Wichtige Hinweise:

1. Sie müssen 3 Aufgaben bearbeiten.

2. In Ihrem Ruby-Prüfungsprojekt finden Sie vier Ordner ***a1, a2, a3*** und ***a4***. Alle Klassen und Methoden einer Aufgabe gehören in das entsprechende Subverzeichnis.

3. Aufgabe 1 und 2 **müssen** bearbeitet werden. Zwischen Aufgabe 3 und 4 müssen Sie **wählen**.

4. Schreiben Sie die 3 Aufgaben, die gewertet werden sollen auf den Zettel mit dem Prüfungs-Account. **Wenn Sie es vergessen oder unterlassen, werte ich nur die ersten beiden Aufgaben!**

5. Schreiben Sie ***in jedes Script, das Sie bearbeiten, Ihre Matrikelnummer***.

6. Zu erreichen sind ***90 Pkt.***

A1 Einzelaufgaben (Punkte 30 Pkt)

1. Gegeben die Definition der Näherungsformel für $\frac{1}{4}$

   $\frac{1}{4} \approx f\_1\_4tel(n) = \sum_{i=1}^{n} \frac{1}{i*(i+1)*(i+2)}$

   a. Berechnen Sie die Formel **iterativ**. Geben Sie ***false*** zurück, wenn die Methode mit ungültigen Werten für n aufgerufen wird. (***5 Pkt***)

   b. Berechnen Sie die Formel **rekursiv**. Geben Sie ***false*** zurück, wenn die Methode mit ungültigen Werten für n aufgerufen wird. (***5 Pkt***)

   Beispielaufrufe entnehmen Sie bitte den RUnit Tests

2. Schreiben Sie eine Methode ***ary_min_2_elems*** für die Klasse ***Array***, die für ein beliebig geschachteltes Array, die Anzahl der Arrays ermittelt, die mindestens 2 Elemente enthalten. Das äußere Array wird mitgezählt. (***10 Pkt***)

   Beispielaufruf mit Ergebnis:

   ```
   [1,[2,[3,4],[[7,[8,9]]]]].ary_min_2_elems()

   Ergebnis: 5
   ```

3. Gegeben eine Hash, der Schlüssel auf eine Menge von Werten abbildet.

   a. Schreiben Sie eine Methode

   organisiere_nach_wert(a_hash) ==> other_hash

   die einen neuen Hash erzeugt, in dem die Werte auf Mengen von Schlüsseln abgebildet werden.

   Beispiel:

   ```
   h = {:a1 => Set.new([:p1,:p2,:p3,:p4]),
        :a2 => Set.new([:p1,:p4,:p5,:p6]),
        :a3 => Set.new([:p1,:p2,:p3]) }


   organisiere_nach_wert(h)
   ```

   Ergebnis:

   ```
   {:p5=>#<Set: {:a2}>,
    :p1=>#<Set: {:a2, :a1, :a3}>,
    :p6=>#<Set: {:a2}>,
    :p2=>#<Set: {:a1, :a3}>,
    :p3=>#<Set: {:a1, :a3}>,
    :p4=>#<Set: {:a2, :a1}>}
   ```

b. Schreiben Sie eine Methode

schluessel_pro_wert(a_hash) ==> other_hash

Die einen neuen Hash berechnet, in dem die Werte auf die Anzahl der Schlüssel abbilden, die in ***a_hash*** auf Werte zeigen. Sie dürfen die Methode aus 3.a verwenden.

Beispiel:

```
h = {:a1 => Set.new([:p1,:p2,:p3,:p4]),
     :a2 => Set.new([:p1,:p4,:p5,:p6]),
     :a3 => Set.new([:p1,:p2,:p3]) }


schluessel_pro_wert(h)
```

Ergebnis:

```
{:p5=>1, :p1=>3, :p6=>1, :p2=>2, :p3=>2, :p4=>2}
```

Hinweise:

- Schreiben Sie Ihre Lösung in das Skript ***A1.rb*** und verwenden Sie nur die in der Aufgabe genannten Methodennamen.
- ***A1Test.rb*** enthält die Unit-Tests zu den Aufgaben.

A2 Gleichheit und Vergleich (Punkte 30 Pkt)

Gegeben ein einfaches nicht vollständiges Modell von Points of Interest (POI). POI's haben eine Geokoordinate und eine Liste von Informationen unterschiedlicher Medientypen (Attachments). Geokoordinaten werden in Breitengrad und Längengrad angegeben. Ein Attachment hat einen Namen und eine Referenz auf den Medieninhalt (eine URI).

Sie sollen die Klassen ***POI, Geokoordinate*** und ***Attachment*** so erweitern, dass folgende Anforderungen erfüllt werden.

1. Gleiche POI's sollen in Arrays wiedergefunden werden. POI's sind gleich wenn alle Eigenschaften der POI's gleich sind.
2. POI's sollen in Mengen verwaltet werden können, ohne dass Dubletten auftreten.
3. POI's sollen angeordnet werden können. Die Ordnung ist wie folgt definiert: Zunächst entscheidet der Name, dann die Geokoordinate und dann die Sortierung der Attachments. Die Ordnung der Geokoordinate ist wie folgt definiert: Zuerst entscheidet der Breitengrad, dann der Längengrad. Für Attachments ist die Ordnung wie folgt definiert: Zuerst entscheidet der Name und dann der Medieninhalt.

Hinweise:

1. Sie finden die vorbereiteten Klassen im Script ***POI.rb*** Verzeichnis ***a2***
2. ***POITest.rb*** enthält die zugehörigen Tests.

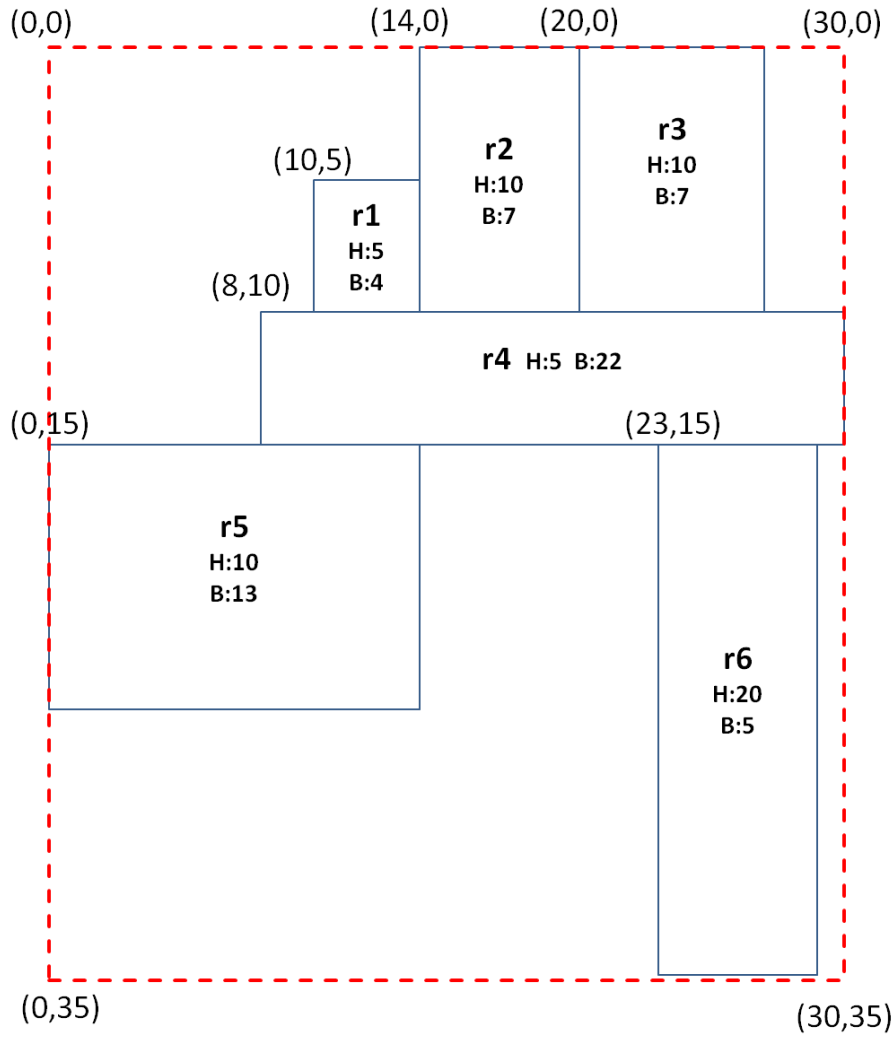
A3 Ein einfaches Stockwerkmodell (30 Punkte)

1. Gegeben ein einfaches Stockwerkmodell eines Gebäudes. ***Stockwerke*** haben eine Ebene und enthalten 1-n Räume. Ein ***Raum*** hat eine Bezeichnung (Symbol) und einen Grundriss, der durch ein Rechteck repräsentiert wird. Sie sollen für die Klasse ***Stockwerk*** die folgenden Methoden schreiben:
    a. ***gesamt_flaeche()***: die Methode berechnet die Gesamtfläche eines Stockwerks (4 Pkt)
    b. ***minimal_umgebendes_rechteck()***: die Methode berechnet das minimale Rechteck, das alle Räume des Stockwerks umgibt. (siehe Zeichnung) (7 Pkt)
    c. ***kleinster_raum()***: die Methode berechnet den kleinsten Raum im Stockwerk. Die Größe eines Raumes ist durch seine Fläche definiert (4 Pkt)
    d. ***anordnen_nach_y_x()***: die Methode liefert ein Array von Räumen zurück, in der die Räume zuerst nach der minimalen y-Koordinate und dann nach ihrer minimalen x-Koordinate sortiert sind. Machen Sie die Objekte der Klasse ***Raum*** dafür geeignet vergleichbar. (6 Pkt)
    e. ***raume_mit_flaeche(flaeche)***: die Methode berechnet eine Liste mit Räumen, deren Flächeninhalt gleich der übergebenen ***flaeche*** ist. (4 Pkt)
    f. ***raum_mit_massen?(raum,toleranz)***: die Methode prüft, ob es im Stockwerk einen Raum gibt, der gleiche ***breite*** und ***hoehe*** wie der übergebene ***raum***, abzüglich /zuzüglich der gegebene ***toleranz*** hat. (5 Pkt)

Hinweise:

- Alle Klassen befinden sich im Script ***Stockwerk.rb***
- Die zugehörigen RUnit-Tests finden Sie in ***Stockwerk Test.rb***.
- Sie müssen die Klasse ***Stockwerk*** **nicht** enumerierbar machen.
- Verwenden Sie geeignete Methoden von Enumerable zur Lösung der Aufgaben.

Beispiel Zeichnung:



A4 Klassen / Objektrekursion / Iteratoren (30 Punkte)

Gegeben ein nicht vollständiges Klassenmodell für einen Baum. Ein Baum besteht aus Knoten, die auf ihre Kindknoten zeigen. Die Blätter des Baumes haben keine Kindknoten. Der Baum hat bereits Methoden zum Einfügen von Inhalten. Sie sollen das Klassenmodell wie folgt erweitern:

1. Definieren Sie eine Gleichheitsrelation (nur Inhaltsgleichheit) für ***Baum*** und ***Knoten***. (5Pkt)
2. Machen Sie den ***Baum*** enumerierbar. Führen Sie den Iterator für den Baum auf einen Iterator für die Klasse ***Knoten*** zurück. Der Block darf dabei **nur auf die Inhalte** der Knoten angewendet werden. (7 Pkt)
3. Schreiben Sie für die Klasse ***Knoten*** eine Methode ***each_node(&b)***: Die Methode iteriert über alle Knoten und wendet den Block auf die Knoten an. (4 Pkt)
4. Schreiben Sie eine Methode ***einfuegen(eltern_knoten_inhalt,inhalt)*** (5 Pkt)
    a. die Methode fügt einen neuen Kindknoten allen Knoten mit Inhalt ***eltern_knoten_inhalt*** hinzu. Verwenden Sie die Methode ***each_node***, um die Elternknoten zu ermitteln, und die Methde ***<<*** der Klasse ***Knoten*** um neue Kindknoten hinzuzufügen. ***<<*** wird ein Inhalt übergeben. Die Methode erzeugt einen neuen Knoten für den Inhalt und fügt den Knoten den Kindknoten hinzu.
    b. die Methode erzeugt einen Fehler (***ArgumentError***), wenn der Typ von ***inhalt*** nicht kompatibel zum Typ ist, der bei der Konstruktion des Baumes übergeben wurde. Verwenden Sie die Methode ***check_typ*** der Klasse ***Baum***.
5. Schreiben Sie eine Methode ***zaehle(inhalt)*** die die Knoten mit Inhalt ***inhalt*** zaehlt. (3 Pkt)
6. Schreiben Sie eine Methode ***alle_inhalte_fuer_bdg(&b)***, die die Knoteninhalte filtert, für die die Bedingung im übergebenen Block zutrifft. (3 Pkt)
7. Schreiben Sie eine Methode ***sortiere***, die die Knoteninhalte sortiert als Array zurückgibt. Definieren Sie eine geeignete Ordnungsrelation in der Klasse ***Knoten.*** Es ist **nicht erlaubt** zuerst ein ***to_a*** auf dem Baum aufzurufen! (3 Pkt)

Hinweise:

- Ergänzen Sie die Klassen, und Methoden des Scriptes ***Baum.rb*** im Verzeichnis ***a4***.
- Die Methoden ***to_s*** in ***Baum*** und ***Knoten*** stellen den Baum formatiert dar. Bitte diese Methoden nicht ändern.
- Die meisten Tests beziehen sich auf gespeicherte Versionen von Bäumen, die mit ***Marshal.load(<name_der_datei>)*** geladen werden. Die Testdaten stehen in den Dateien ***vorlage*** und ***vorlage_integer,*** die im Verzeichnis ***a4*** liegen. Für den äußersten Notfall liegen dort auch Sicherheitskopien.
- Die zugehörigen RUnit-Tests finden Sie in ***BaumTest.rb.***

Baumstruktur in vorlage / vorlage_integer**:**

```
vorlage
            9
        4
    3
            7
        2
1
        7
    2
        5
vorlage integer
    3
1
        5
    2
```