8.1 Adapter-Muster

Für Ihr Kundenmanagement-System benötigen Sie eine Software, die Adressen auf Gültigkeit überprüft. Schreiben Sie eine Schnittstelle **`AdressValidator`**, wie im UML-Diagramm vorgegeben.

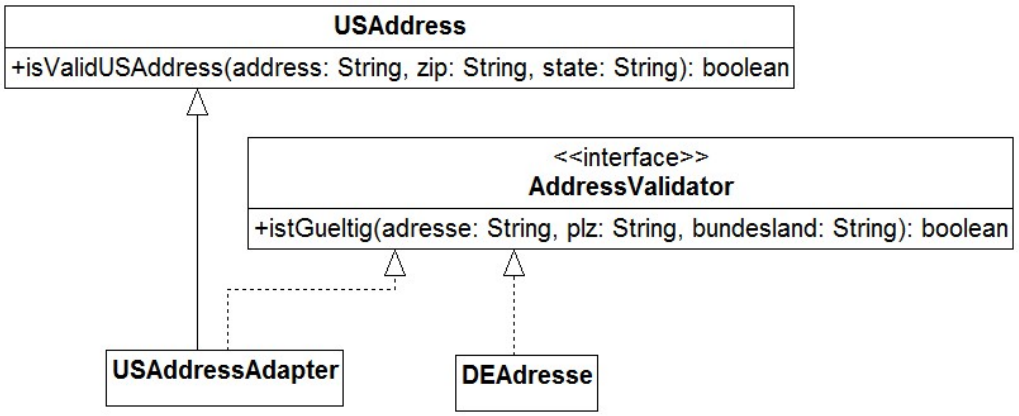


Abbildung 1: Klassendiagramm der Komponente für das Kundenmanagement-System

Die Klasse **`DEAdresse`** implementiert **`AdressValidator`**. Überlegen Sie sich eine sinnvolle Umsetzung der Methode `istGueltig`, abhängig von der Länge der eingegebenen Strings (z.B. Postleitzahl soll fünfstellig sein). Das System soll in der Lage sein Daten von Kunden aus den USA zu verwalten. Stellen Sie sich vor, dass Sie eine externe Bibliothek dafür zur Verfügung haben. Die externe Bibliothek bietet eine Klasse **`USAddress`** als Validator.

```
public class USAddress {

    public boolean isValidUSAddress(String address, String zip,
            String state) {
        if (address.length() < 10)
            return false;
        if (zip.length() < 5)
            return false;
        if (zip.length() > 10)
            return false;
        if (state.length() != 2)
            return false;
        return true;
    }
}
```

Sie wollen die Funktionalität der Klasse verwenden, ohne dabei die Klasse zu modifizieren. Schreiben Sie eine Adapter-Klasse, die **`AdressValidator`** implementiert und die Methode `isValidUSAddress` von **`USAddress`** aufruft. Testen Sie Ihr System. Überprüfen Sie deutsche und amerikanische Adressen auf Gültigkeit. Verwenden Sie dabei `USAddressAdapter` und nicht USAddress.

Lösung:

AddressValidator.java

```
package adapter;

public interface AddressValidator {

    public boolean istGueltig(String adresse, String plz, String
    ↪ bundesland);

}
```

DEAdresse.java

```
package adapter;

public class DEAdresse implements AddressValidator {

    @Override
    public boolean istGueltig(String address, String zip, String
    ↪ bundesland) {
        if (address.length() < 10)
            return false;
        if (zip.length() != 5)
            return false;
        if (bundesland.length() != 2)
            return false;
        return true;

    }

}
```

USAddress.java

```
package adapter;

public class USAddress {

    public boolean isValidUSAddress(String address, String zip, String
    ↪ state) {
        if (address.length() < 10)
            return false;
        if (zip.length() < 5)
            return false;
        if (zip.length() > 10)
```

```
            return false;
        if (state.length() != 2)
            return false;
        return true;
    }

}
```

USAddressAdapter.java

```
package adapter;

public class USAddressAdapter extends USAddress implements
↪  AddressValidator {

    public boolean istGueltig(String addresse, String plz, String
    ↪  bundesland) {
        return isValidUSAddress(addresse, plz, bundesland);
    }

}
```

Test.java

```
package adapter;

public class Test {

    public static void main(String[] args) {
        AddressValidator validator = new DEAdresse();

        boolean valid = false;
        valid = validator.istGueltig("Universitätsstr. 150", "44801",
        ↪  "NRW");
        System.out.println(valid);
        valid = validator.istGueltig("Universitätsstr. 150", "44801",
        ↪  "NW");
        System.out.println(valid);

        validator = new USAddressAdapter();
        valid = validator.istGueltig("1600 Pennsylvania Ave NW,
        ↪  Washington", "20500", "DC");
        System.out.println(valid);
    }

}
```

8.2 Facade-Muster

Als Beispiel für das Facade-Muster soll ein Compiler implementiert werden. Die Klassen, die Sie dafür brauchen, sind im folgenden UML-Klassendiagramm dargestellt.

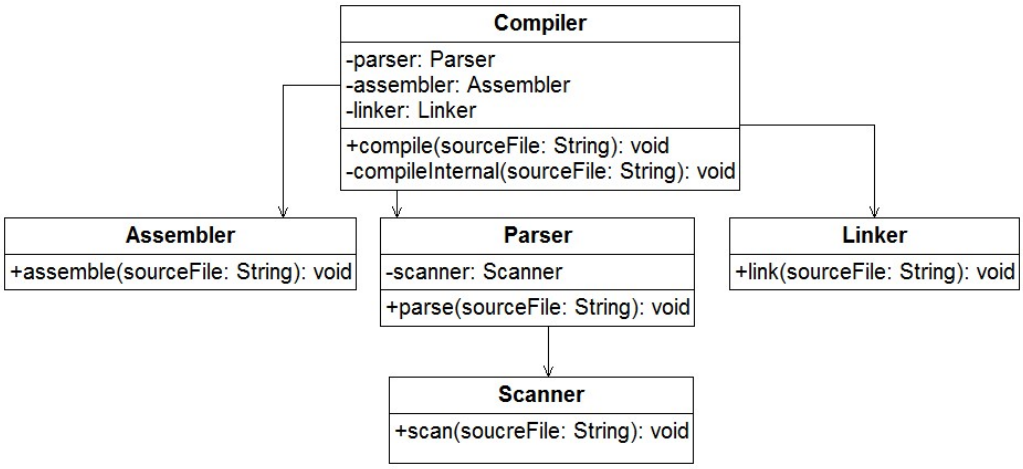


Abbildung 2: Klassendiagramm der Komponente für den Compiler

Der Vorgang der Kompilierung ist wie folgt definiert:

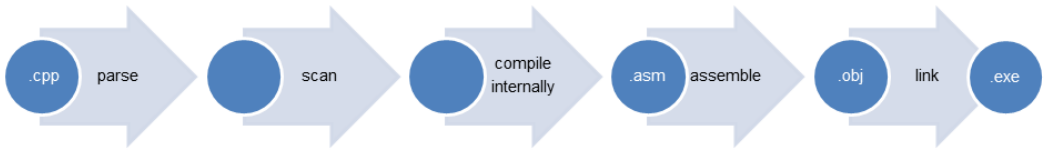


Abbildung 3: Kompilierungsvorgang

Die Methoden `parse`, `scan`, `compileInternal`, `assemble` und `link` sollen als Textausgabe implementiert werden.

Zeichnen Sie ein Sequenzdiagramm, um das Verhalten Ihres Programms graphisch darzustellen.

Lösung:

Scanner.java

```
package facade;

public class Scanner {

    public void scan(String sourceFile){
        System.out.println("Scanning " + sourceFile);
    }
```

```
}
```

Parser.java

```
package facade;

public class Parser {

    private Scanner scanner;

    public void parse(String sourceFile) {
        scanner = new Scanner();
        scanner.scan(sourceFile);
        System.out.println("Parsing " + sourceFile);
    }

}
```

Assembler.java

```
package facade;

public class Assembler {

    public void assemble(String sourceFile) {
        System.out.println("Assembling " + sourceFile);
    }

}
```

Linker.java

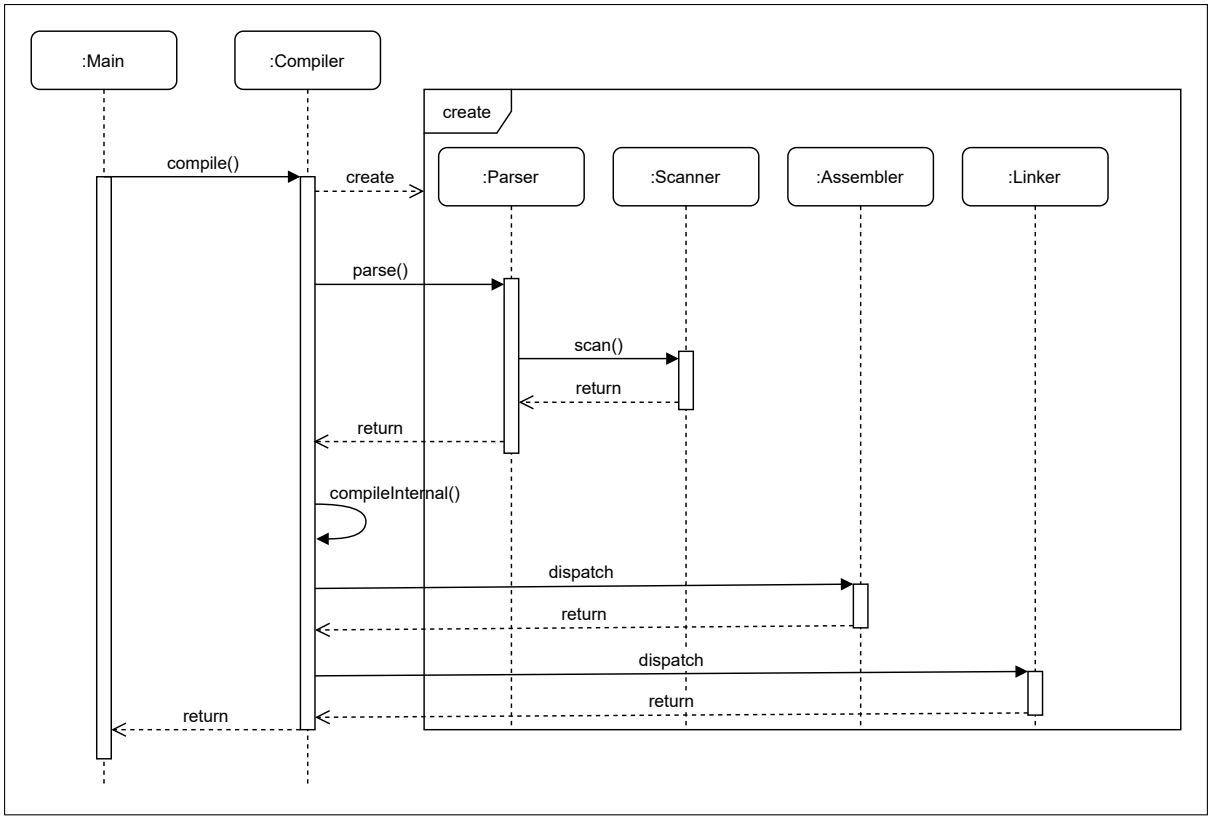
```
package facade;

public class Linker {

    public void link(String sourceFile){
        System.out.println("Linking " + sourceFile);
    }

}
```

Compiler.java

```java
package facade;

public class Compiler {

    private Parser parser;
    private Assembler assembler;
    private Linker linker;

    public void compile(String sourceFile) {
        parser = new Parser();
        assembler = new Assembler();
        linker = new Linker();

        parser.parse(sourceFile);
        compileInternal(sourceFile);
        assembler.assemble(sourceFile);
        linker.link(sourceFile);
    }

    private void compileInternal(String sourceFile) {
        System.out.println("Generating " + sourceFile + ".asm");
    }

}
```

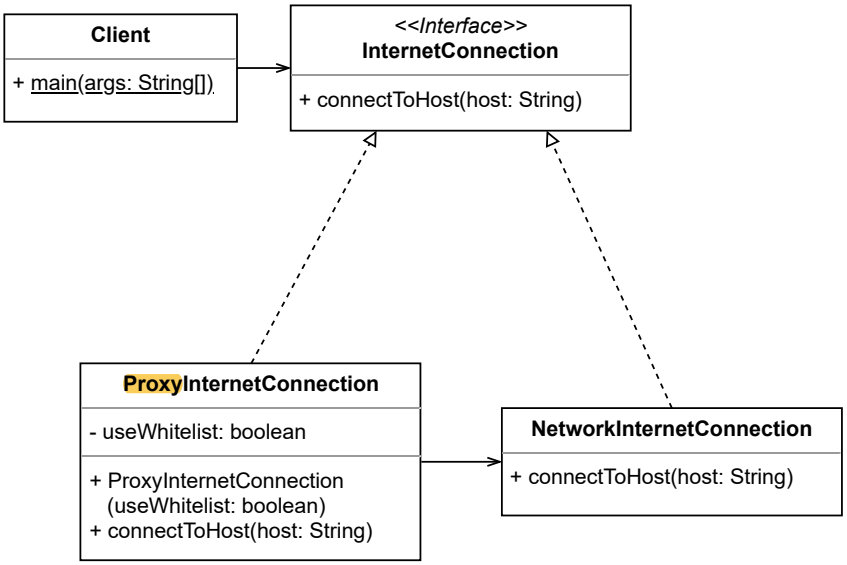
TestKlasse.java

```java
package facade;

public class TestKlasse {

    public static void main(String[] args) {
        Compiler comp = new Compiler();
        comp.compile("HelloWorld_Facade");
    }

}
```


:Main
:Compiler
create
:Parser
:Scanner
:Assembler
:Linker
compile()
create
parse()
scan()
return
return
compileInternal()
dispatch
return
dispatch
return
return

8.3 Proxy-Muster

Sie sollen für das Internet einer Firma einen flexiblen Proxy-Server erstellen, der auf Grund einer Whitelist und einer Blacklist den Zugang zum Internet erlaubt oder blockiert. Die Implementierung soll mit dem Proxy-Muster erfolgen:



Aufgaben

1. Erstellen Sie die Listen whitelist.txt und blacklist.txt und füllen Sie sie mit einigen Einträgen.

2. Setzen Sie die Klassen des UML-Diagramms in Java-Code um. Ob der Proxy-Server im Whitelist- oder Blacklist-Modus arbeitet, soll im Konstruktor übergeben werden.

3. Die Klasse `NetworkInternetConnection` gibt eine Meldung in der Konsole aus, wenn eine Verbindung zum Host möglich ist. Die Klasse `ProxyInternetConnection` leitet diese Anfrage je nach Modus weiter, oder wirft eine Exception, wenn keine Verbindung möglich ist.

4. Testen Sie beide Modi mit mehreren Aufrufen. Fangen Sie eventuell auftretende Exceptions ab und geben Sie dem Benutzer eine Rückmeldung in der Konsole.