Programming in Java (23/24)

– Exercises Day 9 –

Learning goals

Before the next day, you should have achieved the following learning goals:

- Write generic classes with wildcards where sensible
- Work with interfaces and classes from the Java Collections Library
- Understand the requirements on your classes to store them in Java Collections

1 Pair with wildcards

Consider again Exercise 5 of Day 8 with the generic Pair<...> class. In case your constructor with one parameter does not use wildcards in its parameter type, introduce wildcards in its generic type. Can you think of a variable initialisation with a call to the constructor that is now possible, but was not possible before?

2 Different sets, different orders

Add the following Strings to a HashSet<String> in the given order:

"Java", "C", "PHP", "JavaScript", "Python", "Rust", "Haskell", "Go", "Prolog"

Now print the HashSet<String> on the screen. Why are the Strings in the output in this order?

Repeat the above with the same Strings and first a LinkedHashSet<String>, then a TreeSet<String>.

3 Equal phones

Override method equals(Object) from class Object in your classes OldPhone, MobilePhone, and SmartPhone that you creased on earlier days. An OldPhone is equal to another object if the reference is not null, the classes of both objects are the same, and they both have the same brand. A MobilePhone is equal to another reference if the reference is not null, the class of both objects are

the same, they both have the same brand, and they both have the same call history. A `SmartPhone` is equal to another object if the reference is not null, the class of both objects are the same, they both have the same brand, and they both have the same call history. (Recall the DRY principle!)

Hint: for implementing method `equals(Object)`, consider using the final method `getClass()` that all your classes inherit from class `Object`.

4   Hash set weirdness

After you have solved the previous exercise, write a script that creates two `OldPhone` objects. Both phones should have the brand "ACME". Check whether the two phones are equal. Put one of them into a new `HashSet<Phone>` or `LinkedHashSet<Phone>`. Check whether the set contains the other phone. What happens? Why?

5   Implementing hashCode()

In class `OldPhone`, implement method `hashCode()` in a way that is consistent with method `equals(Object)` of `OldPhone`. Make sure that two different `OldPhone` objects are very likely to have different hash codes. Rerun your script from the previous question. What happens? Why?

6   Implementing hashCode() in subclasses

In class `MobilePhone`, implement method `hashCode()` in a way that is consistent with method `equals(Object)`.

7   Points in sets

Recall class Point.

```java
/**
 * Implementation of the geometrical concept of a point in two dimensions.
 * Provides methods to access the coordinates as well as to move a point.
 */
public class Point {
    private int x;
    private int y;

    /**
     * Constructs a new Point with the given coordinates.
     *
     * @param x the x coordinate of the new Point
     * @param y the y coordinate of the new Point
     */
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    /**
     * Getter for the x coordinate of this Point.
     *
     * @return the x coordinate of this Point
     */
    public int getX() {
        return x;
    }

    /**
     * Getter for the y coordinate of this Point.
     *
     * @return the y coordinate of this Point
     */
    public int getY() {
        return y;
    }

    /**
     * Changes the coordinates of this Point to be the same as those of remote.
     *
     * @param remote the Point to which we want to move this Point
     */
    public void moveTo(Point remote) {
```

```
44          this.x = remote.x;
45          this.y = remote.y;
46      }
47  }
```

We want to be able to do the following:

- Use instance method equals(Object) (originally defined in class Object) in class Point such that a Point object is equal to another object if and only if that other object is also an instance of class Point, and both points have the same x coordinates and y coordinates, respectively. (Recall from the JavaDoc https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Object.html#equals(java.lang.Object) that someObject.equals(**null**) is always supposed to return **false**.)

  The code snippet

```
1          Point p1 = new Point(3, 4);
2          Point p2 = new Point(3, 4);
3          System.out.println("p1.equals(p2): expected true; actual "
4                          + p1.equals(p2));
5          List<Point> points = new ArrayList<>();
6          points.add(p1);
7          System.out.println("points.contains(p2): expected true; actual "
8                          + points.contains(p2));
```

  should have the following output:

```
p1.equals(p2): expected true; actual true
points.contains(p2): expected true; actual true
```

- With the above understanding of object equality, add and retrieve Point objects from/to collections of type HashSet<Point> and LinkedHashSet<Point>.

  The code snippet

```
1          Point p1 = new Point(3, 4);
2          Point p2 = new Point(3, 4);
3          Set<Point> set1 = new LinkedHashSet<>();
4          set1.add(p1);
5          System.out.println("set1.contains(p2): expected true; actual "
6                          + set1.contains(p2));
```

  should have the following output:

```
set.contains(p2): expected true; actual true
```

- With the above understanding of object equality, add and retrieve Point objects from/to collections of type TreeSet<Point>.

  The code snippet

```
1            Point p1 = new Point(3, 4);
2            Point p2 = new Point(3, 4);
3            Set<Point> set2 = new TreeSet<>();
4            set2.add(p1);
5            System.out.println("set2.contains(p2): expected true; actual "
6                              + set2.contains(p2));
```

should have the following output:

```
set2.contains(p2): expected true; actual true
```

What do you need to add to class Point for each of the three requirements to make this possible?

8   Working with maps

Write a generic static method count that takes a generic Collection<? **extends** E> as input and returns a Map<E, Integer> as output which maps each entry of the input collection to the number of times it occurs in the input collection.

Would it be a good idea to return a Map<? **extends** E, Integer> instead?

9   Big enough redux

Recall Exercise 11 from Day 5, where your task was to write a small program to store the names and ID numbers of employees for a company *without knowing in advance how many entries you would need*. Now revisit the exercise and try to solve it using suitable interfaces and classes from the Java Collections framework.

What do you observe about the length and complexity of your code now compared to your code from Day 5?

10   Generics with wildcards and subclasses (*)

We want to make the method from Exercise 7 of Day 8 more flexible so that we can call the method both with a List<Animal> and with a List<Dog> as actual parameter. To this end, we add a wildcard to the method parameter type:

```
1    public static Animal doSomething(List<? extends Animal> animals) {
2        return animals.get(0);
3    }
```

Does the compiler accept the modified method? Why (not)?

Would the compiler now let us add a line animals.add(**new** Dog()); or a line animals.add(**new** Animal()); at the beginning of the method body? Why (not)?

Hint: Mind the cats!