Aristotle and Object-Oriented Programming: Why Modern Students Need Traditional Logic

Derek Rayside
Electrical & Computer Engineering
University of Waterloo
Waterloo, Canada
drayside@acm.org

Gerard T. Campbell
Department of Philosophy
St. Jerome's University
Waterloo, Canada
gcampbel@watarts.uwaterloo.ca

Abstract

Classifying is a central activity in object-oriented programming and distinguishes it from procedural programming. Traditional logic, initiated by Aristotle, assigns classification to our first activity in reasoning, whereby *we come to know what a thing is.* Such a grasp of the thing's *whatness* is the foundation for all further reasoning about it.

This connection between Aristotle's way of classifying and object-oriented programming is sometimes acknowledged, but rarely explored in depth.[1] We explore this relation more closely and more carefully, in the hope that a better understanding of classification and programming can be gained from a study of philosophy than from many current text books on object-oriented programming.

1 Traditional Logic and Symbolic Logic

We will use *traditional logic* to refer to what was first set down by Aristotle and was developed and taught as the key of the liberal arts during the ensuing two millennia. *Symbolic logic* will be used to refer to the contemporary approach to reasoning that focuses on method, but which is not concerned with content. The former looks at the acts of reasoning as distinctively human, and so is concerned with truth as well as validity. The latter looks to validity as its chief concern, and so is not concerned with meaning or truth — only function. Since computers trace their lineage to a mathematics, such a logic is crucial in understanding how a computer operates, but not in understanding a programmer's reason.

Software often seems to have one foot firmly rooted in the operational aspects of mathematical functionality, but unsure of where to place the other foot. We feel that software, as a human expression, will eventually set the other foot down on literature and philosophy.

1.1 Traditional Logic's Three Acts of Reason

Traditional logic considers three distinct acts involved in reasoning and develops at great length the art of logic for the sake of performing each of these acts with the right kind of skill, such that when we reason we can do so *with order, with ease, and without error.*[2]

The first two of these acts deal with reason as a faculty of *understanding.* It is worthwhile to note that we refer to reason as the faculty of understanding because reason can grasp 'the natures of things' which *stand under* 'the appearances of things' (grasped by the senses), and the 'causes' of things which *stand under* the 'effects' which are first encountered by the senses and which provoke 'wonder' and the desire to know.[3]

The first act of our intellect consists in the grasping of indivisible or incomplex things (for example, 'dolphin') and by doing so it conceives *what* a thing is. Everything that Aristotle treats of in the *Categories* (*Predicaments*) is devoted to achieving this first act of reason, which culminates in the act of *definition.*

The second act of reason (still dealing with understanding) is reason affirming or denying attributes as belonging to or not belonging to the subject. In other words, to characterizing the subject in some way. As soon as we make such statements, however, we enter into considerations of *truth* and *falsity.* Aristotle's teaching in the *Peri Hermeneias* (*On Interpretation*) is devoted to this second act of reason.

The third act of reason is the one by which this faculty is most perfectly named, for now we deal with the reason or cause (especially the *why* of the subject being

[1] Cf. Grady Booch, *Object-Oriented Analysis and Design, with Applications* 2nd edition, Benjamin/Cummings, 1994.

[2] Cf. Thomas Aquinas, *In Libros Posteriorum Analyticorum Expositio*, Marietti, 1964, n.1. Translation ours.

[3] Cf. Aristotle, *Metaphysics*, Book A, Chapter I, 980a20. 'All men by nature desire to know.' (Translation W.D. Ross.)

SIGCSE 2000 3/00 Austin, TX, USA

as it is). To achieve this kind of knowing we have to discourse (advance by means of a method) from one thing to another, and thereby come-to-know what was previously unknown by means of what is already known. All of Aristotle's other works in logic (*Prior Analytics, Posterior Analytics, Topics, Rhetoric, Poetics, Sophistical Refutations*) are devoted to this process of reasoning.[4]

We summarize and name the three acts of reason as follows:

1. Simple Apprehension: When the first act is named in this way, two considerations need emphasis. The first, that we focus on how our mind apprehends what a thing is. The notion of *apprehend* denotes the activity of reaching out and grasping (like the police *apprehending a criminal*). For our reason to be able to seize the nature of something, we need to know how words are used, how one thing can be like another, how to divide (by means of opposition) something greater so we can see where this nature 'belongs' or 'fits', and how to go about finding the differences which adequately distinguish *this kind of thing* from other things which have attributes in common. All of this is achieved in *the art of defining*. When we use the term *simple* we mean *as one* — not *individually*, for the oneness of singularity is the role of the senses — but as a *one kind of thing*, that is, *universally*.

2. Composition and Division: Having grasped what a thing is, we can begin to talk about it — we can predicate (a predicate is *that which is said of a subject*). So we make statements about the subject in which either we affirm something of the subject (and this is called an act of *composition*), or we deny something of the subject (and this separation of the predicate from the subject is called an act of *division*). When we make these statements, whereby we affirm or deny, these will always be *true or false*. But to be able to judge the truth or falsity of any statement, first we must know *what the thing is*.

3. Argumentation: In any conclusion, a predicate will be affirmed or denied of a subject (and so a conclusion at which we arrive will also be *true or false*). But what will make the argument *valid or invalid* will be the method whereby we are able to arrive at the new conclusion by means of the statements. In the context of an argument, statements are called *propositions* or *premises* to show how they are ordered to the conclusion. We note at this point, however, that an argument with false statements and a true conclusion would not be a valid argument because the premises would not be the *cause* of the conclusion.

[4]While the *Prior Analytics* considers argumentation *formally*, all of the subsequent treatises deal with the *content* of what is reasoned. Cf. Aquinas' *Prologue* to his *Commentary on the Posterior Analytics of Aristotle*, rightly acclaimed for its masterful division and relation of all the parts of logic.

2 Language, Truth and Logic

Good treatises on logic, from Aristotle to Aquinas to Ayer, begin with an examination of language.[5] There is good reason for this: language and our ability to reason are inextricably linked. We cannot think apart from language (and an inability to handle language is an inability to handle thought).

This intimate relation of language and thought is evident even in the term 'logic' itself: *logic* derives from the Greek *logos*, which has, as its first meaning, *word*. Subsequent meanings start here. Since the *word* is a sign of the concept, *logos* has its meaning stretched to signify *concept* (that which is expressed in the *word*). When we use 'words' in logic, we use them as the 'signs of the concepts', which measure our usage. But it does not stop here. *Concepts* represent the *intelligible dimension in things themselves*, which measure our concepts. So in 'anthropology' (*anthropos* + *logos*), we look for the science or the intelligible dimension of mankind. Similarly for 'biology', 'sociology' and all the other 'ologies', including 'theology'. This meaning of logic, as an ordering of concepts, is the most familiar to the modern reader, and probably originates from Cicero the Roman (after Aristotle, before Christ).

But if what is real and unchanging (the intelligible structure in things) is the measure of what we think about it (concept) and speak (word) about it, then it too is a work of reason: not *our* reason, for our reason is the measured, but of *Reason*. Hence, so many of the great thinkers who have gone before us posited that the natural way in which the human mind gets to God is in a mediated way — via things themselves, which express God to the extent that they can. This is the meaning of *Logos* that St John uses when he says 'In the beginning was the *Word*' (John 1:1).[6]

When we use words to signify concepts, we need to do so *conventionally*, that is, even if the signs used are arbitrary, we need to agree on which concepts the signs represent in order to be able to communicate. Translators face a challenge in trying to express concepts for which there is no simple one-to-one correspondence in another language — and the insights of one culture are not always available to another.[7] All languages are artificial (by definition) and each one 'colours' the way we think and talk about things. Nevertheless, language is

[5]The title for this section has been borrowed from Ayer's manifesto on logical positivism, which espouses an entirely different view on the relation of language, truth and logic.

[6]The common English translation is quite poor, both for the translation of *Logos* as *Word*, and *arche* as *beginning*. *Arche* is closer to *first and foremost* or *principle* — its meaning is not temporal.

[7]One advantage of knowing more than one language is that it gives us direct access to different ways of thinking about things.

not about individual ways of thinking: communicating is for the sake of the community. That is why language must be conventional — and that is why we need logic to order our concepts and effect real communication. Logic itself, however, and the ways in which we use a language, are effective only when we learn to respect and acknowledge the relatedness and the differences in things themselves.

Traditional logic is very much aware that words are used to signify concepts in one of three ways: univocally, equivocally, or analogically.

Univocal A word is used univocally when it is said of many things, but as always signifying the same nature in each, i.e., when it signifies a common nature. We can use the word 'bat', for example, to signify 'a small flying mammal.' Whether the individuals of which it is predicated are male or female, or of this colour or that colour, we use 'bat' univocally as long as the same essential nature is signified whenever we use the term in a particular context.

We can also choose (conventionally) to use the word 'bat' to signify 'an instrument used for striking a ball'; and again the word is used univocally whether it be said of a 32oz or a 34oz bat, or of a wooden bat or an aluminum bat, or a baseball bat or a cricket bat. In each imposition, as long as the word 'bat' signifies the same nature in a given context, the term is used univocally.

Equivocal But problems can arise when words have more than one meaning. The example of 'bat', above, shows how a word can signify different kinds of things and so can have more than one meaning. What is important, however, is how we employ the term. We use a word equivocally when, within a given context, we use it to signify *different* natures. Even our ability to enjoy a play-on-words comes when we recognize that a term is being used with dual meanings.

Sometimes when we change the signification of the term without realizing it we engender either error or confusion. For example, there are times when it does need to be made clear whether 'man' is being used *inclusively* or *exclusively*. A more subtle example is the word 'certain': what is really intended if we say that someone has a *certain* grasp of something? That he understands it *somewhat* or *with certitude*?

Metaphorical use of a word is equivocal because in metaphorical use it is not the real nature of a thing which is being signified, but only an accidental likeness to a given nature.[8] When General Rommel was called the desert 'fox', it was not because of a cold, wet, nose, nor pointed ears, nor bushy tail but only because of a slyness of attack. In a similar way, when we call a data structure a 'tree' we use the word metaphorically: it is not a plant made of wood that grows in the ground nor does it have a 'life' of its own.

Niklaus Wirth goes so far as to argue that the use of the word *language* in *programming language* is metaphorical (and, therefore, equivocal).[9] He claims that we use the term 'programming language' instead of 'formalism' or 'formal notation' because of Noam Chomsky's work in representing grammar mathematically, and that it is this accidental resemblance of form that prompts us to the metaphor.

Analogical In computer science we use the word 'program' to signify both the activity of programming and the result of that activity (as a verb and as a noun). There is a real relatedness in this analogical use of language that is quite natural: in our growth in understanding and insight, the appreciation of analogy is a kind of benchmark. A child is quick to seize the relation between 'bat' as 'an instrument for striking a ball' and 'going to bat'. From his experience of the game he may even understand 'going to bat' for someone else in situations other than baseball. If the sign of intelligence in the developing child is the ability to see the similarity between different things, the sign of wisdom in the mature person is the ability to go beyond the similarities and to appreciate the dissimilarities in things that are alike. Accordingly, Aquinas refers to adulthood as the *age of discretion*.[10] We use words *analogically* (notice the word 'logic' in it) because of a real likeness of relation in different things.

Earlier we gave an example of analogy when we talked about logic and *logos* which referred, successively, to the word, the concept, the intelligible nature of the thing (its essence), and the order of the universe. The use of analogy is an important part of our intellectual growth. To borrow an example from St Augustine, the term *father* first names in a concrete, personal way 'one of the principles from whom we have life' (our *biological* father); but the Romans called the teacher 'father' because he was a source of *intellectual* life.[11] The Egyptians called Pharaoh 'the father of his people', a *social* fatherhood, akin to the way that some cultures speak of 'the fatherland'. In some Christian sects the priest is called 'Father' because he initiates his charges into the *spiritual* life through baptism, cares for their development by teaching them principles by which they can live, and feeding them with the Eucharist. Finally, according to St Paul, we realize that the fullness of fatherhood is in *God* and that all other fatherhoods participate in this one.

If we do not keep 'stretching' the meaning of a term

[8] The word 'accident' is discussed in §3.

[9] Cf. *Compiler Construction*, Addison-Wesley, 1996. p8.

[10] Thomas Aquinas, *Summa Contra Gentiles*, Book 3, Chapter 122, n. 2954. Translation ours.

[11] We owe a greater debt to our fathers for the care they provide in *family* life and in teaching us than for their biological contribution.

when there is an essential likeness but, instead, insist on words having only one meaning, we would lose sight of real unity and real relatedness in our world.

Sometimes, however, it is not always clear whether a word is used analogically or whether it is used metaphorically. For example, Edsger Dijkstra, in the following passage, argues that the word 'industry' is used metaphorically with respect to 'software industry'.

> The opportunity for simplification is very encouraging, because in all examples that come to mind the simple and elegant systems tend to be easier and faster to design and get right, more efficient in execution, and much more reliable than the contrived contraptions that have to be debugged into some degree of acceptability. (One of the reasons why the expression 'software industry' can be so misleading is that a major analogy with manufacturing fails to hold: in software, it is often the poor quality of the 'product' that makes it expensive to make! In programming, nothing is cheaper than not introducing bugs in the first place.) The world being what it is, I also expect this opportunity to stay with us for decades to come. Simplicity and elegance are unpopular because they require hard work and discipline to achieve and education to be appreciated. [12]

It is often difficult to draw the distinction between metaphor and analogy when we use words that signify artificial things, such as 'industry', because what is artificial depends upon us for its meaning. Analogical use of words really requires an essential likeness, whereas metaphorical (i.e. equivocal) usage requires only an accidental likeness. However, in day-to-day life it is often enough to recognize that a word is not being used univocally. And let us remember that although there is a relatedness of natures and meaning in analogy, when we deal with argument (and programming is a kind of argument) we must deal with only one meaning of a word at a time. The effective programmer is the one who understands both the use and misuse of likeness.

Understanding the ways in which words can be used to signify concepts is of singular importance in the first act of reason: knowing *what* we are talking about. All later thinking and speaking about a thing depends on performing this first act well. We are unable to reason or communicate effectively if we do not first make the effort to know what each thing is. Shoddiness in the use of language will lead inevitably to shoddiness in thought.

Understanding the ways in which words can be used to signify concepts is of particular importance to the student of computer science, whose future career will depend on the complimentary abilities of expressing complex technical ideas clearly and understanding similar ideas expressed by others. The critical observation underlying Don Knuth's idea of *literate programming* is that these ideas must be expressed to both the computer and to one's colleagues.

When programming we have a choice: either we can name something with a word or we can simply assign a symbol to it.[13] Not surprisingly the time when we are most comfortable using symbols instead of words in programming is when we are automating the computation of a well known mathematical function. Computers trace their lineage to this task: it is what we originally built them for, and it is what we have named them for. However, we now employ computers for a much wider variety of tasks, and so we generally use words more often than symbols when naming things in our programs.

This distinction between words and symbols brings to light the philosophical differences between traditional logic and symbolic logic. Traditional logic views the activity of reasoning both as a kind of understanding and then as an ordering of the concepts understood. Symbolic logic, on the other hand, views the activity of reasoning as mere process or method which can be carried out mechanically without regard for the subject matter under consideration. Traditional logic is concerned with truth, content, *what* we are talking about — a relation whereby our ideas conform to the way things are. Our reason is measured by the reality which it considers. In symbolic logic, on the other hand, the criteria is one of 'truth-value', a way of considering reasoning only as validity of sequence. To knowingly call true that which bears no relation to truth spells the end of the intellectual life for all further discourse becomes futile.

The fact of the matter is that how computers operate is closer to symbolic logic: they perform the abstract manipulation of symbols, with logical consistency, and without regard for meaning. *However, this is not what the programmer does.* Most of the time the programmer's reason is concerned with words that signify concepts and the intelligible order of things. And the concepts signified must be understood *by the programmer* before they can be ordered *by the programmer*.

How well we name a thing is a reflection of how well we have grasped what the thing is; in other words, how well we have performed the first act of reason. While this requires the same hard work and discipline that

[12] *The Tide, Not the Waves*, in *Beyond Calculation: The Next Fifty Years of Computing*, edited by Peter J. Denning and Robert M. Metcalfe, Copernicus (Springer-Verlag), 1997, pp.59–64.

[13] For a very good discussion of the distinction between words and symbols see Charles De Koninck, *The Hollow Universe*, Les Presses de L'Université Laval, 1964. Delivered as the fourth series of Whidden Lectures at McMaster University, 1959.

are necessary for achieving simplicity and elegance, it is *a faculty that can be developed in the student.* This development does, however, require a more meaningful education than simply being told to 'choose meaningful names'.

3 Predicable Relations

Traditional logic, with its focus on grasping *what something is* prior to reasoning to conclusions about it, is necessarily concerned with *predication.* For a logician, 'to predicate' means simply 'to say something of a subject.' These terms, 'subject' and 'predicate', as logical terms, are co-relative: a *subject* is *that of which something is said* and a *predicate* is *that which is said of a subject.* In any logical statement, therefore, a predicate is either *affirmed* or *denied* of a subject and this is done by means of the verb 'to be' as a copula since we are stating something about the very 'being' of the subject. As very simple examples, to say that *humans / are / rational* is to affirm the being of a rational nature to humans, and to say that *humans / are not / machines* is to deny the being of a machine nature of humans. It is always the predicate which is either affirmed or denied by means of the verb copula.

Essence and Accidents Now what is said about the subject either is said about the subject *essentially* or *non-essentially.* If it is said essentially, it is telling us *what the subject is.* If it is said non-essentially (or *accidentally*), it is telling us *something else about the subject,* other than 'what it is', as something inhering in in or *belonging to the subject.* Fred Brooks uses this distinction between *essence* and *accidents* for determining *what software is* in his well known essay *No Silver Bullet.* Traditional logic calls these ways of saying something about a subject either 'essentially' or 'non-essentially', *predicable relations.*[14] Each of the predicable relations involves the saying of something one (hence, *more universal*) of many.

Genus and Species If the something one is said (1) *essentially* and (2) of *many subjects which differ only by quantity or number*, such as 'human' is said of 'human, bat, dolphin, etc.,' then the relation is that of *genus* to the constituent species of which it is composed. What is *species* can only be said of individuals. What is *genus*, is said only of a *less universal* species. Notice then that 'mammal' could be both genus and species at the same time — but only in different relations of universality — for it is a genus with respect to human, but it is a species of animal.

Carolus Linnaeus, an eighteenth century Swedish botanist, adopted the terms species and genus from traditional logic to biology, which is where most people today encounter the use of the terms. But we need to be aware of differences in the newer uses of the terms. In biology, species is used *only for the least universal classification*, whereas in traditional logic it is used (analogously) for *any less universal classification.* Similarly, in biology, genus is used *only for the second least* universal classification, whereas in logic it is used for *any more universal classification.* In biology, for example, we now say that homo sapiens is a species in the genus mammalia, which is in the animal *kingdom.* In biology we would not say that mammalia is a species of animal, for that is the logical use of the word.

The predicable relations of genus and species go by many names in the computer science literature. Some of the terms used for species are:

$$\text{Human is a } \left\{ \begin{array}{l} \textbf{species} \\ \textit{specialization} \\ \textit{sub - set} \\ \textit{sub - class} \\ \textit{derived - class} \\ \textit{child - class} \\ \textit{descendent - class} \end{array} \right\} \text{ of Mammal.}$$

And some of the common words for genus are:

$$\text{Mammal is a } \left\{ \begin{array}{l} \textbf{genus} \\ \textit{generalization} \\ \textit{super - set} \\ \textit{super - class} \\ \textit{base - class} \\ \textit{parent - class} \\ \textit{ancestor - class} \end{array} \right\} \text{ of Human.}$$

The terms *specialization* and *generalization* show their etymology as derived from species and genus, as do the words specific and generic, as well as special and general.

The terms *sub-set* and *super-set* are borrowed from mathematical set theory. With this naming, what we mean by 'humans are a species of mammal' is that some, but not necessarily all, of what are mammals may also be identified as humans.This is easily represented on a Venn diagram, with a large circle to represent all mammals, and a smaller circle inside it to represent those

[14]The notions of the predicable relations were first set out in a treatise of Porphery the Phoenician entitled *Isagoge* and the clear grasp of these relations of the predicate to the subject is one of the most difficult tasks in the first act of reason: *simple apprehension.* We should point out that misunderstandings about these notions led to the great 'problem of universals' at the close of the Middle Ages; some perceptive thinkers such as Richard Weaver (cf. *Ideas Have Consequences*) and Marshall McLuhan attribute the anthropocentric and contemporary insistence on method for its own sake (devoid of content) to the inability to grasp and resolve the problem.

mammals that are humans.

Sub-class and *super-class* are variations on the subset and super-set terms. Bertrand Russell used class to mean set. In most object-oriented programming languages, the word class is used for category or species: hence, we speak of classification, categorization and specification.

The terms *base* and *derived* come from Bjarne Stroustrup (father of C++), because he discovered that many people, particularly those without a background in mathematics, found the sub- and super- terms to be confusing.[15] Stroustrup notes that many find it counter-intuitive that the code for the sub-class (derived) is typically larger than the code for the super-class (base). This confusion is caused by fallacious reasoning: the terms super and sub do not refer to source code length, but to what is more universal and what is more particular.

The terms *child* and *parent* definitely seem to originate from an over-extended metaphor. A classification system may be visualized as a tree; but a classification system is not actually a tree: we are into metaphor. Like the data structure, it shares an accidental resemblance of form with certain types of plants. A family-tree may also be drawn in a like fashion, with prior generations (parents) in the place of genus, and the younger generations (children) in the place of species. From this metaphor (the accidental resemblance between a diagrammatic representation of a classification system and a diagrammatic representation of a family) the terms *parent* and *child* enter into the language of computer science. The terms *descendent* and *ancestor* are also taken from the extended family-tree metaphor.

The relation of universality of a genus to a species, which is commonly known in the vernacular of object-oriented programming as *inheritance*, also seems taken from the extended family-tree metaphor: the child *inherits* certain traits from the parent. We need to recall an earlier point that metaphorical use of language is equivocal; to use the double metaphor of the family-tree verges on fallacy. The parent is not more universal than the child, nor is the child a more specialized version of the parent, nor does a child represent a sub-set of parents: both are individuals bound by the same limits of ordinary human frailty as each of us.

Metaphors show us likenesses — but they are the instrument of our poetic nature, and properly used to excite wonder. They do not provide a well-reasoned grasp of what things are in their own right and in their proper relation. That so much of the terminology of computer science is metaphorical is, in certain respects, an embarrassment, for it can actually constitute an impediment to clear thinking. Ultimately, confusion must result when a programmer, brought up on metaphor, fails to grasp clearly the relations to be set in place.

[15] Bjarne Stroustrup, *The Design and Evolution of C++*, Addison Wesley, 1995 (reprint with corrections), p.49.

Proper and Common Accidents If the something one is said (1) *non-essentially* (not in the line of 'what the subject is') but (2) as *something which is found exclusively in this species*, then it is a *proper accident* or *property*. Notice that proper is the root of property. For example, nursing of the young with milk is a property of all mammals; in fact, mammals are named for this property (from the mammary gland).

If, on the other hand, something one is said (1) *non-essentially* (not in the line of 'what the subject is') but (2) as *something which is not found exclusively in this species* (it is more generic), then it is a *common accident* or simply *accident*. For example, living on land is a non-defining characteristic of many species of animal.

Now, object-oriented programming is a practical rather than a philosophical endeavour, and so most object-oriented programming languages do not make a distinction between proper accidents and common accidents. Often, both are referred to as properties, attributes, or characteristics. However, understanding this difference is of critical importance for both the logician and the programmer, as the basic skills needed for division and definition; for understanding clearly what each thing is.

4 The Many Forms of the Syllogism

Argumentation, the third and final act of reason in coming-to-know, involves arriving at a valid conclusion which unites a subject with a predicate and which follows from what the subject is. From the time of Aristotle until just a few hundred years ago, the syllogism was the only widely used formalism. All of the valid figures and moods of an argument or syllogism are the result of a pure formalism in which the conclusion necessarily follows from certain premises.[16]

In a pure formalism, the validity (consistency) of an argument can be assessed without any reference to what is being reasoned: but just as the conclusion must be caused by the form of the argument, (and this suffices for symbolic logic) a valid argument in traditional logic also demands that the truth of the conclusion must follow from the truth of the premises. In the practical world, if the content of a conclusion matters, then so do the premises. That is what is really meant by the computer science maxim *garbage in, garbage out.*

Ever-increasingly sophisticated formalisms, and the means of assessing their consistency (validity), have

[16] The *figure* of a syllogism is determined by the position of the *middle term* (which is the reason for uniting the predicate with the subject in the conclusion). The *mood* of a syllogism refers to the *quantity* (universal or particular) and the *quality* (affirmative or negative) of each of the statements (premises and conclusion). Every statement has certain properties which flow from its quantity and quality.

been developed in symbolic logic. Nevertheless, the forms of the traditional syllogism are still very well suited to explaining the basic structure of object-oriented programs. Consider the following simple argument, expressed as a syllogism in the first figure:

> All mammals nurse their young.
> All humans are mammals.
> Therefore, all humans nurse their young.

The first two propositions, the premises, can be expressed directly by the programmer in source code, as is shown below. The third line, the conclusion, is implied by the programmer and can be deduced by the computer based solely on the form of the argument: this is part of what makes object-oriented programming so powerful.

Major Premise *All mammals nurse their young.* This proposition predicates a property of the genus mammal. The major premise is always the one that contains the predicate of the conclusion (nursing of the young, in this example). In object-oriented programming, properties are expressed as either fields or methods. A method is most appropriate for this property, as it is an activity that the subject performs:

```
class Mammal {  void nurse() {}  }
```

Minor Premise *All humans are mammals.* This proposition indicates classification: that human is a species of the genus mammal. The minor premise always contains the subject of the conclusion (humans, in this example). As mentioned previously, classification is often referred to as inheritance in object-oriented programming; in Java it is signified primarily with the `extends` keyword:

```
class Human extends Mammal {}
```

Recall that the `extends` keyword was not used in the declaration of `Mammal` (major premise). Scientifically, we would classify mammals as animals, animals as living things, and living things as substances. Substance is as far as this classification can go: it is a *supreme category*. There is no classification of mammals that is more general (i.e. universal) than substance. Substance is one of the ten supreme categories identified in traditional logic. In most object-oriented programming languages the single and only supreme category is *object*. This is what is meant by the common phrase *everything is an object.*[17]

In Java, since the source code declaration of `Mammal` does not specify a genus, the compiler will make `Mammal` a species of the single supreme category `Object`; as if the declaration had been written like this instead:

```
class Mammal extends Object
```

Conclusion *Therefore, all humans nurse their young.* This statement, the conclusion, combines the subject of the minor premise with the predicate of the major premise to arrive at something new from what was previously more known.

This particular conclusion is both true and valid, and advances the philosopher's objective of knowing the natures of things. However, the programmer's objective is not to understand the nature of the species, but rather to be able to draw conclusions about individuals (objects) of the species which can then be used as a basis for action.[18] That is, the programmer wishes to use this conclusion as the major premise for arguments similar to the following:

> All humans nurse their young.
> Mary is human.
> Therefore, Mary nurses her young.

Now, while this argument is formally valid, the conclusion is only probably true. For example, Mary may be too old or too young to nurse, or may not have any children, and so on. Furthermore, it would not be true to draw the same conclusion about James or Ian, even though they are also human. Note that the content of the argument must be understood in order to assess the truth of the conclusion: we must know what we are talking about.

The main characteristic of this argument that distinguishes it from the previous one is that this one is concerned with an individual instead of with the nature of the species. This is the root of the difficulties. Traditional logic and object-oriented programming deal with these difficulties in very different ways: traditional logic understands that the whole of the nature of the species is not to be found in a given individual; object-oriented programming assumes that each individual (object) conforms to the species (class) exactly. This assumption is necessary for computers because they are unable to understand the content of the argument.

As mentioned above, the programmer wishes to draw this conclusion about the individual as a basis for action. The programmer is attempting to solve a practical problem by instructing a computer to act in a particular fashion. This requires moving from the indicative to the imperative: from *can* or *may* to *should*. For a philosopher in the tradition, this move from the indicative to the imperative is the domain of *moral science.*[19]

[17] C++ differs from most object-oriented programming languages in that it does not have a single supreme category.

[18] Cf. Aristotle, *Nicomachean Ethics*, Book 1, Chapter 3, 1095a5.

[19] For an interesting contemporary discussion of this particular point see C.S. Lewis, *The Abolition of Man.*

A Line of Code With the above considerations under our belt we may now examine a line of imperative code:

```
static void baby(Mammal mother) {
    mother.nurse();
}
```

This code snippet defines a function named `baby` which takes one parameter of type `Mammal` that is named `mother`. The single instruction is for the `mother` to `nurse`.

With respect to the previous argument, `mother` stands as a placeholder for Mary; a *reference* or *pointer* in programming parlance. Every time this code is executed `mother` may refer to a different individual: first Mary, then Alexa, then Renée, and so on. This is simply the notion of a parameter, which is common to almost all programming languages.

However, `mother` is not limited to referring to humans: it may refer to any `Mammal`, for example a dolphin or a bat. This is the case because the previous two syllogisms can be adjusted by substituting dolphin for human and Flipper for Mary. Therefore, subject to the considerations given above, we may also conclude that Flipper nurses her young. This feature (that `mother` may refer to any individual of any species of mammal) is unique to object-oriented languages, and is known as *polymorphism*.[20]

Now, dolphin mothers nurse in a different fashion than human mothers: dolphins live underwater, and so must hold their breath while nursing; humans commonly wear clothes, which usually need to be adjusted before and after nursing. Assuming that this distinction is relevant to the problem at hand, one would expect that the class definitions for `Dolphin` and `Human` each contain a `nurse` method with a unique sequence of instructions. The sequence of nursing instructions that is branched to from the `baby` method will depend on the species of the particular `mother` object: if the `mother` is a `Dolphin` then she will hold her breath while nursing; if the `mother` is a `Human`, then she will adjust her clothing before and after nursing.

Notice the use of the words 'she' and 'her' above: they are used correctly to signify the female sex with respect to both women and female dolphins; however, they are used metaphorically with respect to objects in a computer memory. Metaphors of this variety are known as *anthropomorphic*, for they assign human form to something that is not human. In fact, even the term memory when applied to computers is anthropomorphic. These metaphors can be convenient, and helpful in learning, for they explain the thing less familiar (in this case a computer program) in terms of a thing more familiar (in this case a mother). As with all equivocation, however, they can also lead to confusion and error: this happens occasionally in the computer science literature when the writer uses anthropomorphic language and then comes to the fallacious conclusion that there is no difference between man and machine. Those who can no longer see the difference between themselves and a block of memory in a computer are to be pitied.

[20] Etymologically, *polymorphic* means 'having many forms'; as *polynomial* means 'having many terms', and *anthropomorphic* means 'in the form of a human'.

5 The End

Symbolic logic is vital to the student of computer science in coming to understand how machines operate: by the abstract manipulation of symbols, with logical consistency, and without regard for meaning or content. However, in programming we name things with *words* that have *meaning* to the programmer (although not to the machine). It is simply foolish to pretend that human reason is not concerned with meaning, or that programming is not an application of human reason.

Traditional logic is concerned with developing the student's ability to understand what each thing is; that is, with developing the student's ability to handle meaning. This is important to all students insofar as they are human, and therefore their reasoning is concerned with content. It is doubly important to all students of computer science, whose future success will depend upon their ability to express complex technical ideas clearly to both computers and their colleagues. It is triply important to all students of object-oriented programming, and the reasons are twofold: (1) the form of object-oriented programs reflects the form of the syllogism; and (2) traditional logic is where the ideas underlying classification are considered as such, without resorting to metaphor.

Acknowledgements

We wish to express our gratitude to four gentlemen who have graciously given their time to the development of this text: Paul Asman, Scott Kerr, Marcellus Mindel, and Tarver Szwejkowski. Each of them has provided unique insights and the paper is better for their counsel. The anonymous SIGCSE referees provided meaningful feedback that has improved the paper. We would also like to acknowledge readers who have made helpful comments: George Bragg, Bill Cowan, Leigh Davidson, Mike Godfrey, Stewart Ladd, John Mylopoulos, and Stephen Perelgut. Professor Mohamed Kamel and the students of Systems Design 221 deserve special mention for encouraging very early versions of this work. Finally, Professor Kostas Kontogiannis, Stewart Ladd, and the Centre for Advanced Studies at the IBM Toronto Laboratory have been generous with their resources.

We are working on an extended version of this paper.