Learning goals

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

- Create generic classes
- Use generic classes
- Overload methods
- Prevent code repetition in overloaded methods
- Upcast objects to more general types
- Downcast objects to more specific types

1 Don't Repeat Yourself

Look at the following code. Is there anything you can do to make this code better? Hint: you may need to convert between types (e.g. casting).

```
public class Comparator {
    public int getMax(int n, int m) {
        if (n > m) {
            return n;
        } else {
            return m;
        }
    }
    public double getMax(double d1, double d2) {
        if (d1 > d2) {
            return d1;
        } else {
            return d2;
        }
    }
    public String getMax(String number1, String number2) {
        int n1 = Integer.parseInt(number1);
        int n2 = Integer.parseInt(number2);
        if (n1 > n2) {
            return number1;
        } else {
            return number2;
        }
    }
}
```

2 Upcasting, downcasting

For this exercise, you will need to use again some classes and interfaces you created last day: `Phone`, `OldPhone`, `MobilePhone`, `SmartPhone`.

2.1 Start

Create a script that builds a new `SmartPhone` with the line:

```
Smartphone myPhone = new Smartphone();
```

and then uses all its methods.

2.2 Direct upcasting

Change the script so that the `SmartPhone` is created with the line:

```
Mobilephone myPhone = new Smartphone();
```

Compile your code again. Are there any problems? Why do this problems happen? What are the possible solutions?

2.3 Indirect upcasting when calling a method

Pass this object to a method `testPhone(Phone)` that has only one parameter of type `Phone`. What methods can you call on the object inside the method?

2.4 Downcasting

Inside the former method, downcast the object to `Smartphone` so that you can use all the public methods of `Smartphone`.

2.5 Casting exception

Create a `MobilePhone` object and then pass it to method `testPhone(Phone)`. What happens?

3 Generic list

Modify the doubly-linked list that you have created in past weeks to make it generic, i.e. to allow it to have values of its elements of any type.

Once you have it ready, create a class Company that keeps two linked lists, one with the names of the employees and one with their National Insurance Number.

4 Sorted list (*)

Extend your class from the former exercise to create a sorted list. You may need to override the method that adds new elements to the list. The subclass should be generic.

5 Generic stack

Create a generic stack (with methods for pushing, poping, and checking emptiness) that only works with classes that extend `Number` (e.g. Integer and Double, but not String).

6 Generic maps

6.1 Simple map (*)

Create a generic simple map (with methods for putting a key–value pair, getting the value for a key, and removing a key). The key and the value may be any type, and they may be different. Each key can only be linked to one value.

For simplicity, assume that your map can hold a maximum of 1000 pairs. This way, you can use the hashing method you developed in past weeks and base your map on an array.

6.2 Hash table (*)

Create a generic map (with methods for putting a key–value pair, getting the value for a key, and removing a key). The key and the value may be any type, and they may be different. Under each key, the hash table can store any number of values associated to that key.

For simplicity, assume that your map can hold a maximum of 1000 pairs. This way, you can use the hashing method you developed in past weeks and base your map on an array.