

## Script generated by TTT

Title: groh: profile1 (17.06.2016)

Date: Fri Jun 17 09:30:50 CEST 2016

Duration: 74:06 min

Pages: 79

## Main - Methode

- In einer Klasse eine spezielle Klassenmethode **main** mit festgelegter **Signatur** (Name + Sequenz der Parametertypen) und **Modifizieren**

```
public static void main(String[] args)
```

- wird **einmal** beim Programmstart aufgerufen (<---> Computer muss wissen wo er anfangen soll)

```
class BicycleDemo {  
    public static void main(String[] args) {  
        // Create two different Bicycle objects  
        Bicycle bike1 = new Bicycle();  
        Bicycle bike2 = new Bicycle();  
  
        // Invoke methods on those objects  
        bike1.changeCadence(50);  
        bike1.speedUp(10);  
        bike1.changeGear(2);  
    }  
}
```

139

## Overriding

**Overriding:** In einer **Unterklasse** Methode mit **gleichem Namen**, und **gleicher Parameterliste** wie in Oberklasse:

```
class Bicycle {  
    int speed;  
    public void speedUp(int increment) {  
        speed = speed + increment;  
        System.out.println("superclass instance-method");  
    }  
  
    class MountainBike extends Bicycle {  
        public void speedUp(int increment) {  
            super.speedUp(2 * increment); // call overridden method of superclass  
            System.out.println("subclass instance-method");  
        }  
    }  
  
    MountainBike mb = new MountainBike();  
    mb.speedUp(10); // mb.speed == 20
```

Variante mit  
super

→ Ausgabe: superclass instance-method  
subclass instance-method

## Hiding

**Hiding:** Analog zu Overriding aber für **Klassenmethoden (static)**

```
class Bicycle {  
    public static void myClassMethod(int someInt) {  
        System.out.println("superclass class-method");  
    }  
  
    class MountainBike extends Bicycle {  
        public static void myClassMethod(int someInt) {  
            System.out.println("subclass class-method");  
        }  
    }  
  
    Bicycle.myClassMethod(10); // "superclass class-method"  
    MountainBike.myClassMethod(10); // "subclass class-method"
```

**Sinn:** Unterklasse bietet speziellere Version der Methode an (Aspekt von Polymorphie)

## Interfaces <---> Polymorphie : Beispiel

```
interface ICanSting {  
    public void sting();  
}  
  
class LittleBee implements ICanSting {  
    public void sting() {  
        System.out.println("*pieks*");  
    }  
}  
  
class AngryHornet implements ICanSting {  
    public void sting() {  
        System.out.println("*MEGAPIEKS*");  
    }  
}  
  
LittleBee maja = new LittleBee();  
AngryHornet horst = new AngryHornet();  
ICanSting someStinger;  
someStinger = maja;  
someStinger.sting();          // *pieks*  
someStinger = horst;  
someStinger.sting();          // *MEGAPIEKS*
```

## Interfaces <---> Polymorphie : Beispiel

```
interface ICanSting {  
    public void sting();  
}  
  
class LittleBee implements ICanSting {  
    public void sting() {  
        System.out.println("*pieks*");  
    }  
}  
  
class AngryHornet implements ICanSting {  
    public void sting() {  
        System.out.println("*MEGAPIEKS*");  
    }  
}  
  
LittleBee maja = new LittleBee();  
AngryHornet horst = new AngryHornet();  
ICanSting someStinger;  
someStinger = maja;  
someStinger.sting();          // *pieks*  
someStinger = horst;  
someStinger.sting();          // *MEGAPIEKS*
```

149

149

## Interfaces <---> Polymorphie : Beispiel

```
interface ICanSting {  
    public void sting();  
}  
  
class LittleBee implements ICanSting {  
    public void sting() {  
        System.out.println("*pieks*");  
    }  
}  
  
class AngryHornet implements ICanSting {  
    public void sting() {  
        System.out.println("*MEGAPIEKS*");  
    }  
}  
  
LittleBee maja = new LittleBee();  
AngryHornet horst = new AngryHornet();  
ICanSting someStinger;  
someStinger = maja;  
someStinger.sting();          // *pieks*  
someStinger = horst;  
someStinger.sting();          // *MEGAPIEKS*
```

## Ausgabe

Gegeben ist folgende **fehlerhafte Java Methode** expo zur näherungsweisen Berechnung der Funktion  $f(x) = e^x \approx \sum_{i=0}^{30} \frac{x^i}{i!}$

```
double void expo(double x){  
    double result = 1.0;  
    double help = 1.0;  
    double help2 = 1.0;  
    for (int i=1; i<30; i++){  
        help = help * x;  
        help2 = help2 * i;  
        result = result + expo(help / help2);  
    }  
    return result;  
}
```

Umranden bzw. markieren Sie die **drei Fehler** im Code und **verbessern Sie sie!**

149

151

Gegeben ist folgende **fehlerhafte Java Methode** `expo` zur näherungsweisen Berechnung der Funktion  $f(x) = e^x \approx \sum_{i=0}^{30} \frac{x^i}{i!}$

```
double void expo(double x){
    double result = 1.0;
    double help = 1.0;
    double help2 = 1.0;
    for (int i=1; i<30; i++){
        help = help * x;
        help2 = help2 * i;
        result = result + expo(help / help2);
    }
    return result;
}
```

Umranden bzw. markieren Sie die **drei Fehler** im Code und **verbessen Sie sie!**

Gegeben ist folgende **fehlerhafte Java Methode** `expo` zur näherungsweisen Berechnung der Funktion  $f(x) = e^x \approx \sum_{i=0}^{30} \frac{x^i}{i!}$

```
double void expo(double x){
    double result = 1.0;
    double help = 1.0;
    double help2 = 1.0;
    for (int i=1; i<30; i++){
        help = help * x;
        help2 = help2 * i;
        result = result + expo(help / help2);
    }
    return result;
}
```

Umranden bzw. markieren Sie die **drei Fehler** im Code und **verbessen Sie sie!**

Gegeben ist folgende **fehlerhafte Java Methode** `expo` zur näherungsweisen Berechnung der Funktion  $f(x) = e^x \approx \sum_{i=0}^{30} \frac{x^i}{i!}$

```
double void expo(double x){
    double result = 1.0;
    double help = 1.0;
    double help2 = 1.0;
    for (int i=1; i<30; i++){
        help = help * x;
        help2 = help2 * i;
        result = result + expo(help / help2);
    }
    return result;
}
```

Umranden bzw. markieren Sie die **drei Fehler** im Code und **verbessen Sie sie!**

Gegeben ist folgende **fehlerhafte Java Methode** `expo` zur näherungsweisen Berechnung der Funktion  $f(x) = e^x \approx \sum_{i=0}^{30} \frac{x^i}{i!}$

```
double void expo(double x){
    double result = 1.0;
    double help = 1.0;
    double help2 = 1.0;
    for (int i=1; i<30; i++){
        help = help * x;
        help2 = help2 * i;
        result = result + expo(help / help2);
    }
    return result;
}
```

Umranden bzw. markieren Sie die **drei Fehler** im Code und **verbessen Sie sie!**

Gegeben ist folgende **fehlerhafte Java Methode** `expo` zur näherungsweisen Berechnung der Funktion  $f(x) = e^x \approx \sum_{i=0}^{30} \frac{x^i}{i!}$

```
double void expo(double x){
    double result = 1.0;
    double help = 1.0;
    double help2 = 1.0;
    for (int i=1; i<30; i++){
        help = help * x;
        help2 = help2 * i;
        result = result + expo(help / help2);
    }
    return result;
}
```

Umranden bzw. markieren Sie die **drei Fehler** im Code und **verbessen Sie sie!**

Gegeben ist folgende **fehlerhafte Java Methode** `expo` zur näherungsweisen Berechnung der Funktion  $f(x) = e^x \approx \sum_{i=0}^{30} \frac{x^i}{i!}$

```
double void expo(double x){
    double result = 1.0;
    double help = 1.0;
    double help2 = 1.0;
    for (int i=1; i<30; i++){
        help = help * x;
        help2 = help2 * i;
        result = result + expo(help / help2);
    }
    return result;
}
```

Umranden bzw. markieren Sie die **drei Fehler** im Code und **verbessen Sie sie!**

Gegeben ist folgende **fehlerhafte Java Methode** `expo` zur näherungsweisen Berechnung der Funktion  $f(x) = e^x \approx \sum_{i=0}^{30} \frac{x^i}{i!}$

```
double void expo(double x){
    double result = 1.0;
    double help = 1.0;
    double help2 = 1.0;
    for (int i=1; i<30; i++){
        help = help * x;
        help2 = help2 * i;
        result = result + expo(help / help2);
    }
    return result;
}
```

Umranden bzw. markieren Sie die **drei Fehler** im Code und **verbessen Sie sie!**

Gegeben ist folgende **fehlerhafte Java Methode** `expo` zur näherungsweisen Berechnung der Funktion  $f(x) = e^x \approx \sum_{i=0}^{30} \frac{x^i}{i!}$

```
double void expo(double x){
    double result = 1.0;
    double help = 1.0;
    double help2 = 1.0;
    for (int i=1; i<30; i++){
        help = help * x;
        help2 = help2 * i;
        result = result + expo(help / help2);
    }
    return result;
}
```

Umranden bzw. markieren Sie die **drei Fehler** im Code und **verbessen Sie sie!**

Gegeben ist folgende **fehlerhafte Java Methode** `expo` zur näherungsweisen Berechnung der Funktion  $f(x) = e^x \approx \sum_{i=0}^{30} \frac{x^i}{i!}$

```
double void expo(double x){
    double result = 1.0;
    double help = 1.0;
    double help2 = 1.0;
    for (int i=1; i<30; i++){
        help = help * x;
        help2 = help2 * i;
        result = result + expo(help / help2);
    }
    return result;
}
```

Umranden bzw. markieren Sie die **drei Fehler** im Code und **verbessen Sie sie!**

Gegeben ist folgende **fehlerhafte Java Methode** `expo` zur näherungsweisen Berechnung der Funktion  $f(x) = e^x \approx \sum_{i=0}^{30} \frac{x^i}{i!}$

```
double void expo(double x){
    double result = 1.0;
    double help = 1.0;
    double help2 = 1.0;
    for (int i=1; i<30; i++){
        help = help * x;
        help2 = help2 * i;
        result = result + expo(help / help2);
    }
    return result;
}
```

Umranden bzw. markieren Sie die **drei Fehler** im Code und **verbessen Sie sie!**

Gegeben ist folgende **fehlerhafte Java Methode** `expo` zur näherungsweisen Berechnung der Funktion  $f(x) = e^x \approx \sum_{i=0}^{30} \frac{x^i}{i!}$

```
double void expo(double x){
    double result = 1.0;
    double help = 1.0;
    double help2 = 1.0;
    for (int i=1; i<30; i++){
        help = help * x;
        help2 = help2 * i;
        result = result + expo(help / help2);
    }
    return result;
}
```

Umranden bzw. markieren Sie die **drei Fehler** im Code und **verbessen Sie sie!**

Welche **Ausgabe** produziert NotoriousRomanticsDemo?

Welche der drei Konzepte **Overloading, Overriding, Polymorphie** werden im obigen Code benutzt, welche nicht?  
**BEGRÜNDEN** Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {
    public static void main(String[] args) {
        Person manOnTrain = new Person();
        manOnTrain.isInNiceMood = true;
        LovingPerson womanOnTrain = new LovingPerson();
        womanOnTrain.isInNiceMood = true;
        womanOnTrain.sayLovePoem("aehm..."); 
        manOnTrain.commentOnLovePoem("that's so beautiful!");
        Person person = womanOnTrain;
        person.commentOnLovePoem("you like it! :-)");
        Person someBystander = new Person();
        someBystander.commentOnLovePoem(" how cute!");
    }
}

public class LovingPerson extends Person implements Romantic{
    public void sayLovePoem(String what){
        System.out.println(what + "ene mene muh: I'm in love with you!");
    }

    public void commentOnLovePoem(String what){
        System.out.print("no matter what: ");
        this.commentOnLovePoem();
    }

    public void commentOnLovePoem(){
        System.out.println("I am in love");
    }
}

public class Person {
    public boolean isInNiceMood = false;
    public void commentOnLovePoem(String what){
        if(isInNiceMood)
            System.out.println("oiii :-) " + what);
        else
            System.out.println("oerks! :-(");
    }
}

public interface Romantic {
    public void sayLovePoem(String what);
}
```

## Ausgabe

Welche Ausgabe produziert NotoriousRomanticsDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {  
    public static void main(String[] args) {  
        Person manOnTrain = new Person();  
        manOnTrain.isInNiceMood = true;  
        LovingPerson womanOnTrain = new LovingPerson();  
        womanOnTrain.isInNiceMood = true;  
        womanOnTrain.sayLovePoem("aehm...");  
        manOnTrain.commentOnLovePoem("that's so beautiful!");  
        Person person = womanOnTrain;  
        person.commentOnLovePoem("you like it! :-)");  
        Person someBystander = new Person();  
        someBystander.commentOnLovePoem(" how cute!");  
    }  
}  
  
public class LovingPerson extends Person implements Romantic{  
    public void sayLovePoem(String what){  
        System.out.println(what + "ene mene muh: I'm in love with you!");  
    }  
  
    public void commentOnLovePoem(String what){  
        System.out.print("no matter what: ");  
        this.commentOnLovePoem();  
    }  
  
    public void commentOnLovePoem(){  
        System.out.println("I am in love");  
    }  
}  
  
public class Person {  
    public boolean isInNiceMood = false;  
    public void commentOnLovePoem(String what){  
        if(isInNiceMood)  
            System.out.println("oiii :-) " + what);  
        else  
            System.out.println("oerks! :-(");  
    }  
}  
  
public interface Romantic {  
    public void sayLovePoem(String what);  
}
```

## Ausgabe

Welche Ausgabe produziert NotoriousRomanticsDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!



Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {  
    public static void main(String[] args) {  
        Person manOnTrain = new Person();  
        manOnTrain.isInNiceMood = true;  
        LovingPerson womanOnTrain = new LovingPerson();  
        womanOnTrain.isInNiceMood = true;  
        womanOnTrain.sayLovePoem("aehm...");  
        manOnTrain.commentOnLovePoem("that's so beautiful!");  
        Person person = womanOnTrain;  
        person.commentOnLovePoem("you like it! :-)");  
        Person someBystander = new Person();  
        someBystander.commentOnLovePoem(" how cute!");  
    }  
}  
  
public class LovingPerson extends Person implements Romantic{  
    public void sayLovePoem(String what){  
        System.out.println(what + "ene mene muh: I'm in love with you!");  
    }  
  
    public void commentOnLovePoem(String what){  
        System.out.print("no matter what: ");  
        this.commentOnLovePoem();  
    }  
  
    public void commentOnLovePoem(){  
        System.out.println("I am in love");  
    }  
}  
  
public class Person {  
    public boolean isInNiceMood = false;  
    public void commentOnLovePoem(String what){  
        if(isInNiceMood)  
            System.out.println("oiii :-) " + what);  
        else  
            System.out.println("oerks! :-(");  
    }  
}  
  
public interface Romantic {  
    public void sayLovePoem(String what);  
}
```

## Ausgabe

Welche Ausgabe produziert NotoriousRomanticsDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {  
    public static void main(String[] args) {  
        Person manOnTrain = new Person();  
        manOnTrain.isInNiceMood = true;  
        LovingPerson womanOnTrain = new LovingPerson();  
        womanOnTrain.isInNiceMood = true;  
        womanOnTrain.sayLovePoem("aehm...");  
        manOnTrain.commentOnLovePoem("that's so beautiful!");  
        Person person = womanOnTrain;  
        person.commentOnLovePoem("you like it! :-)");  
        Person someBystander = new Person();  
        someBystander.commentOnLovePoem(" how cute!");  
    }  
}  
  
public class LovingPerson extends Person implements Romantic{  
    public void sayLovePoem(String what){  
        System.out.println(what + "ene mene muh: I'm in love with you!");  
    }  
  
    public void commentOnLovePoem(String what){  
        System.out.print("no matter what: ");  
        this.commentOnLovePoem();  
    }  
  
    public void commentOnLovePoem(){  
        System.out.println("I am in love");  
    }  
}  
  
public class Person {  
    public boolean isInNiceMood = false;  
    public void commentOnLovePoem(String what){  
        if(isInNiceMood)  
            System.out.println("oiii :-) " + what);  
        else  
            System.out.println("oerks! :-(");  
    }  
}  
  
public interface Romantic {  
    public void sayLovePoem(String what);  
}
```

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {  
    public static void main(String[] args) {  
        Person manOnTrain = new Person();  
        manOnTrain.isInNiceMood = true;  
        LovingPerson womanOnTrain = new LovingPerson();  
        womanOnTrain.isInNiceMood = true;  
        womanOnTrain.sayLovePoem("aehm...");  
        manOnTrain.commentOnLovePoem("that's so beautiful!");  
        Person person = womanOnTrain;  
        person.commentOnLovePoem("you like it! :-)");  
        Person someBystander = new Person();  
        someBystander.commentOnLovePoem(" how cute!");  
    }  
}  
  
public class LovingPerson extends Person implements Romantic{  
    public void sayLovePoem(String what){  
        System.out.println(what + "ene mene muh: I'm in love with you!");  
    }  
  
    public void commentOnLovePoem(String what){  
        System.out.print("no matter what: ");  
        this.commentOnLovePoem();  
    }  
  
    public void commentOnLovePoem(){  
        System.out.println("I am in love");  
    }  
}  
  
public class Person {  
    public boolean isInNiceMood = false;  
    public void commentOnLovePoem(String what){  
        if(isInNiceMood)  
            System.out.println("oiii :-) " + what);  
        else  
            System.out.println("oerks! :-(");  
    }  
}  
  
public interface Romantic {  
    public void sayLovePoem(String what);  
}
```



Welche Ausgabe produziert NotoriousRomanticsDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {
    public static void main(String[] args) {
        Person manOnTrain = new Person();
        manOnTrain.isInNiceMood = true;
        LovingPerson womanOnTrain = new LovingPerson();
        womanOnTrain.isInNiceMood = true;
        womanOnTrain.sayLovePoem("aehm..."); 
        manOnTrain.commentOnLovePoem("that's so beautiful!");
        Person person = womanOnTrain;
        person.commentOnLovePoem("you like it! :-)");
        Person someBystander = new Person();
        someBystander.commentOnLovePoem(" how cute!");
    }
}

public class LovingPerson extends Person implements Romantic{
    public void sayLovePoem(String what){
        System.out.println(what + "ene mene muh: I'm in love with you!");
    }

    public void commentOnLovePoem(String what){
        System.out.print("no matter what: ");
        this.commentOnLovePoem();
    }

    public void commentOnLovePoem(){
        System.out.println("I am in love");
    }
}

public class Person {
    public boolean isInNiceMood = false;
    public void commentOnLovePoem(String what){
        if(isInNiceMood)
            System.out.println("oiii :-) " + what);
        else
            System.out.println("oerks! :-(");
    }
}

public interface Romantic {
    public void sayLovePoem(String what);
}
```



Welche Ausgabe produziert NotoriousRomanticsDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {
    public static void main(String[] args) {
        Person manOnTrain = new Person();
        manOnTrain.isInNiceMood = true;
        LovingPerson womanOnTrain = new LovingPerson();
        womanOnTrain.isInNiceMood = true;
        womanOnTrain.sayLovePoem("aehm..."); 
        manOnTrain.commentOnLovePoem("that's so beautiful!");
        Person person = womanOnTrain;
        person.commentOnLovePoem("you like it! :-)");
        Person someBystander = new Person();
        someBystander.commentOnLovePoem(" how cute!");
    }
}

public class LovingPerson extends Person implements Romantic{
    public void sayLovePoem(String what){
        System.out.println(what + "ene mene muh: I'm in love with you!");
    }

    public void commentOnLovePoem(String what){
        System.out.print("no matter what: ");
        this.commentOnLovePoem();
    }

    public void commentOnLovePoem(){
        System.out.println("I am in love");
    }
}

public class Person {
    public boolean isInNiceMood = false;
    public void commentOnLovePoem(String what){
        if(isInNiceMood)
            System.out.println("oiii :-) " + what);
        else
            System.out.println("oerks! :-(");
    }
}

public interface Romantic {
    public void sayLovePoem(String what);
}
```



Welche Ausgabe produziert NotoriousRomanticsDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {
    public static void main(String[] args) {
        Person manOnTrain = new Person();
        manOnTrain.isInNiceMood = true;
        LovingPerson womanOnTrain = new LovingPerson();
        womanOnTrain.isInNiceMood = true;
        womanOnTrain.sayLovePoem("aehm..."); 
        manOnTrain.commentOnLovePoem("that's so beautiful!");
        Person person = womanOnTrain;
        person.commentOnLovePoem("you like it! :-)");
        Person someBystander = new Person();
        someBystander.commentOnLovePoem(" how cute!");
    }
}

public class LovingPerson extends Person implements Romantic{
    public void sayLovePoem(String what){
        System.out.println(what + "ene mene muh: I'm in love with you!");
    }

    public void commentOnLovePoem(String what){
        System.out.print("no matter what: ");
        this.commentOnLovePoem();
    }

    public void commentOnLovePoem(){
        System.out.println("I am in love");
    }
}

public class Person {
    public boolean isInNiceMood = false;
    public void commentOnLovePoem(String what){
        if(isInNiceMood)
            System.out.println("oiii :-) " + what);
        else
            System.out.println("oerks! :-(");
    }
}

public interface Romantic {
    public void sayLovePoem(String what);
}
```



Welche Ausgabe produziert NotoriousRomanticsDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {
    public static void main(String[] args) {
        Person manOnTrain = new Person();
        manOnTrain.isInNiceMood = true;
        LovingPerson womanOnTrain = new LovingPerson();
        womanOnTrain.isInNiceMood = true;
        womanOnTrain.sayLovePoem("aehm..."); 
        manOnTrain.commentOnLovePoem("that's so beautiful!");
        Person person = womanOnTrain;
        person.commentOnLovePoem("you like it! :-)");
        Person someBystander = new Person();
        someBystander.commentOnLovePoem(" how cute!");
    }
}

public class LovingPerson extends Person implements Romantic{
    public void sayLovePoem(String what){
        System.out.println(what + "ene mene muh: I'm in love with you!");
    }

    public void commentOnLovePoem(String what){
        System.out.print("no matter what: ");
        this.commentOnLovePoem();
    }

    public void commentOnLovePoem(){
        System.out.println("I am in love");
    }
}

public class Person {
    public boolean isInNiceMood = false;
    public void commentOnLovePoem(String what){
        if(isInNiceMood)
            System.out.println("oiii :-) " + what);
        else
            System.out.println("oerks! :-(");
    }
}

public interface Romantic {
    public void sayLovePoem(String what);
}
```

## Ausgabe

Welche Ausgabe produziert NotoriousRomanticsDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {  
    public static void main(String[] args) {  
        Person manOnTrain = new Person();  
        manOnTrain.isInNiceMood = true;  
        LovingPerson womanOnTrain = new LovingPerson();  
        womanOnTrain.isInNiceMood = true;  
        womanOnTrain.sayLovePoem("aehm...");  
        manOnTrain.commentOnLovePoem("that's so beautiful!");  
        Person person = womanOnTrain;  
        person.commentOnLovePoem("you like it! :-)");  
        Person someBystander = new Person();  
        someBystander.commentOnLovePoem(" how cute!");  
    }  
}  
  
public class LovingPerson extends Person implements Romantic{  
    public void sayLovePoem(String what){  
        System.out.println(what + "ene mene muh: I'm in love with you!");  
    }  
  
    public void commentOnLovePoem(String what){  
        System.out.print("no matter what: ");  
        this.commentOnLovePoem();  
    }  
  
    public void commentOnLovePoem(){  
        System.out.println("I am in love");  
    }  
}  
  
public class Person {  
    public boolean isInNiceMood = false;  
    public void commentOnLovePoem(String what){  
        if(isInNiceMood)  
            System.out.println("oiii :-) " + what);  
        else  
            System.out.println("oerks! :-(");  
    }  
}  
  
public interface Romantic {  
    public void sayLovePoem(String what);  
}
```

## Ausgabe

Welche Ausgabe produziert NotoriousRomanticsDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {  
    public static void main(String[] args) {  
        Person manOnTrain = new Person();  
        manOnTrain.isInNiceMood = true;  
        LovingPerson womanOnTrain = new LovingPerson();  
        womanOnTrain.isInNiceMood = true;  
        womanOnTrain.sayLovePoem("aehm...");  
        manOnTrain.commentOnLovePoem("that's so beautiful!");  
        Person person = womanOnTrain;  
        person.commentOnLovePoem("you like it! :-)");  
        Person someBystander = new Person();  
        someBystander.commentOnLovePoem(" how cute!");  
    }  
}  
  
public class LovingPerson extends Person implements Romantic{  
    public void sayLovePoem(String what){  
        System.out.println(what + "ene mene muh: I'm in love with you!");  
    }  
  
    public void commentOnLovePoem(String what){  
        System.out.print("no matter what: ");  
        this.commentOnLovePoem();  
    }  
  
    public void commentOnLovePoem(){  
        System.out.println("I am in love");  
    }  
}  
  
public class Person {  
    public boolean isInNiceMood = false;  
    public void commentOnLovePoem(String what){  
        if(isInNiceMood)  
            System.out.println("oiii :-) " + what);  
        else  
            System.out.println("oerks! :-(");  
    }  
}  
  
public interface Romantic {  
    public void sayLovePoem(String what);  
}
```

## Ausgabe

Welche Ausgabe produziert NotoriousRomanticsDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {  
    public static void main(String[] args) {  
        Person manOnTrain = new Person();  
        manOnTrain.isInNiceMood = true;  
        LovingPerson womanOnTrain = new LovingPerson();  
        womanOnTrain.isInNiceMood = true;  
        womanOnTrain.sayLovePoem("aehm...");  
        manOnTrain.commentOnLovePoem("that's so beautiful!");  
        Person person = womanOnTrain;  
        person.commentOnLovePoem("you like it! :-)");  
        Person someBystander = new Person();  
        someBystander.commentOnLovePoem(" how cute!");  
    }  
}  
  
public class LovingPerson extends Person implements Romantic{  
    public void sayLovePoem(String what){  
        System.out.println(what + "ene mene muh: I'm in love with you!");  
    }  
}
```

```
public void commentOnLovePoem(String what){  
    System.out.print("no matter what: ");  
    this.commentOnLovePoem();  
}  
  
public void commentOnLovePoem(){  
    System.out.println("I am in love");  
}
```

```
public class Person {  
    public boolean isInNiceMood = false;  
    public void commentOnLovePoem(String what){  
        if(isInNiceMood)  
            System.out.println("oiii :-) " + what);  
        else  
            System.out.println("oerks! :-(");  
    }  
}  
  
public interface Romantic {  
    public void sayLovePoem(String what);  
}
```

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {  
    public static void main(String[] args) {  
        Person manOnTrain = new Person();  
        manOnTrain.isInNiceMood = true;  
        LovingPerson womanOnTrain = new LovingPerson();  
        womanOnTrain.isInNiceMood = true;  
        womanOnTrain.sayLovePoem("aehm...");  
        manOnTrain.commentOnLovePoem("that's so beautiful!");  
        Person person = womanOnTrain;  
        person.commentOnLovePoem("you like it! :-)");  
        Person someBystander = new Person();  
        someBystander.commentOnLovePoem(" how cute!");  
    }  
}  
  
public class LovingPerson extends Person implements Romantic{  
    public void sayLovePoem(String what){  
        System.out.println(what + "ene mene muh: I'm in love with you!");  
    }  
}
```

```
public void commentOnLovePoem(String what){  
    System.out.print("no matter what: ");  
    this.commentOnLovePoem();  
}  
  
public void commentOnLovePoem(){  
    System.out.println("I am in love");  
}
```

```
public class Person {  
    public boolean isInNiceMood = false;  
    public void commentOnLovePoem(String what){  
        if(isInNiceMood)  
            System.out.println("oiii :-) " + what);  
        else  
            System.out.println("oerks! :-(");  
    }  
}  
  
public interface Romantic {  
    public void sayLovePoem(String what);  
}
```



Welche Ausgabe produziert NotoriousRomanticsDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {
    public static void main(String[] args) {
        Person manOnTrain = new Person();
        manOnTrain.isInNiceMood = true;
        LovingPerson womanOnTrain = new LovingPerson();
        womanOnTrain.isInNiceMood = true;
        womanOnTrain.sayLovePoem("aehm..."); 
        manOnTrain.commentOnLovePoem("that's so beautiful!");
        Person person = womanOnTrain;
        person.commentOnLovePoem("you like it! :-)");
        Person someBystander = new Person();
        someBystander.commentOnLovePoem(" how cute!");
    }
}

public class LovingPerson extends Person implements Romantic{
    public void sayLovePoem(String what){
        System.out.println(what + "ene mene muh: I'm in love with you!");
    }

    public void commentOnLovePoem(String what){
        System.out.print("no matter what: ");
        this.commentOnLovePoem();
    }

    public void commentOnLovePoem(){
        System.out.println("I am in love");
    }
}

public class Person {
    public boolean isInNiceMood = false;
    public void commentOnLovePoem(String what){
        if(isInNiceMood)
            System.out.println("oiii :-) " + what);
        else
            System.out.println("oerks! :-(");
    }
}

public interface Romantic {
    public void sayLovePoem(String what);
}
```



Welche Ausgabe produziert NotoriousRomanticsDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {
    public static void main(String[] args) {
        Person manOnTrain = new Person();
        manOnTrain.isInNiceMood = true;
        LovingPerson womanOnTrain = new LovingPerson();
        womanOnTrain.isInNiceMood = true;
        womanOnTrain.sayLovePoem("aehm..."); 
        manOnTrain.commentOnLovePoem("that's so beautiful!");
        Person person = womanOnTrain;
        person.commentOnLovePoem("you like it! :-)");
        Person someBystander = new Person();
        someBystander.commentOnLovePoem(" how cute!");
    }
}

public class LovingPerson extends Person implements Romantic{
    public void sayLovePoem(String what){
        System.out.println(what + "ene mene muh: I'm in love with you!");
    }

    public void commentOnLovePoem(String what){
        System.out.print("no matter what: ");
        this.commentOnLovePoem();
    }

    public void commentOnLovePoem(){
        System.out.println("I am in love");
    }
}

public class Person {
    public boolean isInNiceMood = false;
    public void commentOnLovePoem(String what){
        if(isInNiceMood)
            System.out.println("oiii :-) " + what);
        else
            System.out.println("oerks! :-(");
    }
}

public interface Romantic {
    public void sayLovePoem(String what);
}
```



Welche Ausgabe produziert NotoriousRomanticsDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {
    public static void main(String[] args) {
        Person manOnTrain = new Person();
        manOnTrain.isInNiceMood = true;
        LovingPerson womanOnTrain = new LovingPerson();
        womanOnTrain.isInNiceMood = true;
        womanOnTrain.sayLovePoem("aehm..."); 
        manOnTrain.commentOnLovePoem("that's so beautiful!");
        Person person = womanOnTrain;
        person.commentOnLovePoem("you like it! :-)");
        Person someBystander = new Person();
        someBystander.commentOnLovePoem(" how cute!");
    }
}

public class LovingPerson extends Person implements Romantic{
    public void sayLovePoem(String what){
        System.out.println(what + "ene mene muh: I'm in love with you!");
    }

    public void commentOnLovePoem(String what){
        System.out.print("no matter what: ");
        this.commentOnLovePoem();
    }

    public void commentOnLovePoem(){
        System.out.println("I am in love");
    }
}

public class Person {
    public boolean isInNiceMood = false;
    public void commentOnLovePoem(String what){
        if(isInNiceMood)
            System.out.println("oiii :-) " + what);
        else
            System.out.println("oerks! :-(");
    }
}

public interface Romantic {
    public void sayLovePoem(String what);
}
```



Welche Ausgabe produziert NotoriousRomanticsDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class NotoriousRomanticsDemo {
    public static void main(String[] args) {
        Person manOnTrain = new Person();
        manOnTrain.isInNiceMood = true;
        LovingPerson womanOnTrain = new LovingPerson();
        womanOnTrain.isInNiceMood = true;
        womanOnTrain.sayLovePoem("aehm..."); 
        manOnTrain.commentOnLovePoem("that's so beautiful!");
        Person person = womanOnTrain;
        person.commentOnLovePoem("you like it! :-)");
        Person someBystander = new Person();
        someBystander.commentOnLovePoem(" how cute!");
    }
}

public class LovingPerson extends Person implements Romantic{
    public void sayLovePoem(String what){
        System.out.println(what + "ene mene muh: I'm in love with you!");
    }

    public void commentOnLovePoem(String what){
        System.out.print("no matter what: ");
        this.commentOnLovePoem();
    }

    public void commentOnLovePoem(){
        System.out.println("I am in love");
    }
}

public class Person {
    public boolean isInNiceMood = false;
    public void commentOnLovePoem(String what){
        if(isInNiceMood)
            System.out.println("oiii :-) " + what);
        else
            System.out.println("oerks! :-(");
    }
}

public interface Romantic {
    public void sayLovePoem(String what);
}
```

ahem...ene mene muh: I'm in love with you!  
oiii :-) that's so beautiful!  
no matter what: I am in love  
oerks! :-)

**Overloading:**

ja, denn es gibt die Methode `commentOnLovePoem` in der Klasse `LovingPerson` zwei mal, jeweils mit verschiedenen Signaturen.

**Overriding:**

ja, denn die Methode  
`public void commentOnLovePoem(String what)`  
der Klasse `Person` wird in der Klasse `LovingPerson` überschrieben.

**Polymorphie**

ja, denn das Objekt `womanOnTrain` wird einmal als `Person` und ein anderes Mal als `LovingPerson` angesprochen. Beim Aufruf  
`person.commentOnLovePoem(...)` wird die speziellere Methode der Unterklasse `LovingPerson` ausgeführt.

Welche Ausgabe produziert `FinancialCrisisDemo`?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
**BEGRÜNDEN** Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {
    public static void main(String[] args) {
        MinisterOfFinance schaeuble = new MinisterOfFinance();
        MinisterOfFinance varoufakis = new MinisterOfFinance();
        schaeuble.commentOnCrisis();
        GovernmentMember merkel = new GovernmentMember();
        merkel.commentOnCrisis(" aehhh....");
        GovernmentMember someOtherMember = schaeuble;
        someOtherMember.commentOnCrisis(" aehhh....");
        varoufakis.commentOnCrisis("whatever....i need more money!");
    }
}

public class MinisterOfFinance extends GovernmentMember implements StateRepresentant{
    public void commentOnCrisis(){
        System.out.println("no comment at the moment");
    }

    public void commentOnCrisis(String what){
        System.out.println(what);
    }
}

public class GovernmentMember {
    public void commentOnCrisis(String what){
        System.out.println("i must say that " + what);
    }
}

public interface StateRepresentant {
    public void commentOnCrisis();
}
```

155

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {
    public static void main(String[] args) {
        MinisterOfFinance schaeuble = new MinisterOfFinance();
        MinisterOfFinance varoufakis = new MinisterOfFinance();
        schaeuble.commentOnCrisis();
        GovernmentMember merkel = new GovernmentMember();
        merkel.commentOnCrisis(" aehhh....");
        GovernmentMember someOtherMember = schaeuble;
        someOtherMember.commentOnCrisis(" aehhh....");
        varoufakis.commentOnCrisis("whatever....i need more money!");
    }
}

public class MinisterOfFinance extends GovernmentMember implements StateRepresentant{
    public void commentOnCrisis(){
        System.out.println("no comment at the moment");
    }

    public void commentOnCrisis(String what){
        System.out.println(what);
    }
}

public class GovernmentMember {
    public void commentOnCrisis(String what){
        System.out.println("i must say that " + what);
    }
}

public interface StateRepresentant {
    public void commentOnCrisis();
}
```

Welche Ausgabe produziert `FinancialCrisisDemo`?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
**BEGRÜNDEN** Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {
    public static void main(String[] args) {
        MinisterOfFinance schaeuble = new MinisterOfFinance();
        MinisterOfFinance varoufakis = new MinisterOfFinance();
        schaeuble.commentOnCrisis();
        GovernmentMember merkel = new GovernmentMember();
        merkel.commentOnCrisis(" aehhh....");
        GovernmentMember someOtherMember = schaeuble;
        someOtherMember.commentOnCrisis(" aehhh....");
        varoufakis.commentOnCrisis("whatever....i need more money!");
    }
}

public class MinisterOfFinance extends GovernmentMember implements StateRepresentant{
    public void commentOnCrisis(){
        System.out.println("no comment at the moment");
    }

    public void commentOnCrisis(String what){
        System.out.println(what);
    }
}

public class GovernmentMember {
    public void commentOnCrisis(String what){
        System.out.println("i must say that " + what);
    }
}

public interface StateRepresentant {
    public void commentOnCrisis();
}
```

155

## Aufgabe

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {  
  
    public static void main(String[] args) {  
        MinisterOfFinance schaeuble = new MinisterOfFinance();  
        MinisterOfFinance varoufakis = new MinisterOfFinance();  
        schaeuble.commentOnCrisis();  
        GovernmentMember merkel = new GovernmentMember();  
        merkel.commentOnCrisis(" aehhh...");  
        GovernmentMember someOtherMember = schaeuble;  
        someOtherMember.commentOnCrisis(" aehhh...");  
        varoufakis.commentOnCrisis("whatever....i need more money!");  
    }  
  
    public class MinisterOfFinance extends GovernmentMember implements  
    StateRepresentant{  
        public void commentOnCrisis(){  
            System.out.println("no comment at the moment");  
        }  
  
        public void commentOnCrisis(String what){  
            System.out.println(what);  
        }  
    }  
  
    public class GovernmentMember {  
        public void commentOnCrisis(String what){  
            System.out.println("i must say that " + what);  
        }  
    }  
  
    public interface StateRepresentant {  
        public void commentOnCrisis();  
    }  
}
```

Welche Ausgabe produziert FinancialCrisisDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

## Aufgabe

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {  
  
    public static void main(String[] args) {  
        MinisterOfFinance schaeuble = new MinisterOfFinance();  
        MinisterOfFinance varoufakis = new MinisterOfFinance();  
        schaeuble.commentOnCrisis();  
        GovernmentMember merkel = new GovernmentMember();  
        merkel.commentOnCrisis(" aehhh...");  
        GovernmentMember someOtherMember = schaeuble;  
        someOtherMember.commentOnCrisis(" aehhh...");  
        varoufakis.commentOnCrisis("whatever....i need more money!");  
    }  
  
    public class MinisterOfFinance extends GovernmentMember implements  
    StateRepresentant{  
        public void commentOnCrisis(){  
            System.out.println("no comment at the moment");  
        }  
  
        public void commentOnCrisis(String what){  
            System.out.println(what);  
        }  
    }  
  
    public class GovernmentMember {  
        public void commentOnCrisis(String what){  
            System.out.println("i must say that " + what);  
        }  
    }  
  
    public interface StateRepresentant {  
        public void commentOnCrisis();  
    }  
}
```

Welche Ausgabe produziert FinancialCrisisDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

## Aufgabe

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {  
  
    public static void main(String[] args) {  
        MinisterOfFinance schaeuble = new MinisterOfFinance();  
        MinisterOfFinance varoufakis = new MinisterOfFinance();  
        schaeuble.commentOnCrisis();  
        GovernmentMember merkel = new GovernmentMember();  
        merkel.commentOnCrisis(" aehhh...");  
        GovernmentMember someOtherMember = schaeuble;  
        someOtherMember.commentOnCrisis(" aehhh...");  
        varoufakis.commentOnCrisis("whatever....i need more money!");  
    }  
  
    public class MinisterOfFinance extends GovernmentMember implements  
    StateRepresentant{  
        public void commentOnCrisis(){  
            System.out.println("no comment at the moment");  
        }  
  
        public void commentOnCrisis(String what){  
            System.out.println(what);  
        }  
    }  
  
    public class GovernmentMember {  
        public void commentOnCrisis(String what){  
            System.out.println("i must say that " + what);  
        }  
    }  
  
    public interface StateRepresentant {  
        public void commentOnCrisis();  
    }  
}
```

Welche Ausgabe produziert FinancialCrisisDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

## Aufgabe

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {  
  
    public static void main(String[] args) {  
        MinisterOfFinance schaeuble = new MinisterOfFinance();  
        MinisterOfFinance varoufakis = new MinisterOfFinance();  
        schaeuble.commentOnCrisis();  
        GovernmentMember merkel = new GovernmentMember();  
        merkel.commentOnCrisis(" aehhh...");  
        GovernmentMember someOtherMember = schaeuble;  
        someOtherMember.commentOnCrisis(" aehhh...");  
        varoufakis.commentOnCrisis("whatever....i need more money!");  
    }  
  
    public class MinisterOfFinance extends GovernmentMember implements  
    StateRepresentant{  
        public void commentOnCrisis(){  
            System.out.println("no comment at the moment");  
        }  
  
        public void commentOnCrisis(String what){  
            System.out.println(what);  
        }  
    }  
  
    public class GovernmentMember {  
        public void commentOnCrisis(String what){  
            System.out.println("i must say that " + what);  
        }  
    }  
  
    public interface StateRepresentant {  
        public void commentOnCrisis();  
    }  
}
```

Welche Ausgabe produziert FinancialCrisisDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

## Aufgabe

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {  
  
    public static void main(String[] args) {  
        MinisterOfFinance schaeuble = new MinisterOfFinance();  
        MinisterOfFinance varoufakis = new MinisterOfFinance();  
        schaeuble.commentOnCrisis();  
        GovernmentMember merkel = new GovernmentMember();  
        merkel.commentOnCrisis(" aehhh...");  
        GovernmentMember someOtherMember = schaeuble;  
        someOtherMember.commentOnCrisis(" aehhh...");  
        varoufakis.commentOnCrisis("whatever....i need more money!");  
    }  
  
    public class MinisterOfFinance extends GovernmentMember implements  
    StateRepresentant{  
        public void commentOnCrisis(){  
            System.out.println("no comment at the moment");  
        }  
  
        public void commentOnCrisis(String what){  
            System.out.println(what);  
        }  
    }  
  
    public class GovernmentMember {  
        public void commentOnCrisis(String what){  
            System.out.println("i must say that " + what);  
        }  
    }  
  
    public interface StateRepresentant {  
        public void commentOnCrisis();  
    }  
}
```

Welche Ausgabe produziert FinancialCrisisDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {  
  
    public static void main(String[] args) {  
        MinisterOfFinance schaeuble = new MinisterOfFinance();  
        MinisterOfFinance varoufakis = new MinisterOfFinance();  
        schaeuble.commentOnCrisis();  
        GovernmentMember merkel = new GovernmentMember();  
        merkel.commentOnCrisis(" aehhh...");  
        GovernmentMember someOtherMember = schaeuble;   
        someOtherMember.commentOnCrisis(" aehhh...");  
        varoufakis.commentOnCrisis("whatever....i need more money!");  
    }  
  
    public class MinisterOfFinance extends GovernmentMember implements  
    StateRepresentant{  
        public void commentOnCrisis(){  
            System.out.println("no comment at the moment");  
        }  
  
        public void commentOnCrisis(String what){  
            System.out.println(what);  
        }  
    }  
  
    public class GovernmentMember {  
        public void commentOnCrisis(String what){  
            System.out.println("i must say that " + what);  
        }  
    }  
  
    public interface StateRepresentant {  
        public void commentOnCrisis();  
    }  
}
```

Welche Ausgabe produziert FinancialCrisisDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

## Aufgabe

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {  
  
    public static void main(String[] args) {  
        MinisterOfFinance schaeuble = new MinisterOfFinance();  
        MinisterOfFinance varoufakis = new MinisterOfFinance();  
        schaeuble.commentOnCrisis();  
        GovernmentMember merkel = new GovernmentMember();  
        merkel.commentOnCrisis(" aehhh...");  
        GovernmentMember someOtherMember = schaeuble;  
        someOtherMember.commentOnCrisis(" aehhh...");  
        varoufakis.commentOnCrisis("whatever....i need more money!");  
    }  
  
    public class MinisterOfFinance extends GovernmentMember implements  
    StateRepresentant{  
        public void commentOnCrisis(){  
            System.out.println("no comment at the moment");  
        }  
  
        public void commentOnCrisis(String what){  
            System.out.println(what);  
        }  
    }  
  
    public class GovernmentMember {  
        public void commentOnCrisis(String what){  
            System.out.println("i must say that " + what);  
        }  
    }  
  
    public interface StateRepresentant {  
        public void commentOnCrisis();  
    }  
}
```

Welche Ausgabe produziert FinancialCrisisDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {  
  
    public static void main(String[] args) {  
        MinisterOfFinance schaeuble = new MinisterOfFinance();  
        MinisterOfFinance varoufakis = new MinisterOfFinance();  
        schaeuble.commentOnCrisis();  
        GovernmentMember merkel = new GovernmentMember();  
        merkel.commentOnCrisis(" aehhh...");  
        GovernmentMember someOtherMember = schaeuble;  
        someOtherMember.commentOnCrisis(" aehhh...");  
        varoufakis.commentOnCrisis("whatever....i need more money!");  
    }  
  
    public class MinisterOfFinance extends GovernmentMember implements  
    StateRepresentant{  
        public void commentOnCrisis(){  
            System.out.println("no comment at the moment");  
        }  
  
        public void commentOnCrisis(String what){  
            System.out.println(what);  
        }  
    }  
  
    public class GovernmentMember {  
        public void commentOnCrisis(String what){  
            System.out.println("i must say that " + what);  
        }  
    }  
  
    public interface StateRepresentant {  
        public void commentOnCrisis();  
    }  
}
```

Welche Ausgabe produziert FinancialCrisisDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

155

155

155

## Aufgabe

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {
    public static void main(String[] args) {
        MinisterOfFinance schaeuble = new MinisterOfFinance();
        MinisterOfFinance varoufakis = new MinisterOfFinance();
        schaeuble.commentOnCrisis();
        GovernmentMember merkel = new GovernmentMember();
        merkel.commentOnCrisis(" aehhh... ");
        GovernmentMember someOtherMember = schaeuble;
        someOtherMember.commentOnCrisis(" aehhh... ");
        varoufakis.commentOnCrisis("whatever....i need more money! ");
    }
}

public class MinisterOfFinance extends GovernmentMember implements StateRepresentant{
    public void commentOnCrisis(){
        System.out.println("no comment at the moment");
    }

    public void commentOnCrisis(String what){
        System.out.println(what);
    }
}

public class GovernmentMember {
    public void commentOnCrisis(String what){
        System.out.println("i must say that " + what);
    }
}

public interface StateRepresentant {
    public void commentOnCrisis();
}
```

Welche Ausgabe produziert FinancialCrisisDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

## Aufgabe

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {
    public static void main(String[] args) {
        MinisterOfFinance schaeuble = new MinisterOfFinance();
        MinisterOfFinance varoufakis = new MinisterOfFinance();
        schaeuble.commentOnCrisis();
        GovernmentMember merkel = new GovernmentMember();
        merkel.commentOnCrisis(" aehhh... ");
        GovernmentMember someOtherMember = schaeuble;
        someOtherMember.commentOnCrisis(" aehhh... ");
        varoufakis.commentOnCrisis("whatever....i need more money! ");
    }
}

public class MinisterOfFinance extends GovernmentMember implements StateRepresentant{
    public void commentOnCrisis(){
        System.out.println("no comment at the moment");
    }

    public void commentOnCrisis(String what){
        System.out.println(what);
    }
}

public class GovernmentMember {
    public void commentOnCrisis(String what){
        System.out.println("i must say that " + what);
    }
}

public interface StateRepresentant {
    public void commentOnCrisis();
}
```

Welche Ausgabe produziert FinancialCrisisDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

155

155

## Aufgabe

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {
    public static void main(String[] args) {
        MinisterOfFinance schaeuble = new MinisterOfFinance();
        MinisterOfFinance varoufakis = new MinisterOfFinance();
        schaeuble.commentOnCrisis();
        GovernmentMember merkel = new GovernmentMember();
        merkel.commentOnCrisis(" aehhh... ");
        GovernmentMember someOtherMember = schaeuble;
        someOtherMember.commentOnCrisis(" aehhh... ");
        varoufakis.commentOnCrisis("whatever....i need more money! ");
    }
}

public class MinisterOfFinance extends GovernmentMember implements StateRepresentant{
    public void commentOnCrisis(){
        System.out.println("no comment at the moment");
    }

    public void commentOnCrisis(String what){
        System.out.println(what);
    }
}

public class GovernmentMember {
    public void commentOnCrisis(String what){
        System.out.println("i must say that " + what);
    }
}

public interface StateRepresentant {
    public void commentOnCrisis();
}
```

Welche Ausgabe produziert FinancialCrisisDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

## Aufgabe

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {
    public static void main(String[] args) {
        MinisterOfFinance schaeuble = new MinisterOfFinance();
        MinisterOfFinance varoufakis = new MinisterOfFinance();
        schaeuble.commentOnCrisis();
        GovernmentMember merkel = new GovernmentMember();
        merkel.commentOnCrisis(" aehhh... ");
        GovernmentMember someOtherMember = schaeuble;
        someOtherMember.commentOnCrisis(" aehhh... ");
        varoufakis.commentOnCrisis("whatever....i need more money! ");
    }
}

public class MinisterOfFinance extends GovernmentMember implements StateRepresentant{
    public void commentOnCrisis(){
        System.out.println("no comment at the moment");
    }

    public void commentOnCrisis(String what){
        System.out.println(what);
    }
}

public class GovernmentMember {
    public void commentOnCrisis(String what){
        System.out.println("i must say that " + what);
    }
}

public interface StateRepresentant {
    public void commentOnCrisis();
}
```

Welche Ausgabe produziert FinancialCrisisDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!

155

155

# Aufgabe

Gegeben sei folgender Java Code:

```

public class FinancialCrisisDemo {
    public static void main(String[] args) {
        MinisterOfFinance schaeuble = new MinisterOfFinance();
        MinisterOfFinance varoufakis = new MinisterOfFinance();
        schaeuble.commentOnCrisis();
        GovernmentMember merkel = new GovernmentMember();
        merkel.commentOnCrisis(" aehhh... ");
        GovernmentMember someOtherMember = schaeuble;
        someOtherMember.commentOnCrisis(" aehhh... ");
        varoufakis.commentOnCrisis("whatever....i need more money! ");
    }
}

public class MinisterOfFinance extends GovernmentMember implements
StateRepresentant{
    public void commentOnCrisis(){
        System.out.println("no comment at the moment");
    }

    public void commentOnCrisis(String what){
        System.out.println(what);
    }
}

public class GovernmentMember {
    public void commentOnCrisis(String what){
        System.out.println("i must say that " + what);
    }
}

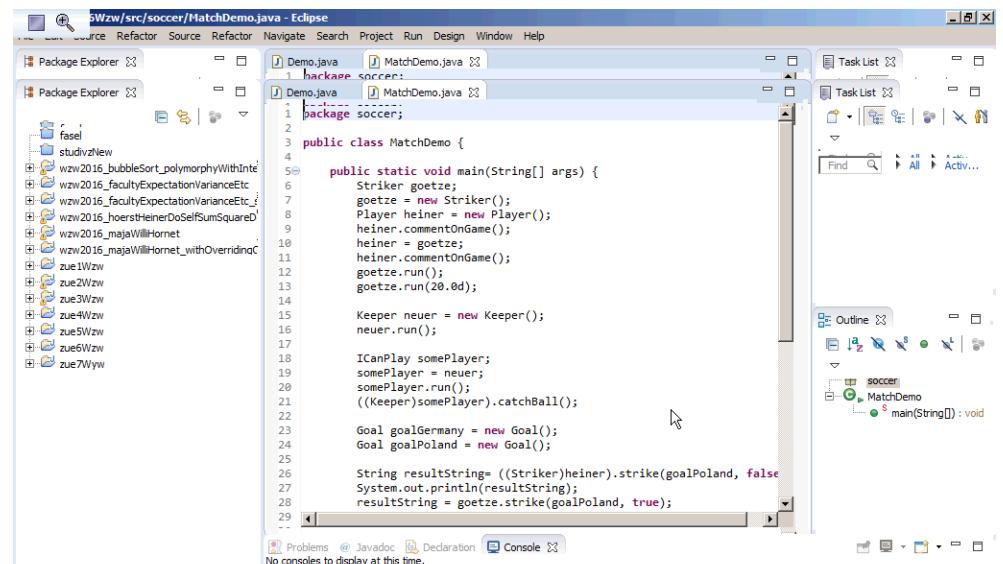
public interface StateRepresentant {
    public void commentOnCrisis();
}

```

155

Welche Ausgabe produziert FinancialCrisisDemo?

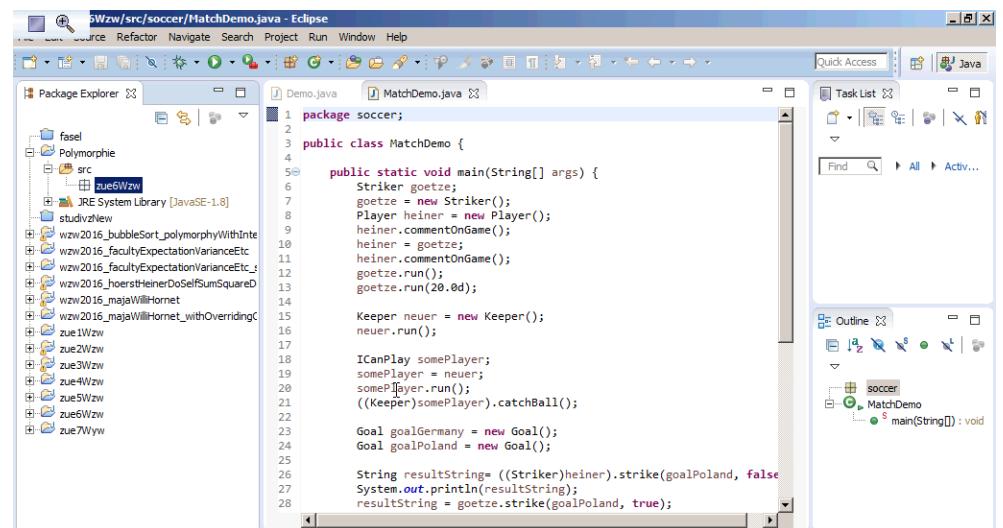
Welche der drei Konzepte Overloading, Overriding, Polymorphismus werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!



```

6Wzw/src/soccer/MatchDemo.java - Eclipse
Source Refactor Search Project Run Window Help
Package Explorer
Demo.java MatchDemo.java
1 package soccer;
2
3 public class MatchDemo {
4
5     public static void main(String[] args) {
6         Striker goetze;
7         goetze = new Striker();
8         Player heiner = new Player();
9         heiner.commentOnGame();
10        heiner = goetze;
11        heiner.commentOnGame();
12        goetze.run();
13        goetze.run(20.0d);
14
15        Keeper neuer = new Keeper();
16        neuer.run();
17
18        ICanPlay somePlayer;
19        somePlayer = neuer;
20        somePlayer.run();
21        ((Keeper)somePlayer).catchBall();
22
23        Goal goalGermany = new Goal();
24        Goal goalPoland = new Goal();
25
26        String resultString= ((Striker)heiner).strike(goalPoland, false);
27        System.out.println(resultString);
28        resultString = goetze.strike(goalPoland, true);
29

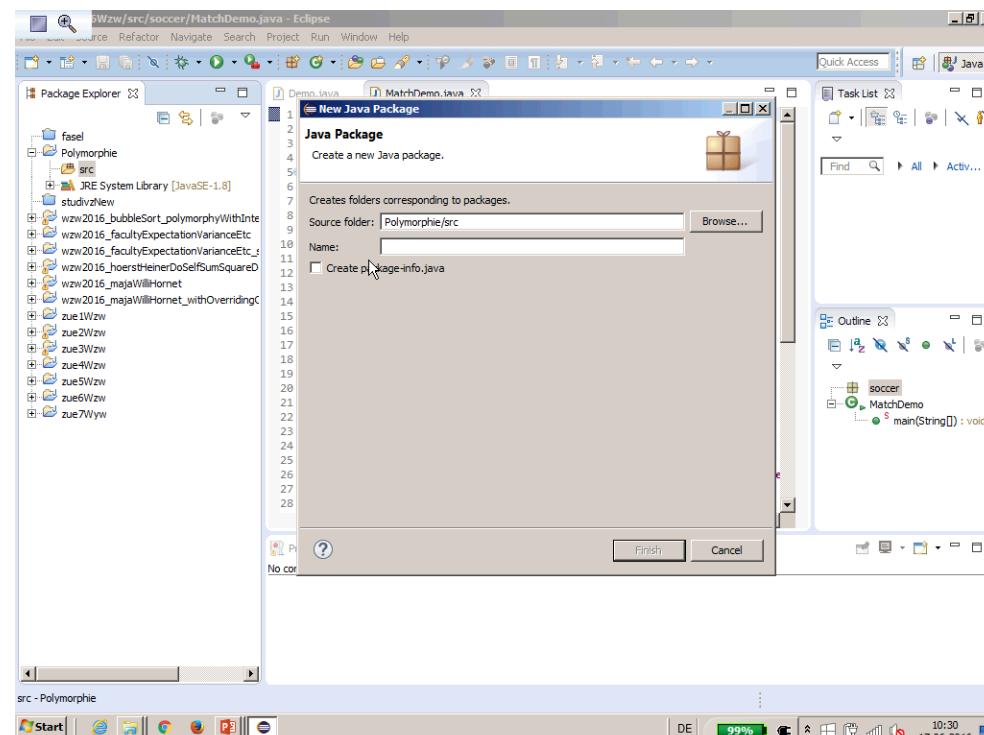
```

```

6Wzw/src/soccer/MatchDemo.java - Eclipse
Source Refactor Search Project Run Window Help
Package Explorer
Demo.java MatchDemo.java
1 package soccer;
2
3 public class MatchDemo {
4
5     public static void main(String[] args) {
6         Striker goetze;
7         goetze = new Striker();
8         Player heiner = new Player();
9         heiner.commentOnGame();
10        heiner = goetze;
11        heiner.commentOnGame();
12        goetze.run();
13        goetze.run(20.0d);
14
15        Keeper neuer = new Keeper();
16        neuer.run();
17
18        ICanPlay somePlayer;
19        somePlayer = neuer;
20        somePlayer.run();
21        ((Keeper)somePlayer).catchBall();
22
23        Goal goalGermany = new Goal();
24        Goal goalPoland = new Goal();
25
26        String resultString= ((Striker)heiner).strike(goalPoland, false);
27        System.out.println(resultString);
28        resultString = goetze.strike(goalPoland, true);
29

```

New Java Package

Java Package

Create a new Java package.

Creates folders corresponding to packages.

Source folder: Polymorphe/src

Name: package-info.java

Create package-info.java

Finish Cancel



Ausgabe

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {
    public static void main(String[] args) {
        MinisterOfFinance schaeuble = new MinisterOfFinance();
        MinisterOfFinance varoufakis = new MinisterOfFinance();
        schaeuble.commentOnCrisis();
        GovernmentMember merkel = new GovernmentMember();
        merkel.commentOnCrisis(" aehhh... ");
        GovernmentMember someOtherMember = schaeuble;
        someOtherMember.commentOnCrisis(" aehhh... ");
        varoufakis.commentOnCrisis("whatever....i need more money! ");
    }
}

public class MinisterOfFinance extends GovernmentMember implements
StateRepresentant{
    public void commentOnCrisis(){
        System.out.println("no comment at the moment");
    }

    public void commentOnCrisis(String what){
        System.out.println(what);
    }
}

public class GovernmentMember {
    public void commentOnCrisis(String what){
        System.out.println("i must say that " + what);
    }
}

public interface StateRepresentant {
    public void commentOnCrisis();
}
```

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {
    public static void main(String[] args) {
        MinisterOfFinance schaeuble = new MinisterOfFinance();
        MinisterOfFinance varoufakis = new MinisterOfFinance();
        schaeuble.commentOnCrisis();
        GovernmentMember merkel = new GovernmentMember();
        merkel.commentOnCrisis(" aehhh... ");
        GovernmentMember someOtherMember = schaeuble;
        someOtherMember.commentOnCrisis(" aehhh... ");
        varoufakis.commentOnCrisis("whatever....i need more money! ");
    }
}

public class MinisterOfFinance extends GovernmentMember implements
StateRepresentant{
    public void commentOnCrisis(){
        System.out.println("no comment at the moment");
    }

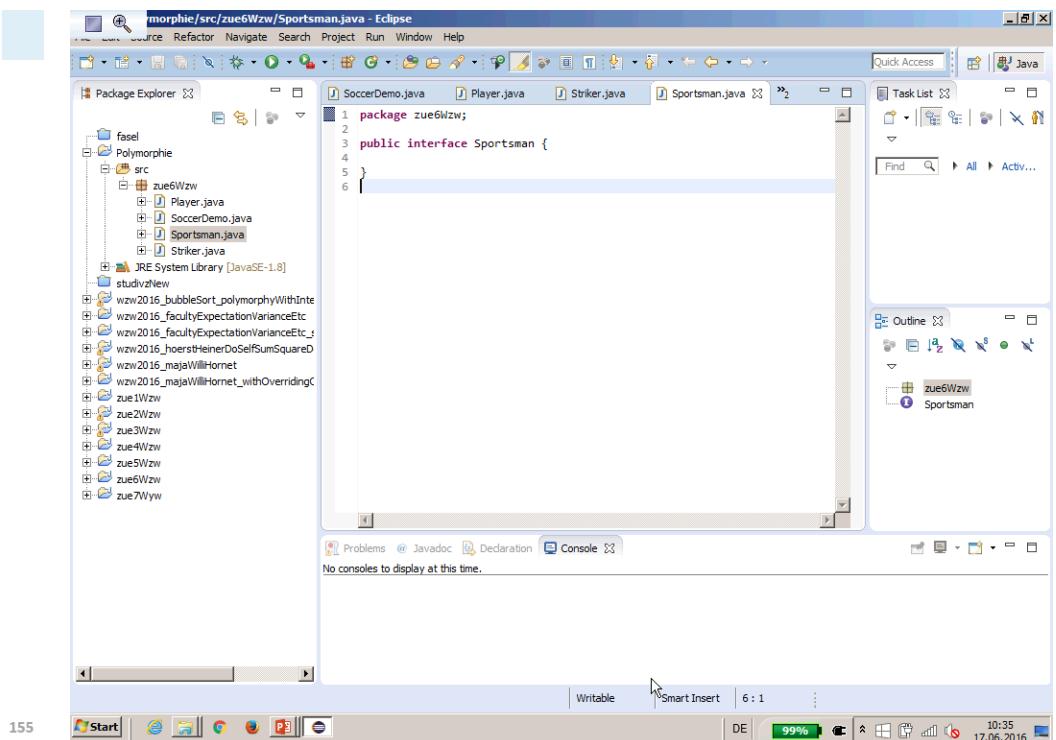
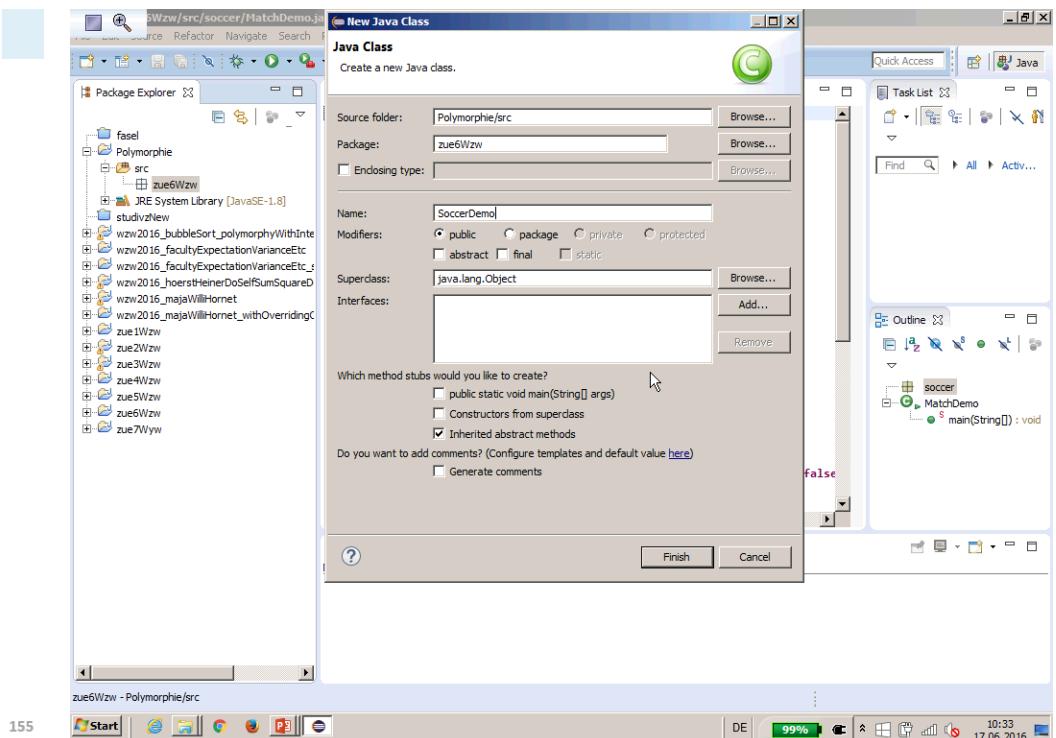
    public void commentOnCrisis(String what){
        System.out.println(what);
    }
}

public class GovernmentMember {
    public void commentOnCrisis(String what){
        System.out.println("i must say that " + what);
    }
}

public interface StateRepresentant {
    public void commentOnCrisis();
}
```

Welche Ausgabe produziert FinancialCrisisDemo?

Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!



Eclipse IDE screenshot showing the Java code for the Sportsman interface:

```
package zue6Wzw;
public interface Sportsman {
    public void doSports();
}
```

The code editor shows the declaration of the `Sportsman` interface with a single method `doSports()`. The Outline view on the right shows the declaration of the `doSports()` method.

Eclipse IDE screenshot showing the Java code for the Sportsman interface:

```
package zue6Wzw;
public interface Sportsman {
    public void doSports();
}
```

The code editor shows the declaration of the `Sportsman` interface with a single method `doSports()`. The Outline view on the right shows the declaration of the `doSports()` method.

Eclipse IDE screenshot showing the Java code for the Sportsman interface:

```
package zue6Wzw;
public interface Sportsman {
    public void doSports();
}
```

The code editor shows the declaration of the `Sportsman` interface with a single method `doSports()`. The Outline view on the right shows the declaration of the `doSports()` method.

Eclipse IDE screenshot showing the Java code for the Sportsman interface:

```
package zue6Wzw;
public interface Sportsman {
    Public void doSports();
}
```

The code editor shows the declaration of the `Sportsman` interface with a single method `doSports()`. The method name is misspelled as `Public void doSports();`. The Outline view on the right shows the declaration of the `doSports()` method.

# Aufgabe

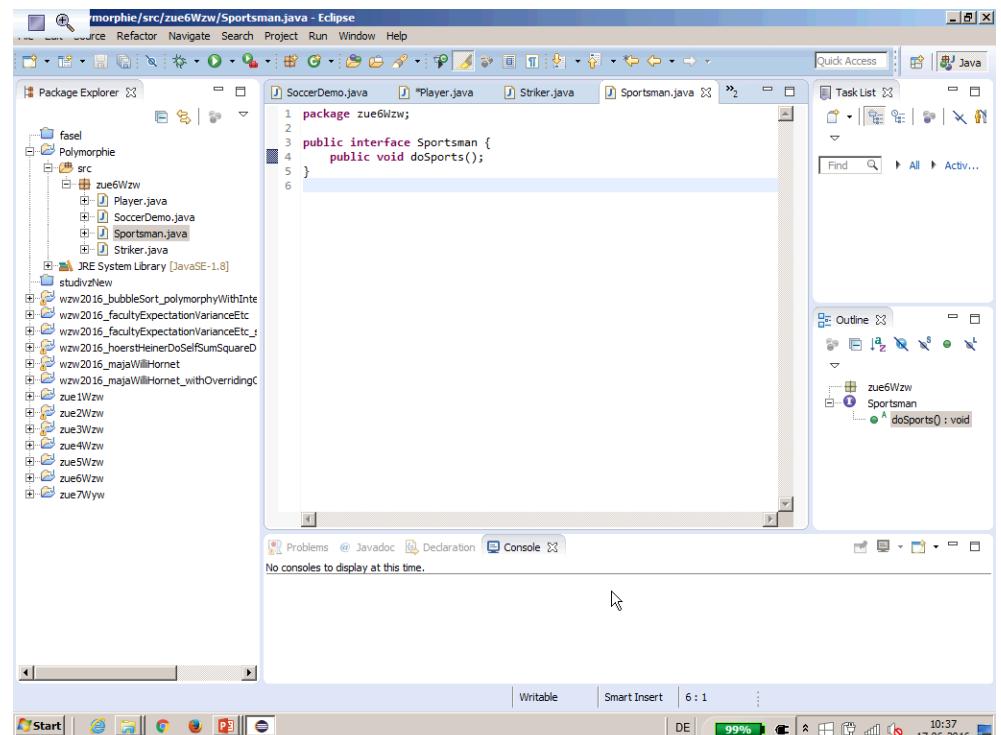
Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {  
    public static void main(String[] args) {  
        MinisterOfFinance schaeuble = new MinisterOfFinance();  
        MinisterOfFinance varoufakis = new MinisterOfFinance();  
        schaeuble.commentOnCrisis();  
        GovernmentMember merkel = new GovernmentMember();  
        merkel.commentOnCrisis(" aehhh...");  
        GovernmentMember someOtherMember = schaeuble;  
        someOtherMember.commentOnCrisis(" aehhh...");  
        varoufakis.commentOnCrisis("whatever....i need more money!");  
    }  
  
    public class MinisterOfFinance extends GovernmentMember implements  
    StateRepresentant{  
        public void commentOnCrisis(){  
            System.out.println("no comment at the moment");  
        }  
  
        public void commentOnCrisis(String what){  
            System.out.println(what);  
        }  
    }  
  
    public class GovernmentMember {  
        public void commentOnCrisis(String what){  
            System.out.println("i must say that " + what);  
        }  
    }  
  
    public interface StateRepresentant {  
        public void commentOnCrisis();  
    }  
}
```

155

Welche Ausgabe produziert FinancialCrisisDemo?

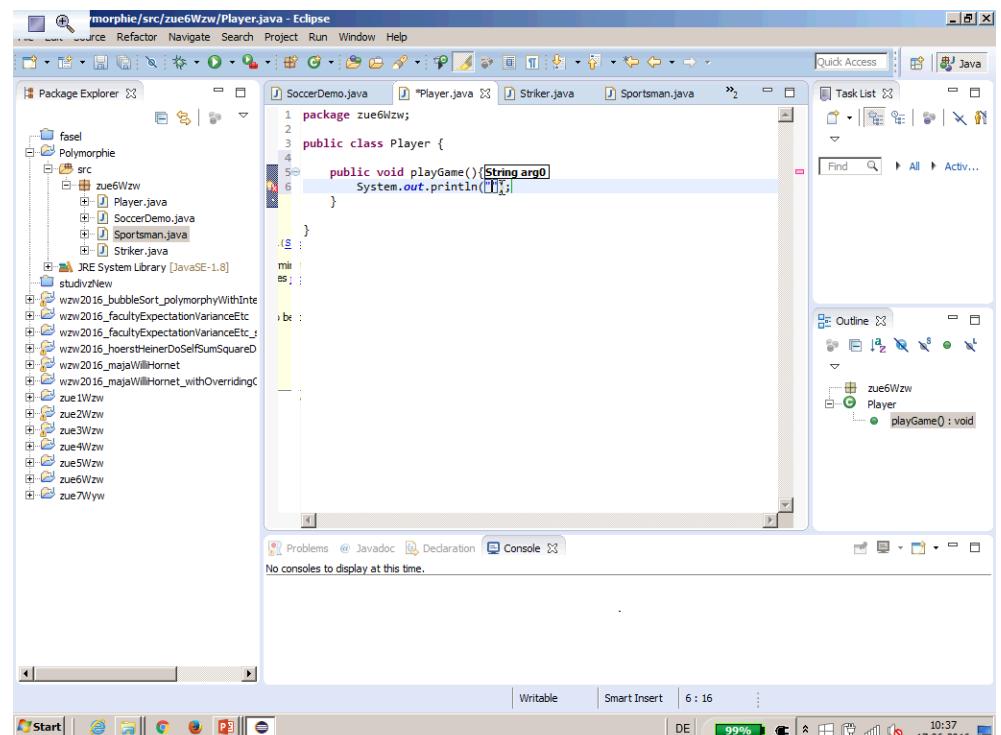
Welche der drei Konzepte Overloading, Overriding, Polymorphismus werden im obigen Code benutzt, welche nicht?  
BEGRÜNDEN Sie jeweils kurz!



The screenshot shows the Eclipse IDE interface. The package explorer on the left shows a project structure with packages like 'fasel' and 'Polymorphe'. The code editor on the right displays the file 'Sportsman.java' which contains a single method declaration: 'public void doSports();'. The status bar at the bottom indicates the file is 99% complete and was last modified on 17.06.2016 at 10:37.

```
package zue6Wzw;  
public interface Sportsman {  
    public void doSports();  
}
```

DE 99% 10:37 17.06.2016



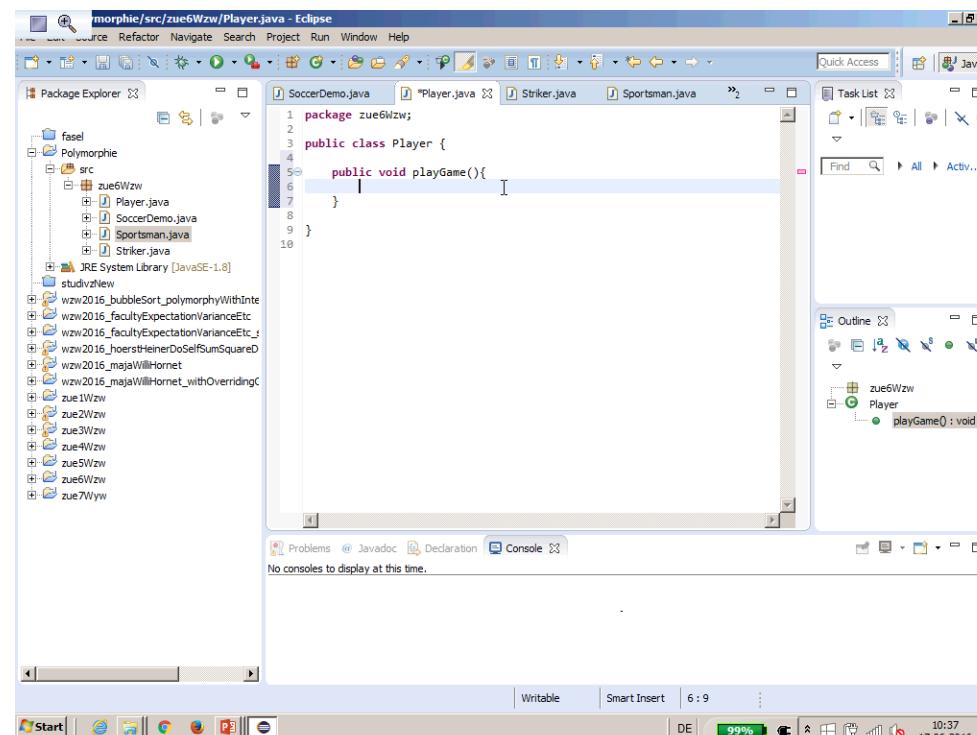
This screenshot shows the Eclipse IDE interface again. The package explorer on the left shows the same project structure. The code editor on the right displays the file 'Player.java' with the following code:

```
package zue6Wzw;  
public class Player {  
    public void playGame(){  
    }  
}
```

At the bottom of the code editor, there is a red squiggle under the word 'String' in the line 'System.out.println(String);'. The status bar at the bottom indicates the file is 6:9 and was last modified on 17.06.2016 at 10:37.

```
package zue6Wzw;  
public class Player {  
    public void playGame(){  
        System.out.println(String);  
    }  
}
```

DE 99% 10:37 17.06.2016



This screenshot shows the Eclipse IDE interface. The package explorer on the left shows the project structure. The code editor on the right displays the file 'Player.java' with the following code:

```
package zue6Wzw;  
public class Player {  
    public void playGame(){  
    }  
}
```

At the bottom of the code editor, there is a red squiggle under the word 'String' in the line 'System.out.println(String);'. The status bar at the bottom indicates the file is 6:9 and was last modified on 17.06.2016 at 10:37.

```
package zue6Wzw;  
public class Player {  
    public void playGame(){  
        System.out.println(String);  
    }  
}
```

DE 99% 10:37 17.06.2016

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows a Java project named "Polymorphe" containing packages like "fasel" and "zue6Wzw".
- Code Editor:** Displays the file "Player.java" with the code provided above.
- Outline View:** Shows the declaration "playGame(): void" under the "Player" class.
- Status Bar:** Shows the date and time as "17.06.2016" and battery level as "99%".

**Aufgabe**

**Welche Ausgabe produziert FinancialCrisisDemo?**

**Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht? BEGRÜNDEN Sie jeweils kurz!**

Gegeben sei folgender Java Code:

```
public class FinancialCrisisDemo {
    public static void main(String[] args) {
        MinisterOfFinance schaeuble = new MinisterOfFinance();
        MinisterOfFinance varoufakis = new MinisterOfFinance();
        schaeuble.commentOnCrisis();
        GovernmentMember merkel = new GovernmentMember();
        merkel.commentOnCrisis(" aehhh... ");
        GovernmentMember someOtherMember = schaeuble;
        someOtherMember.commentOnCrisis(" aehhh... ");
        varoufakis.commentOnCrisis("whatever....i need more money!");
    }
}

public class MinisterOfFinance extends GovernmentMember implements StateRepresentant{
    public void commentOnCrisis(){
        System.out.println("no comment at the moment");
    }

    public void commentOnCrisis(String what){
        System.out.println(what);
    }
}

public class GovernmentMember {
    public void commentOnCrisis(String what){
        System.out.println("i must say that " + what);
    }
}

public interface StateRepresentant {
    public void commentOnCrisis();
}
```

155

**Aufgabe**

**Gegeben sei folgender Java Code:**

```
public class FinancialCrisisDemo {
    public static void main(String[] args) {
        MinisterOfFinance schaeuble = new MinisterOfFinance();
        MinisterOfFinance varoufakis = new MinisterOfFinance();
        schaeuble.commentOnCrisis();
        GovernmentMember merkel = new GovernmentMember();
        merkel.commentOnCrisis(" aehhh... ");
        GovernmentMember someOtherMember = schaeuble;
        someOtherMember.commentOnCrisis(" aehhh... ");
        varoufakis.commentOnCrisis("whatever....i need more money!");
    }
}

public class MinisterOfFinance extends GovernmentMember implements StateRepresentant{
    public void commentOnCrisis(){
        System.out.println("no comment at the moment");
    }

    public void commentOnCrisis(String what){
        System.out.println(what);
    }
}

public class GovernmentMember {
    public void commentOnCrisis(String what){
        System.out.println("i must say that " + what);
    }
}

public interface StateRepresentant {
    public void commentOnCrisis();
}
```

**Welche Ausgabe produziert FinancialCrisisDemo?**

**Welche der drei Konzepte Overloading, Overriding, Polymorphie werden im obigen Code benutzt, welche nicht? BEGRÜNDEN Sie jeweils kurz!**

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows a Java project named "Polymorphe" containing packages like "fasel" and "zue6Wzw".
- Code Editor:** Displays the file "Player.java" with the code provided above.
- Outline View:** Shows the declaration "playGame(): void" under the "Player" class.
- Status Bar:** Shows the date and time as "17.06.2016" and battery level as "99%".

155

155

**morphe/src/zue6Wzw/Striker.java - Eclipse**

```
1 package zue6Wzw;
2
3 public class Striker extends Player implements Sportsman {
4
5 }
```

The type Striker must implement the inherited abstract method Sportsman.doSports()

2 quick fixes available:

- Add unimplemented methods
- Make type Striker abstract

Java

Package Explorer

- fasel
- Polymorphic
- src
- zue6Wzw
- Player.java
- SoccerDemo.java
- Sportsman.java
- Striker.java

JRE System Library [JavaSE-1.8]

studiview

wzw2016\_bubbleSort\_polymorphWithInte

wzw2016\_facultyExpectationVarianceEtc

wzw2016\_facultyExpectationVarianceEtc\_

wzw2016\_hoerstHeinerDoSelfSumSquareD

wzw2016\_majaWillHornet

wzw2016\_majaWillHornet\_withOverridingC

zue1Wzw

zue2Wzw

zue3Wzw

zue4Wzw

zue5Wzw

zue6Wzw

zue7Wyw

Task List

Find All Activ...

Outline

Problems Javadoc Declaration Console

No consoles to display at this time.

Writable Smart Insert 3 : 57

Start

DE 99% 10:40 17.06.2016

**morphe/src/zue6Wzw/Striker.java - Eclipse**

```
1 package zue6Wzw;
2
3 public class Striker extends Player implements Sportsman {
4
5     public void doSports() {
6         System.out.println("I play soccer!");
7     }
8
9 }
10
```

Java

Package Explorer

- fasel
- Polymorphic
- src
- zue6Wzw
- Player.java
- SoccerDemo.java
- Sportsman.java
- Striker.java

JRE System Library [JavaSE-1.8]

studiview

wzw2016\_bubbleSort\_polymorphWithInte

wzw2016\_facultyExpectationVarianceEtc

wzw2016\_facultyExpectationVarianceEtc\_

wzw2016\_hoerstHeinerDoSelfSumSquareD

wzw2016\_majaWillHornet

wzw2016\_majaWillHornet\_withOverridingC

zue1Wzw

zue2Wzw

zue3Wzw

zue4Wzw

zue5Wzw

zue6Wzw

zue7Wyw

Task List

Find All Activ...

Outline

Problems Javadoc Declaration Console

No consoles to display at this time.

Writable Smart Insert 6 : 12

Start

DE 99% 10:41 17.06.2016

**morphe/src/zue6Wzw/Striker.java - Eclipse**

```
1 package zue6Wzw;
2
3 public class Striker extends Player implements Sportsman {
4
5     public void doSports() {
6         System.out.println("I play soccer!");
7     }
8
9 }
10
```

Java

Package Explorer

- fasel
- Polymorphic
- src
- zue6Wzw
- Player.java
- SoccerDemo.java
- Sportsman.java
- Striker.java

JRE System Library [JavaSE-1.8]

studiview

wzw2016\_bubbleSort\_polymorphWithInte

wzw2016\_facultyExpectationVarianceEtc

wzw2016\_facultyExpectationVarianceEtc\_

wzw2016\_hoerstHeinerDoSelfSumSquareD

wzw2016\_majaWillHornet

wzw2016\_majaWillHornet\_withOverridingC

zue1Wzw

zue2Wzw

zue3Wzw

zue4Wzw

zue5Wzw

zue6Wzw

zue7Wyw

Task List

Find All Activ...

Outline

Problems Javadoc Declaration Console

No consoles to display at this time.

Writable Smart Insert 10 : 1

Start

DE 99% 10:41 17.06.2016

**morphe/src/zue6Wzw/SoccerDemo.java - Eclipse**

```
1 package zue6Wzw;
2
3 public class SoccerDemo {
4
5     Public static void main(String[] args) {
6
7     }
8
9 }
10
```

Java

Package Explorer

- fasel
- Polymorphic
- src
- zue6Wzw
- Player.java
- SoccerDemo.java
- Sportsman.java
- Striker.java

JRE System Library [JavaSE-1.8]

studiview

wzw2016\_bubbleSort\_polymorphWithInte

wzw2016\_facultyExpectationVarianceEtc

wzw2016\_facultyExpectationVarianceEtc\_

wzw2016\_hoerstHeinerDoSelfSumSquareD

wzw2016\_majaWillHornet

wzw2016\_majaWillHornet\_withOverridingC

zue1Wzw

zue2Wzw

zue3Wzw

zue4Wzw

zue5Wzw

zue6Wzw

zue7Wyw

Task List

Find All Activ...

Outline

Problems Javadoc Declaration Console

No consoles to display at this time.

Writable Smart Insert 6 : 9

Start

DE 99% 10:42 17.06.2016

Eclipse IDE screenshot showing the Striker.java file in the SoccerDemo project.

```
package zue6Wzw;
public class Striker extends Player implements Sportsman {
    public void doSports() {
        System.out.println("I play soccer!");
    }
}
```

The Outline view shows the Striker class with its doSports() method.

Eclipse IDE screenshot showing the Striker.java file in the SoccerDemo project.

```
package zue6Wzw;
public class Striker extends Player implements Sportsman {
    public void doSports() {
        System.out.println("I play soccer!");
    }
}
```

The Outline view shows the Striker class with its doSports() method.

Eclipse IDE screenshot showing the SoccerDemo.java file in the SoccerDemo project.

```
package zue6Wzw;
public class SoccerDemo {
    public static void main(String[] args) {
        Striker mario = new Striker();
        mario.doSports();
        mario.playGame();
    }
}
```

The Outline view shows the SoccerDemo class with its main() method.

Eclipse IDE screenshot showing the SoccerDemo.java file in the SoccerDemo project.

```
package zue6Wzw;
public class SoccerDemo {
    public static void main(String[] args) {
        Striker mario = new Striker();
        mario.doSports();
        mario.playGame();
    }
}
```

The Outline view shows the SoccerDemo class with its main() method.

Console output:

```
<terminated> SoccerDemo [Java Application] C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (17.06.2016, 10:43:59)
I play soccer!
I have fun!
```