Breaking the Problem Down: Problem Solving in Object-Oriented Computer Science

Sometimes it is just hard to get started. Faced with a long problem or project description, you may wonder why a long list of do this and that was not provided to make your life simpler. Project descriptions usually just include an overview of the project, because there are actually many ways to solve the problem or achieve some purpose.

How to approach a problem

How do we actually approach the problem? One way to think about a problem is to consider it as interactions between entities or components within a system. Two methods of this form of interpretation are the top-down approach and the bottom-up approach. The ***top-down approach*** is sometimes considered the "big picture" approach, because the general idea of the system is first formulated and without getting down to the lowest level, or base, entities. The system, or big picture, is then broken down into slightly smaller subsystems. Those subsystems are then separated again until we reach the very bottom level when no more break down can really occur. Essentially the top-down approach is similar to dissembling a jigsaw puzzle. The ***bottom-up approach*** is the opposite of the top-down approach, because the bottom-up approach first considers the lowest level entities first and their interaction with one another which build subsystems. These subsystems will interact with each to form greater subsystems and slowly build our way up to the highest system. Using the same jigsaw analogy, the bottom-up approach is similar to assembling the puzzle.

The top-down and bottom-up just describes two different general methods of thinking and interpreting some situation. How a given situation is actually perceived and processed will vary with the person and the given, but the idea is to do whatever is best for you to understand.

Problem solving in object-oriented computer science

In object-oriented computer science, the lowest level entities are usually Objects, instances of some class or classes. The interactions of these entities tend to form systems. Thus, it would be useful to spot what will be the entities in some descriptions.

Consider the following project description:

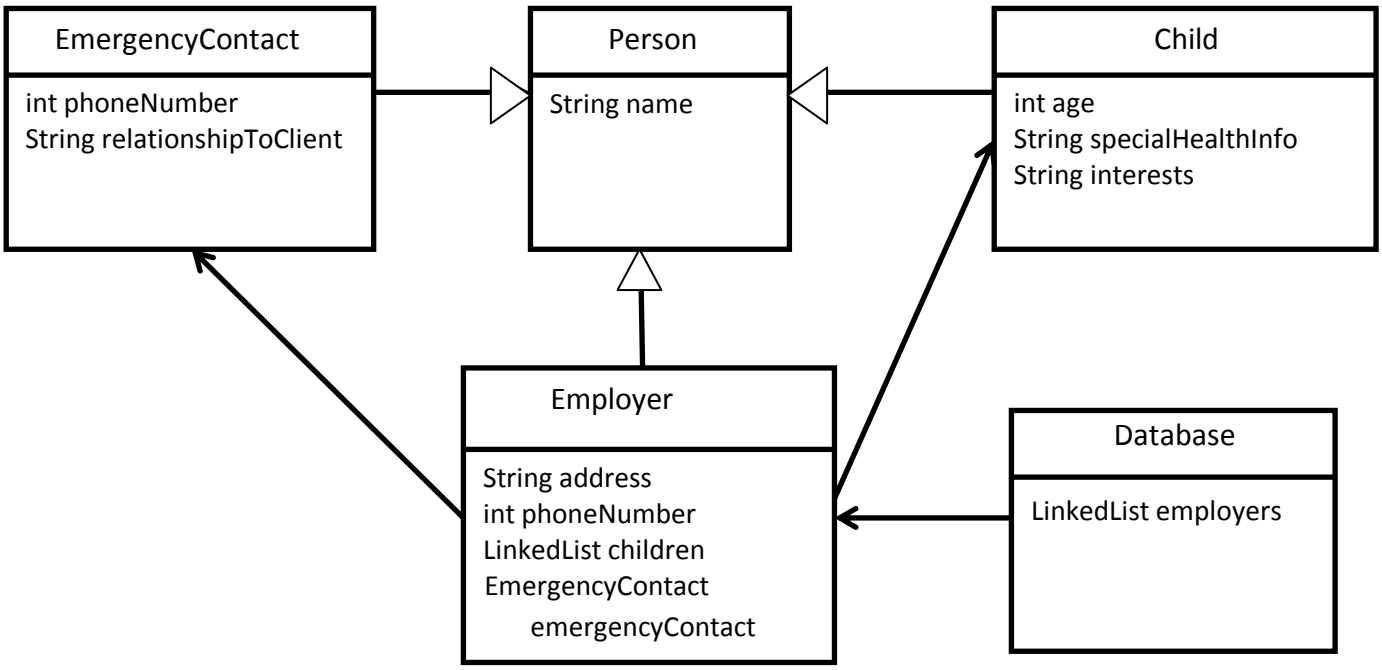
> A part-time babysitter wants you to help her implement some way of organizing and managing her employers' information where it would be easy to add and remove new employers. Each employer is a person with a name, an address, a phone number and at least one child. Each employer also provides a person, an emergency contact, with a name, phone number and relationship to the client. Each child's name, age, special health information and interests should also be noted.

The first line merely gives an overview of what the babysitter would like to have to manage her employers and describes the sort of data structure she would need. Because amount of employers the babysitter has at a given time varies, it would be best to use some sort of dynamic data structure. However it seems that babysitter does not really care how the project is implemented. You could use a static data structure as well, but it would just mean more work for you.

The following lines describe what information actually needs to be track in the data structure. How can we figure out what our entities will be? An easy to way to approach this is to go through and note all of the "nouns" in the description.

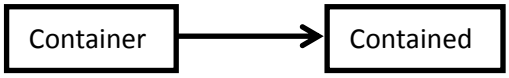
> A part-time babysitter wants you to help her implement some way of organizing and managing her employers' information where it would be easy to add and remove new employers. Each employer is a person with a name, an address, a phone number and at least one child. Each employer also provides a person, an emergency contact, with a name, phone number and relationship to the client. Each child's name, age, special health information and interests should also be noted.

Some of the highlighted nouns in the descriptions will be entities and some actually are attributes that describe that entity. Depending on how involved you want to be, you can have each and every one of the highlighted nouns each be different classes (such as Name, Address, PhoneNumber) or you can have some of them just be Strings to simplify things. You should also notice that an employer, an emergency contact and a child are all considered persons. Do not forget the data structure from the first line! Thus we will have the following diagram:

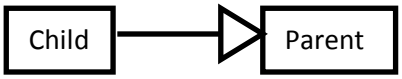


At this point how you to represent the entities and their relationship with one another is entirely up to you, because the only one that needs to understand the diagram is you. However later on in the course, we will discover the need to use more standard diagrams that every person should be able to understand. Let us take a moment to explain some of the relationships used in the above diagram:

The line with the black arrowhead signifies an association between a class that serves as a "container" and the class that is being "contained." For example, the Employer class contains a reference to EmergencyContact called emergencyContact and a LinkedList of Childs called children.



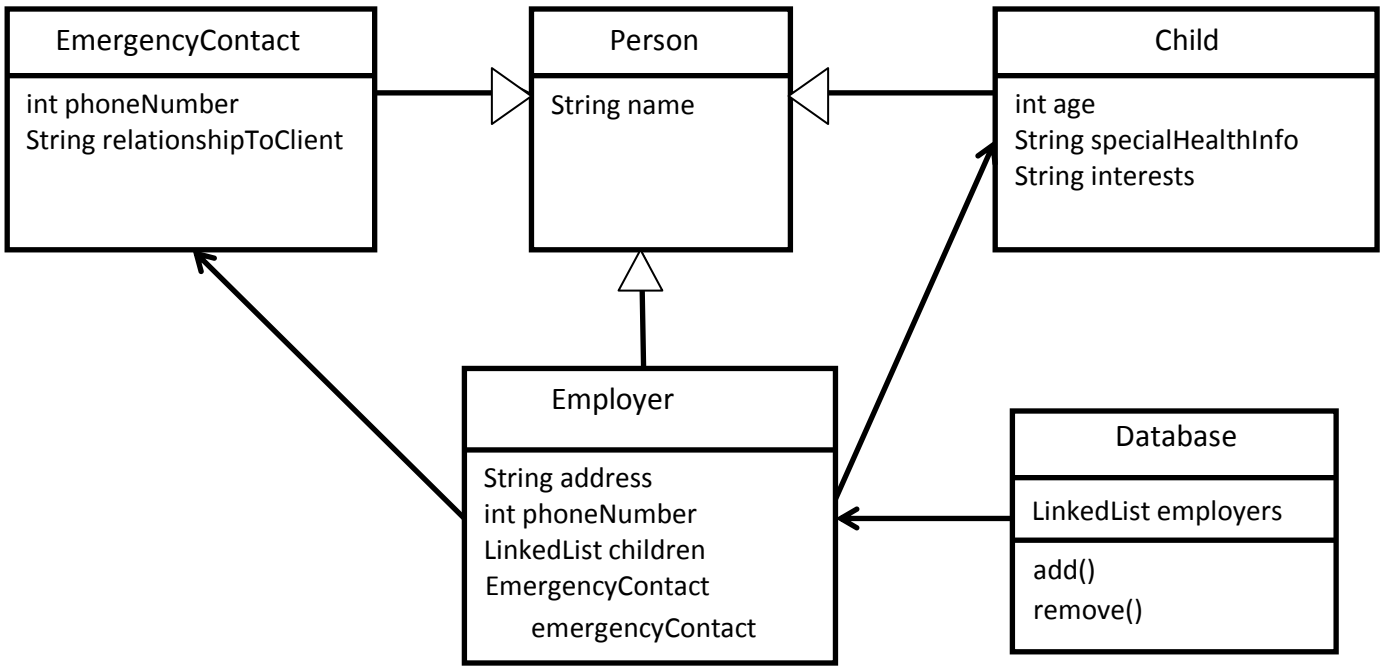
The line with the white arrowhead signifies an inheritance relationship.



The next step is to figure out what each entity can do or what methods we need to use, we go back through the description and look for action verbs associated with the entities.

> A part-time babysitter wants you to help her implement some way of organizing and managing her employers' information where it would be easy to add and remove new employers. Each employer is a person with a name, an address, a phone number and at least one child. Each employer also provides a person, an emergency contact, with a name, phone number and relationship to the client. Each child's name, age, special health information and interests should also be noted.

You should quickly discover that the data structure is the only entity capable of committing any actions. Thus we have to update our diagram:



It is important to remember that the diagram above and those like it are used for initial planning. When you start actually trying to complete the project code-wise, you may discover that you actually need more methods, classes and attributes.

Exercises

For each the following project descriptions, determine the classes you need to write, what methods and variables the classes will contain and how the classes relate to one another. Draw a diagram if necessary.

1. A teacher wants you to help her write a program that will randomly assign each of her students to an activity at the various activity tables available in her classroom. Each activity table knows its activity, the materials needed for the activity, the students at the table and how many chairs are left. The activity tables available are the math table, the art table, the reading table and the computer table and their respective activities are doing math, drawing, reading and playing on the computer.

2. A friend wants you to help create a portion of the logic behind a Pac-Man game. Remember that a Pac-Man knows how to move, eat and die. A Pac-Man can eat dots, the flashing energizer dots or a ghost that has turned blue. A dot only knows its location. There are four normal colors for a ghost: red, pink, orange and cyan. A ghost knows how to move, be eaten, kill Pac-man, turn blue and return to the center of the screen after being eaten. (Do not think about the graphical details or extra portions of the game not mentioned in the description).