Big Picture
Finding Classes and Objects, Attributes, Services, Inheritance, and Delegation, then Filtering.

1. Finding Classes and Objects
• Read the requesting document — look for (pronounceable) nouns in the problem description.
  (Use the clients vocabulary; don’t “improve” it or invent your own.)
• Run through scenarios / Use Cases — for example to play the game first shuffle the deck.
• Look for roles played — teacher, operator, owner, and Sites — office, airport.

2. Finding Attributes (these hold state information)
• For each class/object, describe its values. “What is an object in the class responsible for knowing?”
• Ask each of the following from the perspective of a single object:
  (1) How am I described in general?  consider a warehouse in a shipping program
  (2) How am I described in the problem domain?  a collection of bins for pallets, pickers, orders
  (3) What do I need to know?  about the bins
  (4) What state information do I need to remember over time?  for bin: last removal, current contents
  (5) What states can I be in?  for bin: full, empty
• Attributes of an object are manipulated only by the services of that object or one of its subclasses.
• Make each attribute capture an atomic concept. For example, replace the attributes ownersAddress, ownersName, … with one attribute of class owner.
  motivation: it produces a simpler model for human review, with fewer attribute names, and natural grouping for easier assimilation; code readers will focus on the fact that an address is captured, rather than scanning the list of attributions to determine whether each tightly-related piece is somewhere in the list of attributes.
• Place an attribute in the class/object that it best describes. Consider placing an attribute in a superclass.
• Defer to design: layout, storing intermediate results or tables of common information, etc.

3. Finding Services
Services are behaviors, state changers, and inspectors that commonly come from verb phrases in the problem description. (Programmers tend to be good at this step.)
Look for
  • constructors — (validate initial arguments against constraints)
  • state changers — goToRedAlert()
  • what calculations is the object responsible for? — updateBalance() · · · sortChecks()
  • conversion — degreesFahrenheitToCentigrade()

For each service identify its parameters, its outputs, and any pre-conditions and post-conditions.
4. Finding Inheritance

(1) Consider each class as a super class and for each potential specialization (subclass) ask “are some attributes specific to the subclass only?”

(2) Repeat this process considering each class as a subclass and asking about potential super classes.

(3) If you have a class with a type field, replace the field with subclasses.

5. Finding Delegation (Whole-Part)

Look for three kinds of delegation:

(1) Assembly: truck has-a wheel(s) (i.e., wheel is an instance variable of class Truck)

(2) Containment: truck has-a load

(3) Collection: set has-a element(s)

6. Filtering

AFTER you have your collection of objects check them against the following list:

relevance
Is the class/object, attribute, or service in the problem domain and system’s responsibility?
For example a ticketing program is responsible for a passengers seat, but not their marital status.

need remembrance
(Attributes) Does the system need to remember anything about the objects in the class?
Are attributes relevant? e.g., userName, userHairColor.

need behavior
(Services) Does the object need to provide some behavior (beyond construction)?

status value only?
Does an attribute capture more than just a status value? If not, then just include it in the whole.

(usually) more than one Object in a class
Challenge classes with only a single object. Suspect classes are labeled “this vehicle” or “that vehicle.” However, a single object Radar on a fishing boat is ok.

always applicable attributes
If the attributes can be divided into “always applicable attributes” and “sometimes applicable attributes,” then consider introducing inheritance.
Example 1: a building class always has cost, address, size, but sometimes has number-of-bathrooms and kitchen-size.

If an attribute has the value “not applicable,” then re-think the inheritance hierarchy.
Example 2: max thrust for aircraft. (This may be zero for a glider and thus no new class is warranted; use your judgment.)

always applicable services
Similarly, if services vary for different objects, that is an indication that inheritance could be added.

not merely derived results
Avoid clients-age if clients-birth-date is known. (Retaining age for efficiency is a design decision.)