Java -> Haskell

Übersetzen Sie folgende Java Methoden in die entsprechenden Haskell Funktionen. Es folgt ein Beispiel:

Java:

```
public List<Integer> m1(List<Integer> integers) {
    List<Integer> result = new LinkedList<>();
    for (Integer i : integers) { // für jeden Integer in der Liste
        if (i % 2 == 0) {        // wird geprüft ob er gerade ist
            result.add(i);       // falls ja wird er dem Resultat hinzugefügt
        }
    }
    return result;
}
```

Haskell:

1. Versuch

```
m1 :: [Int] -> [Int]
m1 integers = filter isEven integers
              where isEven n = n `rem` 2 == 0
```

Die Funktion `isEven` brauchen Sie nicht selbst zu definieren, die existiert bereits und heisst **`even`.**

2. Versuch

```
m1 :: [Int] -> [Int]
m1 integers = filter even integers
```

Betrachten wir nochmals den Typ der `filter` Funktion.

```
Prelude> :t filter
filter :: (a -> Bool) -> [a] -> [a]
```

`filter` nimmt als erstes Argument eine Funktion, mit der für jedes Element in der Liste geprüft wird, ob es im Resultat auftauchen soll. Was ist der Typ von `filter even`?

```
Prelude > :t filter even
filter even :: Integral a => [a] -> [a]
```

Perfekt! `filter even` gibt eine Funktion zurück, die eine Liste von ganzzahligen Elementen (Int oder Integer) nimmt und gibt eine gefilterte Liste vom gleichen Typ zurück.

Statt in m1 den integers Parameter zu deklarieren und diesen als zweites Argument der filter Funktion zu übergeben

```
m1 ~~integers~~ = filter even ~~integers~~
```

können Sie den Parameter gleich weglassen – `filter even` erwartet ja die Liste bereits:

3. Versuch

```
m1 :: [Int] -> [Int]
m1 = filter even
```

Hübsch oder?

Nun sind Sie an der Reihe!

Übersetzen Sie die folgenden Java Methoden in Haskell Funktionen. Verwenden Sie Funktionskomposition, Lambda Expressions, Patternmatching und Partial Application um möglichst kompakte Lösungen zu erhalten.

a) Java:

```
class Pair<A,B> {
    public final A fst; public final B snd;
    public Pair(A fst, B snd) {
       this.fst = fst; this.snd = snd;
    }
}

List<Float> getGrades(List<Pair<String,Float>> grades){
    List<Float> result = new LinkedList<>();
    for(Pair<String,Float> g : grades) {
        result.add(g.snd);
    }
    return result;
}
```

Haskell:
Hinweis: Verwenden Sie ein normales Haskell Tupel als Ersatz für die Java Klasse Pair.

b) Java:

```
public List<Integer> m2(List<Integer> integers) {
    List<Integer> result = new LinkedList<>();
    for (Integer i : integers) {
        result.add(i * i);
    }
    return result;
}
```

Haskell:
Hinweis: i * i == i^2

c) Java:

```
public List<String> adultNames(List<Pair<String,Integer>> nameAndAge){
    List<String> result = new LinkedList<>();
    for(Pair<String,Integer> p : nameAndAge) {
        if(p.snd >= 18) {
            result.add(p.fst);
        }
    }
    return result;
}
```

Haskell:
Hinweis: Zuerst wollen Sie die Liste filtern und dann übersetzen. Denken Sie daran, dass (m.f) zuerst f anwendet und auf dessen Resultat dann die Funktion m anwendet.