Approach to object-oriented programming tasks

You have already learned how to proceed with more complex procedural programming tasks. Object-oriented programming also follows the same procedure. The difference now, however, is that the procedural part is converted to object-orientation. The basic idea of object-oriented programming is to merge data that altogether describes a larger coherent object. In addition, the program parts that manipulate this data are grouped into classes. An object therefore only contains the data it represents. The program parts (member functions) that execute operations on the data of the objects and the properties of the data (data members) are defined in classes.

Before we write some code, we want to do an informal modelling of the world for which a program is supposed to be written. To do this, you should proceed as follows:

1. Provide a clear and unambiguous requirement / problem description.
2. On a piece of paper, make a list of the objects that exist in the real world that are relevant for solving the task.
3. Briefly describe the main task of the objects.
4. Create an index card for each object and name the object by assigning a general class name.
5. Assign the most important properties (data members) to each class.
6. Find the appropriate data types for these properties. Are there any classes I have already created, which I can use as data type?
7. We rewrite the designed data as program code by defining our index cards and their properties as classes with data members.
8. Set your data members to *private* as best you can and define get//set methods that read or modify the data members.
9. Create one or more appropriate constructors for the individual classes.
10. Create your objects and initialize them if necessary by calling the constructor.
11. Declare the member functions for the individual classes. Use descriptive names such as *displayData()*.
12. Now start to develop the rough design in the main program.
13. Then proceed to the detailed design. Use your object in appropriate places within the program code and create an appropriate member function with an appropriate return type. Code the algorithm of the member function in the class until the corresponding result appears.
14. Repeat the detailed design cycle until all the member functions you need are defined.
15. Review all test cases and check whether the selected data types are appropriate, whether the program can be further optimized (is there redundancy?) and whether the task has been completely solved.

Example: library program

You want to write a program for the library management, where the data of a user is queried at the beginning of the program start. A menu should then appear in which the user can navigate. Here's what can work:

- *Enter personal data*
- *Borrow a book*
- *Extend loan*
- *Output all data*
- *Return book*
- *Exit*

The personal data is entered into the system from scratch.
The following functions can only be used if the borrower does not yet have a book, otherwise a message appears that the borrower already has a book.

- *The person can borrow a maximum of out of 3 books. For books, only the titles and authors are stored in the system. When borrowing, a return date is specified by which the borrower is supposed to return the book (incorrect date is not queried).*
- *The loan can be extended by 1 month. If the loan starts at the end of the month, the respective month is also set to the end of the month.*
  Example 1: Loan is from 31 Mar 2016 → Extension → New return date: 30 Apr 2016
  Example 2: Loan is from 30 Jan 2016 → Extension → New return date: 30 Feb 2016
  Example 3: Loan is from 20 Dec 2016 → Extension → New return date: 20 Jan 2017
- *When data is output, all loan data (person, book, return date) is displayed.*
- *When the book is returned, the borrower has the possibility to borrow a new book again.*

When the user exits the program, a goodbye message should appear.

Approach:

Based on the example, we would like to answer the following questions one after another:

- **What is do be done?**
  A library management program should be developed. The first task is to create a data model. Afterwards you assign a class name and data members for the objects. The model is then realized in program code, elaborated by member functions and tested until the program runs.
- **What should the rough structure of the program look like?**
  From the task description we can see that we can make a selection in the menu of the program. We need a *switch-case* for this.
  Since the program can be terminated, we can conclude that a *loop* around the menu is needed.
  Based on the menu selection, different actions are performed. These perform operations on the class that accesses the object.
  The objects are initialized at the beginning of the program. In addition, a person's data is entered before the menu is displayed. The different *case scenarios* are therefore the *member functions* of the class that can access the object. These would be the first aspects of the task description.
- **How do I proceed?**
  First we should consider which objects are available in the library management. On the basis of the task description we can identify the following objects and describe them briefly:
  - Person who wants to borrow books.
  - Books that can be borrowed.
  - Actual lending transactions that express that a book has been lent by someone for a certain period of time.
  - Dates, that is, objects that indicate the return date.

  We can then create index cards for these objects. The data members with the corresponding data types have been defined for this purpose:

Person	Book	Date	Loan
String surname	*String title*	*Int day*	*Person borrower*
String first name	*String author*	*Int month*	*Book borrowedBook*
String street		*Int year*	*Date returnDate*
String house number			
Int postal code			
String city			

The obtained data can now be used as classes in our source code. The data members are declared as *private* and a getter-function for each class will output the data. The getter-functions of the other classes are called for the book loans. These functions are declared as *public*. Since we have to enter data for the class Person and Date, we write a function in each class that fills the data members with the data that the user has entered. There are 3 books in the library to choose from, which is why we need a constructor that fills the data member of this three array elements. Now the task description is processed step by step. Starting with the first case scenario, the corresponding member functions are defined and tested. Finally, one should consider the case that if the user has already borrowed a book, he must return the book before he can perform any other operation. Once you have done this, you should check your program for errors (e.g. Are the options in the menu correct? Can the user really only borrow a maximum of 1 book? Is the date set correctly if I extend it by one month (February has only 28 days)? etc.). To be on the safe side, the program can be tested and reviewed by a third person. Since applications should also be simple and familiar to other people, their feedback is crucial.