Übung 10 – Wildcards

Arbeiten Sie im Skript die *Kapitel zu Generics (5+6+7)* durch. Suchen Sie sich weitere Informationen, wo Sie die Ausführungen nicht verstehen. Sprechen Sie mit Ihren Kommilitoninnen und Kommilitonen. Fragen Sie im Forum.
Lesen Sie die Aufgaben vollständig und markieren Sie sich zentrale Aspekte. Verwenden Sie **keine Klassen der Java API** außer als Test für Ihre Implementationen und beachten Sie konsistent die Zugriffrechte.

Aufgabe 1 (Theorie)

- Setzen Sie sich zunächst mit den folgenden Klassen der Java-Api auseinander: `Double`, `Integer`,`Number` und `Object`. Zeichnen Sie die Klassen-Hierarchie.
- Lesen Sie jetzt den entsprechenden Absatz zu Wildcards im Skript und recherchieren Sie in der Literatur.
- Betrachten Sie den gegebenen Source-Code. Wo wird der Compiler Probleme anzeigen und warum? Welche Typen können in den entsprechenden Folgen enthalten sein?

```
public class Wildcards{
  public static void main(String[] args){

    // Generische Folgen
    Folge<Object> objectFolge;
    Folge<Number> numberFolge;
    Folge<Integer> intFolge;

    //Upperbounded
    Folge<? extends Object> extendedobjectFolge = intFolge;
    Folge<? extends Number> extendednumberFolge = intFolge;
    Folge<? extends Integer> extendedintFolge =intFolge;

    extendedobjectFolge.insert(new Object());
    extendedintFolge.insert(new Integer(42));

    extendedobjectFolge = numberFolge;
    extendednumberFolge = numberFolge;
    extendedintFolge = numberFolge;

    extendedobjectFolge = objectFolge;
```

```
    extendednumberFolge = objectFolge;
    extendedintFolge =objectFolge;

    // Lower-bounded
    Folge<? super Object> superobjectFolge = objectFolge;
    Folge<? super Number> supernumberFolge = objectFolge;
    Folge<? super Integer> superintFolge = objectFolge;

    superobjectFolge = numberFolge;
    supernumberFolge = numberFolge;
    superintFolge = numberFolge;

    superobjectFolge.insert(new Object());
    superintFolge.insert(new Integer(42));
    superintFolge.insert(new Object());

    superobjectFolge = intFolge;
    supernumberFolge = intFolge;
    superintFolge = intFolge;

    Folge<?> aFolge = objectFolge;
    aFolge = numberFolge;
    aFolge.insert(new Integer(42));
    aFolge = intFolge;
    aFolge.insert(new Integer(42));
  }
}
```

Aufgabe 2

Verändern Sie Ihre generische Klassenmethode `getMinima()` (Variante mit der natürlichen Ordnung), des vorherigen Blattes mithilfe von Wildcards so, dass auch Aufrufe mit zwei Puffern von Elementen gleichen Typs möglich sind, deren natürliche Ordnung in einer gemeinsamen Oberklasse definiert wird.

Testen Sie Ihre Methode mit Puffern von den aus dem Skript bekannten Klassen `Obst`, etc. Was passiert, wenn Sie zwei `Puffer<Apfel>` an die Methode übergeben?

Aufgabe 3

Verändern Sie Ihre generische Klassenmethode `getMinima()` (Variante mit dem Comparator), des vorherigen Blattes mithilfe von Wildcards so, dass Puffer unterschiedlicher Elemente anhand eines Comparators einer gemeinsamen Oberklasse verglichen werden

können. Also beispielsweise ein `Puffer<Student>` und ein `Puffer<Boxer>` anhand eines `Comparator<Person>`.

Aufgabe 4 (Theorie)

Gegeben sei eine Klasse `Theorie`, die eine `main`-Methode. Es werden darin zunächst folgende Variablen deklariert:

```
Person p;
Person p1 = new Student("Schlau","Susanne", 1111);
Person p2 = new Boxer("Tyson","Mike", 100);
Student s1 = new Student("Klug","Karsten", 2222);
Boxer b = new Boxer("Halmich","Regina", 53);
String str = "Hallo";
Object o = new Object();

Puffer<Student> ps1 = new FolgeMitDynArray<>();
Puffer<Student> ps2 = new SchlangeMitEVL<>();
Puffer<Boxer> pb = new FolgeMitDynArray<>();

Folge<Student> fs = new FolgeMitDynArray<>();
Folge<Person> fp = new FolgeMitDynArray<>();
Comparator<Person> compP = new ComparatorPersonName();
```

Anschließend werden die nachfolgenden Anweisungen ausgeführt. Welche dieser folgenden Anweisungen sind gültig, welche führen zu Fehlern? Begründen Sie Ihre Antwort!

```
p = GenericUtil.gambling(p1,p2);
p = GenericUtil.gambling(p1,s1);
p = GenericUtil.gambling(s1,b);
p = GenericUtil.gambling(p1,str);
o = GenericUtil.gambling(p1,str);
o = GenericUtil.gambling(p1,fp);

GenericUtil.insertInto(pb,ps1);
GenericUtil.insertInto(ps1,ps2);
GenericUtil.insertInto(ps1,fp);

fs = GenericUtil.getMinima(ps1,ps2);
fp = GenericUtil.getMinima(ps1,ps2);

fs = GenericUtil.getMinima(ps1,ps2,compP);
fp = GenericUtil.getMinima(ps1,ps2,compP);
fp = GenericUtil.getMinima(ps1,pb,compP);
```

Hinweis: Gehen Sie davon aus, dass die Methode `getMinima` entsprechend der vorigen Aufgabe implementiert sind.

Aufgabe 5

Wir wollen im Folgenden unsere konkrete generische Klasse `FolgeMitDynArray` um zwei Instanzmethoden erweitern.

- Eine Methode `addAll` die eine Referenz vom Typ `Puffer<T>` annimmt und alle Elemente aus dem übergebenen `Puffer<T>` in die Folge einfügt ohne diese aus dem anderen Puffer zu entfernen. (**Hinweis**: Puffer ist seit dem letzten Blatt `Iterable`.)
- Eine weitere Methode `addAllTo` die eine Referenz vom Typ `Puffer<T>` annimmt und alle Elemente aus der Folge in den Puffer einfügt ohne diese zu entfernen.
- Testen Sie beide Methoden mit einer Instanz vom Typ `Puffer<Person>` und einem `Puffer<Boxer>` als Übergabeparameter. *Welche Probleme meldet der Compiler und warum?* Überlegen Sie welche Typen der Compiler für `T` einsetzt.
- *Wie können Sie im Methodenkopf Wildcards anwenden, sodass der Compilerfehler nicht mehr auftritt? Und warum erscheint das sinnvoll?*

Zusatzaufgabe (Programmierübung Projekt Euler)

Wir betrachten alle ganzzahligen Kombinationen von ab für $2 \leq a \leq 5$ und $2 \leq b \leq 5$:

$2^2 = 4, 2^3 = 8, 2^4 = 16, 2^5 = 32$
$3^2 = 9, 3^3 = 27, 3^4 = 81, 3^5 = 243$
$4^2 = 16, 4^3 = 64, 4^4 = 256, 4^5 = 1024$
$5^2 = 25, 5^3 = 125, 5^4 = 625, 5^5 = 3125$

Wenn wir diese numerisch ordnen und alle Wiederholungen entfernen, erhalten wir die folgende Folge mit 15 verschiedenen Gliedern:

$$4, 8, 9, 16, 25, 27, 32, 64, 81, 125, 243, 256, 625, 1024, 3125$$

Wie viele verschiedene Glieder sind in der Folge gebildet durch ab für $2 \leq a \leq 100$ und $2 \leq b \leq 100$?

`https://projekteuler.de/problems/29`

Lösen Sie das Problem in einer Klasse `Euler29`. Geben Sie die Anzahl der Glieder aus.