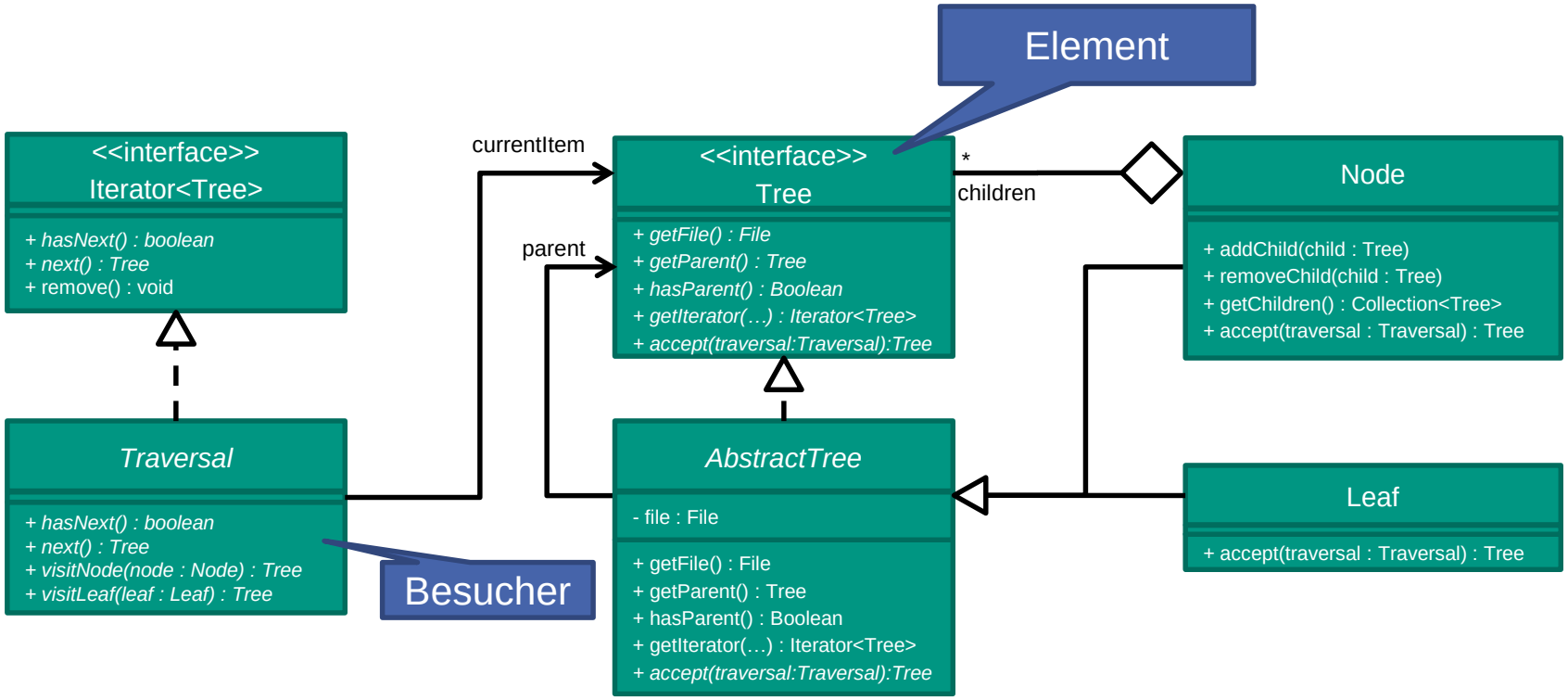
Aufgabe 3: Entwurfsmuster umsetzen



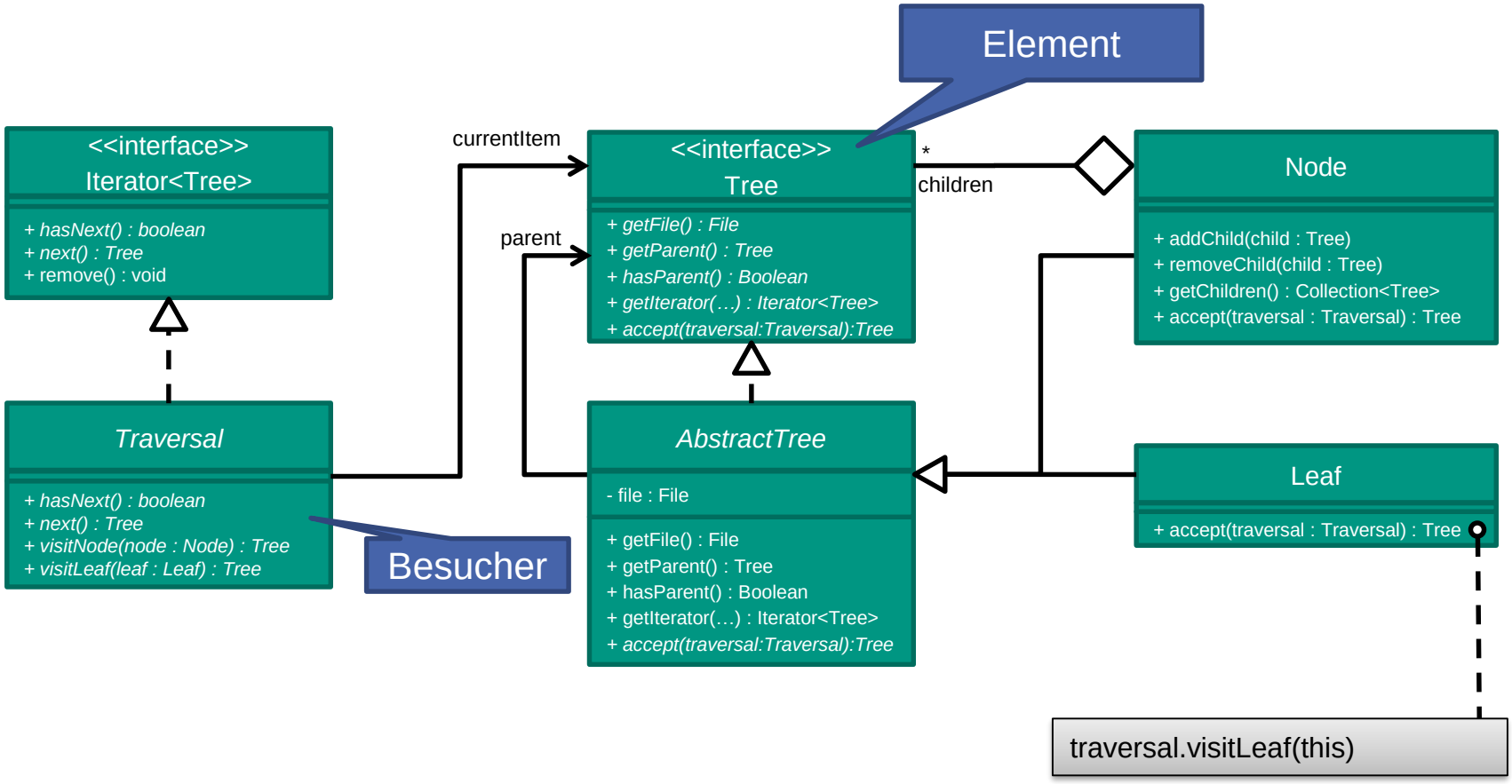
Aufgabe 3: Entwurfsmuster umsetzen



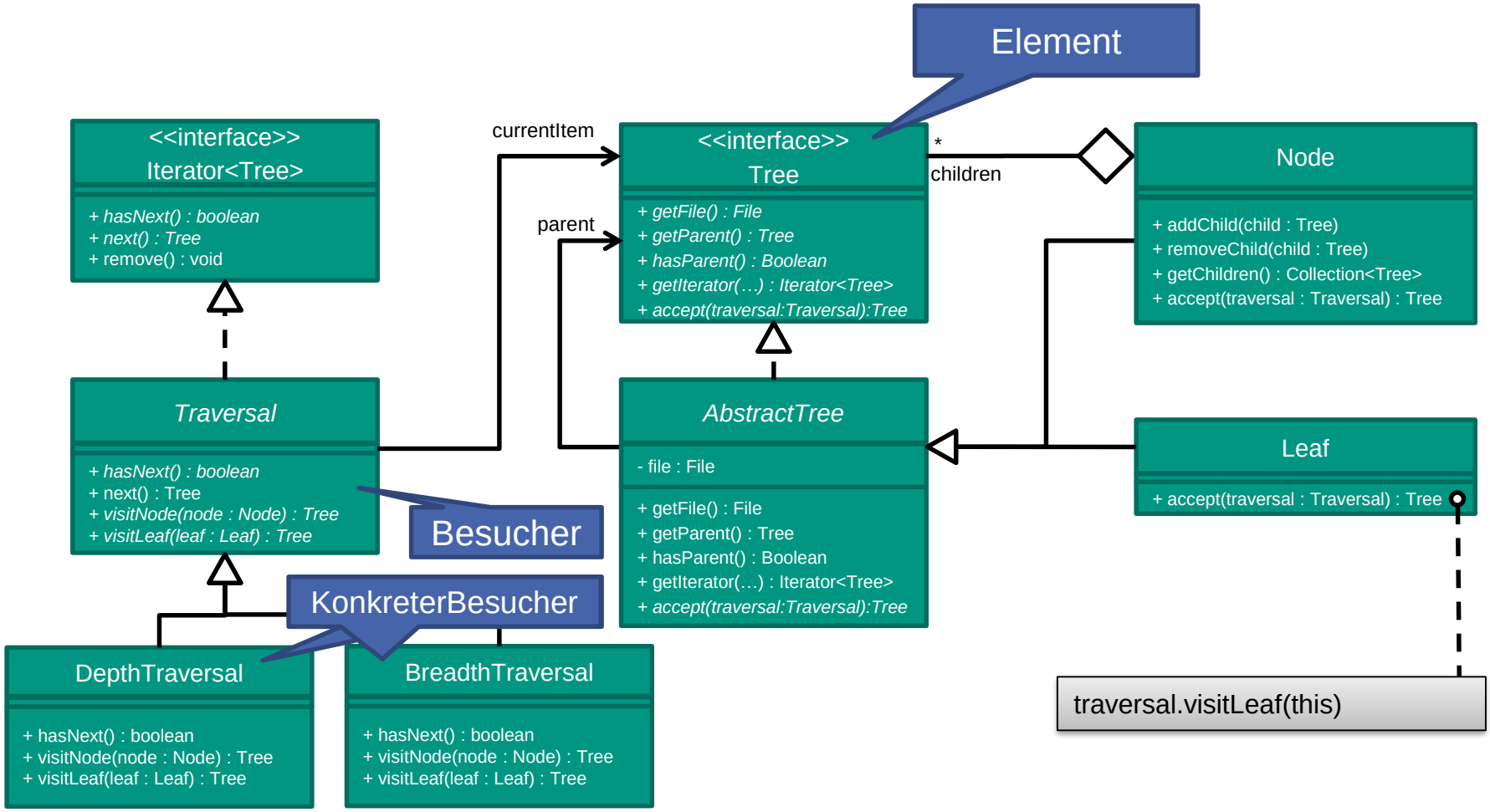
Aufgabe 3: Entwurfsmuster umsetzen



Aufgabe 3: Entwurfsmuster umsetzen



Aufgabe 3: Entwurfsmuster umsetzen



Aufgabe 3: Iterator erstellen

```
public abstract class AbstractTree implements Tree {
// . . .
@Override
  public final Iterator<Tree> getIterator(Class<? extends Traversal> traversal) {
    return TraversalFactory.createTraversal(traversal, this);
  }
}
```

Aufgabe 3: Iterator erstellen

```
public final class TraversalFactory {
  // . . .
  public static Traversal createTraversal(Class<? extends Traversal> traversal,
                                          Tree initialItem) {
    switch (Traversals.fromClass(traversal)) {
    case BREADTH:
      return new BreadthTraversal(initialItem);
    case DEPTH:
    case UNKNOWN:
    default:
      return new DepthTraversal(initialItem);
    }
  }
}
```

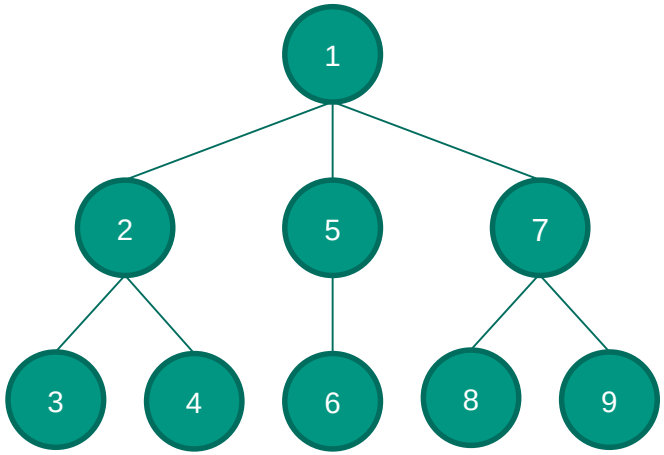
Aufgabe 3b: Hauptreihenfolge

```
public abstract class Traversal implements Iterator<Tree> {
  private Tree currentItem;
  private final Set<Tree> alreadyVisited = new HashSet<>();

  protected Traversal(Tree startItem) {
    this.currentItem = startItem;
  }

  @Override
  public final Tree next() {
    this.currentItem = this.currentItem.accept(this);
    this.alreadyVisited.add(this.currentItem);
    return this.currentItem;
  }

  protected final boolean alreadyVisited(Tree tree) {
    return this.alreadyVisited.contains(tree);
  }
  // ...
}
```

Hier statt in Datenstruktur, wegen mehrerer möglicher Iteratoren gleichzeitig

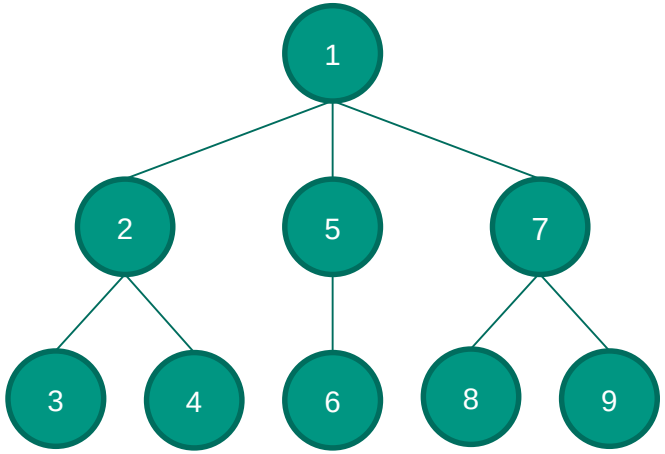
Aufgabe 3b: Hauptreihenfolge

```
public final class DepthTraversal extends Traversal {
  public DepthTraversal(Tree startItem) {
    super(startItem);
  }
  @Override
  public boolean hasNext() {
    return !Objects.isNull(this.getCurrentItem().accept(this));
  }

  @Override
  public Tree visitLeaf(Leaf leaf) {
    if (this.alreadyVisited(leaf)) {
      if (leaf.hasParent()) {
        return leaf.getParent().accept(this);
      }
    } else {
      return leaf;
    }
    return null;
  }
// visitNode nächste Folie
}
```



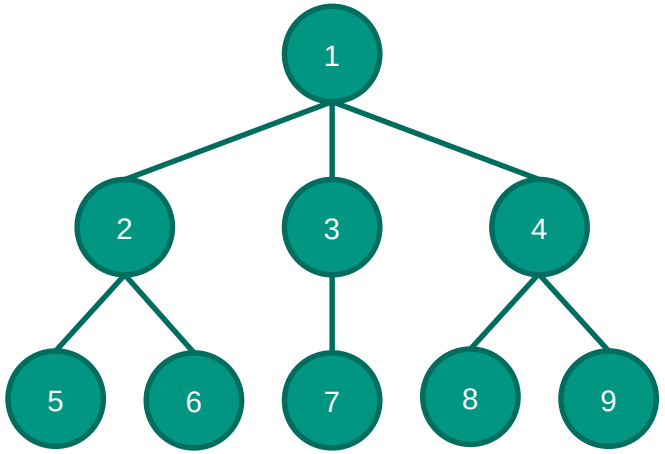
Aufgabe 3b: Hauptreihenfolge

```
public final class DepthTraversal extends Traversal {
  // ...
  @Override
  public Tree visitNode(Node node) {
    if (this.alreadyVisited(node)) {
      for (Tree child : node.getChildren()) {
        if (!this.alreadyVisited(child)) {
          return child.accept(this);
        }
      }
      if (node.hasParent()) {
        return node.getParent().accept(this);
      }
    } else {
      return node;
    }
    return null;
  }

}
```



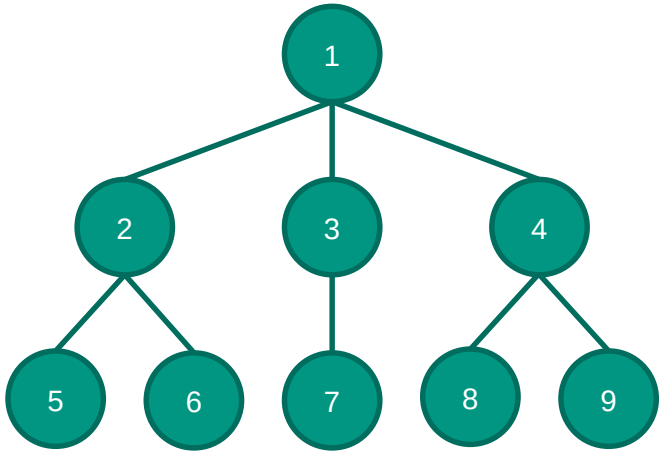
Aufgabe 3c: Ebenenweise

```
public final class BreadthTraversal extends Traversal {
  private final Queue<Tree> toBeVisited = new LinkedList<>();
  public BreadthTraversal(Tree startItem) {
    super(startItem);
  }

  @Override
  public boolean hasNext() {
    return !this.toBeVisited.isEmpty() || (!this.getCurrentItem().hasParent()
           && !this.alreadyVisited(this.getCurrentItem()));
  }

  @Override
  public Tree visitLeaf(Leaf leaf) {
    if (!this.alreadyVisited(leaf)) {
      return leaf;
    } else if (!this.toBeVisited.isEmpty()) {
      return this.toBeVisited.remove().accept(this);
    }
    return null;
  }
}
```



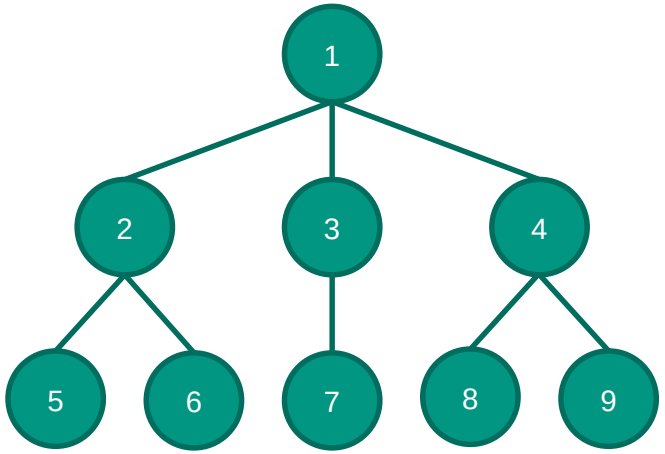
Aufgabe 3c: Ebenenweise

```
public final class BreadthTraversal extends Traversal {
  // ...
  @Override
  public Tree visitNode(Node node) {
    if (!this.alreadyVisited(node)) {
      this.toBeVisited.addAll(node.getChildren());
      return node;
    } else {
      return this.toBeVisited.remove().accept(this);
    }
  }
```



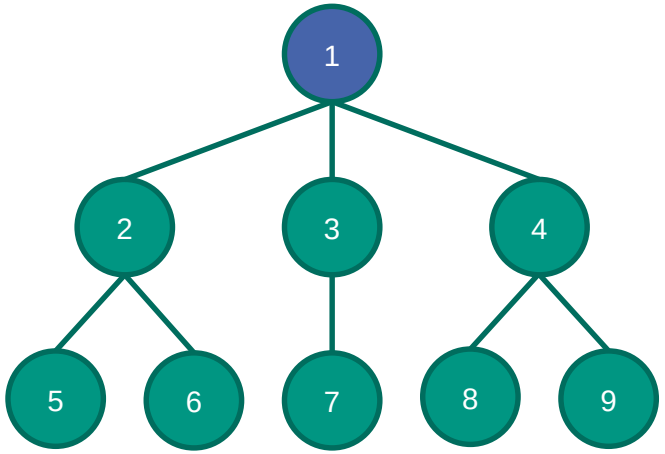
Aufgabe 3c: Ebenenweise

```
public final class BreadthTraversal extends Traversal {
  @Override
  public Tree visitLeaf(Leaf leaf) {
    if (!this.alreadyVisited(leaf)) {
      return leaf;
    } else if (!this.toBeVisited.isEmpty()) {
      return this.toBeVisited.remove().accept(this);
    }
    return null;
  }
  @Override
  public Tree visitNode(Node node) {
    if (!this.alreadyVisited(node)) {
      this.toBeVisited.addAll(node.getChildren());
      return node;
    } else {
      return this.toBeVisited.remove().accept(this);
    }
  }
```



Aufgabe 3c: Ebenenweise

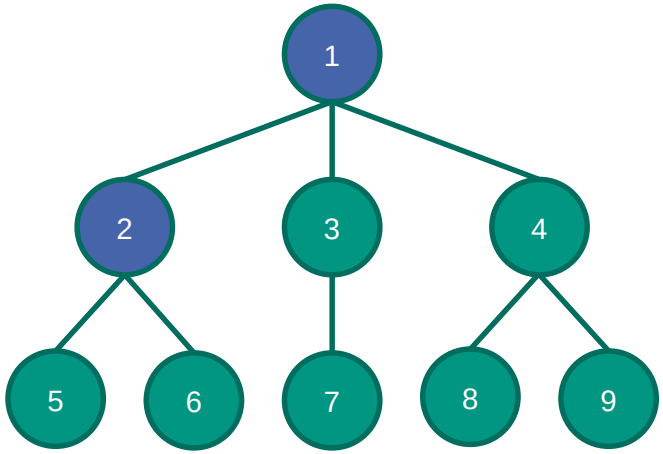
```
public final class BreadthTraversal extends Traversal {
  @Override
  public Tree visitLeaf(Leaf leaf) {
    if (!this.alreadyVisited(leaf)) {
      return leaf;
    } else if (!this.toBeVisited.isEmpty()) {
      return this.toBeVisited.remove().accept(this);
    }
    return null;
  }
  @Override
  public Tree visitNode(Node node) {
    if (!this.alreadyVisited(node)) {
      this.toBeVisited.addAll(node.getChildren());
      return node;
    } else {
      return this.toBeVisited.remove().accept(this);
    }
  }
```



2	3	4					

Aufgabe 3c: Ebenenweise

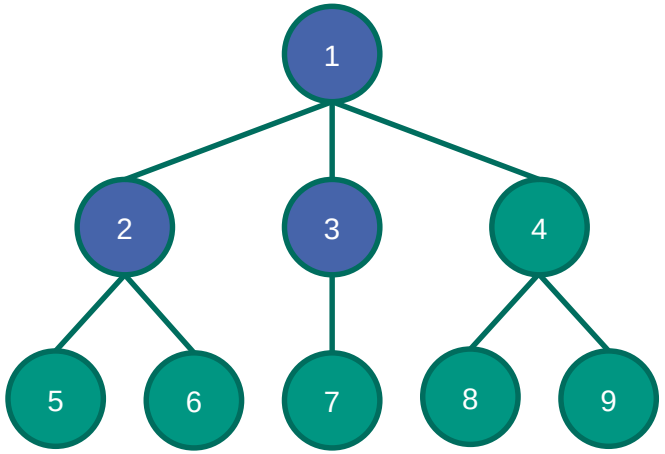
```
public final class BreadthTraversal extends Traversal {
  @Override
  public Tree visitLeaf(Leaf leaf) {
    if (!this.alreadyVisited(leaf)) {
      return leaf;
    } else if (!this.toBeVisited.isEmpty()) {
      return this.toBeVisited.remove().accept(this);
    }
    return null;
  }
  @Override
  public Tree visitNode(Node node) {
    if (!this.alreadyVisited(node)) {
      this.toBeVisited.addAll(node.getChildren());
      return node;
    } else {
      return this.toBeVisited.remove().accept(this);
    }
  }
```



3	4	5	6				

Aufgabe 3c: Ebenenweise

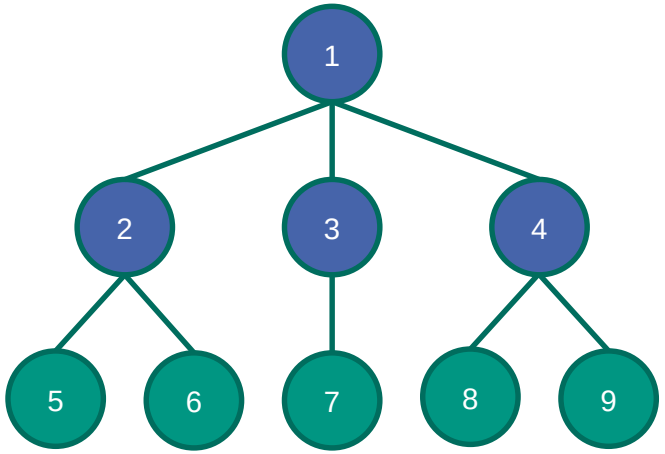
```
public final class BreadthTraversal extends Traversal {
  @Override
  public Tree visitLeaf(Leaf leaf) {
    if (!this.alreadyVisited(leaf)) {
      return leaf;
    } else if (!this.toBeVisited.isEmpty()) {
      return this.toBeVisited.remove().accept(this);
    }
    return null;
  }
  @Override
  public Tree visitNode(Node node) {
    if (!this.alreadyVisited(node)) {
      this.toBeVisited.addAll(node.getChildren());
      return node;
    } else {
      return this.toBeVisited.remove().accept(this);
    }
  }
```



4	5	6	7				

Aufgabe 3c: Ebenenweise

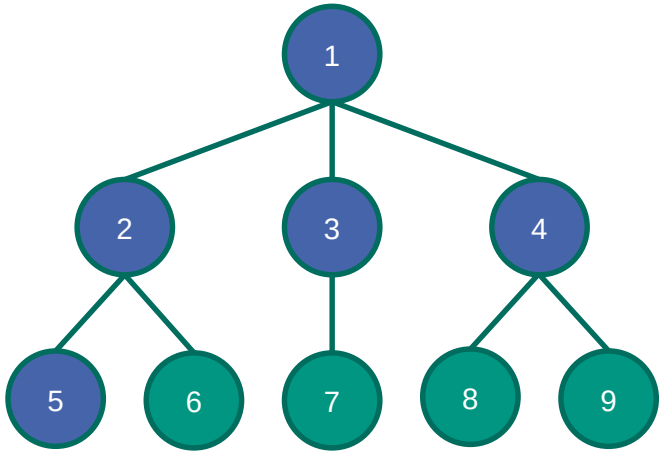
```
public final class BreadthTraversal extends Traversal {
  @Override
  public Tree visitLeaf(Leaf leaf) {
    if (!this.alreadyVisited(leaf)) {
      return leaf;
    } else if (!this.toBeVisited.isEmpty()) {
      return this.toBeVisited.remove().accept(this);
    }
    return null;
  }
  @Override
  public Tree visitNode(Node node) {
    if (!this.alreadyVisited(node)) {
      this.toBeVisited.addAll(node.getChildren());
      return node;
    } else {
      return this.toBeVisited.remove().accept(this);
    }
  }
```



5	6	7	8	9			

Aufgabe 3c: Ebenenweise

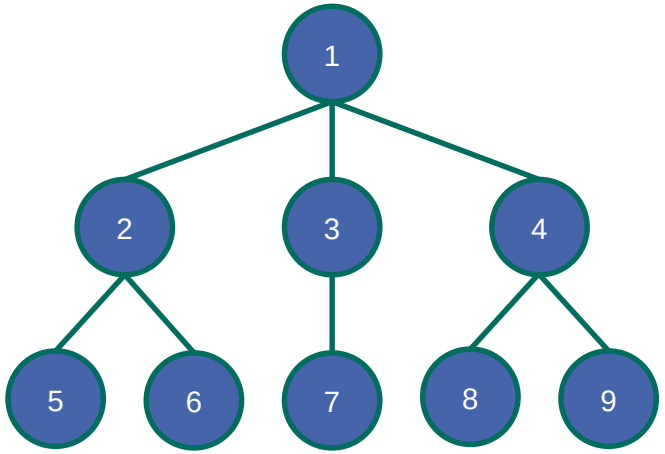
```
public final class BreadthTraversal extends Traversal {
  @Override
  public Tree visitLeaf(Leaf leaf) {
    if (!this.alreadyVisited(leaf)) {
      return leaf;
    } else if (!this.toBeVisited.isEmpty()) {
      return this.toBeVisited.remove().accept(this);
    }
    return null;
  }
  @Override
  public Tree visitNode(Node node) {
    if (!this.alreadyVisited(node)) {
      this.toBeVisited.addAll(node.getChildren());
      return node;
    } else {
      return this.toBeVisited.remove().accept(this);
    }
  }
```



6	7	8	9				

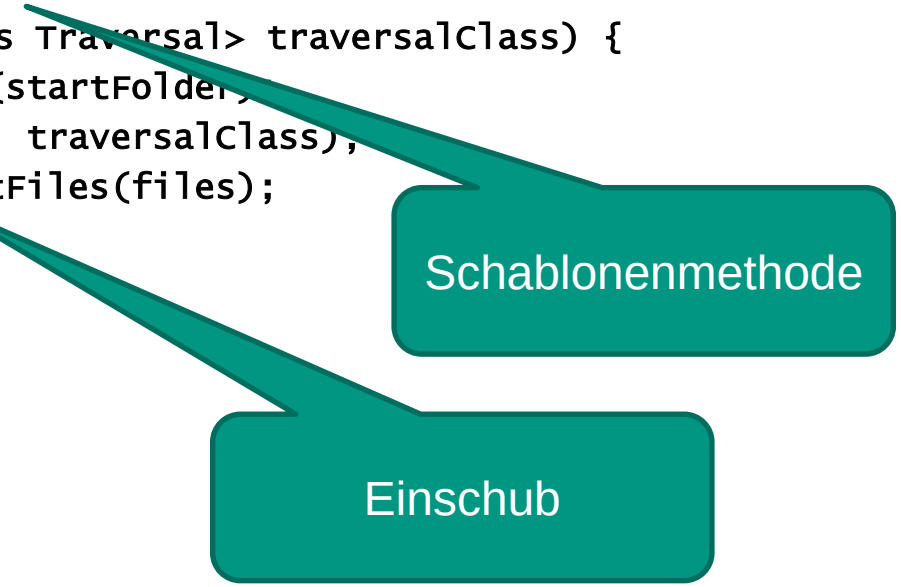
Aufgabe 3c: Ebenenweise (fast-forward)

```
public final class BreadthTraversal extends Traversal {
  @Override
  public Tree visitLeaf(Leaf leaf) {
    if (!this.alreadyVisited(leaf)) {
      return leaf;
    } else if (!this.toBeVisited.isEmpty()) {
      return this.toBeVisited.remove().accept(this);
    }
    return null;
  }
  @Override
  public Tree visitNode(Node node) {
    if (!this.alreadyVisited(node)) {
      this.toBeVisited.addAll(node.getChildren());
      return node;
    } else {
      return this.toBeVisited.remove().accept(this);
    }
  }
```



Aufgabe 3e: Runner

```
public abstract class Runner {
  public final void run(File startFolder,
                        Class<? extends Traversal> traversalClass) {
    Tree tree = this.buildFolderStructure(startFolder);
    List<File> files = this.getFiles(tree, traversalClass);
    List<File> selectedFiles = this.selectFiles(files);
    this.printResults(selectedFiles);
  }
  // nächste Folie
}
```



Aufgabe 3e: Runner

```
public abstract class Runner {
  // ...
  private Tree buildFolderStructure(File startFolder) {
    if (!startFolder.isDirectory()) {
      throw new IllegalArgumentException("start folder needs to be a folder");
    }
    return this.buildFolderStructure(startFolder, null);
  }
  private Tree buildFolderStructure(File current, Tree parent) {
    if (current.isFile()) {
      return new Leaf(current, parent);
    }
    Node folder = new Node(current, parent);

    List<Tree> children = new ArrayList<>();
    for (File child : current.listFiles()) {
      children.add(this.buildFolderStructure(child, folder));
    }
    children.forEach(folder::addChild);
    return folder;
  }
// ...
```

Aufgabe 3e: Runner

```
public abstract class Runner {
  private List<File> getFiles(Tree tree, Class<? extends Traversal> traversalClass) {
    List<File> res = new ArrayList<>();
    tree.getIterator(traversalClass).forEachRemaining(t -> res.add(t.getFile()));
    return res;
  }

  protected abstract List<File> selectFiles(List<File> files);

  private void printResults(List<File> selectedFiles) {
    selectedFiles.forEach(System.out::println);
  }
```

Aufgabe 3e: Runner

```
public final class JPGRunner extends Runner {

  @Override
  protected List<File> selectFiles(List<File> files) {
    List<File> res = new ArrayList<>(files);
    res.removeIf(f -> !f.isFile() || !f.getName().toLowerCase().endsWith(".jpg"));
    return res;
  }
}
```

```
public final class PNGRunner extends Runner {

  @Override
  protected List<File> selectFiles(List<File> files) {
    List<File> res = new ArrayList<>(files);
    res.removeIf(f -> !f.isFile() || !f.getName().toLowerCase().endsWith(".png"));
    return res;
  }

}
```

Aufgabe 3f: Kommandozeilenschnittstelle

```
public static void main(String[] args) {
  CommandLine cmd = App.doCommandLineParsing(args); // try-catch
  boolean jpg = cmd.hasOption(App.JPG_OPT);
  boolean png = cmd.hasOption(App.PNG_OPT);
  boolean bfs = cmd.hasOption(App.BFS_OPT);
  File dir = new File(cmd.getOptionValue(App.DIRECTORY_OPT));

  if (!dir.exists() || !dir.isDirectory()) {
    System.err.println("Illegal start directory");
    System.exit(1);
  }
  if (jpg == png) {
    System.err.println("No/Two Runner(s) has/have been set.");
    System.exit(2);
  }
  Traversals traversals = bfs ? Traversals.BREADTH : Traversals.DEPTH;
  if (jpg) {
    new JPGRunner().run(dir, traversals.getTraversal());
  }
  if (png) {
    new PNGRunner().run(dir, traversals.getTraversal());
  }
}
```

Lesen der Kommandozeilen-Parameter

Überprüfung auf validen Startordner

Überprüfung ob keiner bzw. zwei Runner gewählt (optional)

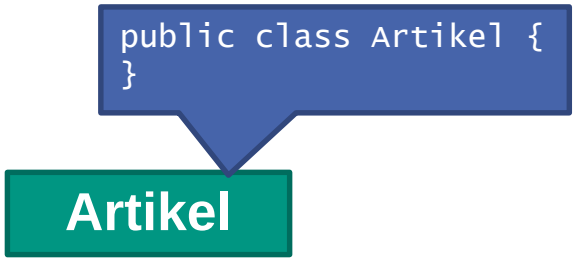
Wähle Traversal

Führe gewählten Runner aus

Implementierung umgekehrt

AUFGABE 4

Aufgabe 4: Entwurfsmuster angewandt

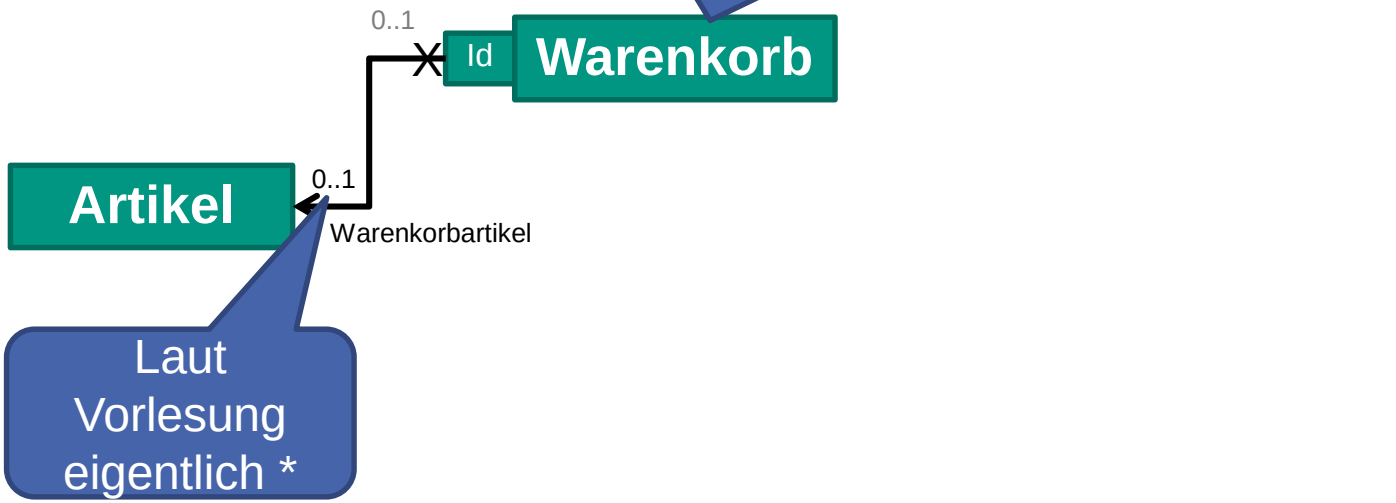


Aufgabe 4: Entwurfsmuster angewandt

Artikel

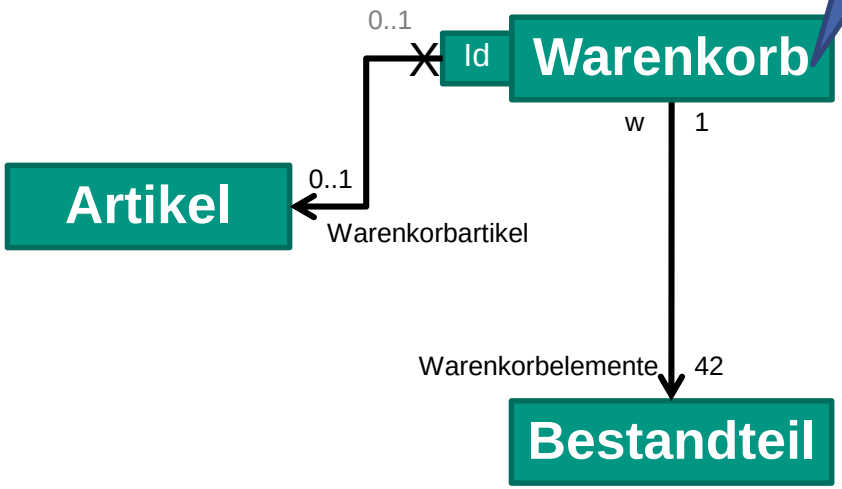
Aufgabe 4: Entwurfsmuster angewandt

```
public class Warenkorb {
  Map<String, Artikel> warenkorbArtikel;
  // ...

  public Warenkorb() {
    warenkorbArtikel = new HashMap<>();
    // ...
  }
}
```



Aufgabe 4: Entwurfsmuster angewandt

```
public class Warenkorb {
  Bestandteil[] warenkorbelemente;
  // ...

  public Warenkorb() {
    warenkorbelemente = new Bestandteil[42];
    for (Bestandteil b : warenkorbelemente) {
      b = new Bestandteil(this);
    }
    // ...
  }
}
```



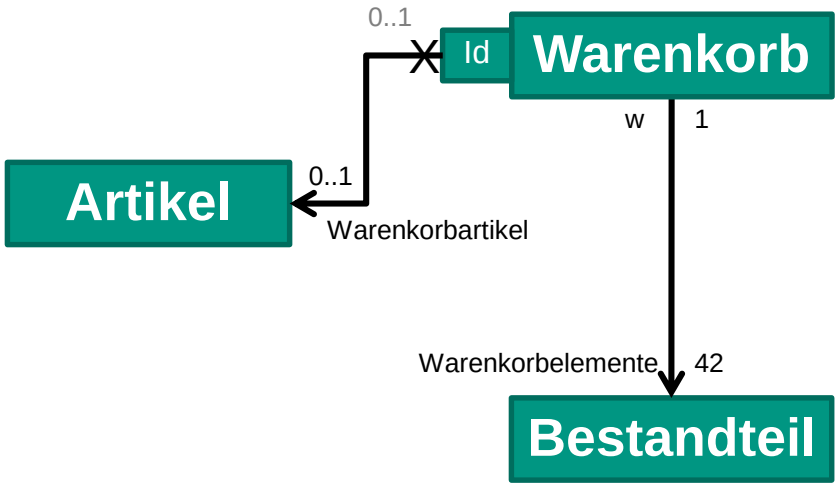
Aufgabe 4: Entwurfsmuster angewandt

```
public class Bestellvorgang {
  // ...
}
```

Bestellvorgang

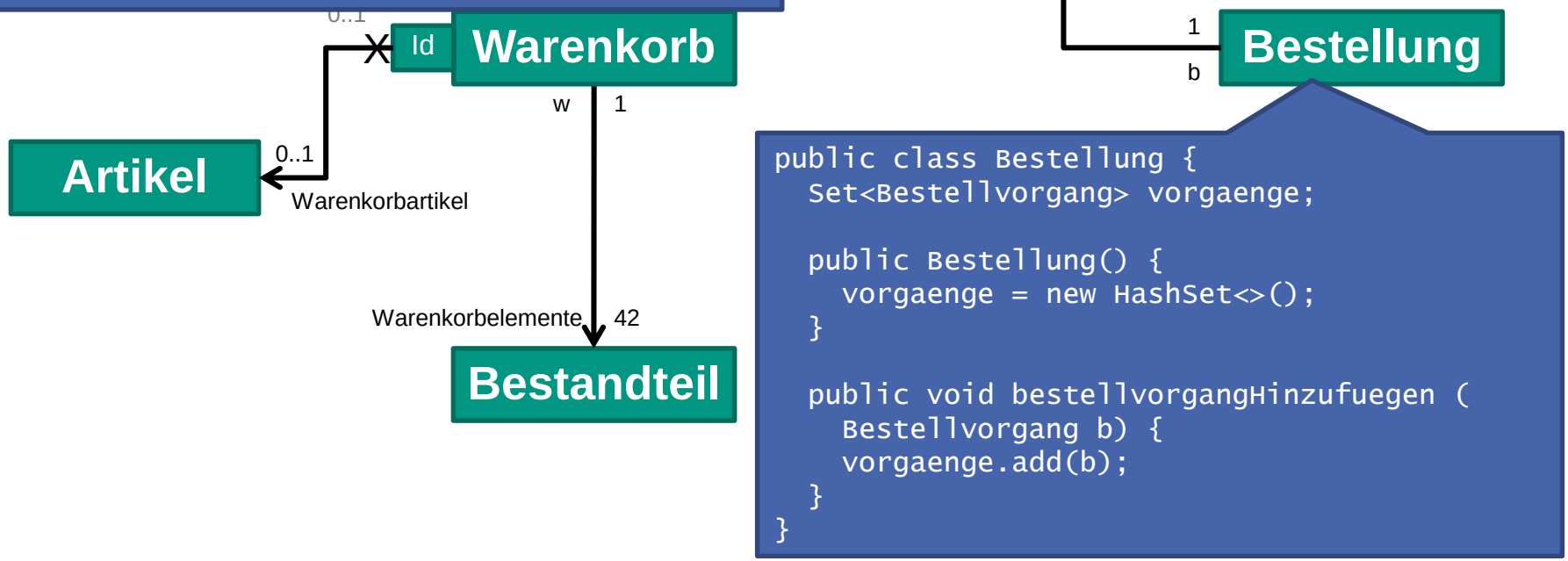
```
public class Bestellung {
  // ...
}
```

Bestellung



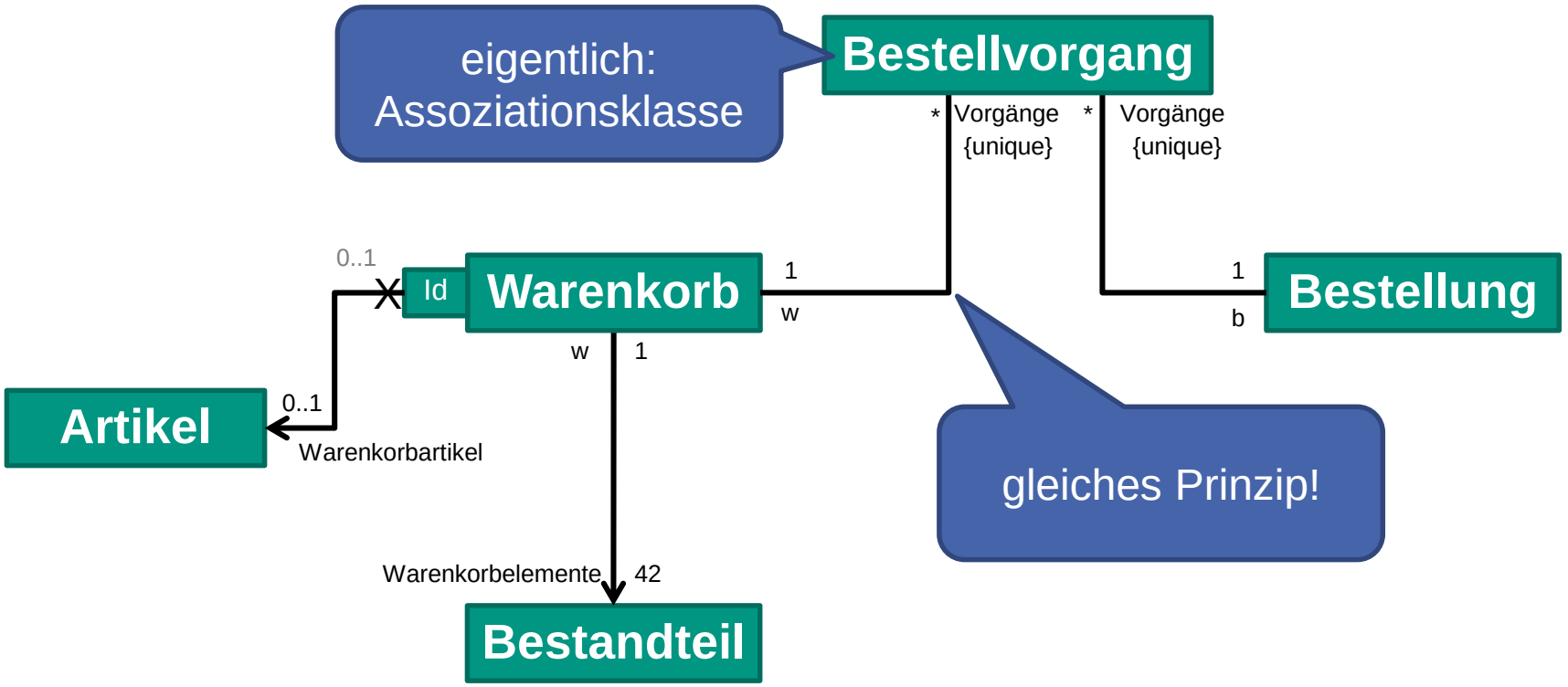
Aufgabe 4: Entwurfsmuster angewandt

```
public class Bestellvorgang {
  // ...
  Bestellung b;

  public Bestellvorgang(Warenkorb w,
    Bestellung b) {
    this.b = b;
    b.bestellvorgangHinzufuegen(this);
    // ...
  }
}
```

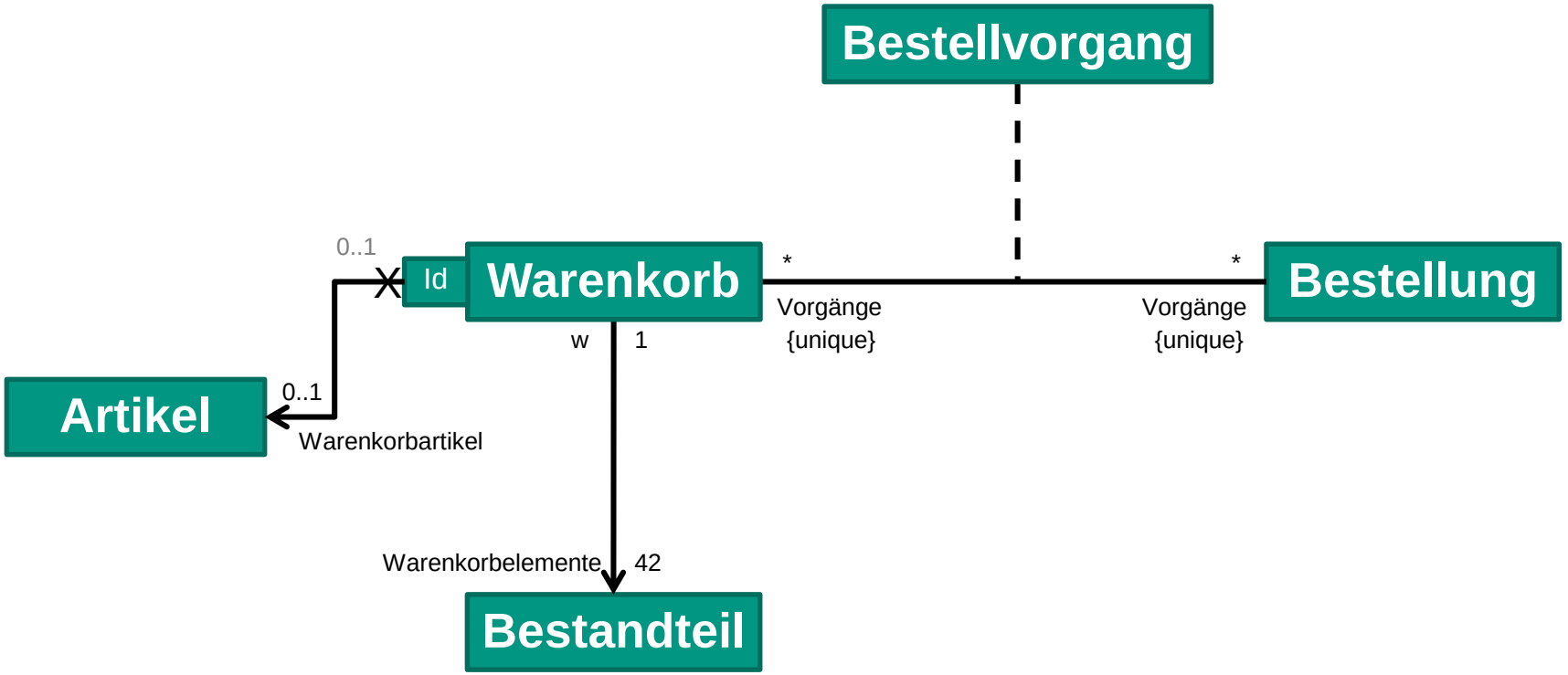



```
public class Bestellung {
  Set<Bestellvorgang> vorgaenge;

  public Bestellung() {
    vorgaenge = new HashSet<>();
  }

  public void bestellvorgangHinzufuegen (
    Bestellvorgang b) {
    vorgaenge.add(b);
  }
}
```

Aufgabe 4: Entwurfsmuster angewandt



Aufgabe 4: Entwurfsmuster angewandt

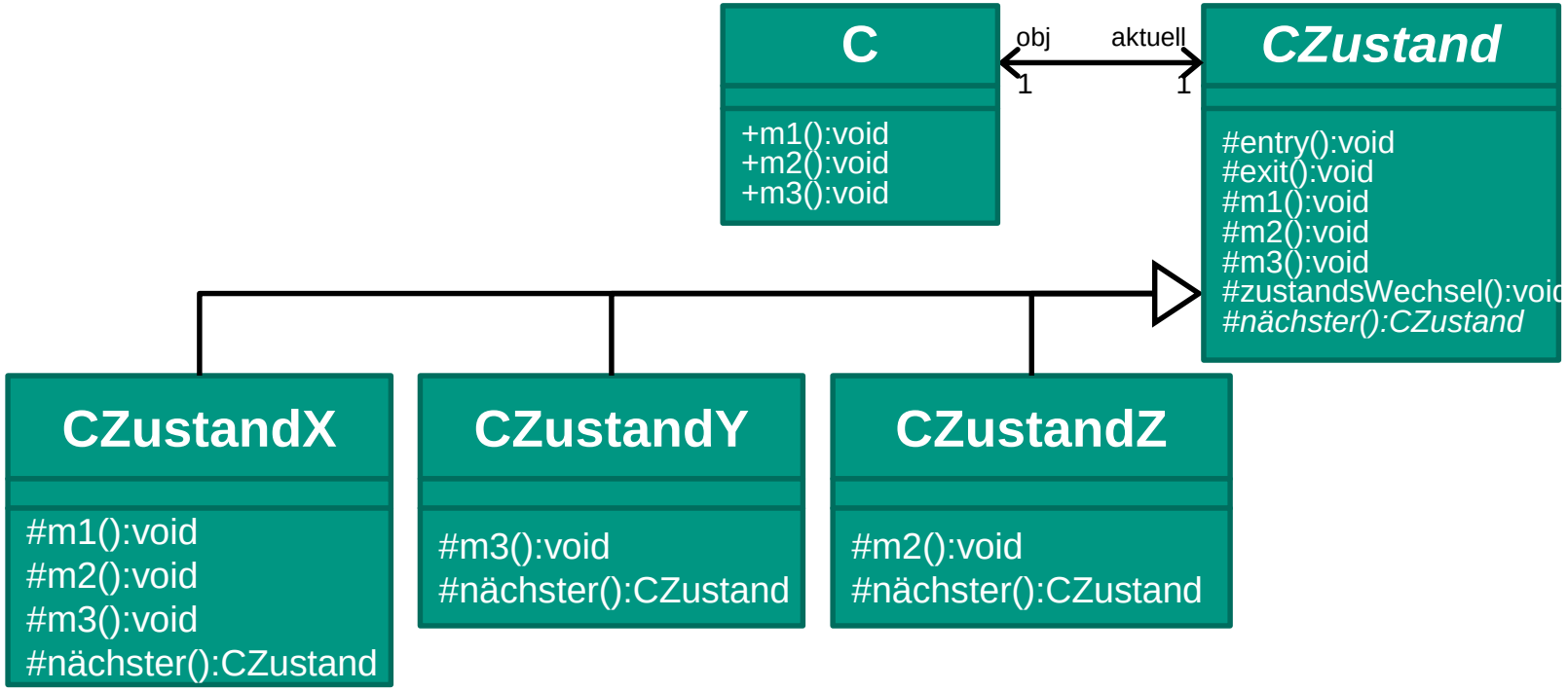


Zustandsdiagramm umsetzen

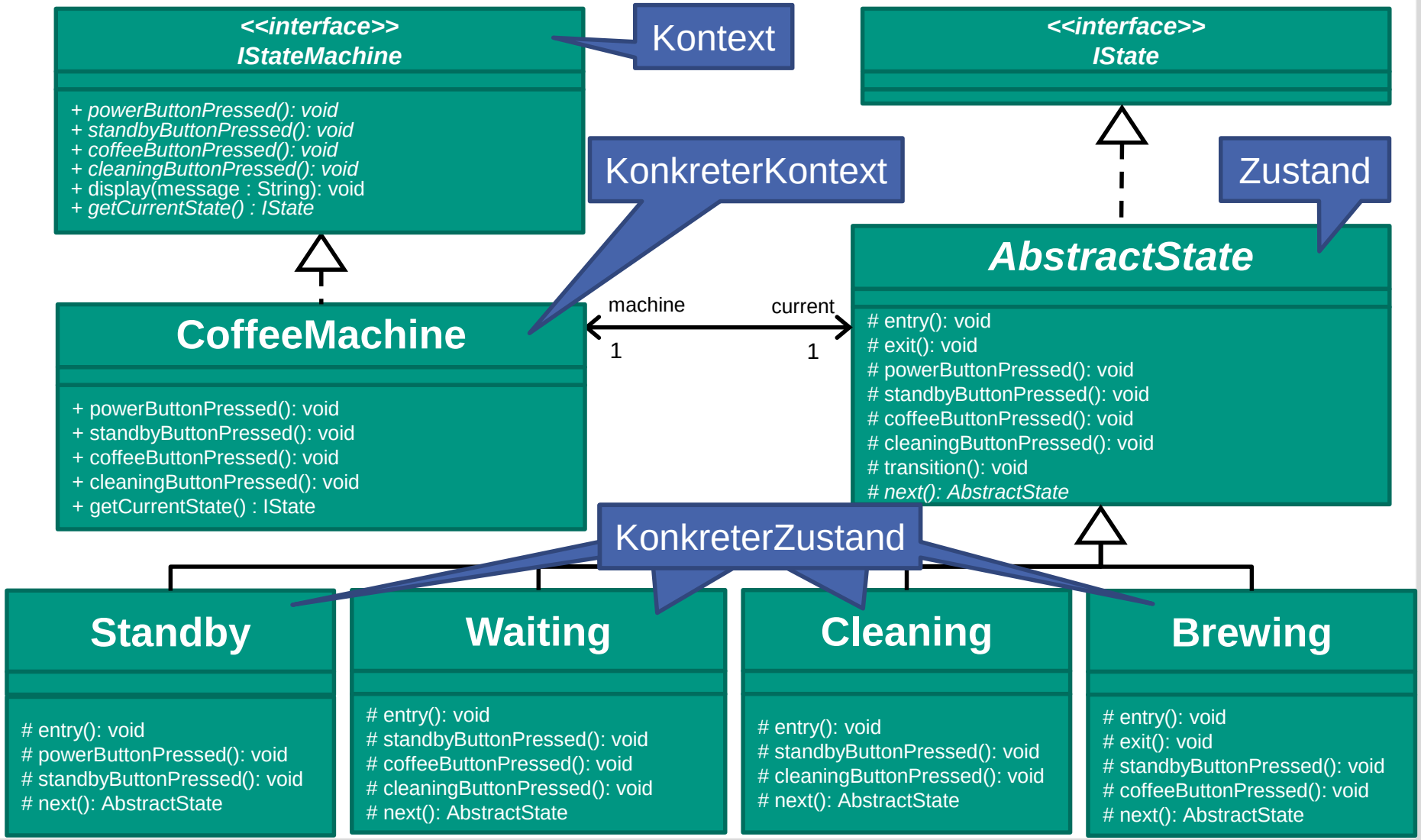
AUFGABE 5

Aufgabe 5: Implementierung eines Zustandsautomaten

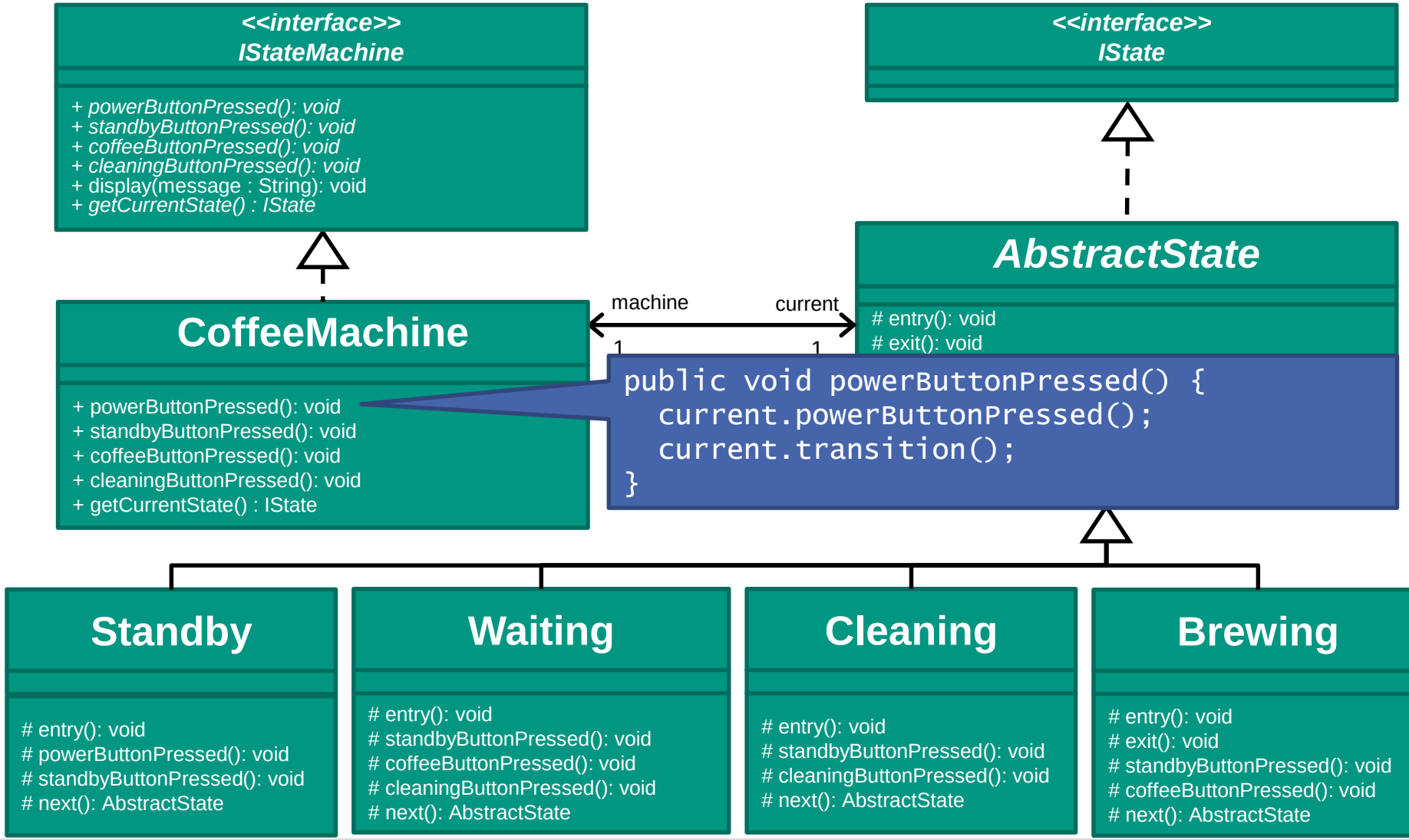
- Das Zustandsmuster wird verwendet, wenn das Verhalten des Objektes von dessen Zustand abhängt und das Objekt sein Verhalten während der Laufzeit, abhängig vom aktuellen Zustand, ändern muss.
- Erweiterbarkeit und leichte Anpassung für andere Maschinen durch:
  - **Ausgelagerte explizite Zustandshaltung**



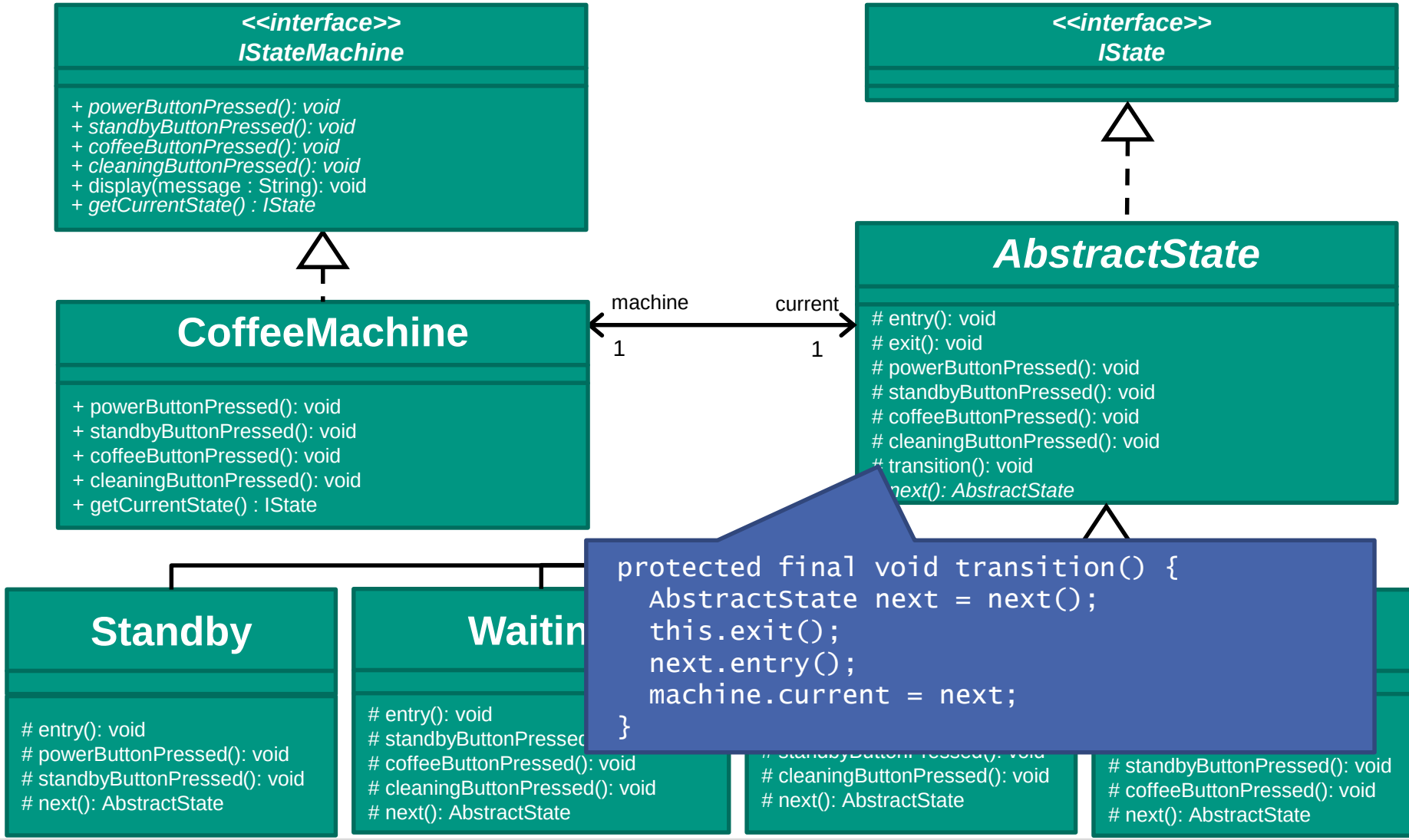
Aufgabe 5: Entwurfsmuster angewandt



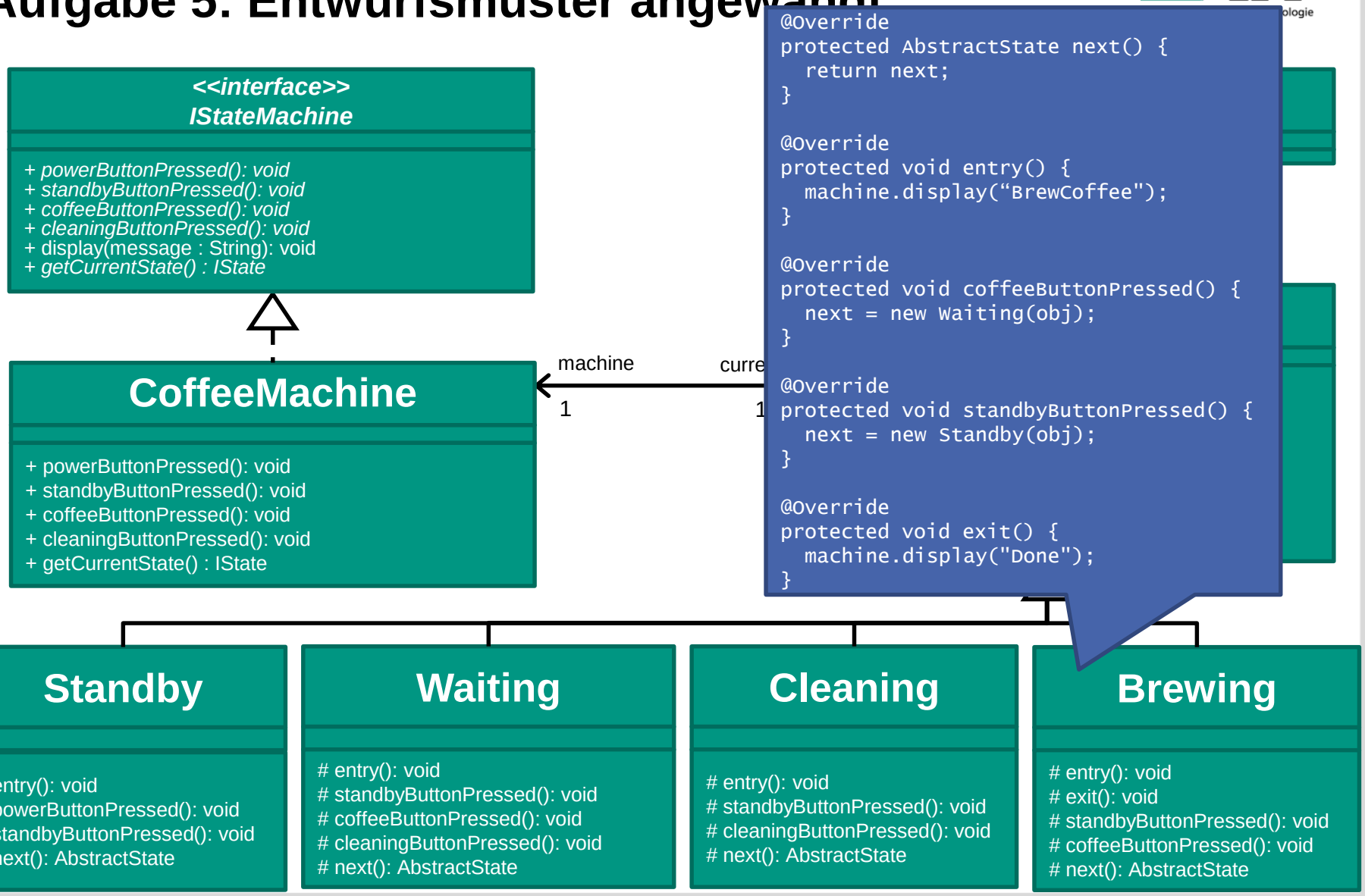
Aufgabe 5: Entwurfsmuster angewandt



Aufgabe 5: Entwurfsmuster angewandt



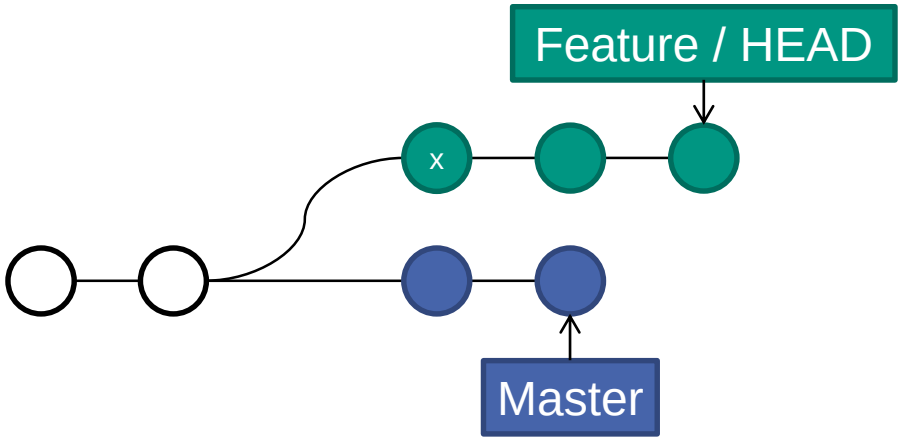
Aufgabe 5: Entwurfsmuster angewandt



Git

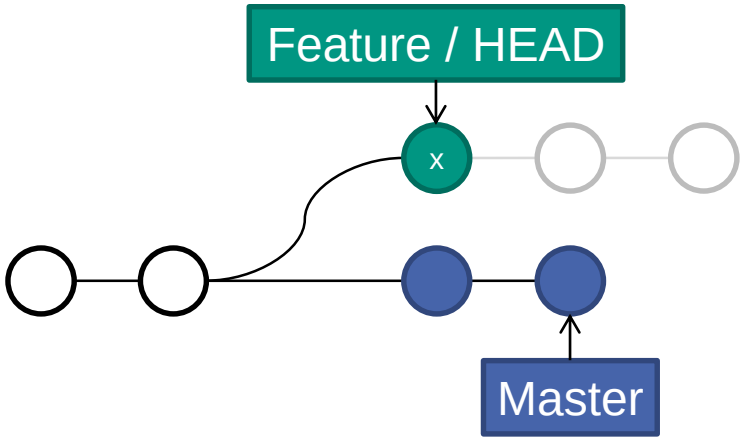
AUFGABE 6

Aufgabe 6 a): Git



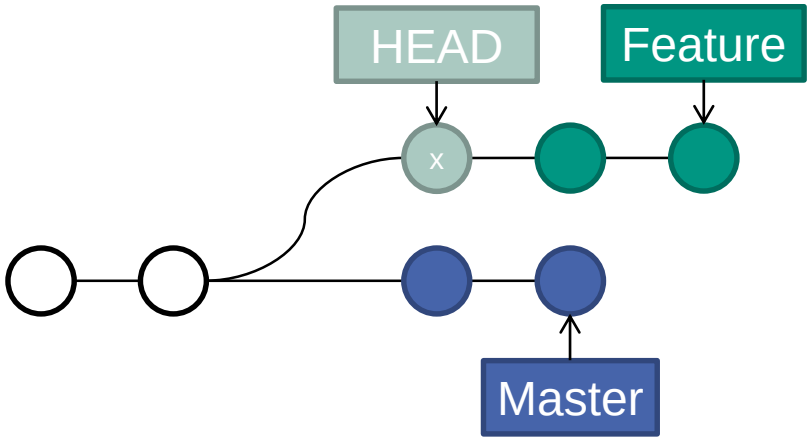
- `git reset HEAD~2`
- `git checkout HEAD~2`
- `git revert HEAD~2..`

Aufgabe 6 a): Git



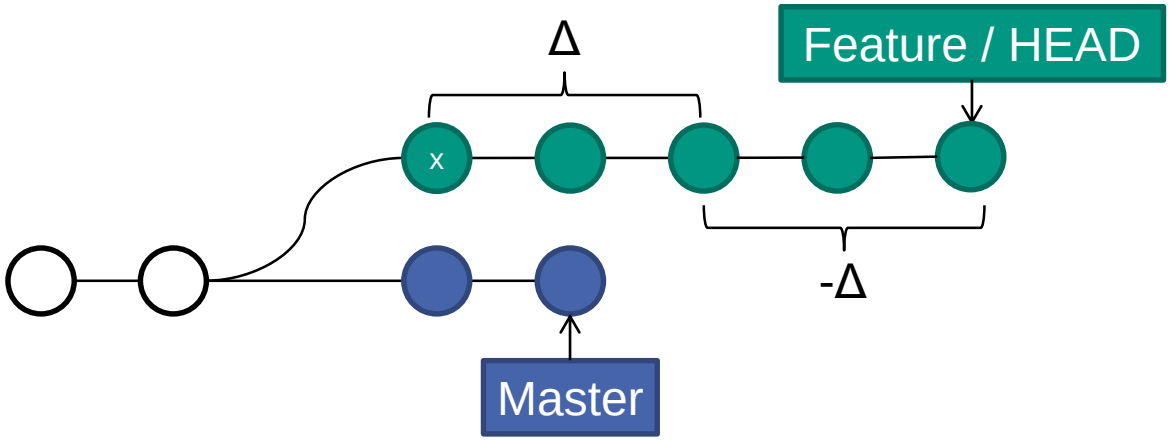
- `git reset HEAD~2`
  - Setzt Spitze des Zweigs auf angegebene Einbuchung
  - Rückgängig gemachte Einbuchungen werden bereinigt

Aufgabe 6 a): Git



- `git checkout HEAD~2`
  - Setzt nur den Kopf-Zeiger auf eine bestimmte Einbuchung
  - Weitere Einbuchungen bleiben erhalten

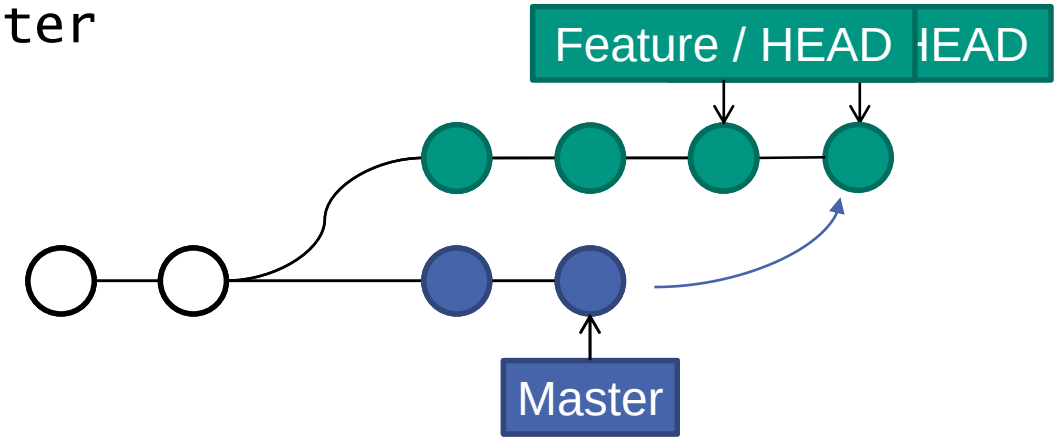
Aufgabe 6 a): Git



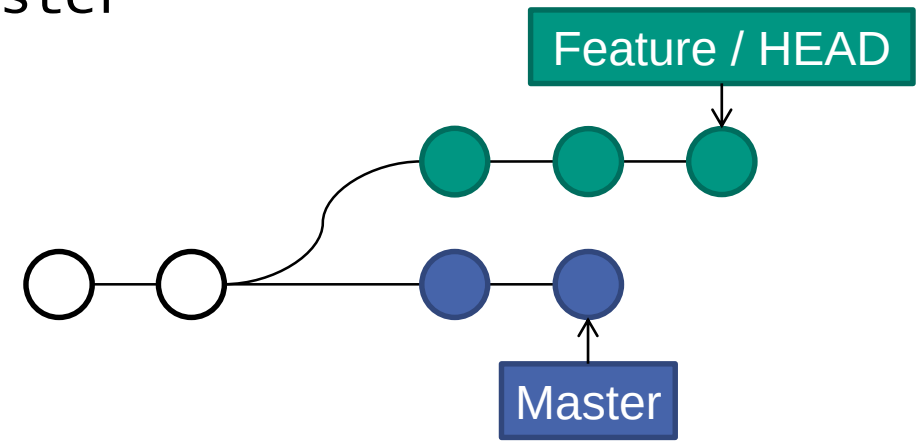
- `git revert HEAD~2..`
  - Macht Einbuchungen rückgängig indem neue Einbuchung mit dem Delta erzeugt wird
  - Historie bleibt erhalten

Aufgabe 6 b): Git

```
git merge master
```

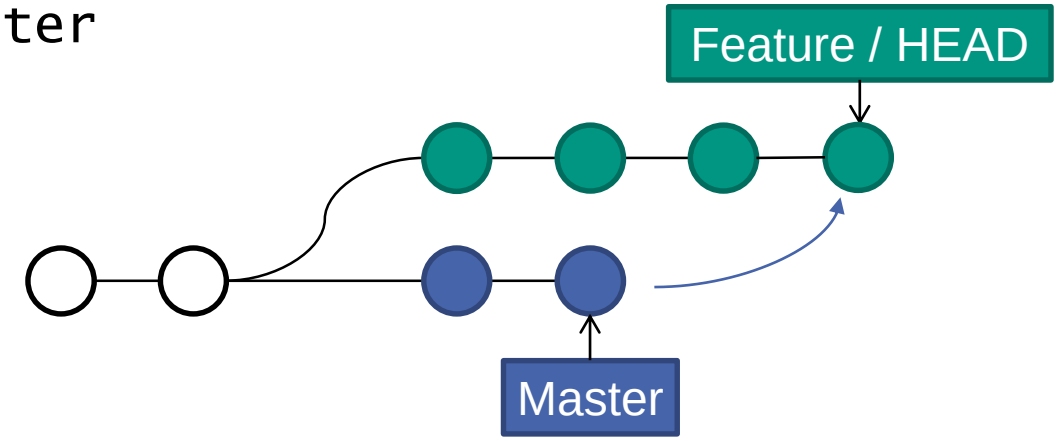



```
git rebase master
```

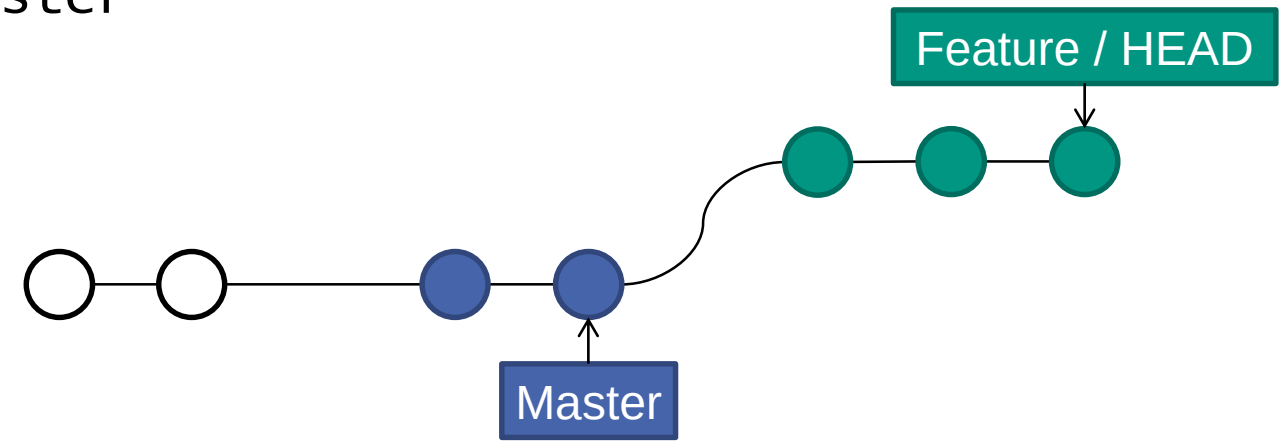


Aufgabe 6 b): Git

```
git merge master
```




```
git rebase master
```



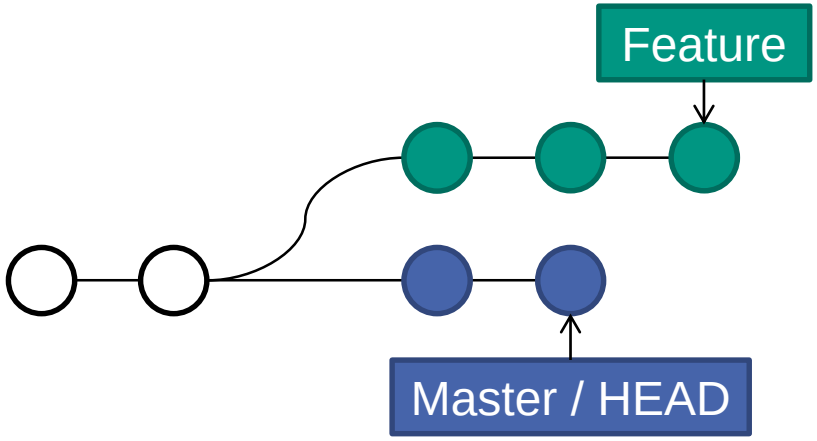
Aufgabe 6 c): Git

- `git merge master`
  - Pro: verändert Historie nicht
  - Con: erzeugt “unnötige” Verschmelzeinbuchungen, nichtlinearer Verlauf

- `git rebase master`
  - Pro: übersichtlichere Historie, keine Verschmelzeinbuchungen
  - Con: Kontext geht verloren (Nachvollziehbarkeit und Sicherheit)
    - rebase sollte nur auf nicht öffentliche Zweige angewendet werden

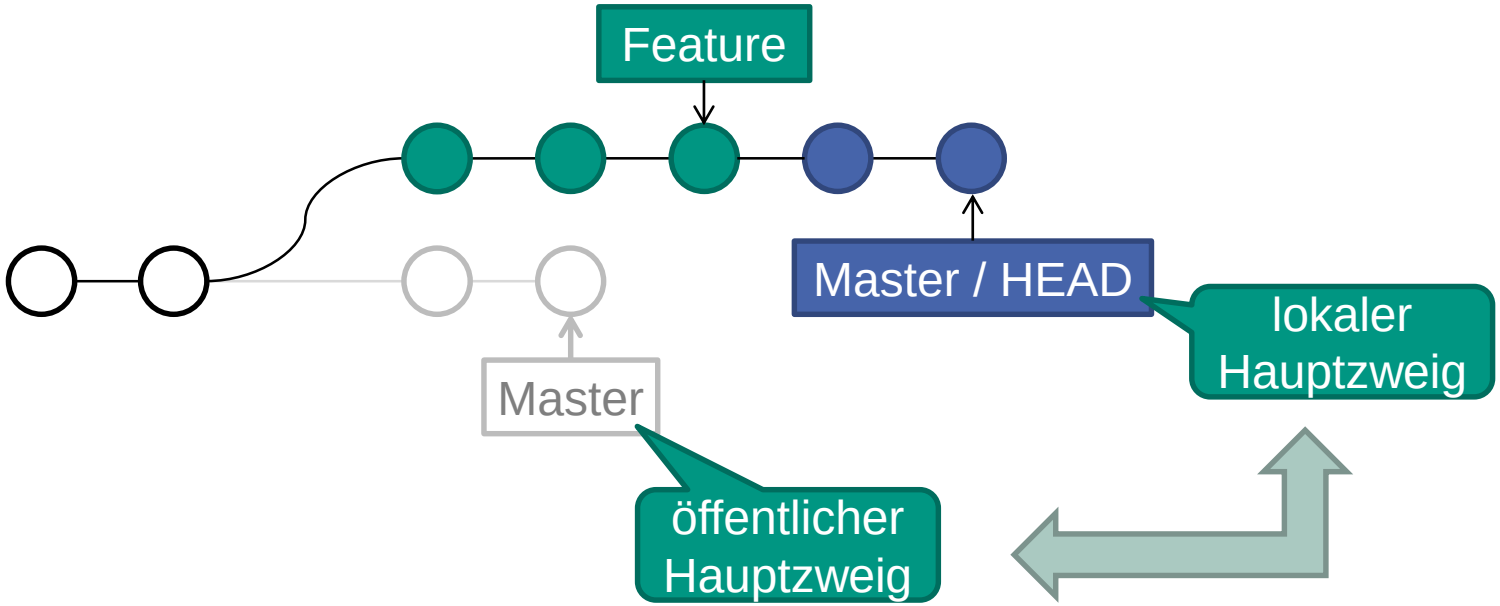
Aufgabe 6 d): Git

- `git rebase feature`



Aufgabe 6 d): Git

- `git rebase feature`



Für Git kein Zusammenhang mehr erkennbar