# Musteraufgaben

| Nr. | Aufgabe |
|---|---|
| 1 | **Enum 1** |
| 2 | **Enum 2** |
| 3 | **Generics 1** |
| 4 | **Generics 2** |
| 5 | **Generics 3** |
| 6 | **Annotation** |
| 7 | **Reflection** |

Hinweis:
An der Prüfung dürfen beliebige Papier-Unterlagen verwendet verwendet werden!

## Aufgabe 1: Enum 1

Definieren Sie eine Enumeration (*enum*) für die vier *Jahreszeiten*.
Definieren Sie eine weitere Enumeration für die *Monate* des Jahres.
Ein Objekt vom Typ *Monat* soll man abfragen können wieviele Tage dieser Monat hat und in welcher Jahreszeit er ist.
Folgendes Code-Fragment soll mit Ihren Enum-Definitionen funktionsfähig sein.

```
Monat monat = Monat.FEBRUAR;
System.out.println(monat.name() + " hat " + monat.getAnzahlTage() + " Tage.");
  if (monat.in(Jahreszeit.WINTER)) {
    System.out.println("Dieser Monat ist im Winter.");
  }

/* Session-Log:
FEBRUAR hat 28 Tage.
Dieser Monat ist im Winter.
*/
```

Hinweise:

- Sie müssen nur die Monate JANUAR und FEBRUAR definieren (Februar einfachheitshalber fix mit 28 Tagen).
- implementieren Sie nur soviel wie nötig damit obiges Code-Fragment funktioniert.

Ihre Implementation:

```
enum Jahreszeit {FRUEHLING, SOMMER, HERBST, WINTER}

enum Monat {

  JANUAR  (31, Jahreszeit.WINTER),
  FEBRUAR (28, Jahreszeit.WINTER),
  MAERZ   (31, Jahreszeit.WINTER),
  APRIL   (30, Jahreszeit.FRUEHLING);
  // ... etc. ...

  private int         mTage;
  private Jahreszeit  mJahreszeit;

  Monat(int pTage, Jahreszeit pJahreszeit) {
    mTage = pTage;
    mJahreszeit = pJahreszeit;
  }

  boolean in(Jahreszeit pJahreszeit) {
    return pJahreszeit == mJahreszeit;
  }

  int getAnzahlTage() {
    return mTage;
  }

}
```

## Aufgabe 2: Enum 2

Welche Objekte werden instanziert bei der Ausführung des folgenden Programmes:

```
enum State {CONFIG, INIT, RUNNING, STOPPED}

class EnumTest {

   static void main(String[] pArgs) {

     State currentState = State.CONFIG;

   }

}
```

***Es werden 4 Objekte vom Typ State instanziert.***
***Pro Aufzählungswert eines.***
***Beispiel für Aufzälungswert CONFIG :***
```
public static final State CONFIG = new State();
```

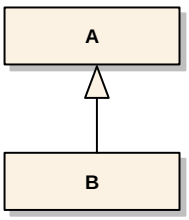
***Hinweis:***
***Es wird kein Objekt EnumTest erzeugt (es wird nur eine statische Methode dieser Klasse ausgeführt)!***

---

## Aufgabe 3: Generics 1

Gegeben:

Die Klasse ***B*** ist *abgeleitet* von der Klasse ***A***.




Im Weiteren folgende Zeile gegeben:

```
List<A> la = new ArrayList<B>();
```

Frage: Ist obige Zeile gültig?

***Nein***

Wenn nein, warum?

***List<B> (oder ArrayList<B>) ist kein Subtyp von List<A>.***

## Aufgabe 4: Generics 2

Gegeben sind folgende Definitionen:

```
class A           { public String toString() {return "A";} }
class B extends A { public String toString() {return "B";} }

List<A> la = null;
List<B> lb = null;
/* ... */
```

Jetzt soll die Methode *meth()* mit den beiden Listen aufgerufen werden:

```
meth(la);
meth(lb);
```

Die Methode *meth()* soll auf allen Objekten der übergebenen Liste die *toString()*-Methode aufrufen und das Resultat auf die Konsole ausgeben.

Implementieren Sie die Methode *meth()* komplett (Hinweis: es sind wenige Zeilen Code) :

```
void meth(List<? extends A> pList) {

  for(A a: pList) {

    System.out.println(a);

  }

}
```

## Aufgabe 5: Generics 3

Gegeben sind folgende Definitionen:

```
class A           { public String toString() {return "A";} }
class B extends A { public String toString() {return "B";} }
class C extends B { public String toString() {return "C";} }

List<A> la = new ArrayList<A>();
addOneBandOneC(la);
List<B> lb = new ArrayList<B>();
addOneBandOneC(lb);
```

Die Methode *addOneBandOneC()* soll der übergebenen Liste genau ein *B*- und ein *C-Objekt* hinzufügen.

Implementieren Sie die Methode *addOneBandOneC()* komplett (Hinweis: es sind wenige Zeilen Code) :

```
void addOneBandOneC(List<? super B> pList) {

  pList.add(new B());

  pList.add(new C());

}
```

## Aufgabe 6: Annotation

Es soll eine Annotation definert werden, mit welcher der gültige *Wertebereich* von Parametern bei Methoden beschrieben werden kann.

Beispiel:
Bei einer Methode `setAlterInJahre(int pAlter)` sei der gültige Wertebereich von `pAlter`:

- min: -1
- max: 130

Die Werte des Wertebereiches (min, max) sollen zur *Laufzeit* des Programmes zur Verfügung stehen.

Definieren Sie eine entsprechende Annotation:

```
@Target(ElementType.PARAMETER)
@Retention(RetentionPolicy.RUNTIME)
@interface ParameterRange {
  int min();
  int max();
}
```

Zeigen Sie, wie diese Annotation bei der Methode `setAlterInJahre()` verwendet würde:

```
void setAlterInJahre(@ParameterRange(min=-1, max=130) int pAlter)
```

## Aufgabe 7: Reflection

Die folgende Methode `loadObject()` soll mittels *Reflection* ein Objekt von der Klasse `pClassName` instanzieren und dann darauf die Methode `pMethName` mit dem Parameter `pDouble` zur Ausführung bringen.

Hinweis:
Sie brauchen keine Fehlerbehandlung zu realisieren.

Implementieren Sie die Methode:

```
void loadObject(String pClassName, String pMethName, double pDouble)
     throws Exception {
```

```
    Class classObj = Class.forName(pClassName);

    Object obj = classObj.newInstance();

    classObj.getDeclaredMethod(pMethName, double.class)
            .invoke(obj, pDouble);
```