Object composition

A field of an object may itself be an object.
Remember that a user-defined type (well, a user-defined class in Java) can be used inside a class that we define? We will say that an object can be composed into another object.

In fact, we will define a class named Customer first and use it as we define another class named Account2, which is a revised version of Account.

(See the owner field of Account2. The type of the owner field is Customer. So, a Customer object is being composed into an object of type Account2. That is, if you create an Account2 object, in it you will find a Customer object composed into it.)

See how Customer is used in Account2.java as well as in UseAccount2.java.

- See Customer.java
- See Account2.java
- See UseAccount2.java

We will see how memory diagrams change when we create Customer class and use it inside Account2.
What would be an alternative if we were to not use composition? Well, we would have to include the fields that make up Customer inside Account2. If Customer is a class that is used only in Account2, that may not be too bad although it still violates the principle of abstraction. However, if Customer is a class that is useful in (many) other classes, then you would have to duplicate the fields of Customer in each of the other classes. That is only half the story still – what if you decided that the fields need to be changed at some point at a later time? You would have to visit each of those classes and make the same modifications everywhere!

Object composition is a very important design principle that people use as they design complex software. It is a means of dealing with complexity and any interesting software is complex.