Reading Technique for Object-Oriented Design

Use the procedure below to perform an initial Object-Oriented design of the new system. This will require specifying at a high level the classes and their responsibilities in the system. As you work, use the questions provided to identify faults in the requirements:

1) Read the requirements through once, and identify some initial classes in the system. Record the names of these classes in the spaces provided on the form.

   To do this, we must think of the system to be built as a model of some aspect of the real world. A class is an abstract representation of a group of entities that will be modeled by the system. Classes have associated responsibilities (actions they perform in the system) and attributes (information which is stored about each entity which belongs to the class). Some authors suggest identifying all of the nouns in the requirements as an initial set of potential classes. At any rate, the important thing to remember is that this is just a first draft that will be flushed out in later steps of the design. The guidelines there may lead you to identify new classes or to throw away ones which you previously identified.

   This is a fuzzy concept but some examples might help:

   a) In a GUI system, we can consider each type of graphical object (e.g. scrollbars, pull-down menus, radio buttons) to belong to a separate class. Each of these classes has certain responsibilities (e.g. they must be able to display themselves in a window and respond with the appropriate functionality to a mouse click) and attributes (e.g. their position within the window, whether they are currently active, possibly a current value).

   b) In a customer database, an obvious choice for a class would be the customer. The customer class would have certain attributes that would be easy to identify (e.g. name, address, phone number, or whatever other information the users of the database are interested in storing). Responsibilities would depend on the functionality the database user wants to perform on customer records (an example might be suggesting new merchandise the customer might be interested in based on his past purchasing history).

2) Assign initial attributes and responsibilities to the classes, and write them in the appropriate spaces on the form beneath the class name. For each class you identified in the previous step, suggest some attributes (i.e. information the system should store for each member of the class) and responsibilities (i.e. the specific functionality it would participate in). Focus only on attributes and responsibilities that are obvious from the requirements or the name of the class. Don't try to be exhaustive – this is just a first guess.

3) Use scenarios to complete the assignment of responsibilities and attributes to classes. Think about each action the user can do. Then ask yourself what class is responsible for handling the user input. If you have already created an appropriate class, add that responsibility to it; otherwise you will have to create a new class which can handle the input. In either case, add any attributes to the class which are necessary to support the functionality. Next think about which other classes the current class will have to communicate with in order to achieve the system functionality which should result, and record these as collaborators of the class on the form provided. Again, if an appropriate class already exists, add the necessary responsibility to its list; otherwise create a new class to which the responsibility can be assigned. Remember to add any necessary attributes. Continue in this way until you have reached the classes which handle the lowest-level, specific functionalities and no more classes are involved. An example may help clarify:

a) Suppose we are designing a GUI system for editing documents. One functionality that should be supported is that the user should be able to click on a scroll bar to move the current document in the window. Therefore, the scroll bar should be modeled as a class with the responsibility of changing the currently visible area of the document. We will need at least one attribute: the current position of the slider on the scroll bar. So, the scroll bar must respond when the user moves the slider, at which point it must figure out what part of the document should now be displayed. We will therefore need a class "Document", which is a collaborating of the scroll bar class, which has the responsibility of computing which part of itself should be displayed in the window. Class "Document" may in turn collaborate with other more specific classes, e.g. "Paragraphs", or "Drawing Objects" that represent the components of a document.

4) Remember that step 3 is an iterative task. As you progress, you may need to split up some large classes (which have many more attributes and responsibilities than other classes) into smaller ones, or delete very small classes (which have few or none attributes and responsibilities) that were suggested earlier but turn out not to be important parts of the system.

5) When you have completed your analysis of the potential classes of the system, use the information you recorded to answer the following questions:

Q1.1 **Are all of the appropriate classes defined in the requirements? Are all the conditions involving the classes specified (e.g. are any requirements missing)?**

Q1.2 **Are classes described in the requirements which had to be deleted because they are really not responsible for any system functionality?**

Q2.1 **Can the data types of all of the attributes be defined (e.g. are the required precision and units specified)?**

Q2.2 **Do the data types which are necessary for the attributes contradict information specified in the general description or elsewhere in the requirements?**

Q3.1 **Are the interfaces between classes involved in collaborations specified and consistent?**

Q3.2 **Are there any points in which you are not clear about which classes are responsible for a functionality because the requirements are not clear or not consistent?**

Q4 **Do the finished classes, collaborations, and attributes make sense from what you know about the application or from what is specified in the general description?**

Acknowledgements
These guidelines owe much to the following works:

Beck, Kent, and Ward Cunningham. (1989) "A Laboratory for Teaching Object-Oriented Thinking", *Proc. OOPSLA89*.

Brummond, Nils. (1996) Object Oriented Analysis and Design using CRC Cards. http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/crc_b/

Jacobson, Ivar, *et al.* (1992) *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Publishing Company.