Inheritance, Generalisation & Specialisation

Inheritance, Generalisation/Specialisation

- Modelling hierarchies of classes
- Important modelling technique in computational finance
- Essential when designing and implementing flexible C++ code
- Ability to create a general base class and specialise it to specific cases

2

Specialisation Process

- We create more specialised classes from more general classes
- The number of levels is infinite (in principle)
- Can create new and modified functionality at ever-increasing levels of detail
- This process is not without its dangers (more on this later)

3

A little Cognitive Psychology

- Cognitive economy: divide the world into classes of things to decrease the amount of information we must learn, remember, perceive
- We tend to categorise and classify the information around us
- Seek a balance between cognitive economy and informativeness

4

Categorization

- Heavily influenced by philosophy and logic (Locke, Frege)
- Object concepts are atomic units that we combine into more complex structures (molecules)
- Four defining views(and they can be implemented in C++)
- Defining attribute; prototype; exemplar; explanation-based view

5

Defining Attribute (Frege)

- A concept has a set of defining attributes or semantic features
- Distinguishes between a concept's intension and extension
- Intension: defines what it means to be a member of the concept (class in C++)
- Extension: set of entities that are members of the concept (objects)

6

Consequences (1/2)

1. The meaning of a concept is captured by the conjunction (AND) of its attributes
2. Attributes are atomic units (primitives)and are the building blocks of concepts
3. Each attribute is necessary and all of them are jointly sufficient to define an instance of the concept

Consequences (2/2)

1. What is and what is not a member of the category is clearly defined (clear-cut boundaries between members and non-members)
2. All members of the concept are equally representative
3. Concept hierarchies: defining attributes of more specific concept are include DAs of the superordinate

Disadvantage of DA View

- Category members are not equally representative in real (C++!) life
- Concept boundaries tend to be changing
- The theory does not predict the three-level structure and basic-level categories
- Some attributes tend to be more important than others
- Concept instability
- What/how to find the defining attributes?

9

Prototype View

- Concepts have a prototype structure (the prototype is the best example of the concept)
- Collection of characteristic attributes
- No delimiting set of necessary or sufficient attributes needed to define concept membership (necessary but not sufficient)
- Concept boundaries are fuzzy; some instances may slip into other categories
- Instances ranged in terms of their typicality

10

Disadvantages

- Incomplete as an account of sort of knowledge people have about concepts
- People tend to know the relationships between attributes rather than just attributes alone
- Does not give a good account of what makes some categories natural and coherent

11

Exemplar-based View

- Instead of working top-down (from concept to instance) we take specific instances to ‘discover’ concepts
- We don’t look at all instances
- (Bird and has-wings versus crow, sparrow)
- The attributes are determined by which instance comes into mind
- Better than the prototype view (it preserves the variability of instances of a category)

12

Disadvantages of Exemplar

- Difficulty with inclusion questions (how to handle abstract knowledge)
- Example: “All birds are creatures”
- Exemplar view (like prototype) depends heavily on similarity
- Example: GUI controls in toolboxes

13

Explanation-based Views

- The other views are attribute-based
- There exist concepts that have very little similarities between attributes
- This view involves more than just attributes
- Concepts contain causal and other background knowledge
- (e.g. dangerous and non-dangerous animals)

14

Some Remarks

- Explanation view is more ‘dynamic’; concepts determined by actions
- Concepts can have attributes but relationships between the attributes)
- (example: light bones enable flight)
- Concept coherence comes from underlying knowledge of concepts, not from similarity alone
- (example; price and hedge derivatives)

15

Conceptual Hierarchies

- Can group concept into hierarchies (e.g. ‘is a chicken an animal?’)
- How do we determine the structure of conceptual hierarchies
- How many levels in the human cognitive system?
- People use about 3 levels (from biological studies)

16

The 3 Levels

- Superordinate (highest): general designations for very general concepts (e.g. furniture)
- Subordinate (lowest): specific types of objects (e.g. my favourite armchair)
- Basic (in-between): correspond to specific concepts (e.g. chair)
- Important later in C++ categories

17

Remarks

- Superordinate level tends to have few attributes
- Economy missing at the subordinate level (too many attributes conveyed)
- Most adults tend to start at the basic level (it's the one acquired first by young children)
- Category members at basic level tend to have the same overall shape; can form a mental picture

18

Concept Instability

- Concept representation changes as a function of the context in which it appears
- Disastrous effects especially in concept hierarchies
- Problem is not so much attributes but behaviour
- Context-sensitive information

19

Relationships with C++

- Developers employ these views when creating classes (implicitly, possibly)
- We need to know which choice we are making and what the consequences are
- Many legacy C++ systems are suffering on account of this ‘blind spot’
- We discuss how to resolve these problems by using an appropriate design

20

Examples in C++

- Payoff and Instrument hierarchies
- 2d Shape hierarchies
- FDM (finite difference) and PDE (partial difference equations) hierarchies
- Data interpolation hierarchies
- Bond instruments
- etc.

21

Aggregation (HASA relationship)

- We can ‘embed’ class C1 in class C2
- Then messages to C2 can be *forwarded/delegated* to C1
- We can emulate inheritance with aggregation
- A form of reusability (object-level)

22

Examples

- A Polyline consists of Points
- A Bond has cash flow and schedules
- A finite difference solver has an embedded PDE object
- An STL stack class has an embedded data store (which can be a vector, list or deque)

23

Summary

- Touched on some issues in conceptual/object-oriented modelling
- Part of broader OO analysis/design
- Dangers of classical attribute-based theory
- Similarities with Design Patterns (Gamma et al)
- We shall see examples in later sections (applications)

24