Übung 1: *Enum* und *Generics*

Aufgabe 1: Enum

a) Beantworten Sie folgende Frage:

Warum ist bei einem ***enum*** ein Vergleich der Art (`meineEnumFarbVariable == Farbe.GELB`) so ohne weiteres möglich ?

b) Bei einem unserer Kunden wird viel mit Münzen gearbeitet.
Er will in den Programmen mit Ausdrücken in der Art *"FUENFER", "ZEHNER", "ZWANZIGER", "FUENFZIGER", "EIN_FRAENKLER"* und *"ZWEI_FRAENKLER"* arbeiten können ;-)
Im Weiteren haben die Münzen neben der Eigenschaft des Wertes in Rappen noch die Eigenschaft des Gewichtes in Gramm.

Definieren Sie einen entsprechenden ***enum***, welcher diesen Anforderungen genügt.

c) Im weiteren soll auch die Münz-Farbe mit den Werten für das Farb-Tripel "Rot-Grün-Blau" pro Münze definiert sein.
Der Kunde will, dass wenn man eine Münze via *System.out* auf die Konsole ausgibt, der Output für einen Fünfer wie folgt aussieht:
"`FUENFER: Gewicht=2 Farbe:GOLD=154-114-50`"

Erstellen Sie die dazu nötigen *enum*'s sodass folgende Test-Sequenz funktioniert:

```
Muenze meineMuenzVariable = Muenze.FUENFER;
if (meineMuenzVariable == Muenze.FUENFER) {
  System.out.println("Das ist ein FUENFER ;-)");
}
System.out.println(meineMuenzVariable);
```

```
/* Session-Log:

Das ist ein FUENFER ;-)
FUENFER: Gewicht=2 Farbe:GOLD=154-114-50

*/
```


Aufgabe 2: Generics

a) Generics und Collections

Erstellen Sie die Klassen *Person*, *Student* und *PersonenListe*, sodass nachfolgende Sequenz fehlerlos funktioniert gemäss dem *Session-Log.*
Wobei:
- *add()* die Personen nach Namen sortiert in die Liste einfügt.
- *merge()* die eigene mit der übergebenen Liste merged.
- die Klasse *Person* das Interface *java.lang.Comparable* implementiert (um nach Namen sortieren zu können).

```
PersonenListe<Person> personenListe = new PersonenListe<Person>();

personenListe.add(new Person("Hans"));
personenListe.add(new Person("Annelis"));
personenListe.print("Personen-Liste  ");

PersonenListe<Student> studentenListe = new PersonenListe<Student>();
studentenListe.add(new Student("Max", 4711));
studentenListe.add(new Student("Anna", 4712));
studentenListe.add(new Student("Clara", 4713));
studentenListe.print("Studenten-Liste ");

PersonenListe<Person> mergeListe = new PersonenListe<Person>();
int length = personenListe.merge(studentenListe, mergeListe);
System.out.print("Number of merged Persons  : " + length);
mergeListe.print("Merge-Liste       ");

/* Session-Log:
Personen-Liste            : Annelis  Hans
Studenten-Liste           : Anna:4712  Clara:4713  Max:4711
Number of merged Persons  : 5
Merge-Liste               : Anna:4712  Annelis  Clara:4713  Hans
Max:4711
*/
```

Hinweise:
- Implementieren Sie die Klassen minimal.
  Es soll nur soweit funktionieren was gemäss Session-Log benutzt wird.
- mit *java.util.Collections.sort(List<T> list)* können Listen sortiert werden.
- auf dem Skripte-Sever befindet sich die Ausgangslage (*Personen.java*).
  Machen Sie daran die nötigen Anpassungen und Erweiterungen.

b) Legacy -> Generic (optional)

Portieren Sie das File *BinarySearchTree.java* (auf dem Skripte-Server) von Java 1.4 nach Java 1.5.

Ziel: in *BinarySearchTree.main()* soll der der Binary-Tree deklariert werden mit:

```
BinarySearchTree<Integer, String> intTree
```

Am Schluss sollen keine Warnungen mehr vorkommen
(ohne @SuppressWarnings ;-).

