1986

Modeling Location for Cadastral Maps Using an Object-Oriented Computer Language

Daniel Kjerne
Portland State University

MODELING LOCATION FOR CADASTRAL MAPS USING AN OBJECT-ORIENTED COMPUTER LANGUAGE

by
Daniel Kjerne

1986

Center for Urban Studies
School of Urban and Public Affairs
Portland State University
Portland, OR 97207-0751
(503) 725-4020
(503) 725-5199 FAX
http://www.upa.pdx.edu/centers.html#CUS

Daniel Kjerne
Geography Department
Portland State University
P.O. Box 751
Portland, OR 97207

MODELING LOCATION FOR CADASTRAL MAPS USING AN OBJECT-ORIENTED COMPUTER LANGUAGE

ABSTRACT. The challenge to designers of multipurpose computer-aided land information systems is to capture enough of the "deep structure" of the problem domain to enable a system to answer user questions and requests in a satisfactory way. Criteria defining "satisfactory" in each case must include considerations of accuracy, completeness, timeliness, and cost. These considerations, when applied to the location in space of parcel boundaries and property corners, present unusual difficulties. This is in large measure due to the fact that important elements of the field measurement process, and the determination of location based on what are essentially logical (legal) abstractions, are problematic or impossible to capture, store, and display either in the graphic object (the map) or in computer storage as presently constituted.

The problem of representing these data is approached through the use of an object-oriented computer language which treats each individual cadastral object as storing internally the method, reference object(s), and measurement(s) by which it was located in the field. The concept is tested in an application of Neon™, an object-oriented language implemented on the Macintosh™ computer.

INTRODUCTION

Since the first National Research Council report on the need for a multipurpose cadastre (8), workers in land records modernization have gained experience with local government agencies and private-sector enterprises in defining the user needs and considerations, both institutional and technical, involved in implementing such a system. The original concept of a single,

unified parcel-based system is being replaced by a concept that relies on coordination of data exchange standards and software integration of users' **layers** for specific tasks. In general, the meaning of this crucial term has been left to the reader to define, but it seems to involve the data-storage expression of a cartographic "theme". That is, it is a set of computer files, a collection of data sets, which is the concern of a particular agency or user, and includes locational and attribute data enabling it to be displayed and analyzed as if it were a map. Chrisman and Niemann proposed such a "layer-based" system with no single permanent basic unit after working with local, state, and federal agencies operating at the county level in Wisconsin (3).

A similar concept appeared in a report prepared for the Multnomah County (Oregon) county assessor (5). Under that concept, the **multipurpose cadastre** is a collective term for that group of closely integrated land information systems used and maintained by the county surveyor, county recorder, and county assessor. Each of these users has its own set of questions and expected form of answers; in this sense each user perceives, works with, and maintains a separate land information system consisting of a layer and its supporting hardware and software. But there is a high degree to which all three users rely on each others' data as well as a large overlap between the sets of questions and answers. Thus, from the point of view of an observer outside these three agencies, the multipurpose cadastre is "one thing".

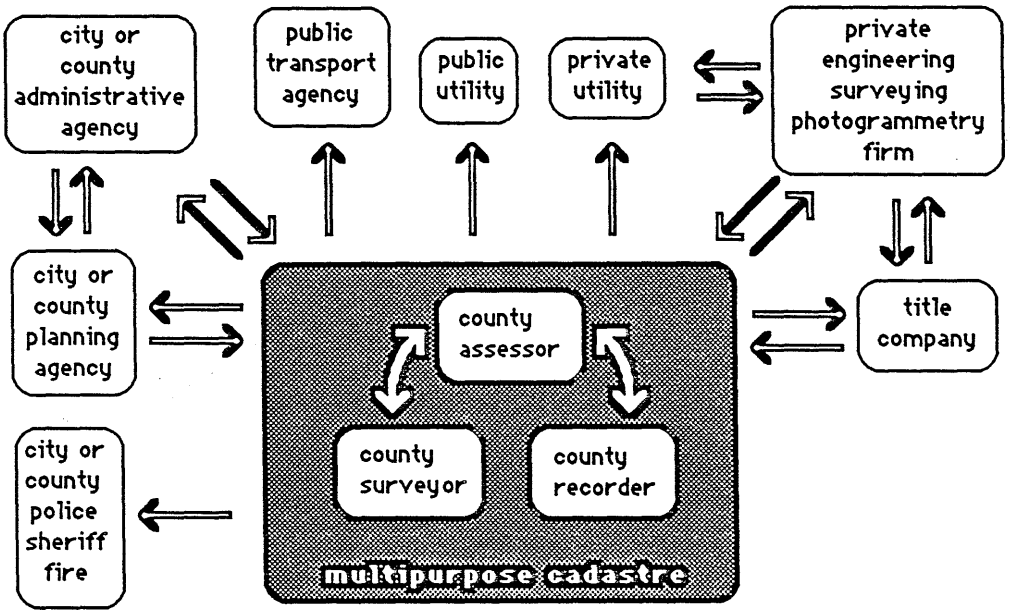
Other users of land records, public and private, rely on data available from the multipurpose cadastre, and supply data to it. As these users develop and implement their own land information systems, there will be an ongoing need to develop and maintain standards to allow data sharing among different systems. Figure 1, portraying the systems and the flows of data between them, illustrates the relationship between the multipurpose cadastre and other land information systems in a local area.

The **cadastral layer** is the special concern of a county assessor. It maps the location of objects such as property corners, boundary lines, street and utility rights of way and easements, and contains linkages to data types such as owners' names, site addresses, land use, flood plain, zoning, jurisdictional boundaries, and a wide variety of other land data. In the non-computerized assessment agency, part of these functions are performed using assessor's maps, which are often used as a base for locating other types of objects by other agencies. The location of cadastral objects is

thus seen as crucial data not only for those maintaining the cadastre, but for other agencies in the community of land records users.

FIGURE 1

THE MULTIPURPOSE CADASTRE AS AN ELEMENT IN A SYSTEM OF LAND INFORMATION SYSTEMS



LEVELS OF DESCRIPTION

The design strategy for a land information system may be thought of as entailing the preparation of a series of descriptions, of increasing specificity, of the entities and relationships between entities which are the subject of the questions directed to the system. The first description will, of necessity, be general in nature; it would only be implementable -- that is, useful in producing answers to questions -- with the aid of a highly sophisticated "knowledge base", hardware, and software configuration. So sophisticated a configuration, in fact, that it only exists now in the form of humans, who can take a verbal description of a problem, a set of data, and tools such as pen, ink and calculator, to produce answers to questions.

Once a description has been achieved at this general level, it must be "translated" into a more specific description so that it can be implemented using a less

sophisticated hardware/software configuration. The objective is to preserve the ability to answer questions, but to, in effect, move the expertise of the higher-level description from the minds of the human experts into the information system.

The number of levels necessary to traverse between the human verbal description and the machine code implementation varies with the type and complexity of the problem. Nyerges (9) discusses the design of a cartographic data base in terms of six levels of description, while pointing out that other workers have described the design process using larger and smaller numbers of levels. For the problem of locating objects in the cadastral layer, the present paper will sketch out descriptions at two different levels of the six described by Nyerges. The descriptions correspond to level 1 and level 4 of Table 1. The first of these, the **information reality** level, is that level used by humans in formulating and solving a problem in a specific area. The second, at the **data structure** level of description, is the highest level of machine implementation, corresponding to a high-level language (or, perhaps, database query language or spreadsheet modeling) description.

TABLE 1

SIX LEVELS OF DESCRIPTION,
FROM LEAST TO MOST SPECIFIC (ADAPTED FROM (9))

	1) information reality
infological models	2) information structure
	3) canonical structure
datalogical models	4) data structure
	5) storage structure
	6) machine encoding

The descriptions (especially the second one) will necessarily be incomplete, in a paper of this scope, but the intention is to suggest how a particular programming language paradigm -- object-oriented design -- can describe the model used by those professionals and technicians who have the problem of cadastral location as their field of expertise: property surveyors and cadastral cartographers.

MODELS OF CADASTRAL OBJECT LOCATION

Information Reality Level of Description

A property surveyor locates **monuments** and **property boundaries**. Monuments may be natural or artificial. Natural monuments are objects such as trees, boulders, streams, ridges, and so on. Certain artificial structures, such as buildings or curbs, are considered durable and stable enough that they are classed as natural monuments. Artificial monuments are usually objects such as iron bars or rods driven in the ground or brass disks set into concrete piers. Natural and artificial monuments, in other words, are physical objects. Boundaries, on the other hand, are abstract objects, which may or may not be marked on the ground by natural or artificial monuments (2, pp. 15-16).

In locating monuments and boundaries, the property surveyor gathers and evaluates evidence, including evidence of title, measurements, testimony, calculations, and so on. The surveyor does not determine ownership -- that is the province of the courts. But the surveyor does determine, based on a preponderence of the evidence, where each cadastral object on the survey in question is located on the ground, and on which evidence that location is based. Thus, once the various levels of evidence have been evaluated, the location of objects in the field and in the office is reducible to a series of actions recorded in field notes and plats, diagrams, and survey reports. In principle, it is possible to locate any object surveyed -- monument or boundary -- with a knowledge of the procedure involved, the objects from which measurements were made, and the value of those measurements.

The **cadastral cartographer** is the technician in an assessor's office who maps cadastral parcels. The descriptions, survey records, and plats prepared by property surveyors are the principal resource used in construction of cadastral maps. While the surveyor is concerned with one or a few parcels of land at one time, the cadastral cartographer must map all the parcels in an area, deciding at the time the map is constructed which parcel descriptions to trust and which to hold in a lesser light.

The hierarchy of evidence used in this process to locate cadastral objects is basically the same as that used by the property surveyor. The cadastral cartographer is concerned with locating the same kinds of objects as the property surveyor, with the addition of a new kind: **control**. These are objects whose location, rather than being determined relative to other

points, is "given." They provide a framework within which to locate groups of monuments and boundaries (located by property surveyors) relative to each other.

In present non-computerized cadastral mapping systems, locations of objects are updated by, essentially, reconstructing the maps from scratch at more or less frequent intervals (10, pp. 335-365). The knowledge base for this cadastral object location system obviously is not totally contained in the drafted cadastral map. When a map is updated, the cartographer must refer to the deed and survey records, recording on the map only the finally determined position of the objects. Thus we may say that the cadastral object location system, in an assessor mapping system, consists of three separate entities in different locations: 1) in the paper maps, which show the most recently derived location of objects; 2) source location data in the survey records and deed descriptions; and 3) in the "expert knowledge" embodied in the surveyors and cadastral cartographers who maintain the system.

Many attempts to computerize the cadastral layer seek to avoid the reconstruction process by digitizing existing assessor's maps and mathematically "rubber sheeting" them to control points. This provides a land base image sufficient for the needs of many users of parcel location data -- for instance, for mapping electical utility distribution networks -- but it fails to provide adequate and consistent spatial registration in the long term. In particular, it is inadequate for property and engineering survey needs and, to a lesser extent, those of assessment mapping in highly developed areas.

It is important to note that the relationships in surveys and deed descriptions involved here are actually deep structural relationships (9, p. 36) which manifest themselves as spatial relationships when the objects in the map are displayed. Any one cadastral object is not related to every other object of concern in the location system, but to a restricted subset: the specific objects used by the property surveyor or the cadastral cartographer in determining its location. It is thus possible for a new survey or deed description to change the derived location of a property boundary or monument without at all affecting the derived location of a nearby monument or property boundary. The second object may have been located using an entirely different description, depending on a different set of control points. Conversely, depending on which survey or deed description established a point, its location could be altered by a change in location of an object located relatively far away. Consequently, rubber sheeting or

least squares adjustment is an inappropriate method for updating cadastral object locations. What is needed is a data structure that captures the structural/spatial relationships inherent in deed descriptions.

A prior report, mentioned above, explored the use of a relational data schema to store the structural/spatial relationships of cadastral objects (5). The basis for that schema was a topological/relational schema presented by Van Demark (11). The object-oriented design approach described below is an attempt to translate the concepts developed in the relational data schema into a more powerful, intuitive medium.

An Object-oriented Data Structure Level Description

Object-oriented programming is based on a paradigm of objects responding to messages, rather than on one of operators performing actions on operands, as is the case with procedural languages. As a programming style, it began in the early 1960's with the development of Simula by the Norwegian Computation Center in Oslo, Norway. Simula was the first language to implement the Class construct. In the early 1970's, the Learning Research Group at Xerox Palo Alto Research Center began implementation of the Smalltalk programming environment. Smalltalk was the first system to be designed completely around the Class/Object concept. Many other languages have since provided support for classes, objects, and subclassing, including CLU, Ada, C++ (an extended version of C), and several versions of LISP, though objects and classes are not as well integrated into these languages as they are in Smalltalk (6).

Under this paradigm, the computer is conceptually divided into a number of **objects**, or **instances**, each of which can itself act like a small computer, and be given a role like that of an actor in a play (7). Each **class** of objects may have its own private data types, or **internal variables**, and actions, or **methods**, which objects of that class perform upon receiving a suitable message. This segmentation allows a great deal of flexibility in the system considered as a whole, since new classes of objects can be defined without worrying about the side effects their behavior might have on the private data or actions of other, already defined classes of objects. (Coping with such side effects is definitely a consideration when programming with procedural languages). The main feature of object-oriented design, though, is that it allows a fairly direct translation of the behavior of an object conceptualized at a higher level of description to that of an object class at the programming level of description.

This feature allows the structural/spatial relationship noted in the information reality description of cadastral objects to be implemented at the data structure level. For instance, a particular class of object, such as a property corner, can be defined which will "know" that its location is determined by a bearing and distance from another monument. It will contain a private method to compute its location in response to a message from another object, "Give me your location." The value of the bearing, the value of the distance, and the identity of the monument are all private data belonging to that property corner. These data cannot be changed without a specific message to that object. But if the direction of north or the location of the reference monument were to change, the location of the property corner would also shift, without having to alter data belonging to the corner.

Figure 2 indicates another feature of object-oriented design: **inheritance** of object behavior. An object class in a lower position on the tree has the same internal data types and can perform the same methods as objects higher on the tree, in addition to possessing methods and data types unique to itself. In this way, it is easy to define new object classes which are like already existing objects, but have additional characteristics. An object of class ControlPoint can do certain things specific to its class: for instance, it can display itself in a certain format (perhaps by reference to a bitmap) in response to a message. In addition, the ControlPoint class of objects inherits the methods and data types of the ZeroCell class of objects. An object of this type can accept a message to store its location in x,y coordinates and will return its location in response to another message. The ZeroCell, OneCell, and TwoCell object classes are also at the appropriate level for storage of topological data and methods.

As another example, MonumentPointAngleDistance objects are located by a specific location rule ("by distance A from object B, at an angle C from object D, turned from object E"). Each location rule corresponds to a separate class of objects; thus, MonumentPointOffset objects might be located by "intersection of the offset A from the line formed by objects B and C and the offset D from the linear object E." All the MonumentPoint[RuleName] object classes are subclasses of the MonumentPoint object class, which has a set of private methods and data variables inherited by all its subclasses (one such might be, for instance, a method for displaying itself).

FIGURE 2

HIERARCHY OF INHERITANCE OF OBJECT CHARACTERISTICS FOR CADASTRAL OBJECT CLASSES

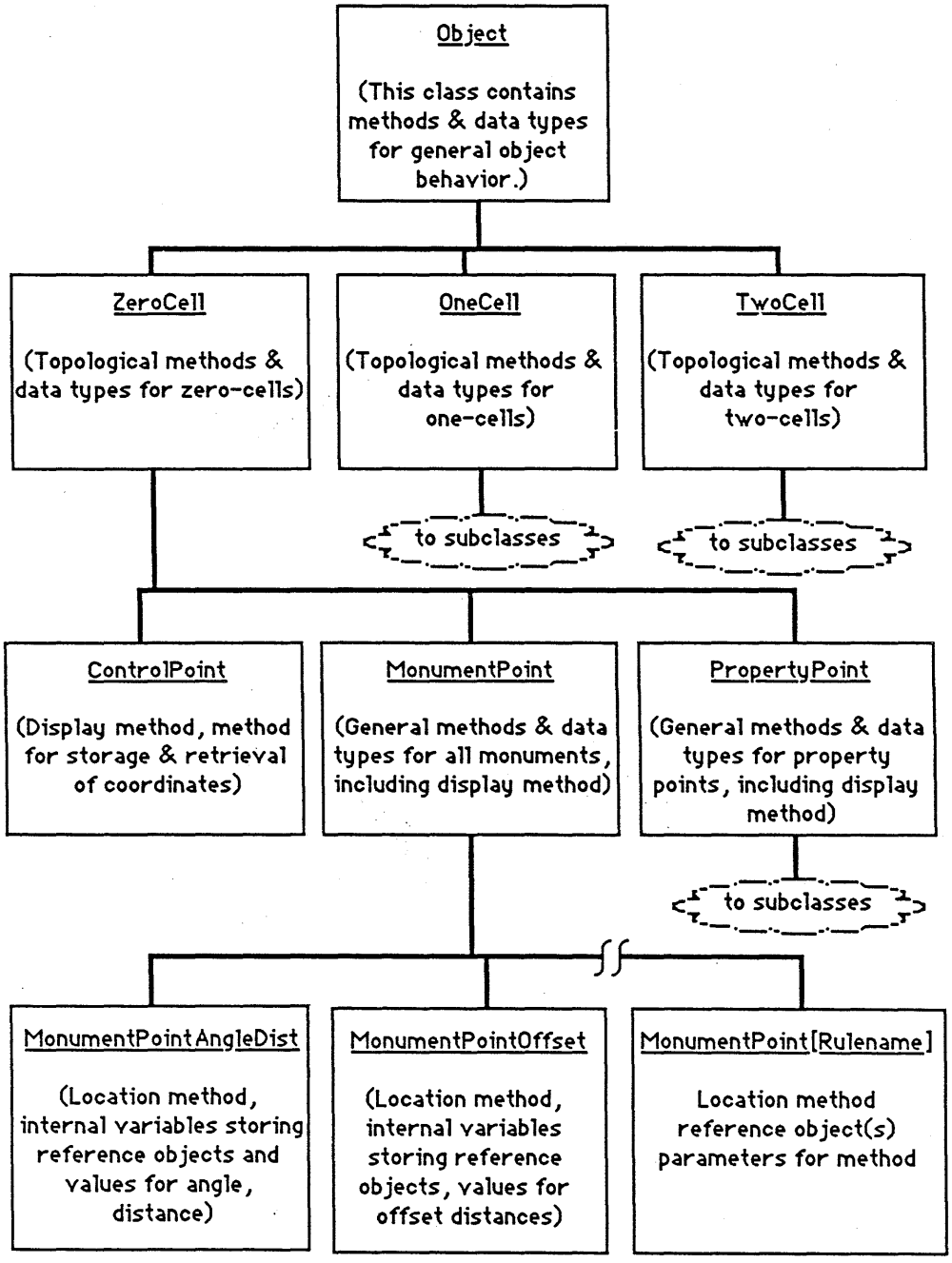


Figure 2 is only faintly suggestive of what would be the true extent of the cadastral object location hierarchy: in addition to the ControlPoint and MonumentPoint subclasses of the ZeroCell class, a third subclass, PropertyPoint, is indicated. This class of objects also inherits characteristics of the ZeroCell class, and would have a series of subclass object classes (not shown) with different locational rules, similar to those of the MonumentPoint[RuleName] classes. In addition, linear monuments and property boundary classes (again, not shown) would be defined under the OneCell object class, with subsidiary classes, each with its own locational rule.

Each combination of locational rule, reference object(s), and parameter(s) determines location for each object. General rules are applied by humans -- property surveyors and cadastral cartographers -- to specific instances, finding one and only one location for each object. In other words, this is not an "expert system" capable of evaluating constraint rules (1), but, essentially, a system capable of storing the location decision made by the person compiling the cadastral map.

Listings 1-4 in the Appendix are object class definitions of some of the objects indicated in Figure 2 and in the discussion above. They are written in what might be called "Neon pseudocode". Neon™ is an object-oriented language derived from FORTH and SmallTalk-80, implemented on the Apple Macintosh™ computer, while the term "pseudocode" simply means that the details of calculation and bitpushing have been left out in order to make the basic idea a little more clear. Listings 5-9 are working Neon code for many of the same objects, and hence appear somewhat more arcane.

One of the many possible locational rules for cadastral objects, each with its own object class, is suggested by the object class definition for MonIdent (Listing 10). Since it models the locational behavior of an object located at the identical spot as its reference object, this object class doesn't store its location as x,y coordinates. Instead, it has an internal variable that is a pointer to its reference object. When a MonIdent receives a **locate:** message, it consults its private **locate:** method, which tells it to send another **locate:** message to the object pointed to in its RefPt variable. If the pointed-to object is an object of type CtlPt, that object's own **locate:** method will have it return the value in its Location variable.

On the other hand, if the pointed-to object is of the class MonAngDist (Listing 4), its locate: method actually calculates coordinates based on the values in

its internal variables and the location of its reference object.

CONCLUSIONS AND DIRECTIONS FOR FUTURE INVESTIGATION

At this writing, classes of objects have been defined in Neon that "know" their location by reference to another object(s) and will move their displayed position if the reference object(s) location changes. It is possible to model cadastral object location as it is understood by property surveyors and cadastral cartographers. The use of an object-oriented design renders such definitions straightforward. There remain important questions having to do with the practicalities of implementation.

First, much more work must be completed on the schema of object location rules. The true scope of the number of object classes involved needs to be determined, as well as how such a schema would be integrated with the other functionalities of a land information system. At present, it appears that between thirty and one hundred different object classes would be required to locate cadastral objects. Should one use an object-oriented design in the area of cadastral object location only, grafting this capability to existing geographic information system design (4), or would it be effective to build a whole system using a unified object-oriented design approach? The latter approach will require adaptation of spatial data handling operators existing in procedural languages to methods internal to objects.

The second area of investigation involves consideration of economic impacts. That is, what are the comparative costs and benefits of a highly structured cadastral layer as compared to a simple image file? Given that updating a cadastral layer ineluctably involves reconstruction of location from source records, is it more cost-effective to integrate the locational structure within the data base, so that reconstruction is continuous, or to rely on manual reconstruction and redigitization of maps at intervals?

At present, efforts are continuing to flesh out the schema of object classes indicated in Figure 3. The hope is to demonstrate, using part of an assessor's tax map, the feasibility of mapping objects with a whole range of locational rules.

REFERENCES

(1) Borning, Alan, 1981. "The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory." ACM Transactions on Programming Languages and Systems, Vol. 3, no. 4 (October 1981), pp. 353-387.

(2) Brown, C.M., 1969. Boundary Control and Legal Principles, 2nd ed. New York: John Wiley & Sons.

(3) Chrisman, N., and B.J. Niemann, Jr., 1985. "Alternative Routes to a Multipurpose Cadastre: Merging Institutional and Technical Reasoning." Auto-Carto 7 Proceedings, pp. 84-94. Falls Church, Virginia: ACSM-ASPRS.

(4) Cox, B.J., 1984. "Message/Object Programming: An Evolutionary Change in Programming Technology." IEEE Software, January 1984, pp. 50-61.

(5) Dueker, K.J., L.M. Conrad, and D. Kjerne, 1985. Draft User Needs Assessment and Technical Issues for a Multipurpose Cadastre for Multnomah County, Oregon. Portland, Oregon: Center for Urban Studies, Portland State University.

(6) Horn, Bruce, "Neon, Version 1.0" (software review). Dr. Dobb's Journal, October 1985, pp. 96-100.

(7) Kay, Alan, 1984. "Computer Software." Scientific American, Vol. 251, no. 3 (September 1984), pp. 53-59.

(8) National Research Council (NRC), 1980. Need for a Multipurpose Cadastre. Washington, D.C.: National Academy Press.

(9) Nyerges, T., 1981. "Cartographic Information Modeling as a Theoretical Basis for Cartographic Data Base Structures." Paper presented at the Second International Hypergraph-based Data Structures Seminar, Richmond, Va., 9-13 March 1981.

(10) Oregon Department of Revenue (ODR), 1981. Manual of Cadastral Map Standards, Concepts, and Cartographic Procedures, 2nd ed. Salem, Oregon: ODR.

(11) Van Demark, P., 1985. "A Model Schema for Geographic Base Layer." Survey sent to Spatial Oriented Reference System Association (SORSA) members.

APPENDIX

----Listing 1----

```
:CLASS Object          <Super Meta

( This object class has general methods, like GET: and
  PUT:, which allow the storage of data into internal
  variables. All objects in an object-oriented language
  inherit, directly or indirectly, from the Object class
  )

;CLASS                                  \ Terminates class
                                        \ definition
```

----Listing 2----

```
:CLASS ZeroCell        <Super Object    \ Begins class
                                        \ definition;
                                        \ indicates superclass

       Point           Location         \ These lines are for
                                        \ the definition
                                        \ of internal
                                        \ variables.  The
                                        \ variable "Location"
                                        \ (a Point data
                                        \ type) holds x,y
                                        \ coordinates.

;CLASS
```

----Listing 3----

```
:CLASS ControlPoint    <Super ZeroCell

                                        \ no new internal
                                        \ variables

:M                                      \ beginning of
                                        \ internal method
                                        \ definition

  ( theX theY -- )                      \ this comment shows
                                        \ the condition of the
                                        \ stack before and
                                        \ after method
                                        \ execution

DEFINE:                                 \ name of method

Put: Location                           \ this method takes
                                        \ two input values and
                                        \ puts them in the
```

```
                                        \ variable "Location"

;M                                      \ end of method
                                        \ definition
  ( -- theX theY )

:M LOCATE: Get: Location ;M             \ This method returns
                                        \ the values in
                                        \ "Location"

;CLASS
```

----Listing 4----

```
:CLASS MonAngDist    <Super ZeroCell

( This type of monument locates itself by azimuth and
  distance from another ZeroCell object.)

        Angle           Azimuth
        Real            Distance
        Ptr             RefPt

  (theRefPt theDistance theAngle -- )
:M DEFINE: Put: Azimuth Put: Distance Put: RefPt

( Define:, like the method of the same name in the
  Control object class definition, takes input values
  and stores them in internal variables.  Instead of
  storing x,y coordinates, this method stores the values
  used to compute the coordinates. )

;M

  ( -- theX theY )
:M LOCATE:

( computes the sine of Azimuth times Distance, adds to
  x-coordinate of RefPt, puts on stack; computes the
  cosine of Azimuth times Distance, adds to y-coordinate
  of RefPt, puts on stack )

;M

;CLASS
```

----Listing 5----

```
( ZeroCell -- point object class. Should have handle to
  list of OneCells bounding each ZeroCell, as well as
  location. dk 12 May 86 )

:CLASS ZeroCell      <Super Object

       Point         Location

;CLASS
```

----Listing 6----

```
( qdbitmap -- from qd1 -- 03/06/86 dk )

Decimal

( define the quickDraw bitmap object )

:CLASS qdBitMap      <Super  Object

       Var           BaseAddr
       Int           RowBytes
       Rect          BndsRect

( addr n l t r b --- )
  :M  PUT: Put: bndsRect  Put: RowBytes Put: BaseAddr ;M

( -- addr )                          \ gets abs addr of
                                     \ BndsRect
  :M  BNDGET: Addr: BndsRect +base ;M

;CLASS

( the following "colon definitions" define new words (á
  la FORTH) )

: SPOT  8 210 gotoxy ;

: .OK  -curs spot 12 spaces spot +curs ;

: +pair { x1 y1 x2 y2 -- x1+x2 y1+y2 } x1 x2 + y1 y2 + ;

:  CoordMsg  ." The coordinates are " ;
```

-----Listing 7-----

```
( load_bitmaps -- array and bitmap sources for control
  point and monument templates. dk 12 May 86 )

4 Array Ctlimage
: xx to: Ctlimage ;
hex
```

```
18002400 0 xx
42008100 1 xx
81004200 2 xx
24001800 3 xx
decimal
forget xx
QDBitmap CtlptSource
abs: Ctlimage 4+ 2 0 0 8 8 Put: CtlptSource
2 Array MonImage
: xx to: MonImage ;
hex
6000F000 0 xx
F0006000 1 xx
decimal
forget xx
QDBitMap MonSource
abs: MonImage 4+ 2 0 0 4 4 Put: MonSource
```

-----Listing 8-----

```
( Ctlpt -- this uses a separate bitmap, CtlptSource. )

:CLASS  Ctlpt        <Super ZeroCell
        Var          DestBits
        Int          Mode
        Rect         DestRect

:M GETPORT: Abs: DestBits call GetPort 2 +: DestBits  ;M

        ( theMode -- )
:M  CHMOD: Put: Mode  ;M

        ( theX theY -- )
:M  DEFINE:  { xloc yloc -- } xloc yloc Put: Location
             xloc yloc -4 -4 +pair putTop: DestRect
             xloc yloc 4 4 +pair putBot: DestRect .ok
;M

       ( -- theX theY )
:M  LOCATE:  Get: Location .ok ;M

:M  DRAW:   GetPort: self  Abs: CtlptSource Get:
  DestBits Bndget: CtlptSource Abs: DestRect Int: Mode 0
  call CopyBits .ok ;M

:M  REF:  CoordMsg Get: Location . . cr .ok ;M

;CLASS
```

----Listing 9----

```
( Mon -- contains display method for all monuments )

:CLASS  Mon          <Super ZeroCell
        Var          DestBits
        Int          Mode
        Rect         DestRect

:M GETPORT: Abs: DestBits call GetPort 2 +: DestBits

;M

        ( theMode -- )
:M  CHMOD: Put: Mode  ;M

:M  DISPLAY:   GetPort: self  Abs: MonSource Get:
  DestBits Bndget: MonSource Abs: DestRect Int: Mode 0
  call CopyBits .ok ;M

;CLASS
```

----Listing 10----

```
( MonIdent -- this is a type of monument located at a
  ZeroCell-type object -- dk 12 May 86 )

: RefMsg  ." Reference point is " ;

:CLASS  MonIdent     <Super Mon

        Var          RefPt

        ( theRefPt -- )
:M  DEFINE: Put: RefPt .ok ;M


:M  LOCATE: { \ theObj -- x y } Get: RefPt -> theObj
  Locate: theObj
;M


:M  DRAW: { \ xloc yloc -- }
       Locate: Self -> yloc -> xloc
       xloc yloc -2 -2 +pair putTop: DestRect
       xloc yloc  2  2 +pair PutBot: DestRect
       Display: Super
;M

        ( -- )                      \ prints name of refPt
:M  REF: RefMsg Get: RefPt 3 - >name id. cr .ok ;M

;CLASS
```
