Focus on Object-Oriented Design: Finding the Classes and Their Responsibilities

CONCEPT: One of the first steps in creating an object-oriented application is determining the classes that are necessary, and their responsibilities within the application.

So far you have learned the basics of writing a class, creating an object from the class, and using the object to perform operations. This knowledge is necessary to create an object oriented application, but it is not the first step in designing the application. The first step is to analyze the problem that you are trying to solve and determine the classes that you will need. In this section we will discuss a simple technique for finding the classes in a problem and determining their responsibilities.

Finding the Classes

When developing an object-oriented application, one of your first tasks is to identify the classes that you will need to create. Typically, your goal is to identify the different types of real-world objects that are present in the problem, and then create classes for those types of objects within your application.

Over the years, software professionals have developed numerous techniques for finding the classes in a given problem. One simple and popular technique involves the following steps.

1. Get a written description of the problem domain.

2. Identify all the nouns (including pronouns and noun phrases) in the description. Each of these is a potential class. Refine the list to include only the classes that are relevant to the problem.

Let's take a closer look at each of these steps.

Write a Description of the Problem Domain

The *problem domain* is the set of real-world objects, parties, and major events related to the problem. If you adequately understand the nature of the problem you are trying to solve, you can write a description of the problem domain yourself. If you do not thoroughly understand the nature of the problem, you should have an expert write the description for you.

For example, suppose we are programming an application that the manager of Joe's Automotive Shop will use to print service quotes for customers. Here is a description that an expert, perhaps Joe himself, might have written:

Joe's Automotive Shop services foreign cars and specializes in servicing cars made by Mercedes, Porsche, and BMW. When a customer brings a car to the shop, the manager gets the customer's name, address, and telephone number. The manager then determines the make, model, and year of the car, and gives the customer a service quote. The service quote shows the estimated parts charges, estimated labor charges, sales tax, and total estimated charges.

The problem domain description should include any of the following:

- Physical objects such as vehicles, machines, or products
- Any role played by a person, such as manager, employee, customer, teacher, student, etc.
- The results of a business event, such as a customer order, or in this case a service quote
- Recordkeeping items, such as customer histories and payroll records

Identify All of the Nouns

The next step is to identify all of the nouns and noun phrases. (If the description contains pronouns, include them too.) Here's another look at the previous problem domain description. This time the nouns and noun phrases appear in bold.

Joe's Automotive Shop services **foreign cars**, and specializes in servicing **cars** made by **Mercedes**, **Porsche**, and **BMW**. When a **customer** brings a **car** to the **shop**, the **manager** gets the **customer**'s **name**, **address**, and **telephone number**. The **manager** then determines the **make**, **model**, and **year** of the **car**, and gives the **customer** a **service quote**. The **service quote** shows the **estimated parts charges**, **estimated labor charges**, **sales tax**, and **total estimated charges**.

Notice that some of the nouns are repeated. The following list shows all of the nouns without duplicating any of them.

address
BMW
car
cars
customer
estimated labor charges
estimated parts charges
foreign cars
Joe's Automotive Shop
make
manager
Mercedes
model
name
Porsche
sales tax
service quote
shop
telephone number
total estimated charges
year

Refine the List of Nouns

The nouns that appear in the problem description are merely candidates to become classes. It might not be necessary to make classes for them all. The next step is to refine the list to include only the classes that are necessary to solve the particular problem at hand.

We will look at the common reasons that a noun can be eliminated from the list of potential classes.

1. Some of the nouns really mean the same thing.

In this example, the following sets of nouns refer to the same thing:

- **cars** and **foreign cars**

  These both refer to the general concept of a car.
- **Joe's Automotive Shop** and **shop**

  Both of these refer to the company "Joe's Automotive Shop."

We can settle on a single class for each of these. In this example we will arbitrarily eliminate **foreign cars** from the list, and use the word **cars**. Likewise we will eliminate **Joe's Automotive Shop** from the list and use the word **shop**. The updated list of potential classes is:

address
BMW
car
cars
customer
estimated labor charges
estimated parts charges
~~foreign cars~~
~~Joe's Automotive Shop~~
make
manager
Mercedes
model
name
Porsche
sales tax
service quote
shop
telephone number
total estimated charges
year

Because **cars** and **foreign cars** mean the same thing in this problem, we have eliminated **foreign cars**. Also, because **Joe's Automotive Shop** and **shop** mean the same thing, we have eliminated **Joe's Automotive Shop**.

2. Some nouns might represent items that we do not need to be concerned with in order to solve the problem.

A quick review of the problem description reminds us of what our application should do: print a service quote. In this example we can eliminate two unnecessary classes from the list:

- We can cross **shop** off the list because our application only needs to be concerned with individual service quotes. It doesn't need to work with or determine any company-wide information. If the problem description asked us to keep a total of all the service quotes, then it would make sense to have a class for the shop.

• We will not need a class for the **manager** because the problem statement does not direct us to process any information about the manager. If there were multiple shop managers, and the problem description had asked us to record which manager generated each service quote, then it would make sense to have a class for the manager.

The updated list of potential classes at this point is:

address
BMW
car
cars
customer
estimated labor charges
estimated parts charges
~~foreign cars~~
~~Joe's Automotive Shop~~
make
~~manager~~
Mercedes
model
name
Porsche
sales tax
service quote
~~shop~~
telephone number
total estimated charges
year

Our problem description does not direct us to process any information about the **shop**, or any information about the **manager**, so we have eliminated those from the list.

Some of the nouns might represent objects, not classes.

We can eliminate **Mercedes**, **Porsche**, and **BMW** as classes because, in this example, they all represent specific cars, and can be considered instances of a **cars** class. Also, we can eliminate the word **car** from the list. In the description it refers to a specific car brought to the shop by a customer. Therefore, it would also represent an instance of a **cars** class. At this point the updated list of potential classes is:

address
~~BMW~~
~~car~~
cars
customer
estimated labor charges
estimated parts charges
~~foreign cars~~
~~Joe's Automotive Shop~~
~~manager~~
make
~~Mercedes~~
model
name
~~Porsche~~
sales tax
service quote
~~shop~~
telephone number
total estimated charges
year

We have eliminated **Mercedes**, **Porsche**, **BMW**, and **car** because they are all instances of a **cars** class. That means that these nouns identify objects, not classes.

NOTE: Some object-oriented designers take note of whether a noun is plural or singular. Sometimes a plural noun will indicate a class and a singular noun will indicate an object.

4. Some of the nouns might represent simple values that can be stored in a variable and do not require a class.

Remember, a class contains attributes and member functions. Attributes are related items that are stored within an object of the class, and define the object's state. Member functions are actions or behaviors that may be performed by an object of the class. If a noun represents a type of item that would not have any identifiable attributes or member functions, then it can probably be eliminated from the list. To help determine whether a noun represents an item that would have attributes and member functions, ask the following questions about it:

- Would you use a group of related values to represent the item's state?
- Are there any obvious actions to be performed by the item?

If the answers to both of these questions are no, then the noun probably represents a value that can be stored in a simple variable. If we apply this test to each of the nouns that remain in our list, we can conclude that the following are probably not classes: **address**, **estimated labor charges**, **estimated parts charges**, **make**, **model**, **name**, **sales tax**, **telephone number**, **total estimated charges** and **year**. These are all simple string or numeric values that can be stored in variables. Here is the updated list of potential classes:

~~address~~
~~BMW~~
~~car~~
cars
customer
~~estimated labor charges~~
~~estimated parts charges~~
~~foreign cars~~
~~Joe's Automotive Shop~~
~~make~~
~~manager~~
~~Mercedes~~
~~model~~
~~name~~
~~Porsche~~
~~sales tax~~
service quote
~~shop~~
~~telephone number~~
~~total estimated charges~~
~~year~~

We have eliminated **address**, **estimated labor charges**, **estimated parts charges**, **make**, **model**, **name**, **sales tax**, **telephone number**, **total estimated charges**, and **year** as classes because they represent simple values that can be stored in variables.

As you can see from the list, we have eliminated everything except **cars**, **customer**, and **service quote**. This means that in our application, we will need classes to represent cars, customers, and service quotes. Ultimately, we will write a Car class, a Customer class, and a ServiceQuote class.

Identifying a Class's Responsibilities

Once the classes have been identified, the next task is to identify each class's responsibilities.

A class's *responsibilities* are:

- the things that the class is responsible for knowing
- the actions that the class is responsible for doing

When you have identified the things that a class is responsible for knowing, then you have identified the class's attributes. Likewise, when you have identified the actions that a class is responsible for doing, you have identified its member functions.

It is often helpful to ask the questions "In the context of this problem, what must the class know? What must the class do?" The first place to look for the answers is in the description of the problem domain. Many of the things that a class must know and do will be mentioned. Some class responsibilities, however, might not be directly mentioned in the problem domain, so brainstorming is often required. Let's apply this methodology to the classes we previously identified from our problem domain.

The Customer class

In the context of our problem domain, what must the Customer class know? The description directly mentions the following items, which are all attributes of a customer:

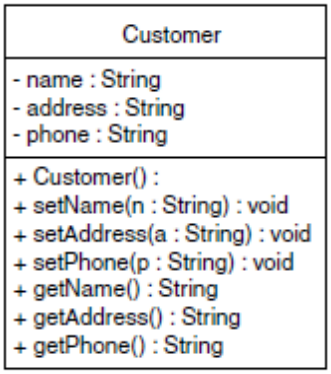
- the customer's name
- the customer's address
- the customer's telephone number

These are all values that can be represented as strings and stored in the class's member variables. The Customer class can potentially know many other things. One mistake that can be made at this point is to identify too many things that an object is responsible for knowing. In some applications, a Customer class might know the customer's email address. This particular problem domain does not mention that the customer's email address is used for any purpose, so we should not include it as a responsibility.

Now let's identify the class's member functions. In the context of our problem domain, what must the Customer class do? The only obvious actions are to

- create an object of the Customer class
- set and get the customer's name
- set and get the customer's address
- set and get the customer's telephone number

From this list we can see that the Customer class will have a constructor, as well as accessor and mutator functions for each of its attributes. The below figure shows a UML diagram for the Customer class.



The Car Class

In the context of our problem domain, what must an object of the Car class know? The following items are all attributes of a car, and are mentioned in the problem domain:

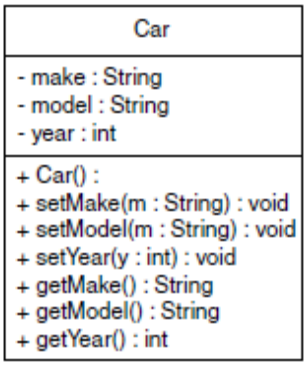
- the car's make
- the car's model
- the car's year

Now let's identify the class's member functions. In the context of our problem domain, what must the Car class do? Once again, the only obvious actions are the standard set of member functions that we will find in most classes (constructors, accessors, and mutators).

Specifically, the actions are:

- create an object of the Car class
- set and get the car's make
- set and get the car's model
- set and get the car's year

The below figure shows a UML diagram for the Car class at this point.



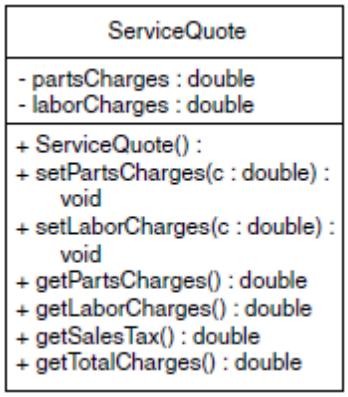
The ServiceQuote Class

In the context of our problem domain, what must an object of the ServiceQuote class know? The problem domain mentions the following items:

- the estimated parts charges
- the estimated labor charges
- the sales tax
- the total estimated charges

Careful thought and a little brainstorming will reveal that two of these items are the results of calculations: sales tax and total estimated charges. These items are dependent on the values of the estimated parts and labor charges. In order to avoid the risk of holding stale data, we will not store these values in member variables. Rather, we will provide member functions that calculate these values and return them.

The other member functions that we will need for this class are a constructor and the accessors and mutators for the estimated parts charges and estimated labor charges attributes. The below figure shows a UML diagram for the ServiceQuote class.



This Is Only the Beginning

You should look at the process that we have discussed in this section as merely a starting point. It's important to realize that designing an object-oriented application is an iterative process. It may take you several attempts to identify all of the classes that you will need, and determine all of their responsibilities. As the design process unfolds, you will gain a deeper understanding of the problem, and consequently you will see ways to improve the design.

****

Look at the following description of a problem domain:

A doctor sees patients in her practice. When a patient comes to the practice, the doctor performs one or more procedures on the patient. Each procedure that the doctor performs has a description and a standard fee. As the patient leaves the practice, he or she receives a statement from the office manager. The statement shows the patient's name and address, as well as the procedures that were performed, and the total charge for the procedures.

Assume that you are writing an application to generate a statement that can be printed and given to the patient.

A) Identify all of the potential classes in this problem domain.

B) Refine the list to include only the necessary class or classes for this problem.

c) Identify the responsibilities of the class or classes that you identified in step B.

Look at the following description of a problem domain:

The bank offers the following types of accounts to its customers: savings accounts, checking accounts, and money market accounts. Customers are allowed to deposit money into an account (thereby increasing its balance), withdraw money from an account (thereby decreasing its balance), and earn interest on the account. Each account has an interest rate.

Assume that you are writing an application that will calculate the amount of interest earned for a bank account.

A) Identify the potential classes in this problem domain.

B) Refine the list to include only the necessary class or classes for this problem.

C) Identify the responsibilities of the class or classes.

If the items on the following list appeared in a problem domain description, which would be potential classes?

Animal
Medication
Nurse
Inoculate
Operate
Advertise
Doctor
Invoice
Measure
Patient
Client
Customer

Determining Class Collaborations with CRC Cards

During the object-oriented design process, you can determine many of the collaborations that will be necessary between classes by examining the responsibilities of the classes.

A class's responsibilities are

- the things that the class is responsible for knowing
- the actions that the class is responsible for doing

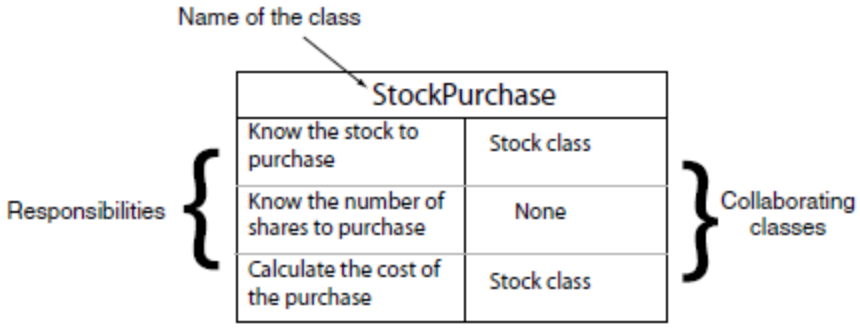
Often you will determine that the class must collaborate with another class in order to fulfill one or more of its responsibilities. One popular method of discovering a class's responsibilities and collaborations is by creating CRC cards. *CRC* stands for Class, Responsibilities, and Collaborations.

You can use simple index cards for this procedure. Once you have gone through the process of finding the classes, set aside one index card for each class. At the top of the index card, write the name of the class. Divide the rest of the card into two columns. In the left column, write each of the class's responsibilities. As you write each responsibility, think about whether the class needs to collaborate with another class to fulfill that responsibility.

Ask yourself questions such as

- Will an object of this class need to get data from another object in order to fulfill this responsibility?
- Will an object of this class need to request another object to perform an operation in order to fulfill this responsibility?

If collaboration is required, write the name of the collaborating class in the right column, next to the responsibility that requires it. If no collaboration is required for a responsibility, simply write "None" in the right column, or leave it blank. The below figure shows an example CRC card for the StockPurchase class.



From the CRC card shown in the figure, we can see that the StockPurhcase class has the following responsibilities and collaborations:

- Responsibility: To know the stock to purchase
  Collaboration: The Stock class

- Responsibility: To know the number of shares to purchase
  Collaboration: None

- Responsibility: To calculate the cost of the purchase
  Collaboration: The Stock class

When you have completed a CRC card for each class in the application, you will have a good idea of each class's responsibilities and how the classes must interact.

Inventory Item – Class diagram