Listing 1: Uebung_06/src/HigherOrderFunctions.java

```java
import java.util.*;
import java.util.function.Function;
import java.util.function.Predicate;
import java.util.stream.Collectors;

public class HigherOrderFunctions {

    public static <E> List<E> filterNull(List<E> list) {
        return list.stream().filter(Objects::nonNull).collect(Collectors.toList());
    }

    public static <E> int count(List<E> list) {
        return list.stream().reduce(0, (subtotal, element) -> subtotal + 1,
            Integer::sum);
    }

    public static <E extends Comparable<? super E>> Optional<E> min(List<E> list) {
        return list.stream().reduce((x, y) -> x.compareTo(y) < 0 ? x : y);
    }

    public static <E> List<E> takeWhile(List<E> list, Predicate<? super E> predicate) {
        List<E> result = new ArrayList<>();
        for (E item : list) {
            if (!predicate.test(item)) break;
            result.add(item);
        }
        return result;
    }

    public static <E> List<E> skipWhile(List<E> list, Predicate<? super E> predicate) {
        boolean keep = false;
        List<E> result = new ArrayList<>();
        for (E item : list) {
            if (!keep && !predicate.test(item)) {
                keep = true;
            }
            if (keep) result.add(item);
        }
        return result;
    }

    public static <E, K> Map<K, List<E>> group(List<E> list, Function<? super E, ?
        extends K> groupingFn) {
        return list.stream().collect(Collectors.groupingBy(groupingFn));
    }

    static class House implements Comparable<House> {
        private String address;
        private String city;
        private double price;

        public String getAddress() {
            return address;
        }

        public String getCity() {
            return city;
        }

        public double getPrice() {
            return price;
        }
        public void setAddress(String address) {
            this.address = address;
        }

        public void setCity(String city) {
            this.city = city;
        }

        public void setPrice(double price) {
            this.price = price;
        }

        public House(String address, String city, double price) {
            this.address = address;
            this.city = city;
            this.price = price;
        }

        @Override
        public int compareTo(House other) {
            return Double.compare(this.price, other.price);
        }

        @Override
        public String toString() {
            return address + "␣?␣" + String.format("%.2f?", price);
        }
    }

    // Existing higher-order functions...

    public static void main(String[] args) {
        List<House> houses = List.of(
                new House("Hummerstraße␣12", "Linz", 340000),
                new House("Meixnergasse␣1a", "Wels", 480000),
                new House("Flammweg␣2", "Wels", 800000),
                new House("Maria-Hilfer-Straße␣17", "Wien", 5700345),
                new House("Landstraße␣10", "Linz", 950000),
                new House("Herrengasse␣5", "Graz", 650000),
                new House("Hauptplatz␣1", "Linz", 1050000)
        );

        // 1. House with the lowest sale price
        System.out.println("House␣with␣the␣lowest␣price:␣" + min(houses).orElse(null));

        // 2. Take houses until the price exceeds 1,000,000?
        List<House> expensiveHouses = takeWhile(houses, house -> house.price <=
            1000000);
        System.out.println("Houses␣until␣price␣>␣1,000,000?:␣" + expensiveHouses);

        // 3. Skip houses until one is in Linz
        List<House> remainingHouses = skipWhile(houses, house ->
            !house.city.equals("Linz"));
        System.out.println("Houses␣after␣first␣in␣Linz:␣" + remainingHouses);

        // 4. Group houses by city
        Map<String, List<House>> groupedHouses = group(houses, house -> house.city);
        System.out.println("Houses␣grouped␣by␣city:");
        groupedHouses.forEach((city, cityHouses) -> {
            System.out.println(city + "␣->␣" + cityHouses);
        });
    }
}
```