Gegeben sei folgender regulärer Ausdruck:

```
String regex = ‘‘a(b*c)’’;
```

Welche Ausdrücke liefern den Wert „true“ zurück?

Pattern.matches(regex, ”abc”);	☐
Pattern.matches(regex, ”a”);	☐
Pattern.matches(regex, ”abeeeec”);	☐
Nichts davon gilt	☐

Gegeben sei folgender String:

```
String text = ‘‘Semestralklausur’’;
```

Welche Ausdrücke liefern den Wert „true“ zurück?

Pattern.matches(”[a-z]*”, text);	☐
Pattern.matches(”\\.*”, text);	☐
Pattern.matches(”[^\s][a-zA-Z0-9]*[0-9]?”, text);	☐
Nichts davon gilt	☐

Aufgabe 1

Jeweils ALLE richtigen Antworten ankreuzen.

Reguläre Ausdrücke beziehen sich hier ausschließlich auf das entsprechende Konzept von Java.

Welche Aussagen über reguläre Ausdrücke treffen zu?

Gierige Operatoren „matchen" die kürzeste passende Zeichenkette.	☐
"." ist ein gieriger Operator.*	☐
Pattern.compile("A.?B").split("ABCDEB")[1] liefert "CDEB" zurück.*	☐
Nichts davon gilt	☐

Gegeben sei folgender regulärer Ausdruck:

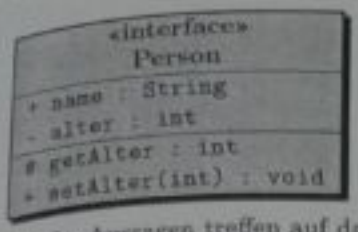
```
String regex = "[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|26[1-9]";
```

Welche Ausdrücke liefern den Wert „true" zurück?

Pattern.matches(regex, "06");	☐
Pattern.matches(regex, "206");	☐
Pattern.matches(regex, "260");	☐
Nichts davon gilt	☐

Aufgabe

Jeweils ALLE richtigen Antworten ankreuzen.



Welche Aussagen treffen auf das oben abgebildete UML-Diagramm zu?

Ein Objekt einer Implementierung dieses Interfaces ist immer `static`.	☐
Eine Klasse, die das Interface Person implementiert (`implements Person`), enthält immer eine Methode: `private void getAlter() { ... }`	☐
Die Variable `alter` ist eine Konstante.	☐
Nichts davon gilt	☐

Welche Aussagen treffen auf den `ASCII`-Standard zu?

Definiert *256* verschiedene Zeichen.	☐
Die `Unicode`-Kodierung erweitert die `ASCII`-Kodierung.	☐
Ist eine 7-Bit-Zeichenkodierung.	☐
Nichts davon gilt	☐

Welche Aussagen treffen auf `JAVA` zu?

Alle Variablen in Java sind automatisch vom Typ `Object`.	☐
Wenn `C` eine abstrakte Klasse ist, kann eine Variable so deklariert werden: `C x = new C();`	☐
Autoboxing bedeutet, dass automatisch zwischen primitiven Typen und ihren Wrapper-Klassen konvertiert wird.	☐
Nichts davon gilt	☐

Welche Aussagen treffen auf `Arrays` in Java zu?

Arrays sind keine Objekte in Java.	☐
Arrays können mehrere Rückgabewerte haben.	☐
Folgender Ausdruck erzeugt eine leere 3x3 Matrix: `int[] matrix = new int[3*3];`	☐
Nichts davon gilt	☐

Gegeben sei: `String[] s = {"s", "0", "3.1415", "EiP"};`
Geben Sie jeweils den Typ und den Wert der folgenden Ausdrücke an.

- `s[s.length - 1].equals("eip")`

Typ: ..... Wert: .....

- `Double.parseDouble(s[1] + "3.1415")`

Typ: ..... Wert: .....

- `s[9/4]`

Typ: ..... Wert: .....

- `"1" == s[1] || !(false & !true)`

Typ: ..... Wert: .....

Aufgabe 3

Beweisen Sie mit den Regeln des Hoare-Kalküls die partielle Korrektheit des folgenden Programmstücks. Geben Sie jeweils auch an, welche Regel Sie anwenden.

```
// Vorbedingung: true
if(a <= b) { x = a; y = b; }
else { x = b; y = s; }
// Nachbedingung: min(a, b) == x && max(a, b) == y
```

Die Main-Klasse wird nun wie folgt abgeändert:

```
1  class Main {
2
3    public static void main(String[] args) {
4      for(int i=0; i<args.length; i++) {
5        A a = new A();
6        a.start();
7        try{
8          a.join();
9        } catch(InterruptedException e) {
10         System.exit(1);
11         }
12     }
13     for(int i=0; i<args.length; i++) {
14       Thread b = new Thread(new B());
15       b.run();
16       try{
           b.join();
         } catch(InterruptedException e) {
           System.exit(1);
         }
       }
     }
   }
```

Was sind jetzt mögliche Ausgaben des Aufrufes:
`java Main 1 2 3 3`

A1 A2 A3 A4 B1 B2 B3 B4	☐
A1 A2 A4 A3 B1 B2 B3 B4	☐
A1 A2 A3 A4 B1 B2 B4 B3	☐
A1 A2 A4 A3 B1 B2 B4 B3	☐
Nichts davon gilt	☐

Aufgabe 4

Jeweils ALLE richtigen Antworten ankreuzen.

```
1  class A extends Thread {
2
3    protected static int number=0;
4
5
6    public A () {
7      super("A");
8    }
9
10   private int addNumber() {
11     return ++number;
12   }
13
14   public void run() {
15     System.out.println(
16       this.getName() + addNumber());
17   }
18 }
```

```
1  class B implements Runnable {
2
3    protected static int number=0;
4    protected static String name;
5
6    public B () {
7      this.name = "B";
8    }
9
10   private int addNumber() {
11     return ++number;
12   }
13
14   public void run() {
15     System.out.println(
16       this.name + addNumber());
17   }
18 }
```

```
1  class Main {
2
3    public static void main(String[] args) {
4      for(int i=0; i<args.length; i++) {
5        A a = new A();
6        a.start();
7      }
8      for(int i=0; i<args.length; i++) {
9        Thread b = new Thread(new B());
10       b.run();
11     }
12   }
13 }
```

Was sind mögliche Ausgaben des Aufrufes:
java Main 1 2 3 3

A1 A2 A3 A4 B1 B2 B3 B4	☐
A1 A2 A4 A3 B1 B2 B3 B4	☐
A1 A2 A3 A4 B1 B2 B4 B3	☐
A1 A2 A4 A3 B1 B2 B4 B3	☐
Nichts davon gilt	☐

Aufgabe 5

```
1 public class Max {
2   public
3         T max(T x, T y) {
4       if (x.compareTo(y) >= 0) return x;
5       else return y;
6   }
7 }
```

Die Klasse *Max* soll das Maximum für verschiedene Java-Typen definieren. Ergänzen Sie den obigen Quelltext zwischen Zeile 2 und 3, so dass die Klasse *Max* das *Comparable*-Interface verwendet und mögliche Typfehler in Aufrufen von *max* bereits beim Übersetzen dieser Aufrufe erkannt werden.

Eine Klasse *Rational* repräsentiert Quotienten von ganzen Zahlen. Das Maximum von zwei *Rational*-Objekten *a* und *b* soll wie folgt berechnet werden können: `(new Max()).max(a,b);`
Geben Sie **nur den Kopf** (vgl. Zeile 1 der Klasse Max) der Klasse *Rational* an.

Die Klasse *Rational* habe zwei Instanzvariablen **int** *zaehler*, *nenner*; zur Repräsentation von $\frac{\text{zaehler}}{\text{nenner}}$.
Definieren Sie **nur** die Methode der Klasse *Rational*, die notwendig ist, damit *max* das Maximum von zwei *Rational*-Objekten berechnen kann.

```
class MainMax {

  public enum Obst {
    APFEL, BIRNE, KIRSCHE, PFIRSICH, PFLAUME
  }
  public static void main(String[] args) {
    System.out.println((new Max()).max(Obst.KIRSCHE.ordinal(), 3));
  }
}
```

Das Programm wird mit `java MainMax` aufgerufen. Was ist die Ausgabe?

Aufgabe 6

Beschreiben Sie, zu welchen Fehlermeldungen es beim Kompilieren des untenstehenden Programmes kommt. Geben Sie jeweils an, warum der Fehler auftritt und wie dieser am sinnvollsten (d.h. insbesondere ohne Weglassen von Codezeilen) zu beheben wäre.
Einer der insgesamt 5 Fehler ist bereits als Muster angegeben.

```
1  interface Term {
2    public abstract double wert();
3    public abstract String infix();
4  }
```

```
5  abstract class ZweiStelligeOperation implements Term {
6    private Term ersterOperand, zweiterOperand;
7
8    public ZweiStelligeOperation(Term ersterOperand, Term zweiterOperand) {
9      this.ersterOperand = ersterOperand;
10     this.zweiterOperand = zweiterOperand;
11   }
12 }
```

```
13 class Produkt extends ZweiStelligeOperation {
14
15   public Produkt(Term ersterSummand, Term zweiterSummand) {
16     super();
17   }
18
19   int wert() {
20     return ersterOperand.wert() * zweiterOperand.wert();
21   }
22 }
```

Fehler: Instanzvariablen „ersterOperand" und „zweiterOperand" sind nicht sichtbar in Zeile 20.
Abhilfe: In Zeile 6 „private" durch „protected" ersetzen.

Fehler: .....

.....

Abhilfe: .....

.....

Fehler: .....

.....

Abhilfe: .....

.....

Fehler: .....

.....

Aufgabe 7

Untenstehend finden Sie ein Java-Programm, welches einwandfrei kompiliert, aber bei der Ausführung einen Laufzeitfehler auslöst. Geben Sie an, in welcher Zeile der Laufzeitfehler auftritt und von welcher Art (z.B. ArithmeticException oder „Division durch 0“) er ist bzw. wodurch er ausgelöst wird.

```
1  public class Ausgeben {
2
3    public static void main(String[] args) {
4      System.out.println(join(args, ","));
5    }
6
7    private static String join(String[] array, String separator) {
8      String verknuepft;
9      verknuepft = array[0];
10     for (int i = 1; i <= array.length; ++i)
11       verknuepft += separator + array[i];
12     return verknuepft;
13   }
14 }
```

Das Programm wird mit `java Ausgeben Eier Milch Käse` aufgerufen.

Aufgabe 8

```
public class Kraftfahrzeug {
  public String output() {
    return "KFZ";
  }
  public void print() {
    System.out.print("Kraftfahrzeug: ");
    System.out.println(output());
  }
}
```

```
public class Motorrad
             extends Kraftfahrzeug {
  public void print() {
    System.out.print("Motorrad: ");
    System.out.println(output());
  }
}
```

```
class Personenkraftwagen
      extends Kraftfahrzeug {
  public String output() {
    return "PKW";
  }
}
```

```
public class Cabriolet
            extends Personenkraftwagen {
  public void print() {
    System.out.print("Cabriolet: ");
    System.out.println(output());
  }
}
```

```
Kraftfahrzeug k = new Kraftfahrzeug(); Motorrad m = new Motorrad();
Personenkraftwagen p = new Personenkraftwagen(); Cabriolet c = new Cabriolet();
```

Was wird nach diesen Deklarationen mit folgenden Anweisungen ausgegeben?

`k = m; k.print();` // Ausgabe:.....

`k = p; k.print();` // Ausgabe:.....

`k = c; k.print();` // Ausgabe:.....

Jeweils ALLE richtigen Antworten ankreu

Welche Deklarationen sind erlaubt?

`Kraftfahrzeug k1 = new Motorrad();`	☐
`Kraftfahrzeug k2 = new Cabriolet();`	☐
`Personenkraftwagen p3 = new Kraftfahrzeug();`	☐
`Personenkraftwagen p4 = (Cabriolet) new Motorrad();`	☐
chts davon gilt	☐

Aufgabe 9

Gegeben seien zwei Arrays a und b vom Typ int[ ], beide mit der selben Länge n. Die beiden folgenden Algorithmen sollen testen, ob die Arrays disjunkt sind. Für die Komplexitätsanalyse sollen jeweils nur die Vergleiche zwischen Elementen der Arrays gezählt werden.

Algorithmus **A**

```
boolean disjunkt = true;
int i=0, j=0;
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        if (a[i]==b[j]) {
            disjunkt=false;
        }
    }
}
return disjunkt;
```

Algorithmus **B** falls beide Arrays sortiert sind

```
boolean disjunkt = true;
int i=0, j=0;
while (i<n && j<n) {
      if (a[i]<b[j]) { i++; }
         else if(a[i]>b[j]) { j++; }
         else /* a[i]==b[j] */ {
              disjunkt=false;
              i++; j++; }
}
return disjunkt;
```

Wie viele Vergleiche zwischen Array-Elementen macht Algorithmus A? .....

Welche Komplexität (O-Notation) hat Algorithmus A? .....

Wie oft wird die Schleife in Algorithmus B höchstens durchlaufen? .....

Welche Komplexität (O-Notation) hat Algorithmus B? .....

Die Arrays sollen mit InsertionSort sortiert werden. Ergänzen Sie den untenstehenden Algorithmus direkt **im Quellcode** entsprechend.

```
public static void insertionSort(int[] array) {
   int n = array.length;
   int i, j, t;
   for (i=1; i<n; ++i) {
       j=i;
       t=array[j];
   // ab hier ergaenzen
```

```
       array[j]=t;
   }
```

Jeweils ALLE richtigen Antworten ankreuzen.

Welche der folgenden Aussagen treffen auf InsertionSort zu?

InsertionSort ist ein *stabiles* Sortierverfahren.	☐
InsertionSort ist ein *in-place* Sortierverfahren.	☐
InsertionSort ist parallelisierbar.	☐
InsertionSort hat eine *best case* Komplexität von $O(n \log_2(n))$.	☐
Nichts davon gilt	☐