This session is mainly concerned with abstract classes and interfaces, besides a brief view of `enum` types. In particular, we see how abstract classes and interfaces are useful, also looking at more example on polymorphism.

Polymorphism is strictly related to the *upcasting* of an object of a derived class to a reference of the base class. In this way, it can be treated as an object of the base class at compilation time, allowing us for example to pass it to methods that accept references to object of the base class in the argument list. To have a better understanding of how upcasting works, we look at the code in

`com.andreamazzon.session5.explicitupcasting.goalkeepers`.

The last time we saw an example of polymorphism in `com.andreamazzon.session4.polymorphism.shapes`. However, we saw that we had to give a dummy implementation of the method `computeArea()` in the base class `Shape`, because we are not able to compute the area for a general shape, but only for a specific shape like `Circle`, `Square` or `Triangle`.

When the base class has a method which can be implemented in a meaningful way only in derived classes, we define the method to be `abstract`, and the class to be `abstract` as well: we just give the returning type, the name and the argument list of the method, without implementation. We see a first example of how this works in

`com.andreamazzon.session5.abstractclasses.basicexample`,

and we look at how this can lead to a better solution of the shape example in

`com.andreamazzon.session5.abstractclasses.shapes`.

In both the examples you see again polymorphism: we have methods whose argument list is represented by object of the interface type, accepting anyway objects of the classes implementing those interfaces.

But what if we want a class to inherit from two parent classes? This is not possible, mainly for a reason highlighted in

`com.andreamazzon.session5.nomultipleinheritance`.

The two parent classes could indeed have both a method with same name, same argument list and same implementation. In this case, it would be impossible to say which implementation should be inherited from the derived class.

This would not be a problem, however, if the two parent classes are *fully abstract*, i.e., if they do not have any implemented method, so, only abstract methods. Interfaces are what we are looking for. An interface allows the creator to establish the form for a class: method names, argument lists, and return types, but no method bodies. An interface can also contain fields, but these are implicitly static and final. In this sense, an interface provides only a form, but no implementation. We write that a class `implements` an interface, and you can think about this as if it `extends` the fully abstract class represented by that interface. Any class *implementing* an interface must give the implementation of all the methods defined in the interface, or define them to be `abstract`. Moreover, they have to be `public`.

We see a first example of use of interfaces in

`com.andreamazzon.session5.interfaceexample`,

also providing you the syntax you have to use when you construct classes implementing an interface.

In

`com.andreamazzon.session5.multipleinterfaceimplementation`

we see that it is possible for a class to implement more than one interface, something that is prevented to classes with inheritance. Not only: as we see in

`com.andreamazzon.session5.inheritinginterfaces`,

an interface can extend more that one interface. However, problems may arise when a class implements two interfaces which both have a method with same name, same argument list but different return type. We see an example of such a behaviour in

`com.andreamazzon.session5.nameclasheswithinterfaces`.

At this point, we do an excursion from interfaces in order to look at a new type, i.e., `enum`: this is useful when you have to group together and use a set of constant values (enumerated types, typically strings), and you can see a first example in

`com.andreamazzon.session5.enumtype.currencies`.

Another example of `enum` type, combined with interfaces and polymorphism, can be found in

`com.andreamazzon.session5.interfaceandpolymorphism.band`.

Time permitting (otherwise this will be the first code discussed in session 6) we will have a look at a bit more complicated example where inheritance from `abstract` classes and composition are combined together. This can be found in

`com.andreamazzon.session5.abstractclasses.simulators`

and

`com.andreamazzon.session5.abstractclasses.usingsimulators`.

For every package listed here, you can find a more detailed description of the code itself and of the main topic shown in the `package-info` file.