

CSSE2003

Software Engineering Studio

Lecture 9 UML Sequence Diagrams

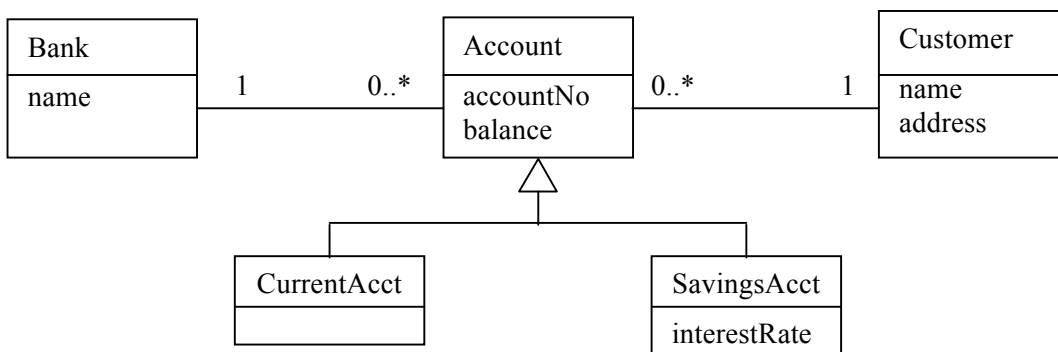
Supplementary Materials Sample Solutions

Semester 2, 2009

School of Information Technology and Electrical Engineering
The University of Queensland

Exercise 1

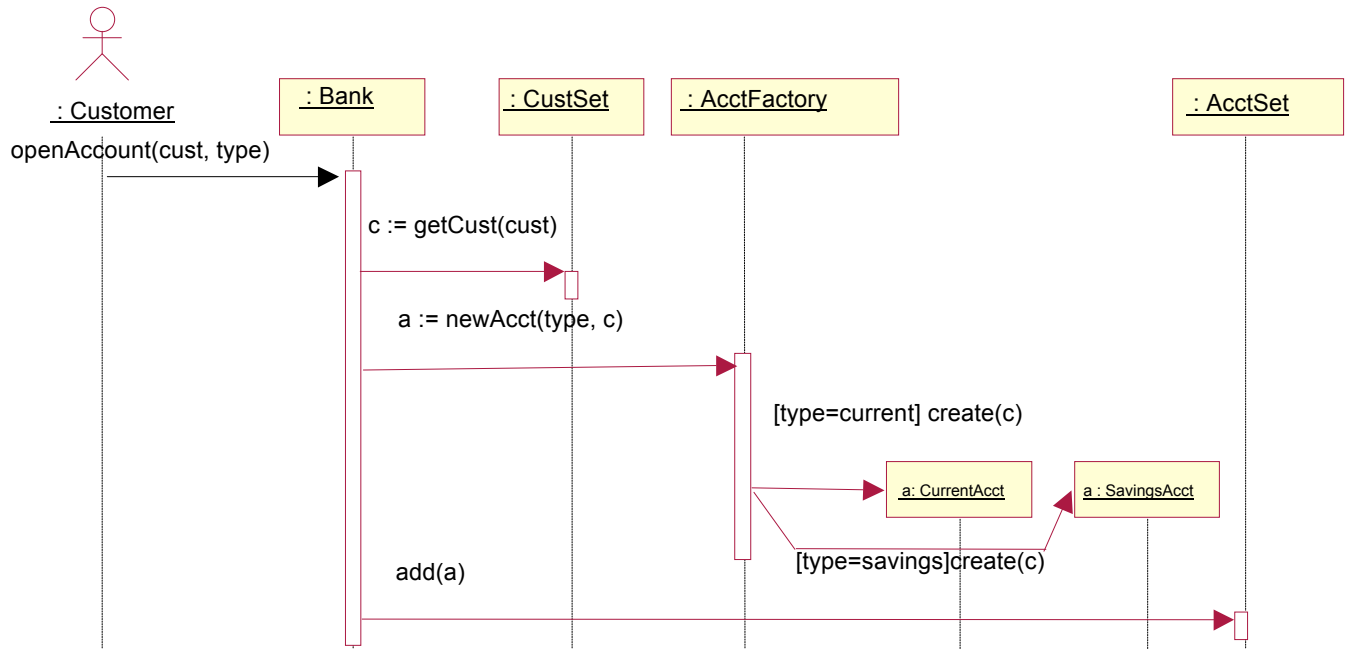
This exercise is based on Chapter 10 of “Developing Java Software” by Winder and Roberts. A bank manages a collection of customer accounts of which there are several different kinds, such as current account and savings account. Some kinds of accounts pay interest. Accounts may be opened or closed. Money may be deposited and withdrawn from an account or the current balance requested. The UML domain model is



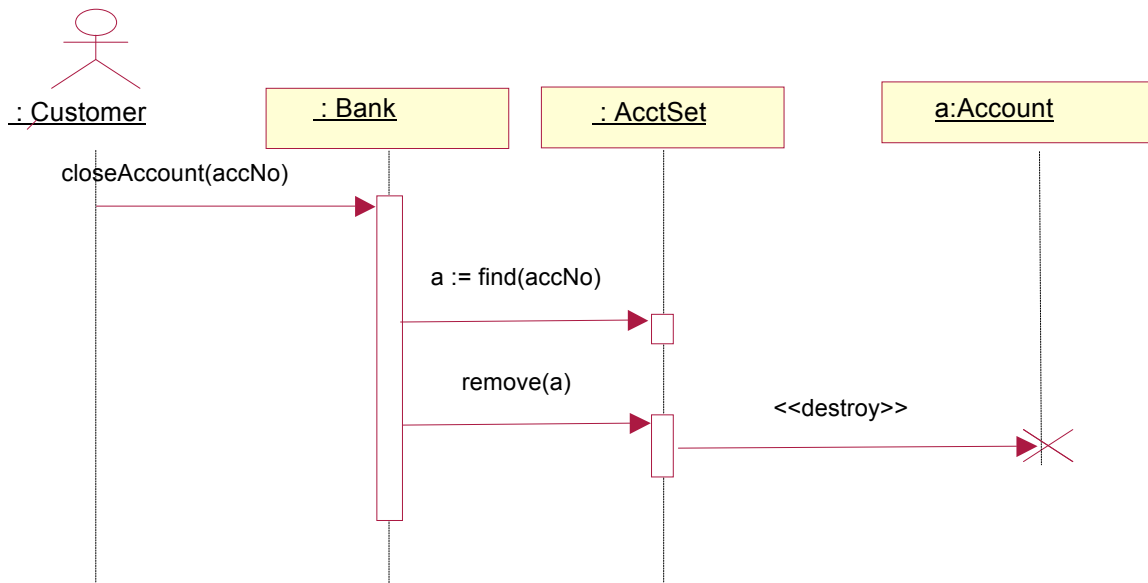
Draw sequence diagrams for the following banking operations, *openAccount(cust,type)*, *closeAccount(accNo)*, *checkBalance(accNo)*, *depositMoney(accNo,amt)* and *withdrawMoney(accNo,amt)*. You will need to introduce new classes (beyond the above domain model) in your design.

Note that the following sequence diagrams are *sample* solutions only.

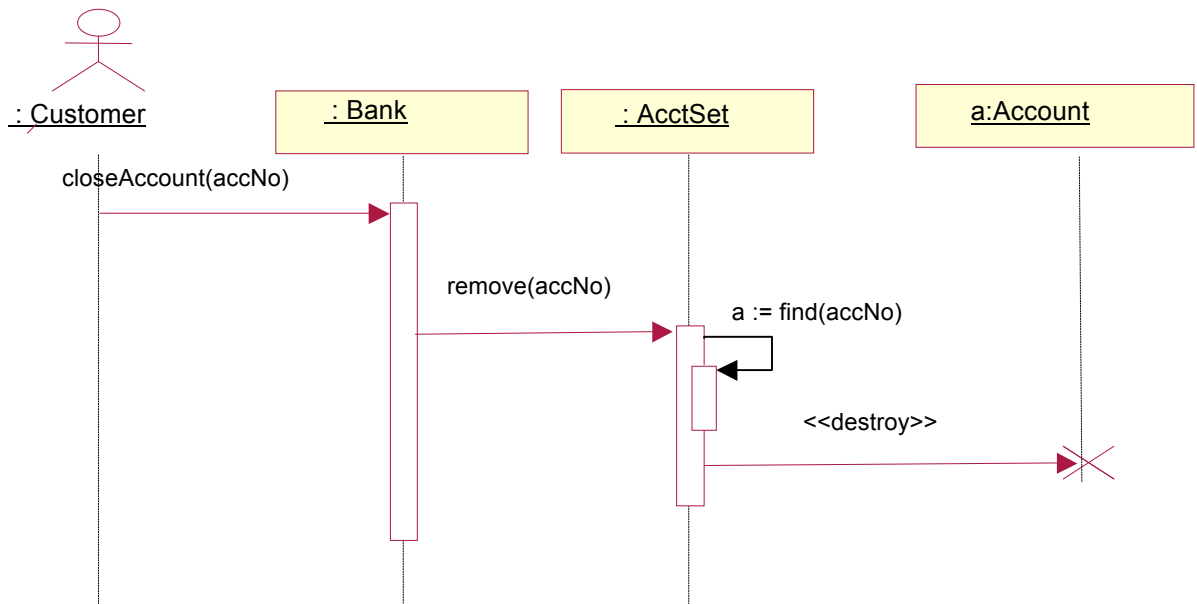
Open a new account



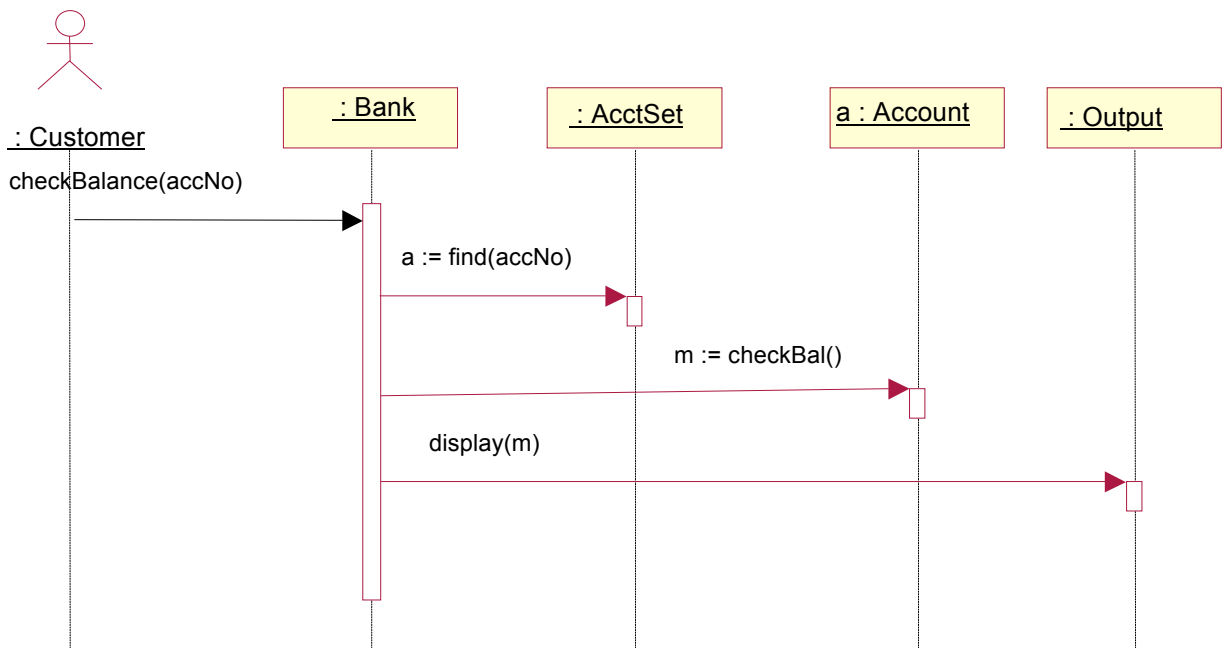
Close an existing account



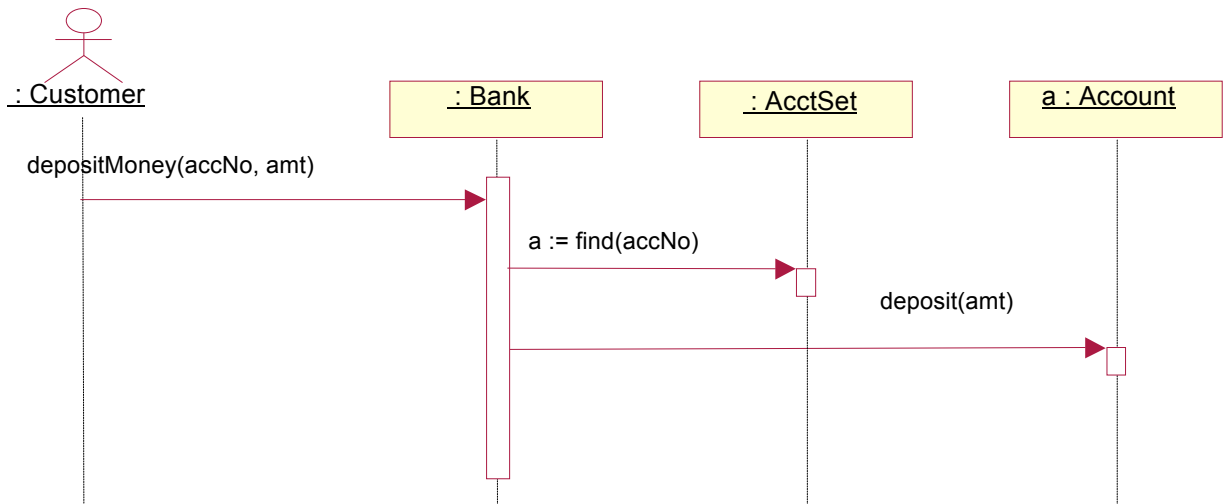
An alternative solution is to have the AcctSet object remove the account by calling find on itself first.



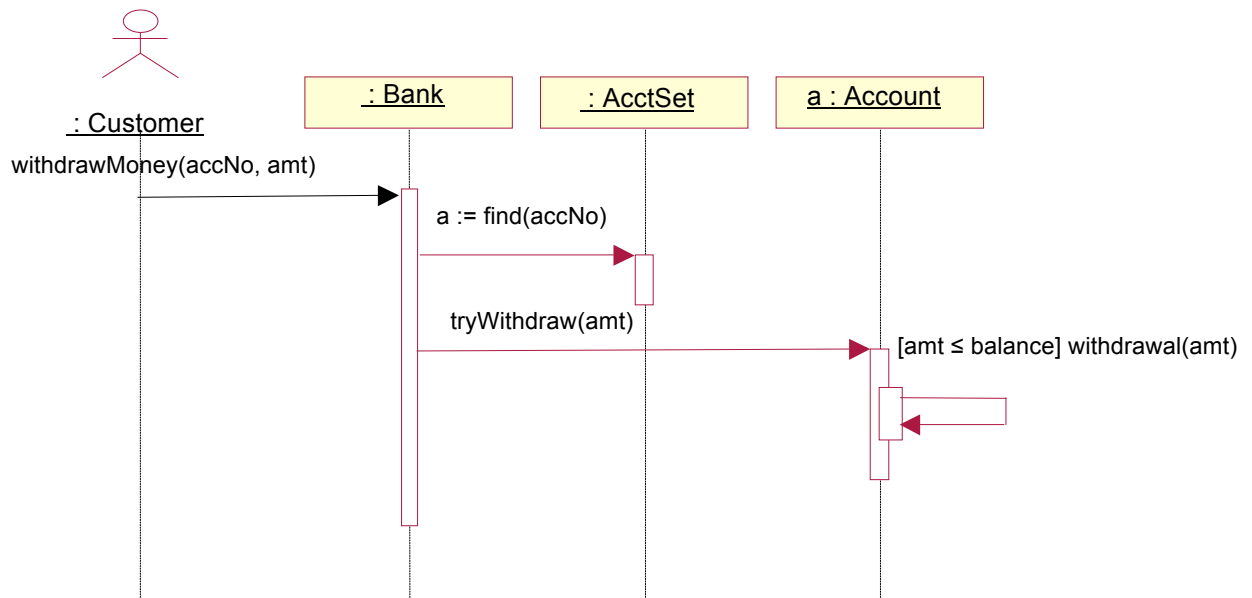
Check the balance of an account



Deposit money into an account



Withdraw money from an account



Exercise 2

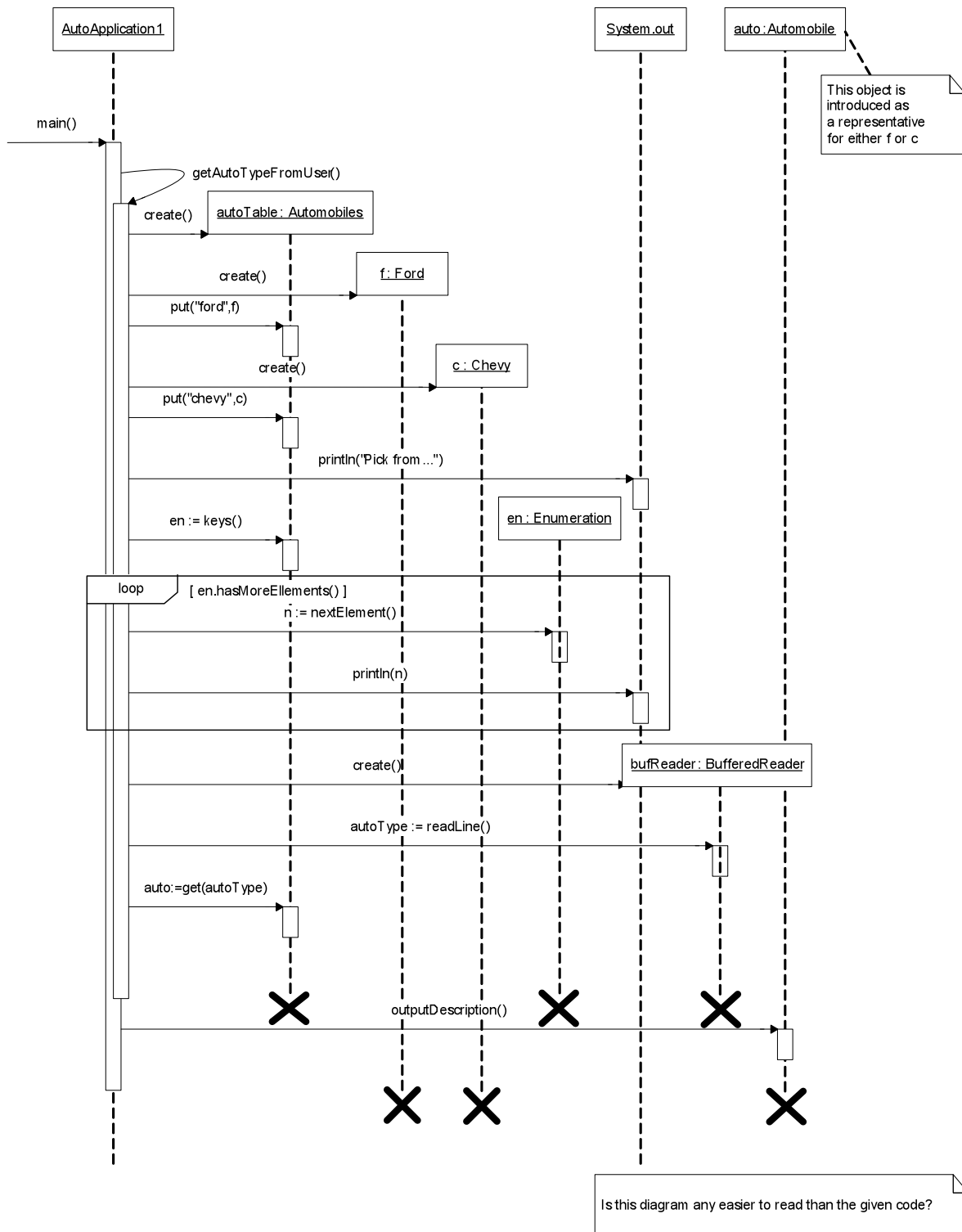
Draw a sequence diagram for these *AutoApplication* classes. Remember that the purpose of a sequence diagram is not to draw an algorithm in full detail but to present an abstraction that shows how participating objects and classes interact. In this case, the participation is in the service of a use case in which a user specifies an automobile of interest and the *AutoApplication* provides a description of it. This question can be answered at various levels of abstraction.

```
class AutoApplication1 {  
  
    private static Automobile auto;  
  
    private static void getAutoTypeFromUser() throws IOException {  
  
        Automobiles autoTable = new Automobiles();  
        autoTable.put("ford", new Ford() );  
        autoTable.put("chevy", new Chevy() );  
  
        System.out.println( "Pick from the following:" );  
        Enumeration en = autoTable.keys();  
        while (en.hasMoreElements()) {  
            System.out.println(en.nextElement());  
        }  
        BufferedReader bufReader =  
            new BufferedReader( new InputStreamReader(System.in) );  
        String autoType = bufReader.readLine();  
        auto = (Automobile)autoTable.get(autoType);  
        if(auto == null) {  
            System.out.println("A ford is assumed.");  
            auto = new Ford();  
        }  
    }  
  
    public static void main( String[] args ) throws IOException {  
        getAutoTypeFromUser();  
        auto.outputDescription();  
    }  
}
```

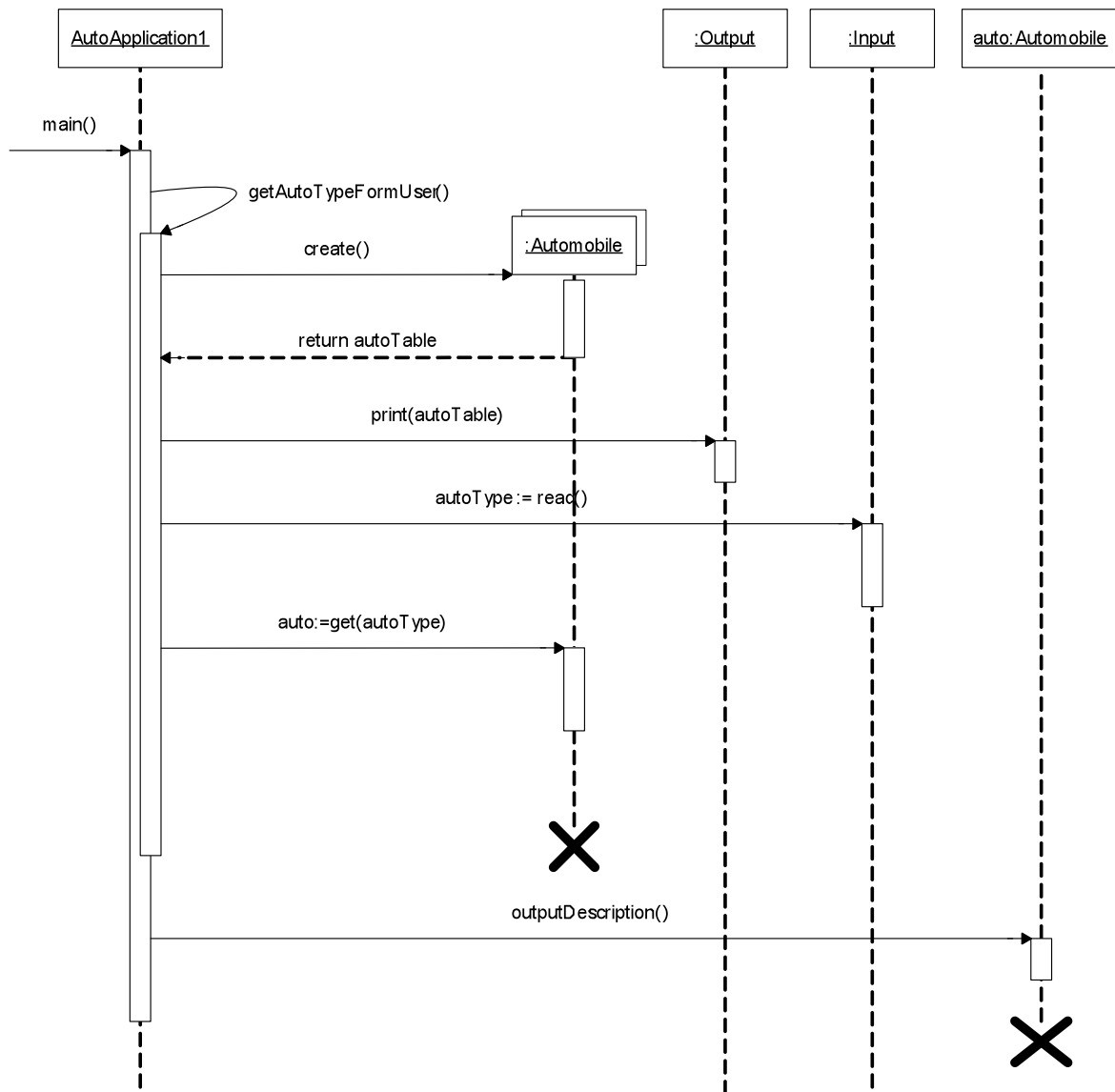
There are multiple solutions to this problem.

(Some of these solutions are technically correct but do not provide any useful information to the reader.)

The first is a very concrete representation closely matching the source code.



The second version is more abstract because it uses a multi-object to represent the *Automobiles* class and an *Automobile* object to represent all of the auto types. This version is implementation independent because it does use language specific objects; e.g. Java's 'System.out' is replaced by a generic 'Output' object.



Finally, there is a model so abstract it makes a good example of “correct work of zero academic merit” :).

