

Klassenaufgaben

① Was wird vererbt?

	Methoden		Attribute	
	Static	Non-Static	Static	Non-Static
Klasse	✓	✓	✓	✓
Interface	✗	✓*	✓	—
Enum	—	—	—	—

— Enums sind immer final und Interfaces haben keine non-static Attribute * Default und abstrakte Methoden

② Was wird überschrieben (override)?

Was wird versteckt (hide)

Klasse n-s Methoden Attribute (s und n-s) s Methoden
 Interface n-s Methoden Attribute (s und n-s)

Gegeben sei eine Methode main in einer Java Klasse.

```
public static void main(String[] args) {  
    /* body */  
}
```

Die folgenden Anweisungen sollen als "Body" (Rumpf) anstelle des Kommentars `/* body */` eingefügt werden. Geben Sie für jede Anweisung an, was für eine Ausgabe erzeugt wird – entweder was gedruckt wird, oder ob ein Laufzeitfehler auftritt (schreiben Sie "Exception"), oder ob der Compiler einen Fehler feststellt (schreiben Sie "Compile-Fehler"). Achten Sie auf die korrekte Formatierung der verschiedenen Typen, also z.B. 7.0 statt 7 für eine reelle Zahl (double).


1. `System.out.println(11 + 16 / 4 * 2 + (4 + ">") + 4 * 2);`

$$11 + 8 + ("4 >") + 8 = 194 > 8$$

2. `System.out.println(20 / 10 % 6 + 3 / 2 + (double) 5 / 4 + 3 / 4);`

$$2 + 1 + 1.25 + 0 = 4.25$$

3. `System.out.println((11 % 4) > 2 && 9 > (16 / (2 / 4)) && 1 % 8 < 0);`

$$3 > 2 \ \&\& \ 9 > \underbrace{(16 / 0)} \ \&\& \ 1 < 0$$


Exception

Gegeben sind die Precondition und Postcondition für das folgende Programm

```
public int compute(int v, int n) {
```

```
    // Precondition: n >= 0
```

Sollte $(n == 0 \parallel (n > 0 \ \&\& \ v == 2))$ sein (oder stärker)

```
    int x;  
    int tmp;
```

```
    x = 1;  
    tmp = 1;
```

```
    // Loop Invariante:
```

```
    while (x <= n) {  
        tmp = tmp * v;  
        x = x + 1;  
    }
```

```
    // Postcondition: tmp == 2n
```

```
    return tmp;
```

```
}
```

$$2^n == tmp \cdot v^{n-x+1}$$



$$2^{x-1} == tmp \ \&\& \ x <= n+1 \ \&\& \ \left(\begin{array}{l} v == 2 \\ \&\& \\ n > 0 \end{array} \parallel n == 0 \right)$$

$$\&\& \ x \geq 1$$

Bitte geben Sie die Loop Invariante an.

Loop Invariante: _____

$$2^{x-1} == tmp \ \&\& \ x \geq 1 \ \&\& \ x <= n+1 \ \&\& \ \left(v == 2 \parallel n == 0 \right)$$

Wichtig: $P_1 \Rightarrow P_2$ P_2 is weaker than P_1

Vervollständigen Sie die Lücken im untenstehenden Programmcode, sodass die main Methode in der InheritanceTest Klasse ohne Fehler kompiliert und ausgeführt werden kann, ohne eine Exception zu werfen. Der zu erwartende Konsolen-Output von main ist unten angegeben. Sie dürfen keine weiteren Klassen, Methoden, oder Interfaces hinzufügen.

Tipp: Nicht in allen Lücken *muß* etwas stehen aber in allen Lücken *kann* etwas stehen.

```
class InheritanceTest {

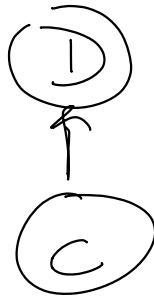
    public static void main(String[] args) {
        Z ref1 = new B();
        ref1.bar();
        System.out.println("++");
        Z ref2 = new A();
        ((A) ref2).bar();
        System.out.println("++");

        C c1 = new C();
        System.out.println("C.foo():");
        c1.foo();
        System.out.println("--");
        D d1 = new D();
        if (d1 instanceof C) {
            ((C)d1).test();
        } else {
            d1.foo();
        }
    }
}
```

Jedesmal wenn die Methode main in der Klasse InheritanceTest ausgeführt wird (alle Klassen sind in dem selben Package), sollte folgender Output auf die Konsole geschrieben werden:

```
Hello
Bingo
++
Hello
++
C.foo():
Here
--
Found
```

Hello



=> C extends D

```
class C extends D {

    public void foo() {
        System.out.println("Here");
    }

    public void test(){
        System.out.println("Test");
    }

}
```

```
class D extends A {

    public void foo() {
        super.foo();
    }

}
```

```
class Z {

    public void bar() {
        System.out.println("Hello");
    }

}
```

```
class A extends Z {

    int a1 = 0;

    A() {}
    A(int v) {
        a1 = v;
    }
    Where bar ???
    public void foo() {
        System.out.println("Found");
    }

}
```

```
class B extends A {

    B() { }

    B(int w) {
        super(w);
    }

    public void bar() {
        super.bar();
        System.out.println("Bingo");
    }

}
```

Gegeben seien diese Klassen und Interfaces in separaten Dateien (im default Package):

```
class Caniformia {
    String name = "Hundeartige";
    String shortName = "HA";

    public String toString() {
        return shortName;
    }
}

class Canidae extends Caniformia {
    String name = "Hunde";
    String shortName = "H";

    public void jagdbar() {
        System.out.println("J2_" + shortName);
    }

    public void geschuetzt() {
        System.out.println("G2_" + shortName);
    }
}

class Arctoidea extends Caniformia {
    String name = "NichtHunde";

    public void jagdbar() {
        System.out.println("J3_" + shortName);
        praesent();
    }

    public void praesent() {
        System.out.println("P3_" + shortName);
    }
}

class Ursidae extends Arctoidea {
    String name = "Baeren";
    String shortName = "B";

    public void geschuetzt() {
        System.out.println("G4_" + shortName);
    }
}

class Pinnipedia extends Arctoidea {
    String name = "Robben";
    String shortName = "R";

    public void geschuetzt() {
        System.out.println("G5_" + shortName);
    }

    public void praesent() {
        System.out.println("P5_" + shortName);
    }

    public String toString() {
        return shortName;
    }
}

class Otariidae extends Pinnipedia {
    String name = "Ohrenrobben";
    String shortName = "O";

    public void geschuetzt() {
        super.geschuetzt();
        System.out.println("G6_" + shortName);
    }
}

class Odobenidae extends Pinnipedia {
    String name = "Walrosse";
    String shortName = "W";

    public void geschuetzt() {
        System.out.println("G7_" + shortName);
    }
}
```

In einer Klasse Explore in dem selben Package befindet sich die Methode main.

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

    /* Body */

}
```

1. `Object tier = new Odobenidae();`
`((Pinnipedia) tier).praesent();`

P5, R

2. `Caniformia cx = new Canidae();`
`cx.jagdbar();`

Compile Error

3. `Caniformia cw = new Ursidae();`
`System.out.println(cw);`

HA

4. `Arctoidea cy = new Ursidae();`
`System.out.println((Arctoidea) cy);`
`if (cy instanceof Pinnipedia) {`
 `cy.praesent();`
`}`



HA

5. `Ursidae uz = new Ursidae();`
`uz.geschuetzt();`

G4, B

6. `Pinnipedia pb = new Otariidae();`
`((Odobenidae) pb).geschuetzt();`

Runtime Exception

7. `Pinnipedia pc = new Otariidae();`
`pc.praesent();`

P5, R

8. `Arctoidea av = new Odobenidae();`
`av.praesent();`

P5, R

9. `Arctoidea at = new Odobenidae();`
`Pinnipedia pu = (Pinnipedia)at;`
`pu.praesent();`

P5, R