Learning goals

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

- Understand the concept of inheritance
- Extend classes
- Override methods
- Use `super` both for method calling and construction
- Understand the use of `final` for classes and methods.
- Understand the meaning `private`, `public`, and `protected`.

Note: Many exercises below instruct you to create methods. Unless the exercise description says otherwise, a very simple implementation (e.g. just printing something on screen) will be enough. The point today is on practicing inheritance, not over-complicated algorithms for smartphones, musical instruments, etc.

1 Extension, extension...

Create a class `OldPhone` that implements the following interface.

```
/**
 * A phone makes calls
 */
public interface Phone {
    /**
     * Just print on the screen: "Calling <number>...".
     */
    void call(String number);
}
```

Now create a class `MobilePhone` that extends `OldPhone` and adds methods for things like `ringAlarm(String)` and `playGame(String)`. This class keeps a list of the last ten numbers that have been called, which can be printed with the method `printLastNumbers()`.

Then create a class `SmartPhone` that extends `MobilePhone` and adds methods for `browseWeb(String)` and `findPosition()`, the latter returning a (fictitious) GPS-found position.

Create a small script called `PhoneLauncher` in which you create a `SmartPhone` and use all its methods, including those inherited from its ancestor classes.

```
public class PhoneLauncher {
    public static void main(String[] args) {
        PhoneLauncher launcher = new PhoneLauncher();
        launcher.launch();
    }
    public void launch() {
        // your code creating and using SmartPhone here...
    }
}
```

2 Overriding

Save money by routing your international calls through the internet! Modify your class `SmartPhone` so that it overrides the method `call(String)` inherited from `OldPhone`. If the string parameter starts with "00", the method should output "Calling <number> through the internet to save money"; otherwise, the method should do the same as the original method (hint: use `super`).

3 Passing information to ancestor classes

Add the following field, constructor, and method to `OldPhone`:

```
private String brand = null;
public OldPhone(String brand) {
    this.brand = brand;
}
public String getBrand() {
    return brand;
}
// ... there is no setter for brand
```

Add the appropriate constructors to `MobilePhone` and `SmartPhone` in order to be able to call the method `getBrand()` from an object of class `SmartPhone` and obtain the right answer, i.e. the brand provided in the constructor.

4 Visibility

4.1 Increasing visibility ???

Change the visibility of `playGame(String)` to `private` and check whether the script you wrote in the former exercise still works. Why does this happen? What are the minimal changes that you need to make on class `SmartPhone` so that the script still works?

4.2 Reducing visibility

Some parents are concerned that their children spend too much time playing with their smartphones. Create a class `RestrictedSmartPhone` that overrides `playGame(String)` to make it `private` and thus non-visible to external classes and scripts. Is this possible? Why?

5 Multiple inheritance

Create a class `MusicalInstrument` with a method `play()`. Now create another class `WoodenObject` with a method `burn()`.

Create a class `Guitar` that is at the same time a musical instrument and a wooden object. How would you do it in Java?

6 Java magic

Can you see what is wrong in the following code (most JavaDoc comments ommited for the sake of space)?

```
// Teacher.java
public class Teacher {
    private String name;
    public Teacher(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void teach(String lessonName) {
        System.out.println("Teaching lesson: " + lessonName);
    }
}
(...)
```

```
// Lecturer.java
/**
 * A lecturer has both teaching and research responsibilities
 */
public class Lecturer extends Teacher {
    public void doResearch(String topic) {
        System.out.println("Doing research on: " + topic);
    }
}
```

If it is not evident, try to compile the code.

If it compiles without problems, write a script that creates an object of class `Lecturer` and uses its two methods. If it does not, modify class `Lecturer` so that the program compiles, and then write the script to use its two methods.

7 Final means no change

Create a class that extends `String` and adds a method `printEven()` that prints on screen the even-numbered characters of the string. Try to compile it and see what happens.

8 Noah's Ark (*)

Design and implement an application that represents the day that Noah's Ark was open, just before the rain started. In your script, create an animal of each species and then call them all in. Every animal must implement a method `call()` that prints on screen the appropriate message. You should keep in mind the following requirements:

- The application should contain at least: bears, beetles, cats, crocodiles, dogs, dolphins, eagles, flies, frogs, lizards, monkeys, pigeons, salmon, sharks, snakes, whales.
- All animals have at least a method `call()` and a method `reproduce()` (for after the rain).
- Insects, fish, amphibians, reptiles, and birds lay eggs (`layEggs()`). Mammals cannot lay eggs but give birth[1] (`giveBirth()`). The method `reproduce()` should call the appropriate method in each case.
- When called, all animals answer (i.e. print on screen) "<name of the animal> coming...". The exceptions are aquatic animals, which are not affected by the rain and answer "<name of the animal> will not come..."; and flying animals, that answer "<name of the animal> now flying, will come later when tired...". Method `call()` **must not** be implemented in every class: use inheritance to reuse methods and constructors to pass information to parent classes.
- All animals make a sound. If Animal is an interface in your design, `makeSound()` must be a method in there; if `Animal` is an abstract class, it must be an abstract method. The method can then be implemented by descendant classes.

[1] There were no platypus in Noah's Ark.