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 Modifiern

```
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);
    }
}
```


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");
    }
}
```

Variante mit `super`

```
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
```

⇨ Ausgabe:
```
superclass instance-method
subclass instance-method
```

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

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"
```

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

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

Aufgabe

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!**

Aufgabe

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!**

Aufgabe

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!**

151

Aufgabe

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!**

151

Aufgabe

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!**

151

Aufgabe

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!**

Aufgabe

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!**

151

Aufgabe

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!**

151

Aufgabe

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!**

151

Aufgabe

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!**

Aufgabe

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!**

Aufgabe

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!**

Aufgabe

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);
}
```

Aufgabe

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**:

```java
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);
}
```


Aufgabe

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**:

```java
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);
}
```


Aufgabe

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**:

```java
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);
}
```


Aufgabe

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**:

```java
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);
}
```

Aufgabe

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);
}
```


Aufgabe

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);
}
```


Aufgabe

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);
}
```


Aufgabe

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);
}
```

Aufgabe

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);
}
```


Aufgabe

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);
}
```

Aufgabe

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);
}
```

Aufgabe

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);
}
```

Aufgabe

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);
}
```


Aufgabe

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);
}
```

Aufgabe

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);
}
```

Aufgabe

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);
}
```

Aufgabe

aehm...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.

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();
}
```


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();
}
```

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

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

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

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();
}
```


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

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

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

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();
}
```


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

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

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

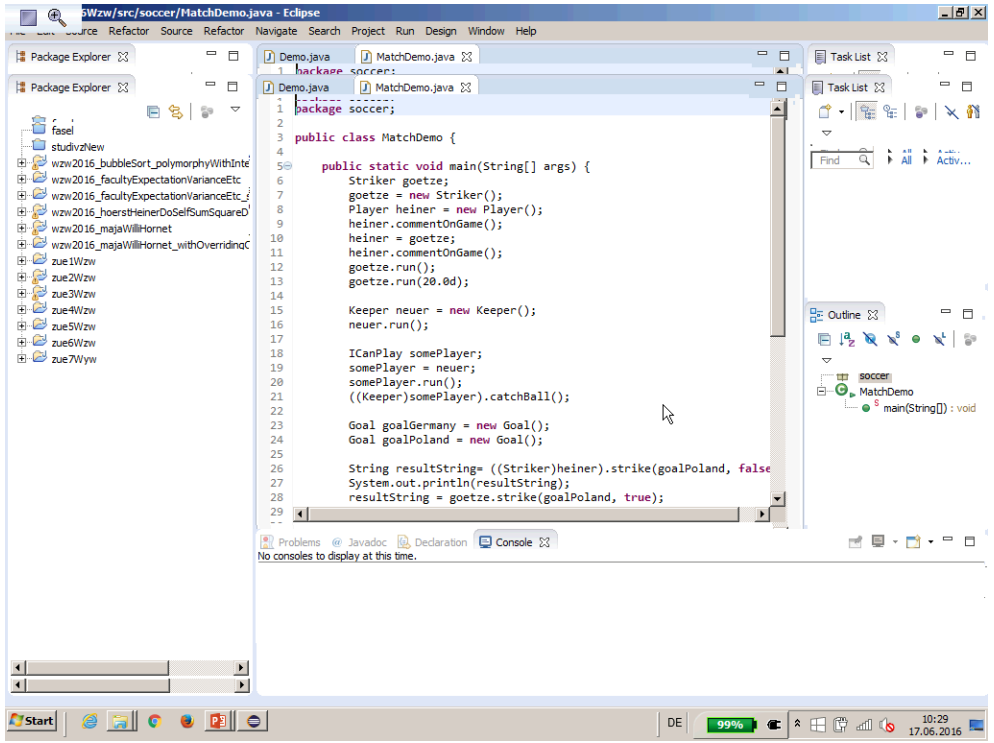
Aufgabe

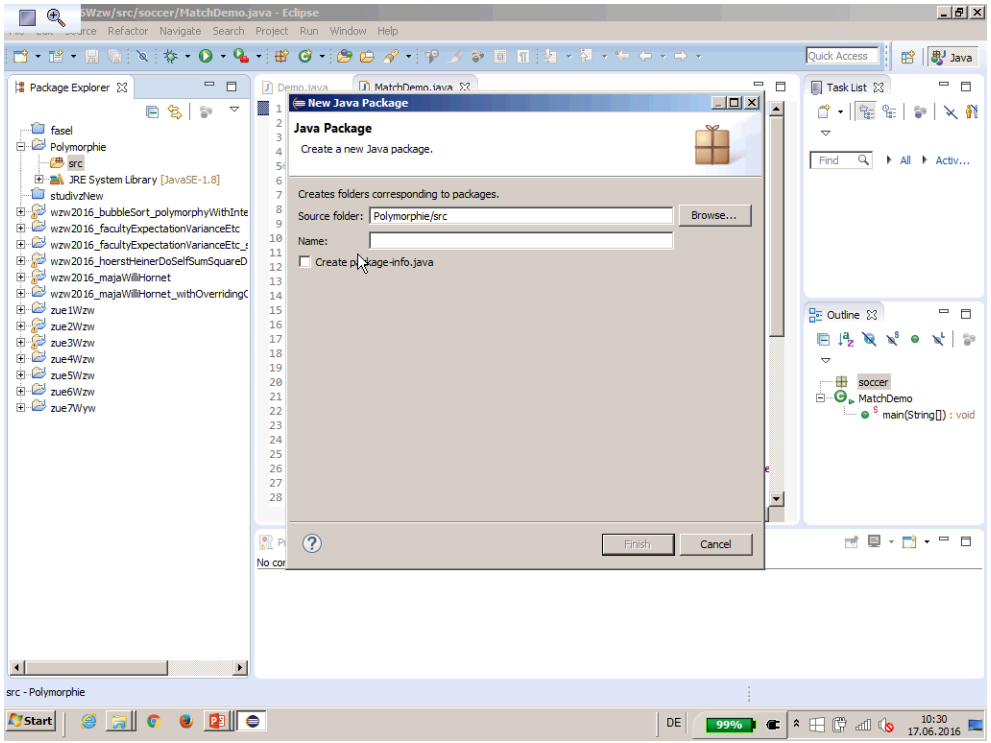
Welche Ausgabe produziert FinancialCrisisDemo?

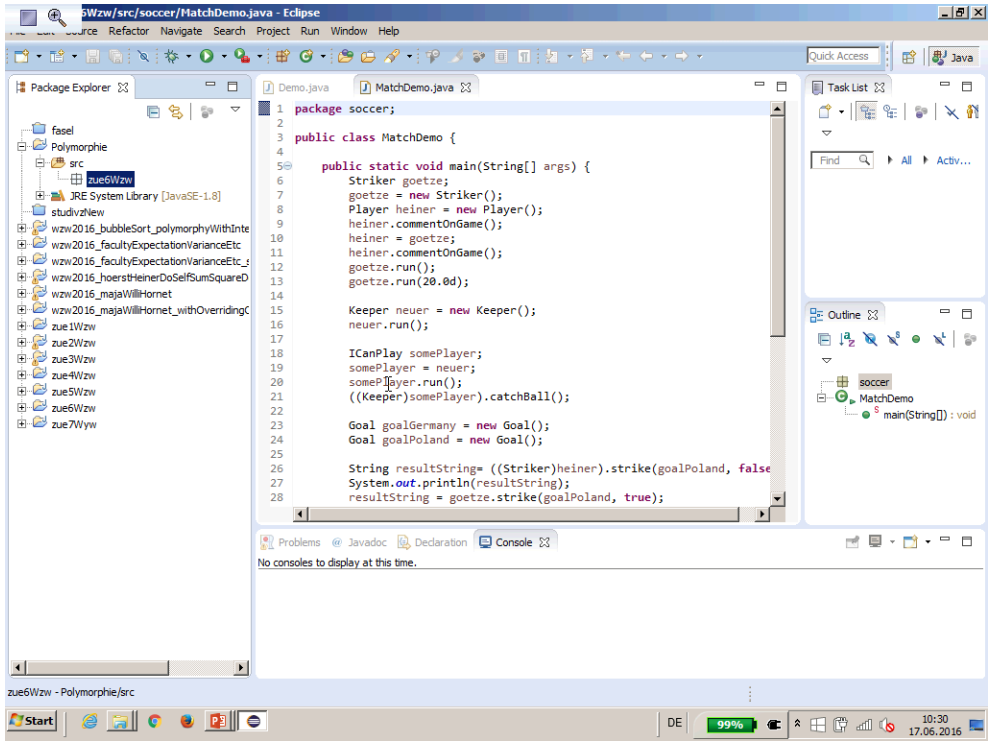
Welche der drei Konzepte **Overloading**, **Overriding**, **Polymorphie** werden im obigen Code benutzt, welche nicht? **BEGRÜNDEN** Sie jeweils kurz!

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();
}
```





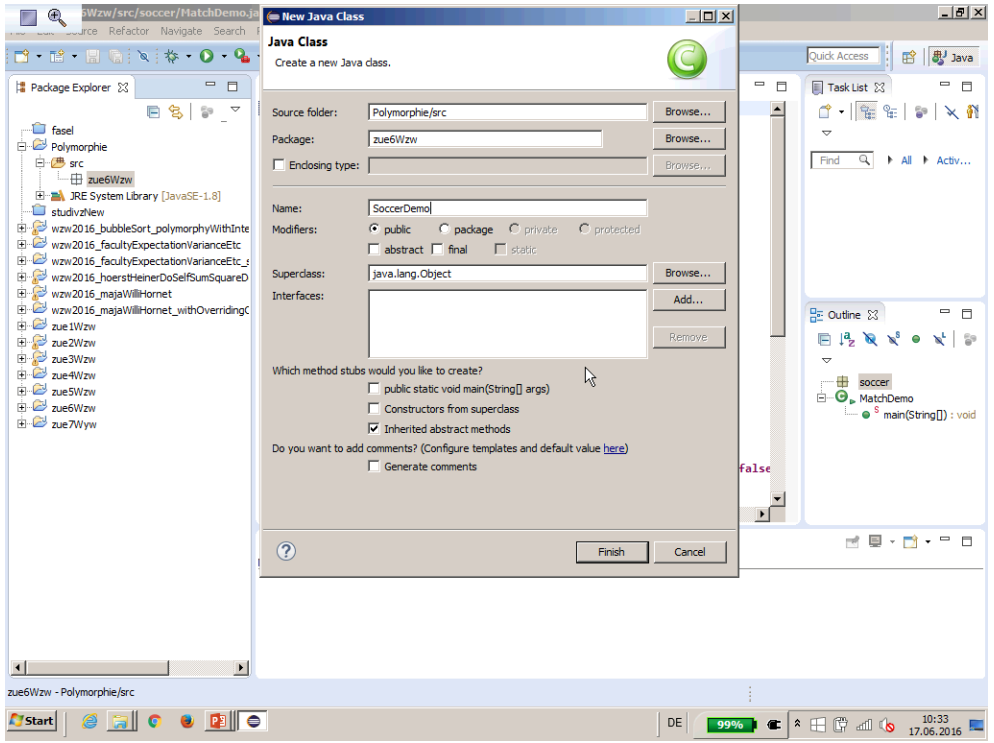
Aufgabe

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();
}
```



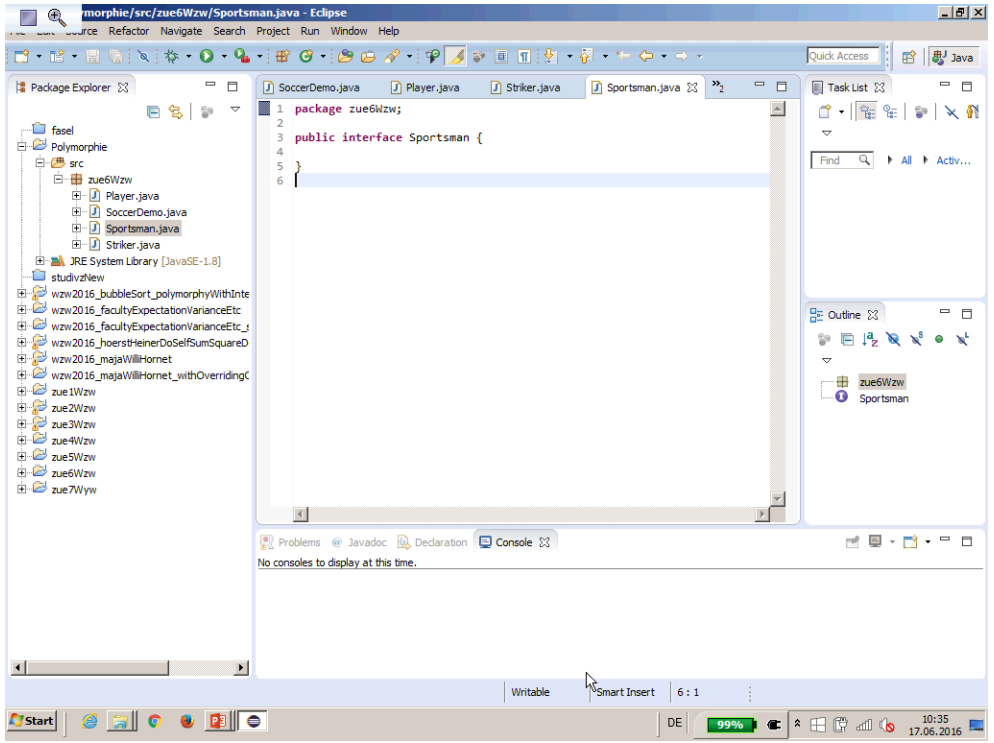
Aufgabe

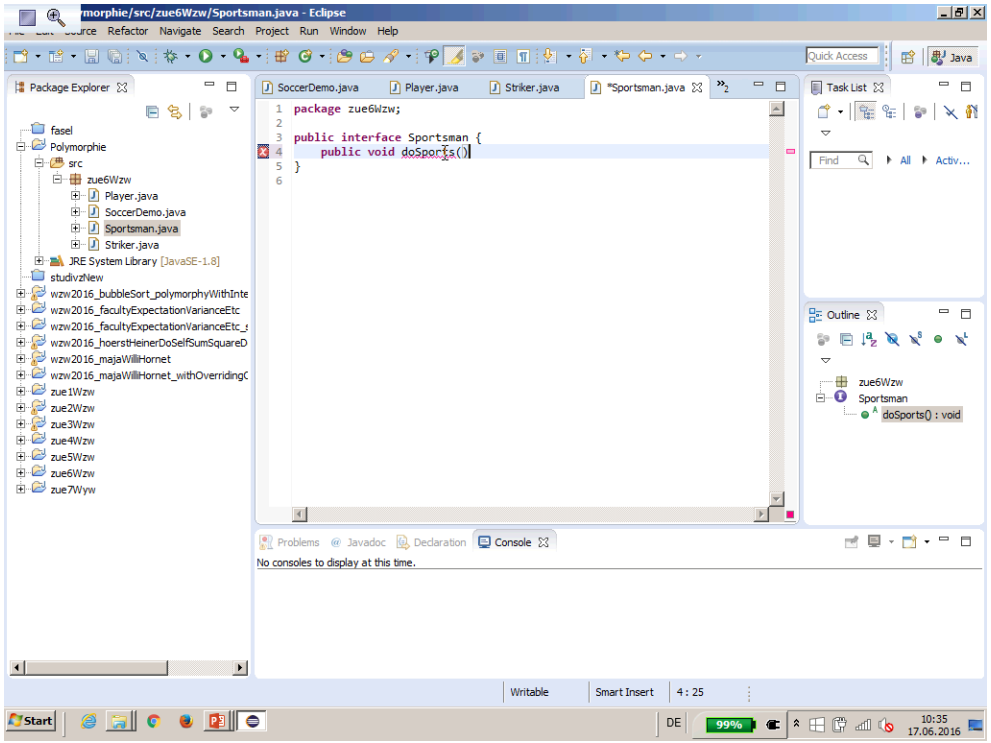
Welche Ausgabe produziert FinancialCrisisDemo?

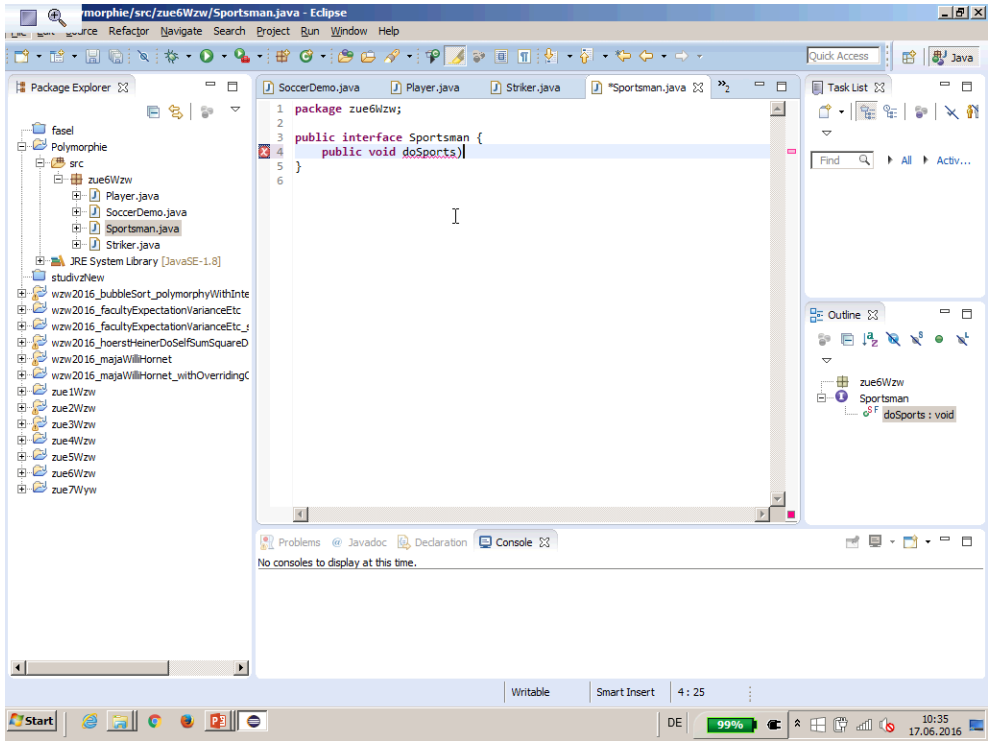
Welche der drei Konzepte **Overloading**, **Overriding**, **Polymorphie** werden im obigen Code benutzt, welche nicht? **BEGRÜNDEN** Sie jeweils kurz!

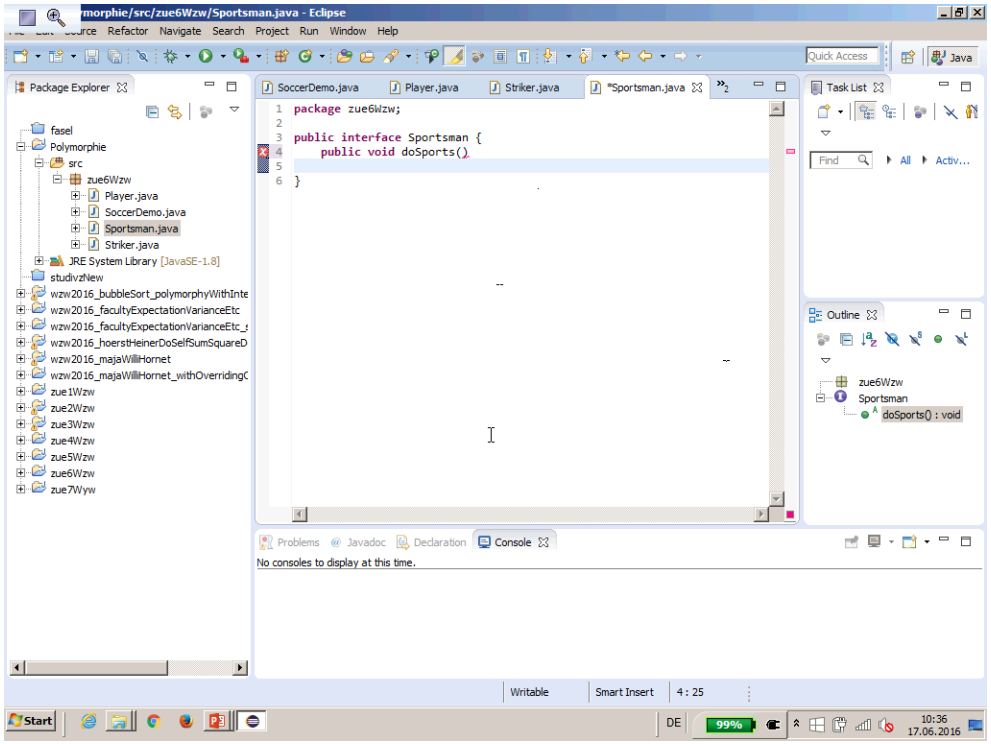
Gegeben sei folgender **Java Code**:

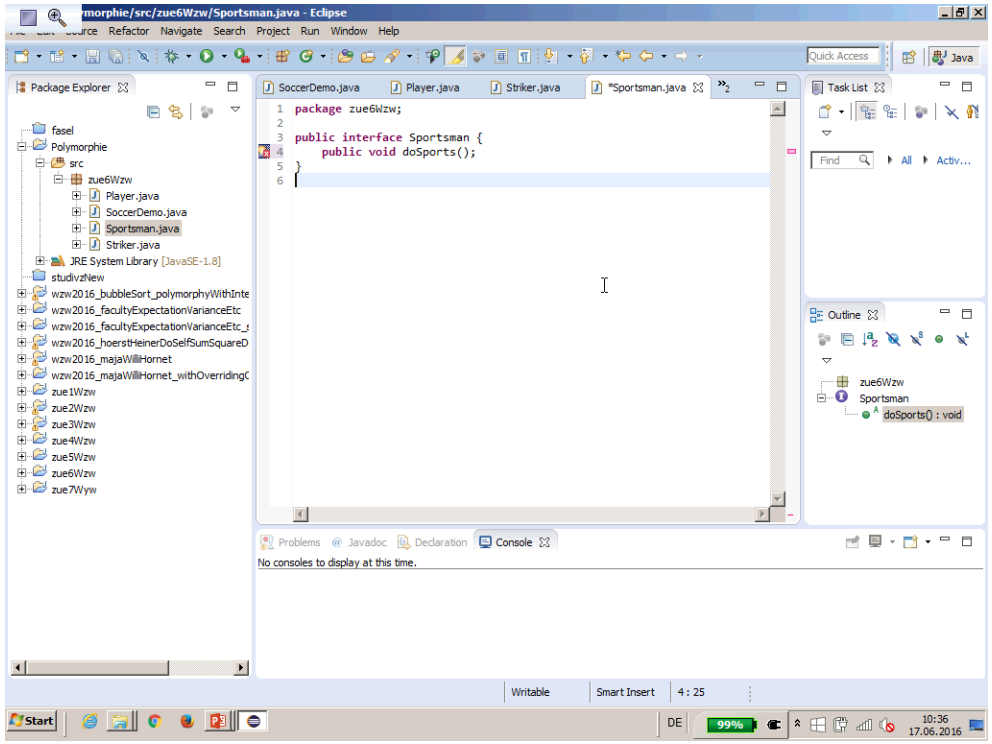
```
public class FinancialCrisisDemo {

	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();
}
```




morphie/src/zue6Wzw/Sportsman.java - Eclipse
Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
fasel
Polymorphie
src
zue6Wzw
Player.java
SoccerDemo.java
Sportsman.java
Striker.java
JRE System Library [JavaSE-1.8]
studivzNew
wzw2016_bubbleSort_polymorphyWithInte
wzw2016_facultyExpectationVarianceEtc
wzw2016_facultyExpectationVarianceEtc_
wzw2016_hoerstHeinerDoSelfSumSquareD
wzw2016_majaWillHornet
wzw2016_majaWillHornet_withOverridingC
zue1Wzw
zue2Wzw
zue3Wzw
zue4Wzw
zue5Wzw
zue6Wzw
zue7Wyw
SoccerDemo.java
Player.java
Striker.java
*Sportsman.java
1 package zue6Wzw;
2
3 public interface Sportsman {
4 public void doSports()
5 }
6
Task List
Find
All
Activ...
Outline
zue6Wzw
Sportsman
doSports() : void
Problems
Javadoc
Declaration
Console
No consoles to display at this time.
Writable
Smart Insert
4 : 25
Start
DE
99%
10:35
17.06.2016


morphie/src/zue6Wzw/Sportsman.java - Eclipse
Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
fasel
Polymorphie
src
zue6Wzw
Player.java
SoccerDemo.java
Sportsman.java
Striker.java
JRE System Library [JavaSE-1.8]
studivzNew
wzw2016_bubbleSort_polymorphyWithInte
wzw2016_facultyExpectationVarianceEtc
wzw2016_facultyExpectationVarianceEtc_
wzw2016_hoerstHeinerDoSelfSumSquareD
wzw2016_majaWillHornet
wzw2016_majaWillHornet_withOverridingC
zue1Wzw
zue2Wzw
zue3Wzw
zue4Wzw
zue5Wzw
zue6Wzw
zue7Wyw
SoccerDemo.java
Player.java
Striker.java
*Sportsman.java
1 package zue6Wzw;
2
3 public interface Sportsman {
4 public void doSports)
5 }
6
Task List
Find
All
Activ...
Outline
zue6Wzw
Sportsman
doSports : void
Problems
Javadoc
Declaration
Console
No consoles to display at this time.
Writable
Smart Insert
4 : 25
Start
DE
99%
10:35
17.06.2016


morphie/src/zue6Wzw/Sportsman.java - Eclipse
Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
fasel
Polymorphie
src
zue6Wzw
Player.java
SoccerDemo.java
Sportsman.java
Striker.java
JRE System Library [JavaSE-1.8]
studivzNew
wzw2016_bubbleSort_polymorphyWithInte
wzw2016_facultyExpectationVarianceEtc
wzw2016_facultyExpectationVarianceEtc_
wzw2016_hoerstHeinerDoSelfSumSquareD
wzw2016_majaWillHornet
wzw2016_majaWillHornet_withOverridingC
zue1Wzw
zue2Wzw
zue3Wzw
zue4Wzw
zue5Wzw
zue6Wzw
zue7Wyw
SoccerDemo.java
Player.java
Striker.java
*Sportsman.java
1 package zue6Wzw;
2
3 public interface Sportsman {
4 public void doSports()
5
6 }
Task List
Find
All
Activ...
Outline
zue6Wzw
Sportsman
doSports() : void
Problems
Javadoc
Declaration
Console
No consoles to display at this time.
Writable
Smart Insert
4 : 25
Start
DE
99%
10:36
17.06.2016


morphie/src/zue6Wzw/Sportsman.java - Eclipse
Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
fasel
Polymorphie
src
zue6Wzw
Player.java
SoccerDemo.java
Sportsman.java
Striker.java
JRE System Library [JavaSE-1.8]
studivzNew
wzw2016_bubbleSort_polymorphyWithInte
wzw2016_facultyExpectationVarianceEtc
wzw2016_facultyExpectationVarianceEtc_
wzw2016_hoerstHeinerDoSelfSumSquareD
wzw2016_majaWillHornet
wzw2016_majaWillHornet_withOverridingC
zue1Wzw
zue2Wzw
zue3Wzw
zue4Wzw
zue5Wzw
zue6Wzw
zue7Wyw
SoccerDemo.java
Player.java
Striker.java
*Sportsman.java
1 package zue6Wzw;
2
3 public interface Sportsman {
4 public void doSports();
5 }
6
Task List
Find
All
Activ...
Outline
zue6Wzw
Sportsman
doSports() : void
Problems
Javadoc
Declaration
Console
No consoles to display at this time.
Writable
Smart Insert
4 : 25
Start
DE
99%
10:36
17.06.2016

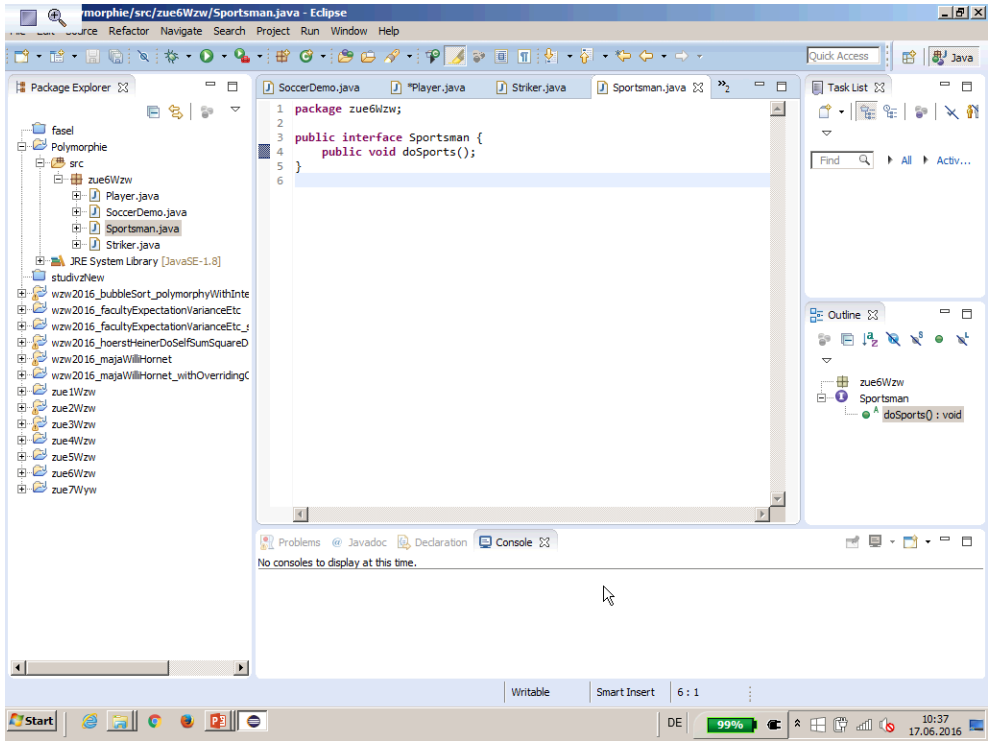
Aufgabe

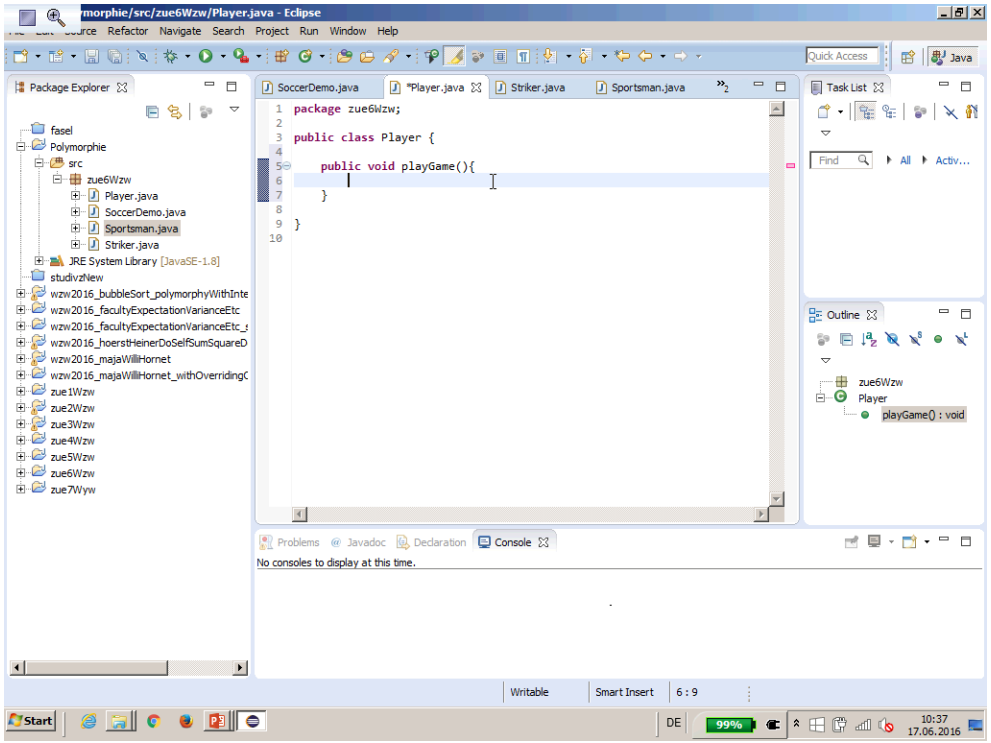
Gegeben sei folgender **Java Code**:

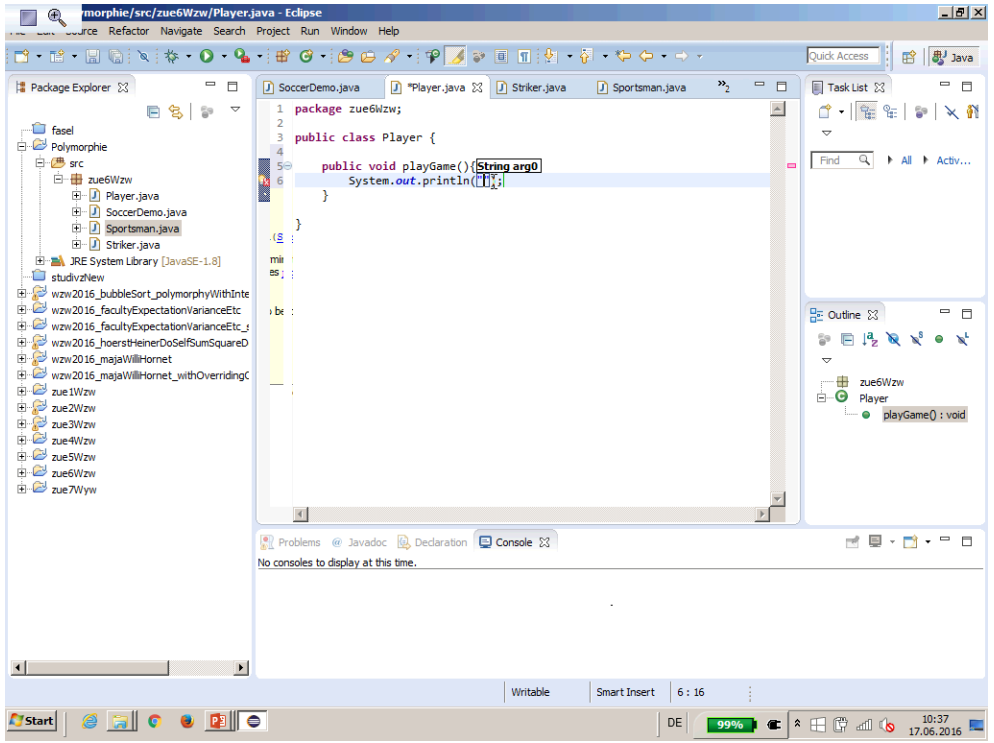
```
public class FinancialCrisisDemo {

	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();
}
```

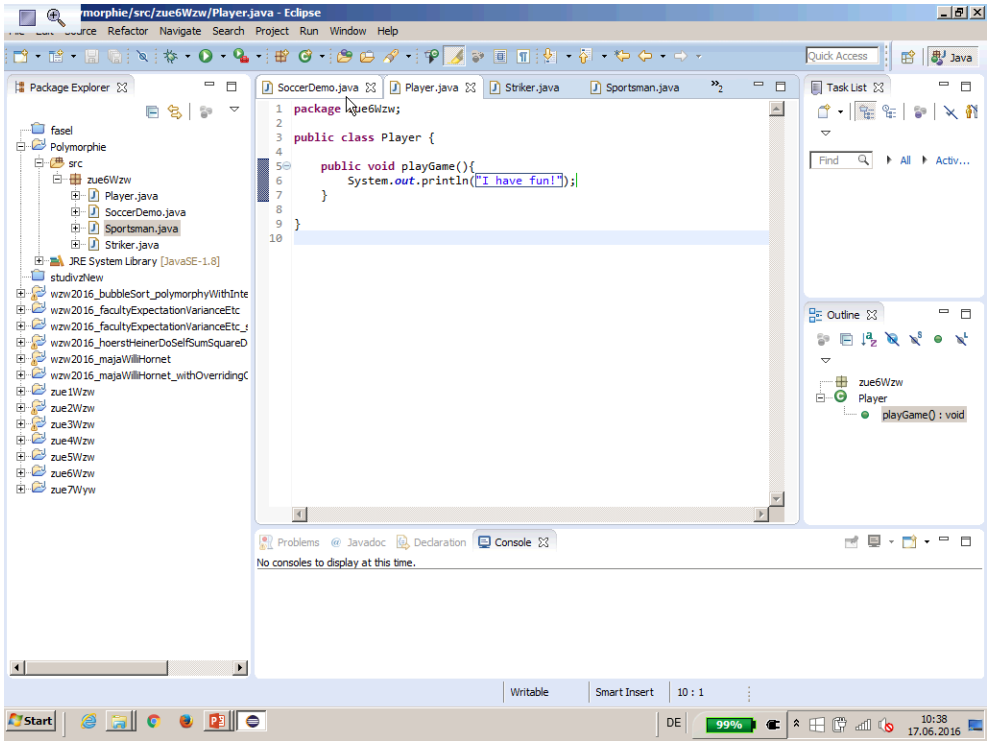

Welche Ausgabe produziert FinancialCrisisDemo?

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









Aufgabe

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();
}
```


Aufgabe

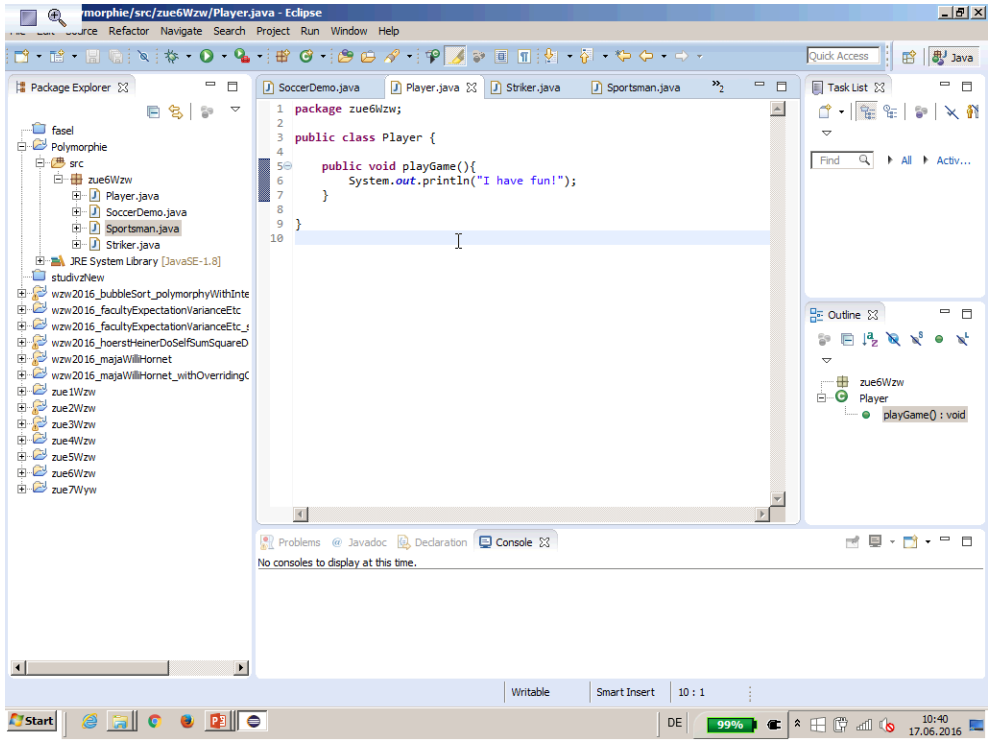
Welche Ausgabe produziert FinancialCrisisDemo?

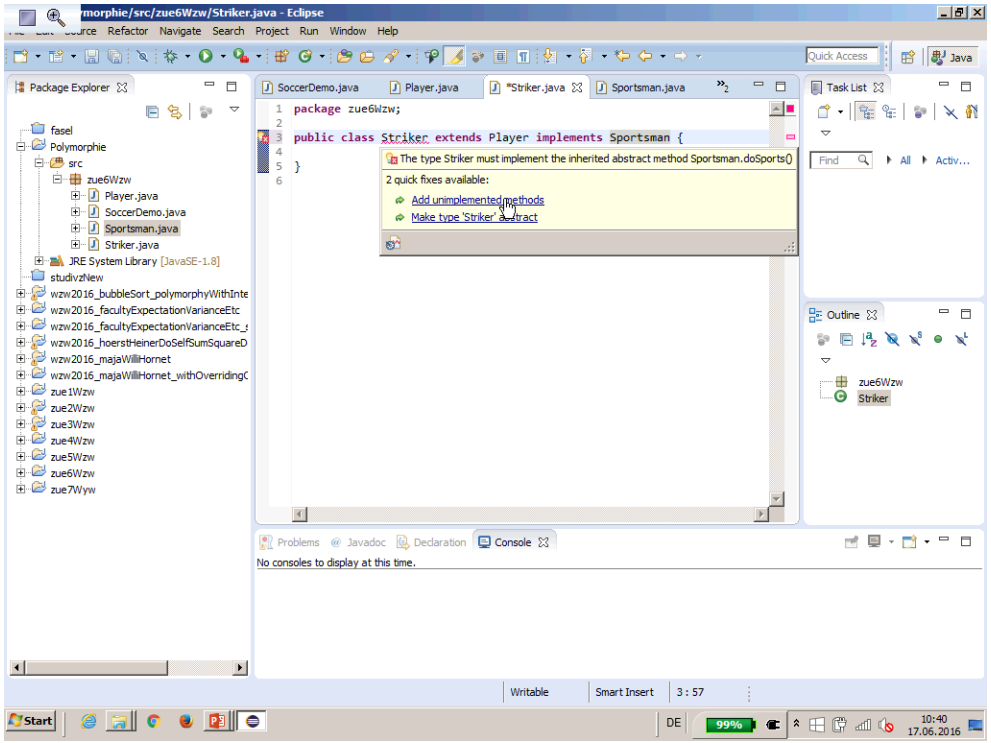
Welche der drei Konzepte **Overloading**, **Overriding**, **Polymorphie** werden im obigen Code benutzt, welche nicht? **BEGRÜNDEN** Sie jeweils kurz!

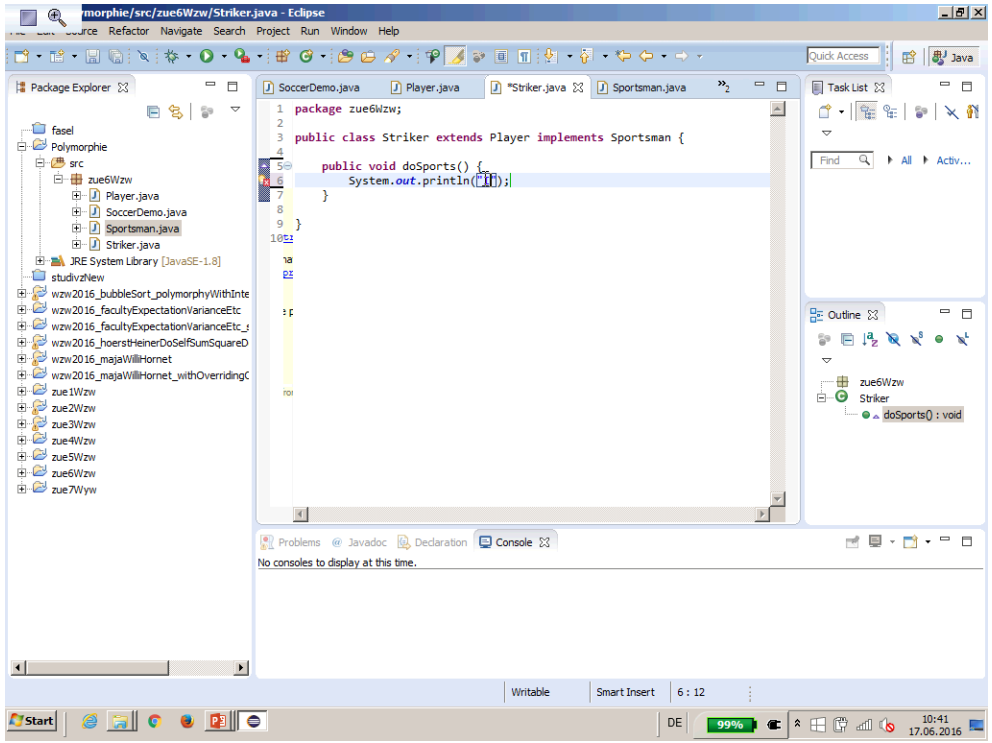
Gegeben sei folgender **Java Code**:

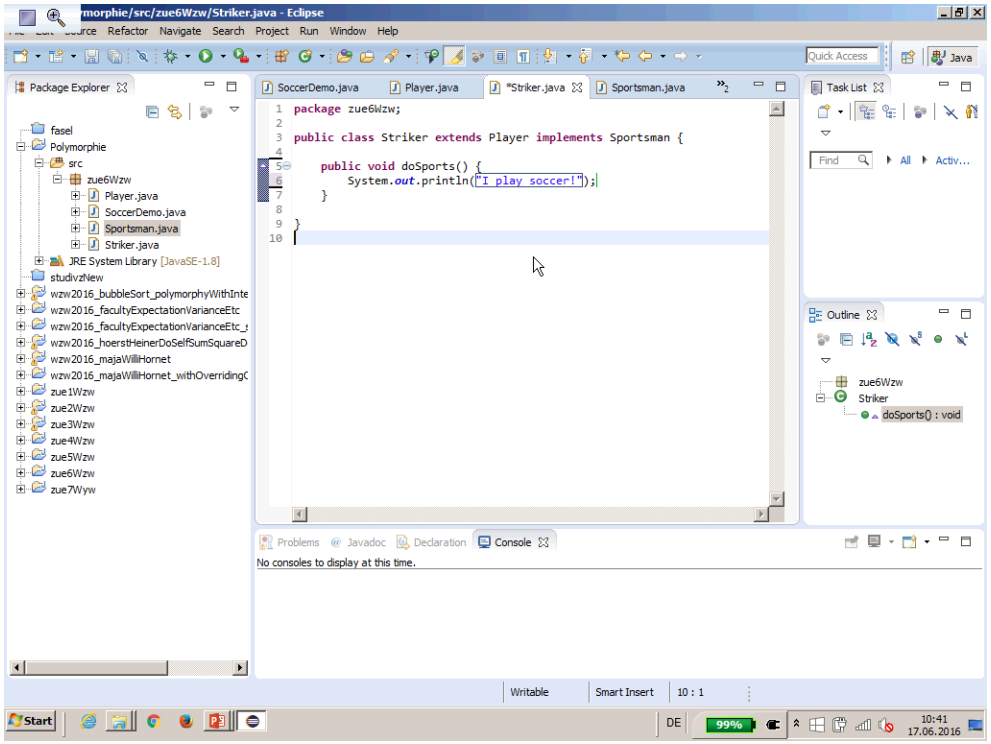
```
public class FinancialCrisisDemo {

	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();
}
```

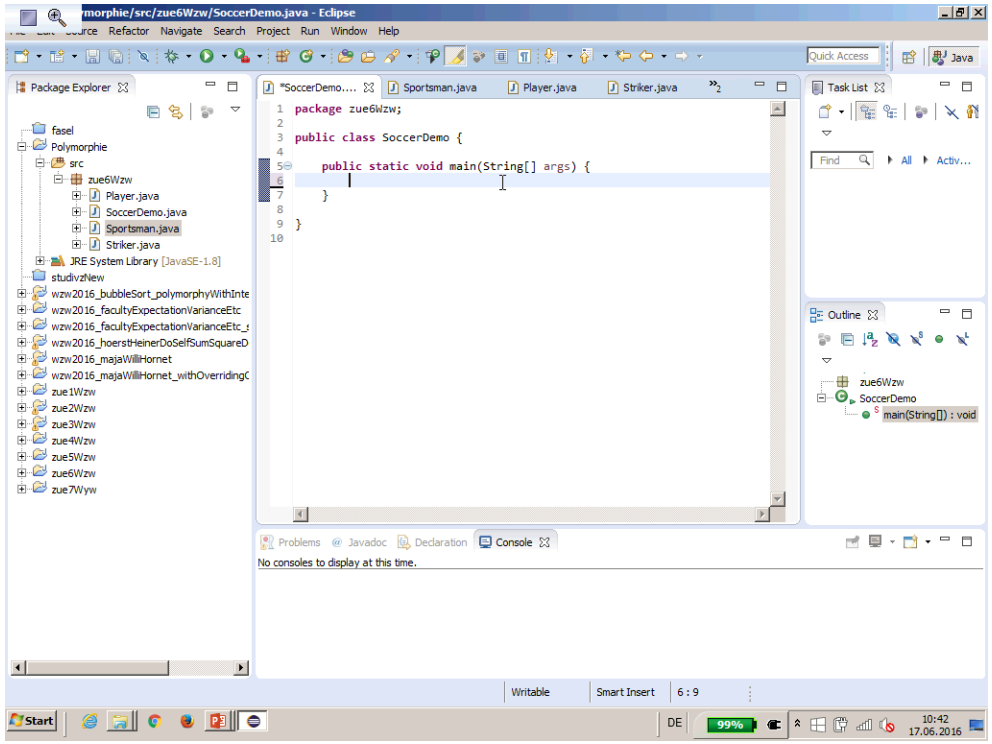
155

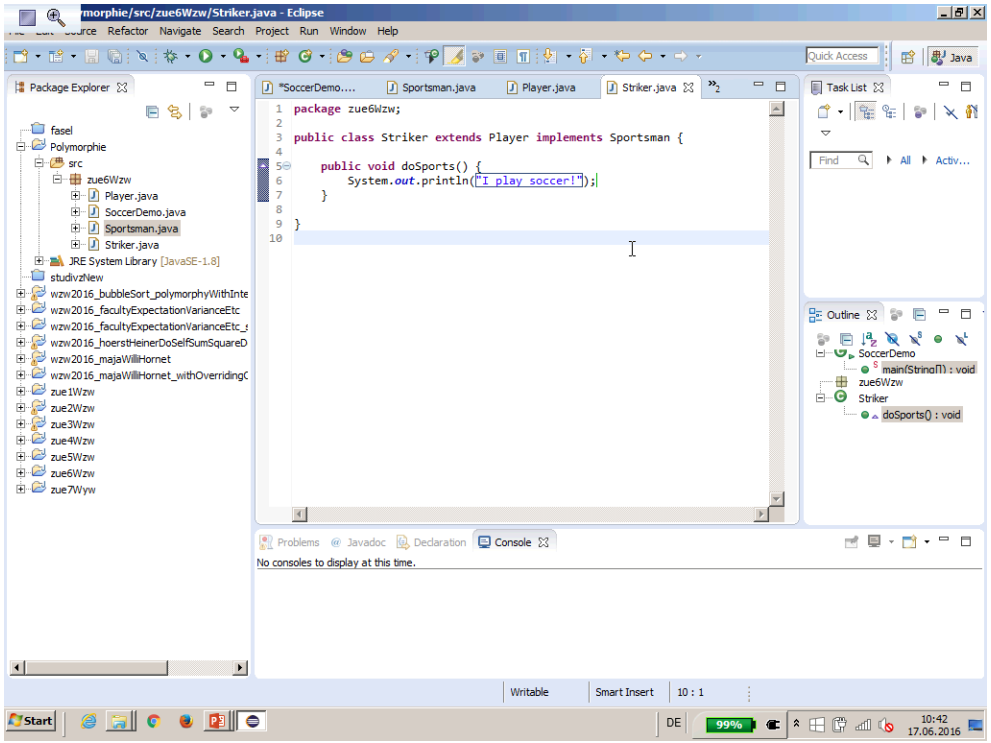


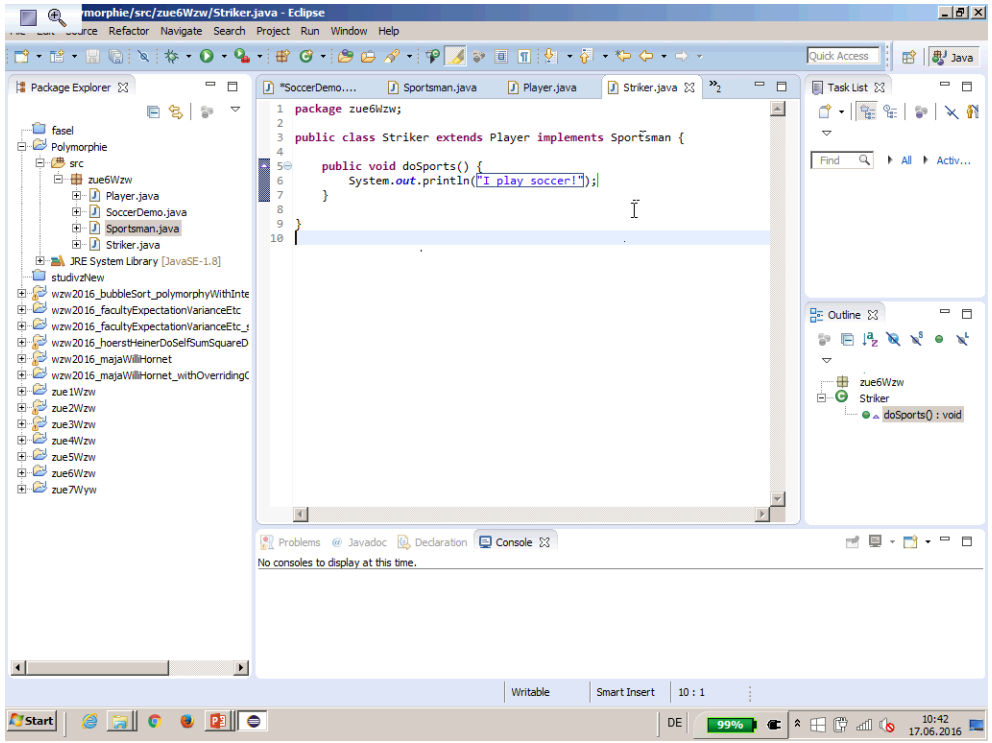

morphie/src/zue6Wzw/Striker.java - Eclipse
package zue6Wzw;
public class Striker extends Player implements Sportsman {
}
The type Striker must implement the inherited abstract method Sportsman.doSports()
2 quick fixes available:
Add unimplemented methods
Make type 'Striker' abstract
Writable
Smart Insert
3 : 57

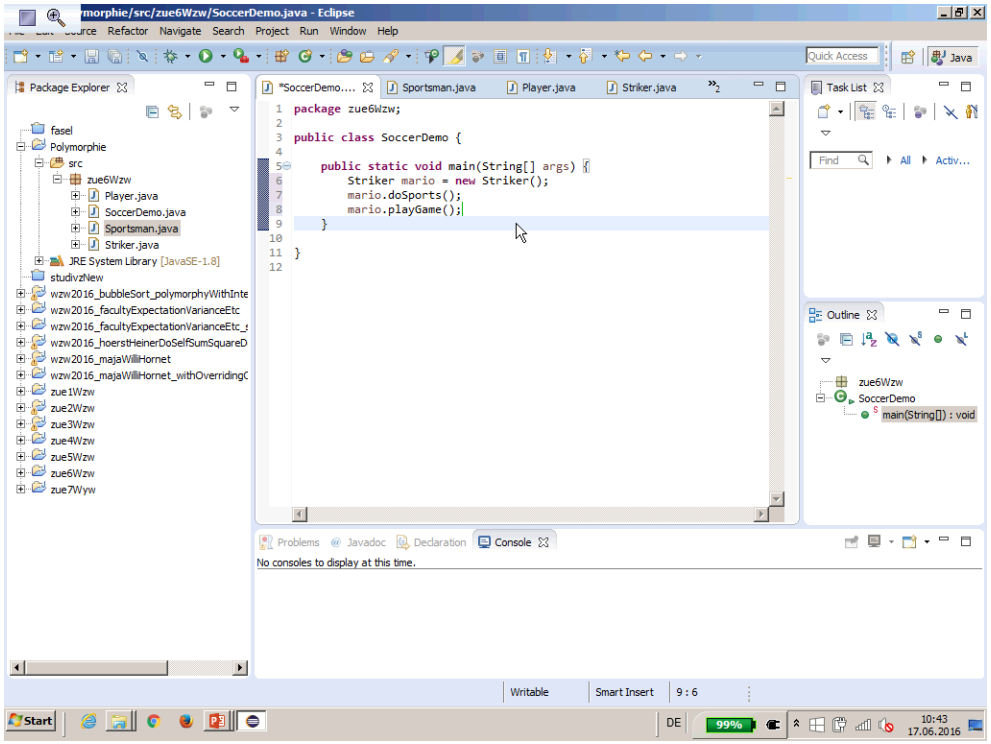

morphie/src/zue6Wzw/Striker.java - Eclipse
package zue6Wzw;
public class Striker extends Player implements Sportsman {
public void doSports() {
System.out.println();
}
}
Writable
Smart Insert
6 : 12

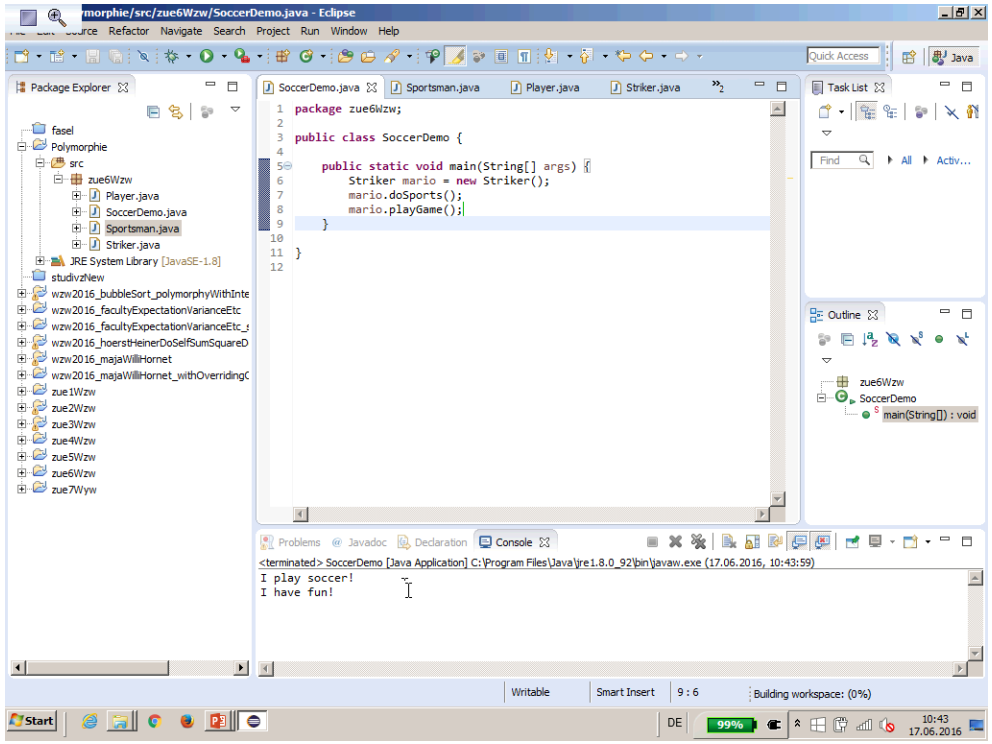

morphie/src/zue6Wzw/Striker.java - Eclipse
package zue6Wzw;
public class Striker extends Player implements Sportsman {
public void doSports() {
System.out.println("I play soccer!");
}
}
Writable
Smart Insert
10 : 1


morphie/src/zue6Wzw/SoccerDemo.java - Eclipse
package zue6Wzw;
public class SoccerDemo {
public static void main(String[] args) {
}
}
Writable
Smart Insert
6 : 9


morphie/src/zue6Wzw/Striker.java - Eclipse
Refactor Navigate Search Project Run Window Help
Package Explorer
*SoccerDemo.... Sportsman.java Player.java Striker.java
Task List
Outline
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
Problems Javadoc Declaration Console
No consoles to display at this time.
Writable Smart Insert 10 : 1


morphie/src/zue6Wzw/Striker.java - Eclipse
Refactor Navigate Search Project Run Window Help
Package Explorer
*SoccerDemo.... Sportsman.java Player.java Striker.java
Task List
Outline
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
Problems Javadoc Declaration Console
No consoles to display at this time.
Writable Smart Insert 10 : 1


morphie/src/zue6Wzw/SoccerDemo.java - Eclipse
Refactor Navigate Search Project Run Window Help
Package Explorer
*SoccerDemo.... Sportsman.java Player.java Striker.java
Task List
Outline
1 package zue6Wzw;
2
3 public class SoccerDemo {
4
5 public static void main(String[] args) {
6 Striker mario = new Striker();
7 mario.doSports();
8 mario.playGame();
9 }
10
11 }
12
Problems Javadoc Declaration Console
No consoles to display at this time.
Writable Smart Insert 9 : 6


morphie/src/zue6Wzw/SoccerDemo.java - Eclipse
Refactor Navigate Search Project Run Window Help
Package Explorer
SoccerDemo.java Sportsman.java Player.java Striker.java
Task List
Outline
1 package zue6Wzw;
2
3 public class SoccerDemo {
4
5 public static void main(String[] args) {
6 Striker mario = new Striker();
7 mario.doSports();
8 mario.playGame();
9 }
10
11 }
12
Problems Javadoc Declaration Console
<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!
Writable Smart Insert 9 : 6
Building workspace: (0%)