PR1 – Formular für Lesenotizen

WS2020/21

Nachname	Vorname	Matrikelnummer	Abgabedatum:
Abdel Kader	Schehat	1630110	10.12.2020

Collections (L.8.3-L.8.4) & Rekursion (L.9.1)

Lernzielfragen:

a) Wie lautet die Ausgabe des folgenden Codes?

```
TreeSet<Double> s= new TreeSet<Double>();
s.add(5.0);
s.add(4.0);
s.add(4.5);
System.out.println(s);
```

Ausgabe: [4.0, 4.5, 5.0]

b) Warum kann man ein TreeSet<Double> und ein TreeSet<Boolean> anlegen, ein TreeSet<Point> aber nicht?

- Alles was eine lexikographische Ordnung hat kann man an Objekttypen verwenden

c) Schreiben Sie Programmcode, der zu jedem Land die Einwohnerzahl speichert

```
import java.util.HashMap;
public class Ac {
    public static void main(String[] args) {
        HashMap<String, String> einwohner = new HashMap<String, String>();
        einwohner.put("Deutschland", "83 Mio.");
        einwohner.put("Frankreich", "77 Mio.");
        System.out.println(einwohner);
    }
}
```

d) Kann die Datenstruktur der vorstehenden Aufgabe auch invertiert werden? Argumentieren Sie

- Nein, da hier die Länder eindeutig sind, aber mehrere Länder könnten die gleiche Anzahl an Einwohner haben und dann haben wir ein Problem, da es nicht mehr eindeutig ist.
- Man könnte durch logische Verknüpfungen mehrere Länder zusammenfassen,

e) Schreiben Sie eine rekursive Methode, die den Rest der Ganzzahldivision mittels einfacher Subtraktionen und Vergleiche berechnet:

```
import java.util.Scanner;
public class Ae {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Gebe 2 Zahlen ein: ");
        int a = console.nextInt();
        int b = console.nextInt();
        System.out.println("Rest von " + a + " / " + b + ": " + rest(a, b));
    }

    public static int rest(int a, int b) {
        if (a < b) {
            return a;
        }
        return rest(a - b, b);
    }
}
```

Set

$A = (A \cap B) \cup C$

```
HashSet<Integer> A = new HashSet<Integer>();
Collections.addAll(A, 1, 5, 6);
HashSet<Integer> B = new HashSet<Integer>();
Collections.addAll(B, 7, 5, 4, 8, 1);
HashSet<Integer> C = new HashSet<Integer>();
Collections.addAll(C, 6, 1, 9);
A.retainAll(B);
A.addAll(C);
System.out.println(A.toString());
```

Maps

Assoziiert jeden Schlüssel (key) mit einem Wert (value)

Methode	Beschreibung
`clear()`	Entfernt alle Schlüssel und Werte
`containsKey(`*key*`)`	liefert `true`, wenn der gegebene Schlüssel in der Map existiert
`containsValue(`*value*`)`	liefert `true`, wenn der gegebene Wert in der Map existiert
`get(`*key*`)`	Liefert den Wert, der zum gegebenen Schlüssel gehört (`null`, falls nicht gefunden)
`isEmpty()`	liefert `true`, wenn die Map keine Schlüssel oder Werte enthält
`keySet()`	Liefert eine Menge aller Schlüssel
`put(`*key*`, `*value*`)`	Ordnet dem gegebenen Schlüssel den gegebenen Wert zu
`putAll(`*map*`)`	Fügt alle Schlüssel-Wert-Paare aus der gegebenen Map in diese Map ein
`remove(`*key*`)`	Löscht den gegebenen Schlüssel und den zugehörigen Wert
`size()`	Liefert die Anzahl der Schlüssel-Wert-Paare in der Map
`values()`	Liefert eine Collection aller Werte

Maps invertieren:

```
import java.util.HashMap;
public class LesAuf_847 {
    public static void main(String[] args) {
        HashMap<String, String> byName = new HashMap<String, String>();
        byName.put("James Bond", "007");
        byName.put("Chanel", "No. 5");
        byName.put("Notruf", "112");
        byName.put("Porsche", "911");
        System.out.println(byName);

        HashMap<String, String> byPhone = new HashMap<String, String>();
        for (String key : byName.keySet()) {
            byPhone.put(byName.get(key), key);
        }
        System.out.println(byPhone);
    }
}
```

Rekursion: Programmiertechnik, bei der eine Methode sich selbst aufruft

Iteration: Programmiertechnik, bei der man Wiederholung durch Schleifen umsetzt

Rekursive Methoden:

- Einfacher Fall – Problem ohne Rekursion lösbar => & Abbruchbedingung
- Rekursiver Fall – Problem durch Rekursion lösbar => bis zur Abbruchbedingung

⇨ Nachteil: hoher Speicherverbrauch, viel mehr als bei Iteration

Bsp.: ggT Rekursion

```
public class LesAuf_916 {
    public static void main(String[] args) {
        System.out.println(ggT(5, 21));
    }

    public static int ggT(int a, int b) {
        if (b==0) {
            return a;
        }
        return ggT(b, a%b);
    }
}
```