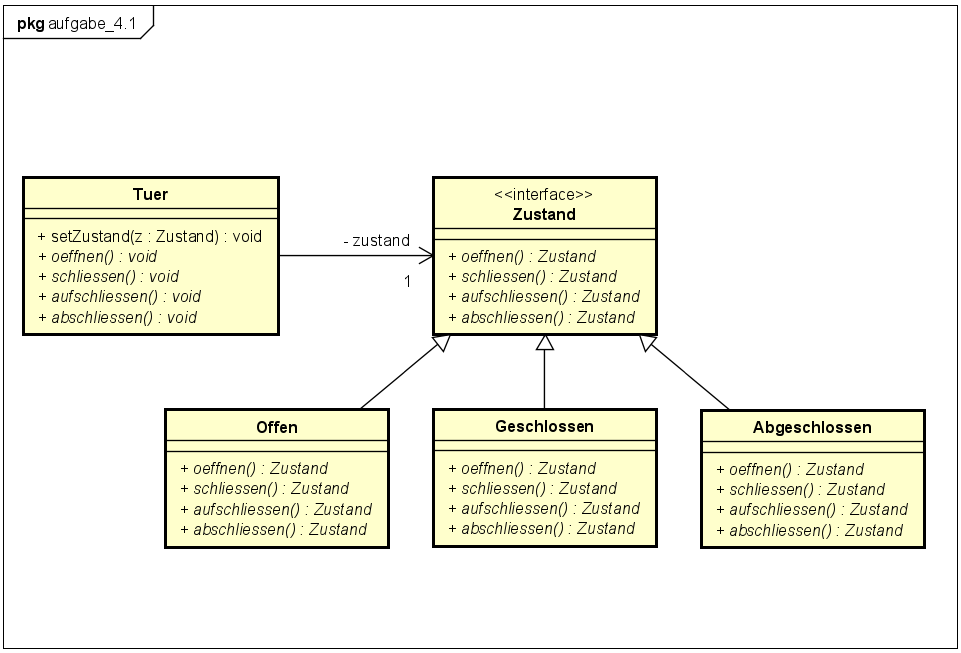
Aufgabe 4.1



public class prog {  
 public static void main(String[] args) {  
 Tuer tuer = new Tuer();  
  
 tuer.oeffnen();  
 tuer.abschliessen();  
 tuer.aufschliessen();  
 tuer.schliessen();  
  
 tuer.schliessen();  
 tuer.aufschliessen();  
 tuer.oeffnen();  
  
 tuer.schliessen();  
  
 tuer.abschliessen();  
  
 tuer.oeffnen();  
 tuer.schliessen();  
 tuer.abschliessen();  
 tuer.aufschliessen();  
 }  
}

public class Tuer {  
 private Zustand zustand = new Offen();  
  
 public void oeffnen() {  
 zustand = this.zustand.oeffnen();  
 }  
  
 public void schliessen() {  
 zustand = this.zustand.schliessen();  
 }  
  
 public void aufschliessen() {  
 zustand = this.zustand.aufschliessen();  
 }  
  
 public void abschliessen() {  
 zustand = this.zustand.abschliessen();  
 }  
  
 public void setZustand(Zustand zustand) {  
 this.zustand = zustand;  
 }  
}

public interface Zustand {  
 Zustand oeffnen();  
 Zustand schliessen();  
 Zustand aufschliessen();  
 Zustand abschliessen();  
}

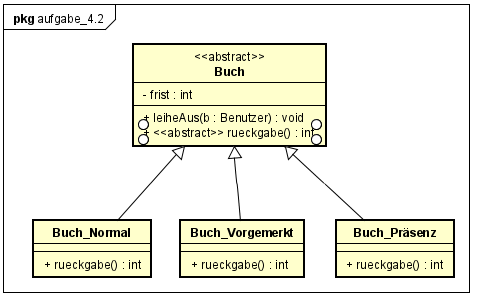
public class Geschlossen implements Zustand {  
 @Override  
 public Zustand oeffnen() {  
 System.out.println("Tür geöffnet! Zustand gewechselt!");  
 return new Offen();  
 }  
  
 @Override  
 public Zustand schliessen() {  
 System.out.println("Tür bereits geschlossen!");  
 return this;  
 }  
  
 @Override  
 public Zustand aufschliessen() {  
 System.out.println("Tür ist nicht abgeschlossen!");  
 return this;  
 }  
  
 @Override  
 public Zustand abschliessen() {  
 System.out.println("Tür abgeschlossen! Zustand gewechselt!");  
 return new Abgeschlossen();  
 }  
}

public class Offen implements Zustand {  
 @Override  
 public Zustand oeffnen() {  
 System.out.println("Tür bereits offen!");  
 return this;  
 }  
  
 @Override  
 public Zustand schliessen() {  
 System.out.println("Tuer geschlossen! Zustand gewechselt!");  
 return new Geschlossen();  
 }  
  
 @Override  
 public Zustand aufschliessen() {  
 System.out.println("Tür ist nicht abgeschlossen!");  
 return this;  
 }  
  
 @Override  
 public Zustand abschliessen() {  
 System.out.println("Tür noch offen!");  
 return this;  
 }  
}

public class Abgeschlossen implements Zustand {  
 @Override  
 public Zustand oeffnen() {  
 System.out.println("Tür ist noch abgeschlossen!");  
 return this;  
 }  
  
 @Override  
 public Zustand schliessen() {  
 System.out.println("Tür bereits geschlossen und abgeschlossen!");  
 return this;  
 }  
  
 @Override  
 public Zustand aufschliessen() {  
 System.out.println("Tür aufgeschlossen! Zustand gewechselt!");  
 return new Geschlossen();  
 }  
  
 @Override  
 public Zustand abschliessen() {  
 System.out.println("Tür bereits abgeschlossen!");  
 return this;  
 }  
}

Aufgabe 2)

public class Buch\_a {  
 private String type;  
 private int frist;  
  
 public Buch\_a(String type) {  
 this.type = type;  
 }  
  
 public void leiheAus(Benutzer b) {  
 if (b.isBerechtigt()) {  
 switch (type) {  
 case "Normal":  
 this.frist = 4 \* 7;  
 break;  
 case "Vorgemerkt":  
 this.frist = 2 \* 7;  
 break;  
 case "Präsenz":  
 this.frist = 1;  
 break;  
 }  
 }  
 }  
}



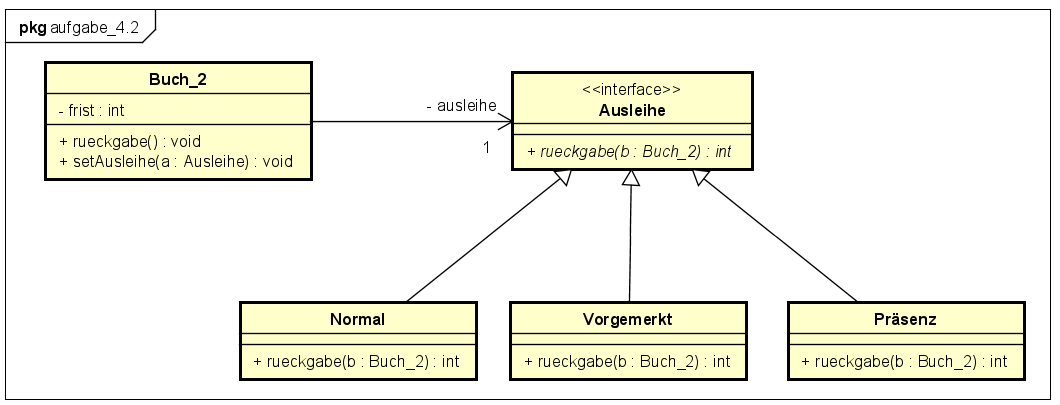
public abstract class Buch {  
 private int frist;  
 abstract int rueckgabe();  
  
 public void leiheAus(Benutzer b) {  
 if (b.isBerechtigt()) {  
 this.frist = this.rueckgabe();  
 }  
 }  
  
 public int getFrist() {  
 return frist;  
 }  
  
 public void setFrist(int frist) {  
 this.frist = frist;  
 }  
}

public class Buch\_Normal extends Buch {  
 @Override  
 int rueckgabe() {  
 return 4 \* 7;  
 }  
}

public class Buch\_Vorgemerkt extends Buch {  
 @Override  
 int rueckgabe() {  
 return 2\*7;  
 }  
}

public class Buch\_Präsenz extends Buch {  
 @Override  
 int rueckgabe() {  
 return 1;  
 }  
}

public class prog {  
 public static void main(String[] args) {  
 Benutzer b = new Benutzer(true);  
  
 Buch\_Normal b\_normal = new Buch\_Normal();  
 b\_normal.leiheAus(b);  
  
 Buch\_Vorgemerkt b\_vorgemerkt = new Buch\_Vorgemerkt();  
 b\_vorgemerkt.leiheAus(b);  
  
 Buch\_Präsenz b\_praesenz = new Buch\_Präsenz();  
 b\_praesenz.leiheAus(b);  
 }  
}



public class Buch\_2 {  
 private int frist;  
 private Ausleihe ausleihe;  
  
 public Buch\_2(Ausleihe ausleihe) {  
 this.ausleihe = ausleihe;  
 }  
  
 public void rueckgabe() {  
 this.frist = this.ausleihe.rueckgabe();  
 System.out.println("Frist liegt bei " + frist + " Tagen!");  
 }  
  
 public void setAusleihe(Ausleihe ausleihe) {  
 this.ausleihe = ausleihe;  
 }  
}

public interface Ausleihe {  
 int rueckgabe();  
}

public class Normal implements Ausleihe {  
 @Override  
 public int rueckgabe() {  
 return 4\*7;  
 }  
}

public class Vorgemerkt implements Ausleihe {  
 @Override  
 public int rueckgabe() {  
 return 2\*7;  
 }  
}

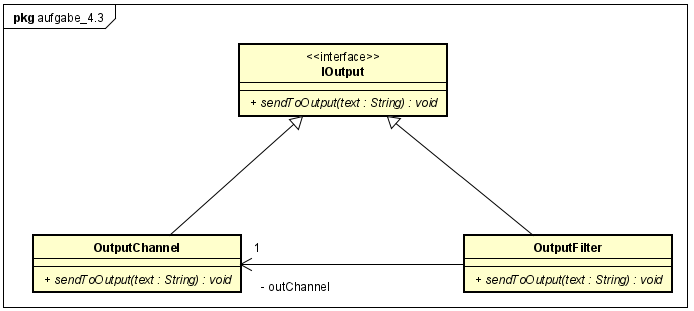
public class Praesenz implements Ausleihe {  
 @Override  
 public int rueckgabe() {  
 return 1;  
 }  
}

public class prog {  
 public static void main(String[] args) {  
 Normal normal = new Normal();  
 Vorgemerkt vorgemerkt = new Vorgemerkt();  
 Praesenz praesenz = new Praesenz();  
  
 Buch\_2 buch\_1 = new Buch\_2(normal);  
 Buch\_2 buch\_2 = new Buch\_2(vorgemerkt);  
 Buch\_2 buch\_3 = new Buch\_2(praesenz);  
  
 buch\_1.rueckgabe();  
 buch\_2.rueckgabe();  
 buch\_3.rueckgabe();  
 }  
}

Template-Pattern: Ein Buch Objekt pro Art des Ausleihens, welches instanziiert wird, mit einer Oberklasse Buch welches das Standardverhalten implementiert. Kann Berechnung zur Laufzeit nicht ändern, müsste neues Buch Objekt eines anderen Typs erstellen

Strategy-Pattern: Ein Buch Objekt, welches das Standartverhalten implementiert und instanziiert wird. Enthält als Attribut die Berechnungsvorschrift für die Rückgabe. Kann also seine Berechnung ändern zur Laufzeit

Aufgabe 4.3)



b) Proxy-Pattern wird implementiert

c)

IOutput: Objekt

OutptFilter: Proxy

OutputChannel: RealObjekt