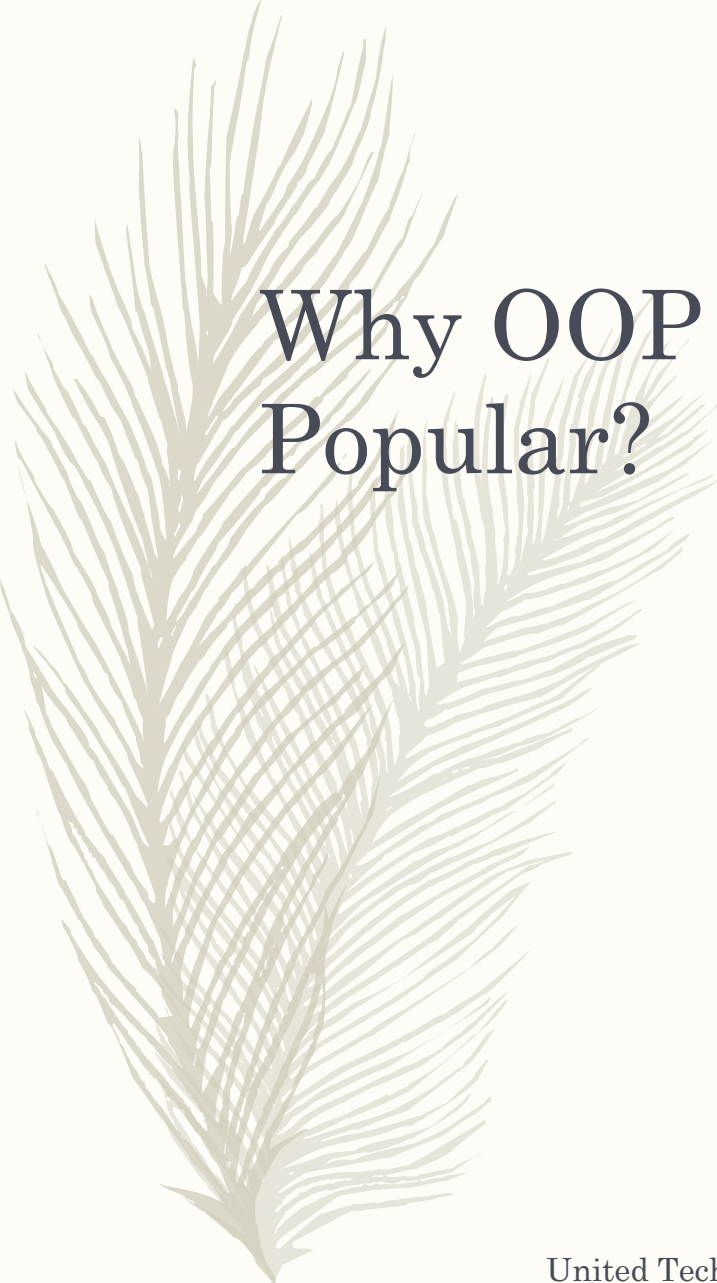




Thinking of Object-Oriented

Contents

- ❑ Why OOP is Popular?
- ❑ Object Oriented Programming: a new paradigm
- ❑ A way of viewing world agent
- ❑ Types of classes
- ❑ Computation as simulation
- ❑ Coping with complexities
- ❑ Non-linear behavior of complexities
- ❑ Abstraction mechanism



Why OOP is Popular?

- ❖ OOP is a *revolutionary idea*, totally unlike anything that has come before in programming.
- ❖ OOP is an *evolutionary step*, following naturally on the heels of earlier programming abstractions.
- ❖ Object-oriented programming scales very well, from the most trivial of problems to the most complex tasks.
- ❖ It provides a form of abstraction that resonates with techniques people use to solve problems in their everyday life.
- ❖ Object-oriented languages there are an increasingly large number of libraries that assist in the development of applications for many domains.
- ❖ The managers and programmers alike hope that a C or Pascal programmer can be changed into a C++ or Delphi programmer with no more effort than the addition of a few characters to their job title. But object-oriented programming is a new way of thinking about what it means to compute, about how we can structure information and communicate our intentions both to each other and to the machine.

Object Oriented Programming: *a new paradigm*

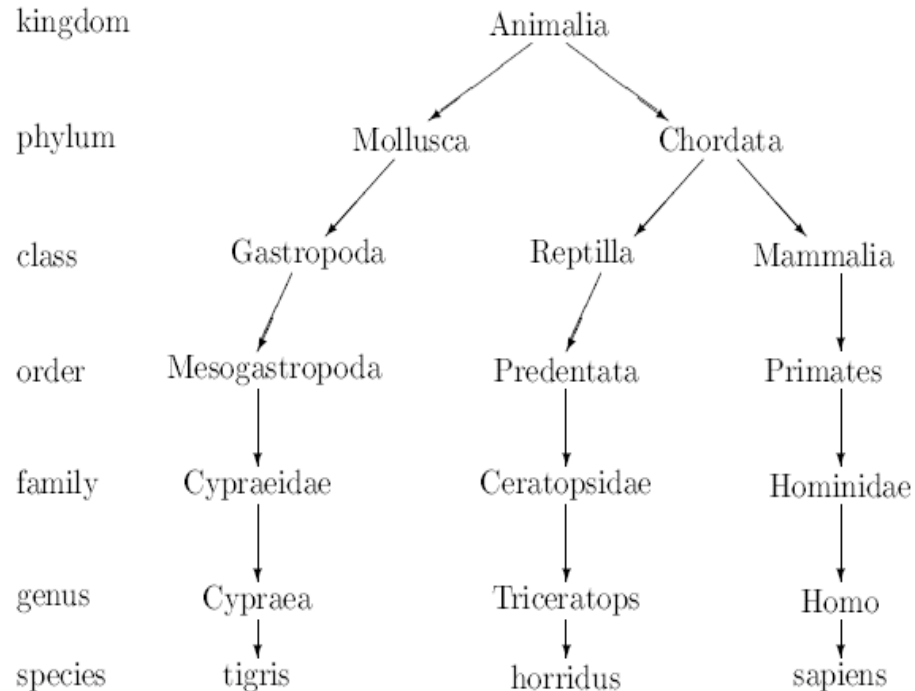


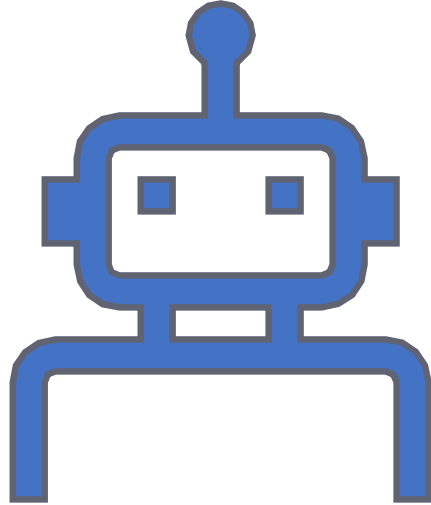
Figure The Linnaean Inheritance Hierarchy

- ❖ A programming paradigm is a way of conceptualizing what it means to perform computation and how tasks to be carried out on a computer should be structured and organized.
- ❖ Although new to computation, the organizing technique that lies at the heart of object-oriented programming can be traced back at least as far as Carolus Linnaeus (1707-1778). It was Linnaeus, you will recall, who categorized biological organisms using the idea of phylum, genus, species, and so on.



A New Paradigm cont.

- ❖ Paradoxically, the style of problem solving embodied in the object-oriented technique is frequently the method used to address problems in everyday life.
- ❖ Thus, computer novices are often able to grasp the basic ideas of object-oriented programming easily, whereas people who are more computer literate are often blocked by their own preconceptions. Alan Kay, for example, found that it was often easier to teach Smalltalk to children than to computer professionals [Kay 1977].



A Way of Viewing the World

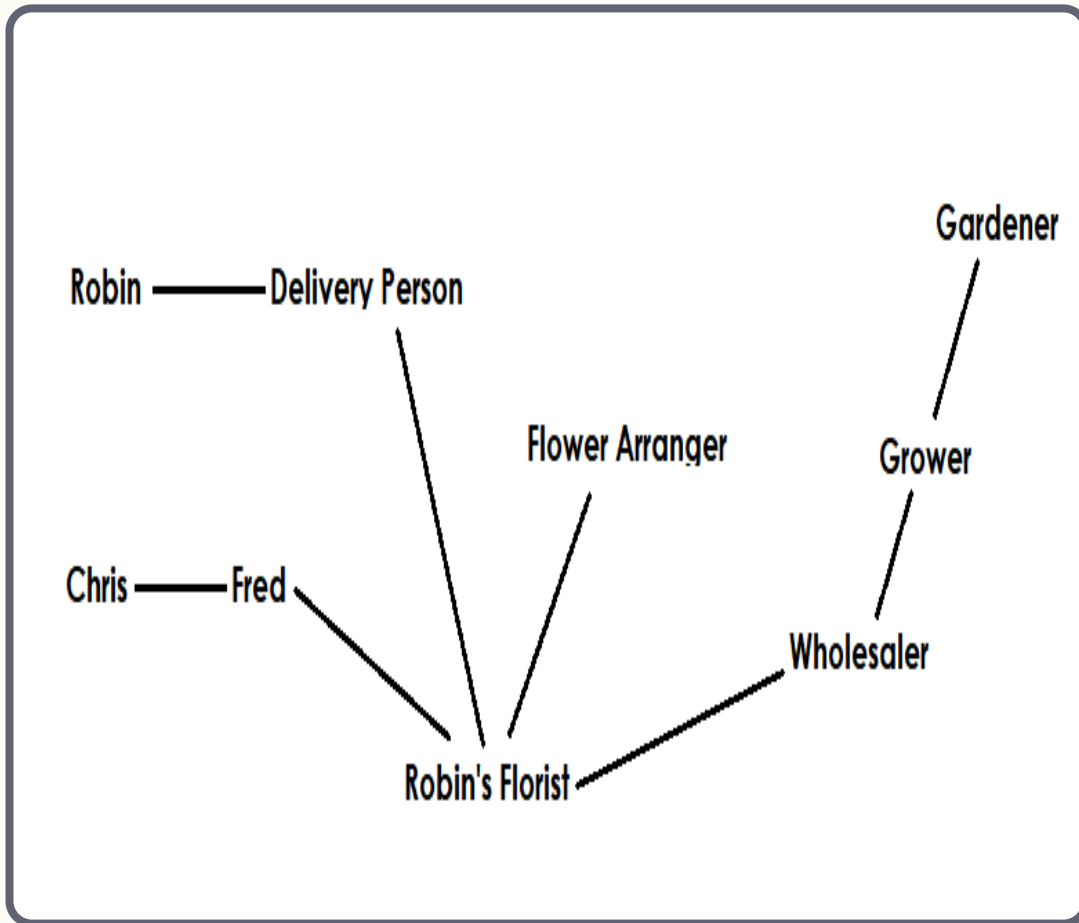
- ❖ Let us consider first how we might go about handling a real-world situation and then ask how we could make the computer more closely model the techniques employed.
- ❖ **Chris** wishes to send flowers to a friend named **Robin**, who lives in another city. Because of the distance, Chris cannot simply pick the flowers and take them to Robin in person.
- ❖ Nevertheless, it is a task that is easily solved. Chris simply walks to a nearby flower shop, run by a florist named **Fred**. Chris will tell Fred the *kinds of flowers* to send to Robin, and *the address* to which they should be delivered. Chris can then be *assured* that the flowers will be delivered expediently and automatically.



Agents and Communities

- ❖ Let us emphasize that the mechanism that was used to solve the problem was to find an appropriate agent (namely, Fred) and to pass to this agent a message containing a request. It is the responsibility of Fred to satisfy the request.
- ❖ There is some method--some algorithm or set of operations--used by Fred to do this. Chris does not need to know the particular method that Fred will use to satisfy the request; indeed, often the person making a request does not want to know the details. This information is usually hidden from inspection.
- ❖ An investigation, however, might uncover the fact that Fred delivers a slightly different message to another florist in the city where Robin lives.
- ❖ That florist, in turn, perhaps has a subordinate who makes the flower arrangement. The florist then passes the flowers, along with yet another message, to a delivery person, and so on.
- ❖ Earlier, the florist in Robin's city had obtained her flowers from a flower wholesaler who, in turn, had interactions with the flower growers, each of whom had to manage a team of gardeners.

Agents and Communities cont.



- ❖ The solution to this problem required the help of many other individuals without their help, the problem could not be easily solved.
- ❖ We phrase this in a general fashion as : *An object-oriented program is structured as a community of interacting agents, called objects. Each object has a role to play. Each object provides a service, or performs an action, that is used by other members of the community.*



Messages and Methods

- ❖ The chain reaction that ultimately resulted in the solution to Chris's problem began with a request given to the florist Fred. This request lead to other requests, which lead to still more requests, until the flowers ultimately reached Chris's friend Robin. We see, therefore, that members of this community interact with each other by making requests.
- ❖ *Action is initiated in object-oriented programming by the transmission of a message to an agent (an object) responsible for the action. The message encodes the request for an action and is accompanied by any additional information (arguments) needed to carry out the request.*
- ❖ *The receiver is the object to whom the message is sent. If the receiver accepts the message, it accepts the responsibility to carry out the indicated action. In response to a message, the receiver will perform some method to satisfy the request.*
- ❖ *The client sending the request need not know the actual means by which the request will be honored.*
- ❖ *An important part of object-oriented programming is the development of reusable components, and an important first step in the use of reusable components is a willingness to trust software written by others.*



Messages vs. Procedure Calls

- ❖ In both cases, there is a set of well-defined steps that will be initiated following the request. But there are two important distinctions.
- ❖ The first is that in a message there is a designated receiver for that message; the receiver is some object to which the message is sent. In a procedure call, there is no designated receiver.
- ❖ The second is that the interpretation of the message (that is, the method used to respond to the message) is determined by the receiver and can vary with different receivers. Chris could give a message to a friend named Elizabeth, for example, and she will understand it and a satisfactory outcome will be produced. However, the method Elizabeth uses to satisfy the request will be different from that used by Fred in response to the same request.
- ❖ If Chris were to ask Kenneth, a dentist, to send flowers to Robin, Kenneth may not have a method for solving that problem. If he understands the request at all, he will probably issue an appropriate error diagnostic.



Messages vs. Procedure Calls cont.

*In message passing, there is a **designated receiver**, and the interpretation—the selection of a method to execute in response to the message—may vary with different receivers. Usually, the specific receiver for any given message will not be known until run time, so the determination of which method to invoke cannot be made until then. Thus, we say there is **late binding** between the message (function or procedure name) and the code fragment (method) used to respond to the message. This situation contrasts with the very early (**compile-time or link-time**) binding of name to code fragment in conventional procedure calls.*



Responsibilities

- ❖ A fundamental concept in object-oriented programming is *to describe behavior in terms of responsibilities*.
- ❖ Chris's request for action indicates only the desired outcome (flowers sent to Robin). Fred is free to pursue any technique that achieves the desired objective, and in doing so will not be hampered by interference from Chris.
- ❖ By discussing a problem in terms of *responsibilities* increases the *level of abstraction*. This permits greater *independence* between objects, a critical factor in solving complex problems. The entire collection of responsibilities associated with an object is often described by the term ***protocol***.
- ❖ A traditional program often operates by acting on data structures, for example changing fields in an array or record. In contrast, an object-oriented program requests data structures (that is, objects) to perform a service.
- ❖ ***Ask not what you can do to your data structures, but rather ask what your data structures can do for you.***



Classes and Instances

- ❖ We can use the term “**Florist**” to represent the category (or class) of all florists.
- ❖ Chris can make certain assumptions based on the previous experience with other florists, and hence Chris can expect that Fred, being an instance of this category, will fit the *general patterns*.
- ❖ *All objects are instances of a class.*
- ❖ *The method invoked by an object in response to a message is determined by the class of the receiver.*
- ❖ *All objects of a given class use the same method in response to similar messages.*

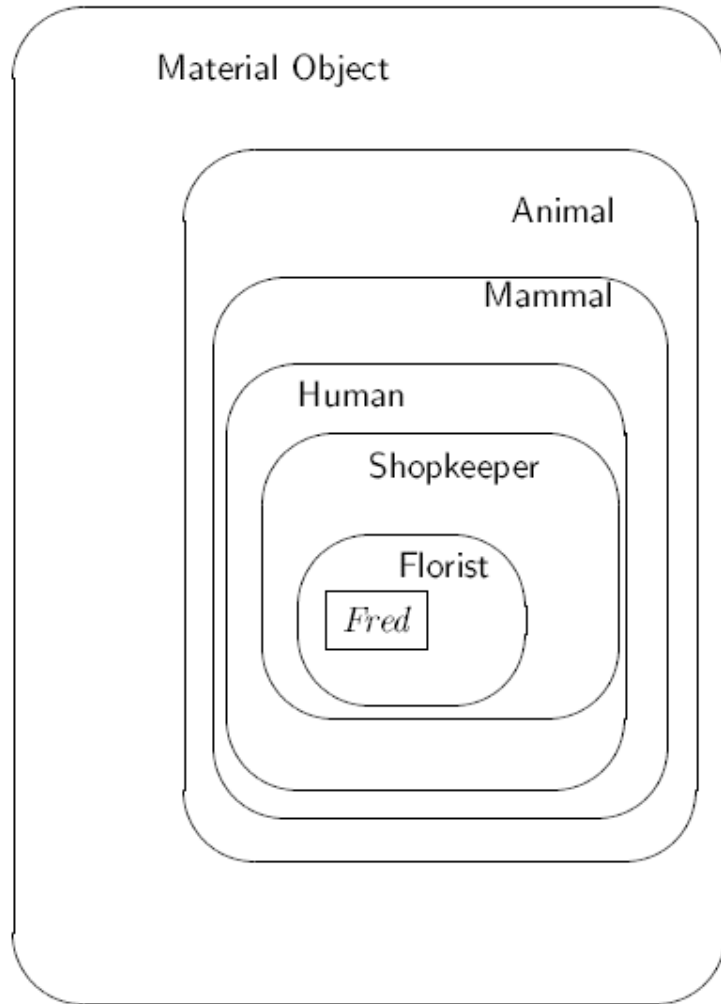


Fig: The categories surrounding Fred.

Class Hierarchies-- Inheritance

- ❖ Chris has more information about Fred not necessarily because Fred is a florist but because he is a shopkeeper. Since the category Florist is a more specialized form of the category Shopkeeper, any knowledge Chris has of Shopkeepers is also true of Florists and hence of Fred.
- ❖ One way to think about how Chris has organized knowledge of Fred is in terms of a hierarchy of categories (see Figure).
- ❖ Fred is a Florist, but Florist is a specialized form of Shopkeeper. Furthermore, a Shopkeeper is also a Human; so Chris knows, for example, that Fred is probably bipedal. A Human is a Mammal (therefore they nurse their young and have hair), and a Mammal is an Animal (therefore it breathes oxygen), and an Animal is a Material Object (therefore it has mass and weight).
- ❖ Thus, quite a lot of knowledge that Chris has that is applicable to Fred is not directly associated with him, or even with the category Florist.

Class Hierarchies– Inheritance cont.

- ❖ The principle that knowledge of a more general category is also applicable to a more specific category is called *inheritance*. We say that the class Florist will inherit attributes of the class (or category) Shopkeeper.
- ❖ More **abstract classes** (such as Material Object or Animal) listed near the top of the tree, and more **specific classes**, and finally individuals, are listed near the bottom. Figure shows this class hierarchy for Fred. This same hierarchy also includes Elizabeth, Chris's dog Fido, Phyl the platypus who lives at the zoo, and the flowers the Chris is sending to Robin.
- ❖ Classes can be organized into a hierarchical inheritance structure.
- ❖ A child class (or subclass) will inherit attributes from a parent class higher in the tree.
- ❖ An abstract parent class is a class (such as Mammal) for which there are no direct instances; it is used only to create subclasses.

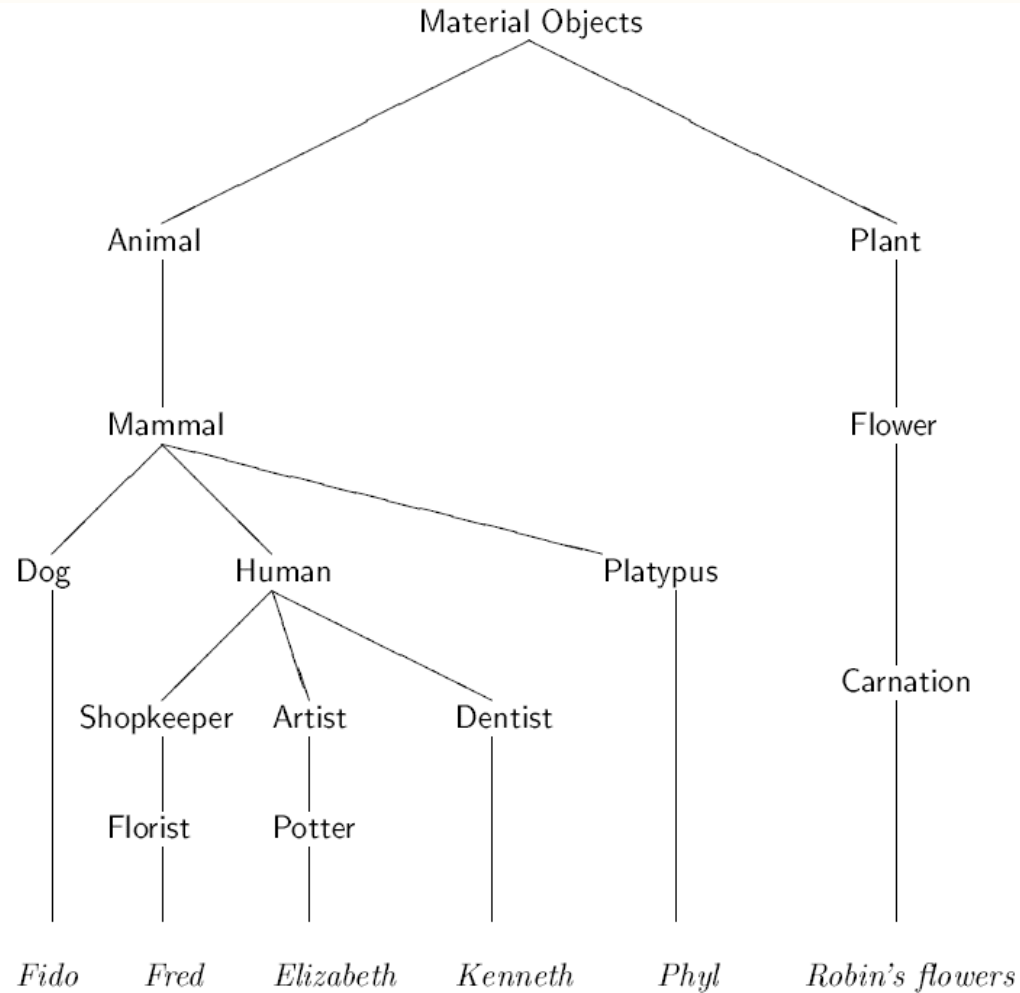
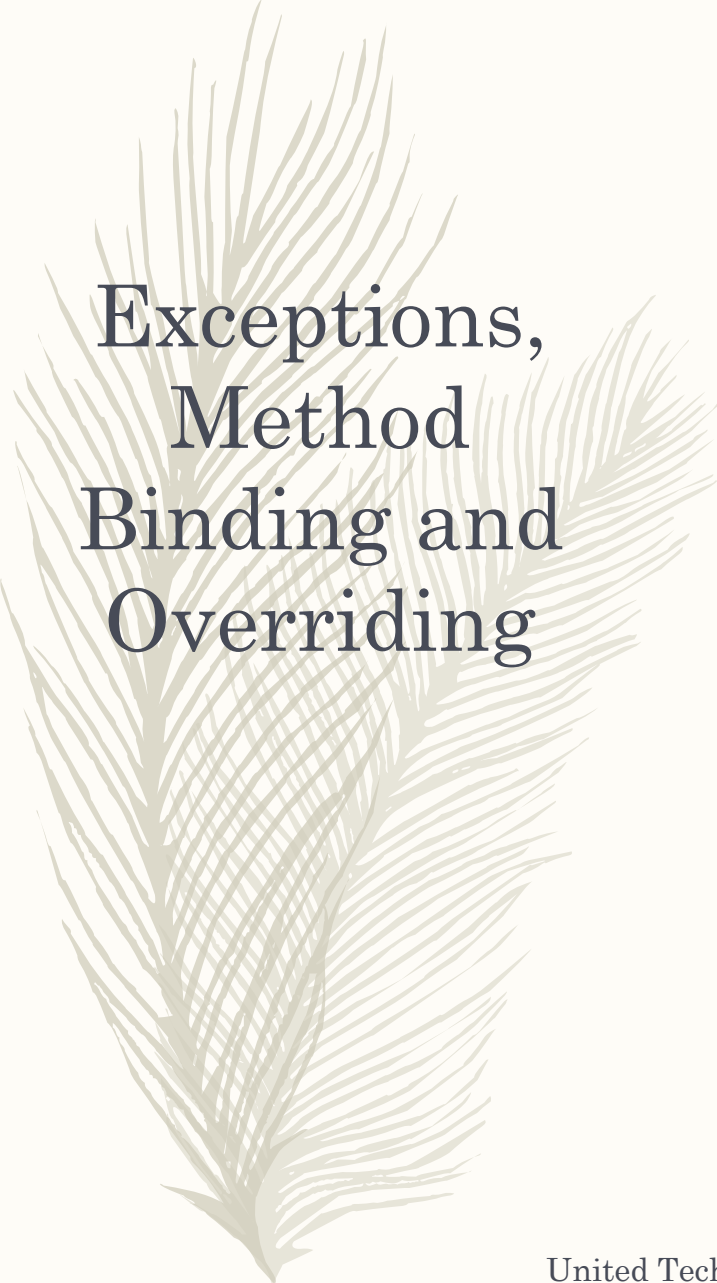



Figure A class hierarchy for various material objects.



Exceptions, Method Binding and Overriding

- ❖ Phyl the platypus presents a problem for our simple organizing structure. Chris knows that mammals give birth to live children, and Phyl is certainly a Mammal, yet Phyl (or rather his mate Phyllis) lays eggs. To accommodate this, we need to find a technique to encode *exceptions* to a general rule.
- ❖ The information contained in a subclass can *override information* inherited from a parent class.
- ❖ Implementations of this approach takes the form of a method in a subclass having the same name as a method in the parent class, combined with a rule stating how to conduct the search for a method to match a specific message.
- ❖ The search for a method to invoke in response to a given message begins with the class of the receiver. If no appropriate method is found, the search is conducted in the parent class of this class.
- ❖ The search continues up the parent class chain until either a method is found, or the parent class chain is exhausted. In the former case the method is executed; in the latter case, an error message is issued.
- ❖ If methods with the same name can be found higher in the class hierarchy, the method executed is said to ***override*** the inherited behavior.
- ❖ The fact that both Elizabeth and Fred will react to Chris's messages, but use different methods to respond, is one form of ***polymorphism***.
- ❖ Chris does not, and need not, know exactly what method Fred will use to honor the request is an example of ***information hiding***.



Summary of Object- Oriented Concepts

Alan Kay, considered by some to be the father of object-oriented programming, identified the following characteristics as fundamental to OOP [Kay 1993]:

1. *Everything is an object.*
2. *Computation is performed by objects communicating with each other, requesting that other objects perform actions. Objects communicate by sending and receiving messages. A message is a request for action bundled with whatever arguments may be necessary to complete the task.*
3. *Each object has its own memory, which consists of other objects.*
4. *Every object is an instance of a class. A class simply represents a grouping of similar objects, such as integers or lists.*
5. *The class is the repository for behavior associated with an object. That is, all objects that are instances of the same class can perform the same actions.*
6. *Classes are organized into a singly rooted tree structure, called the inheritance hierarchy. Memory and behavior associated with instances of a class are automatically available to any class associated with a descendant in this tree structure.*

Computation as Simulation

- ❖ The traditional model describing the behavior of a computer executing a program is a *process-state* or *pigeon-hole model*.
- ❖ In this view, the computer is a data manager, following some pattern of instructions, wandering through memory, pulling values out of various slots (memory addresses), transforming them in some manner, and pushing the results back into other slots (see Figure).
- ❖ By examining the values in the slots, one can determine the state of the machine or the results produced by a computation.

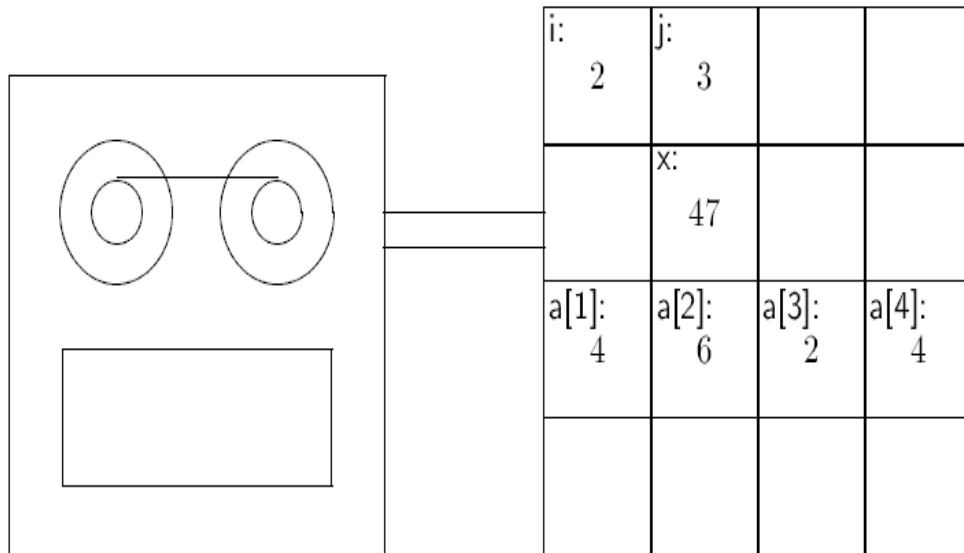
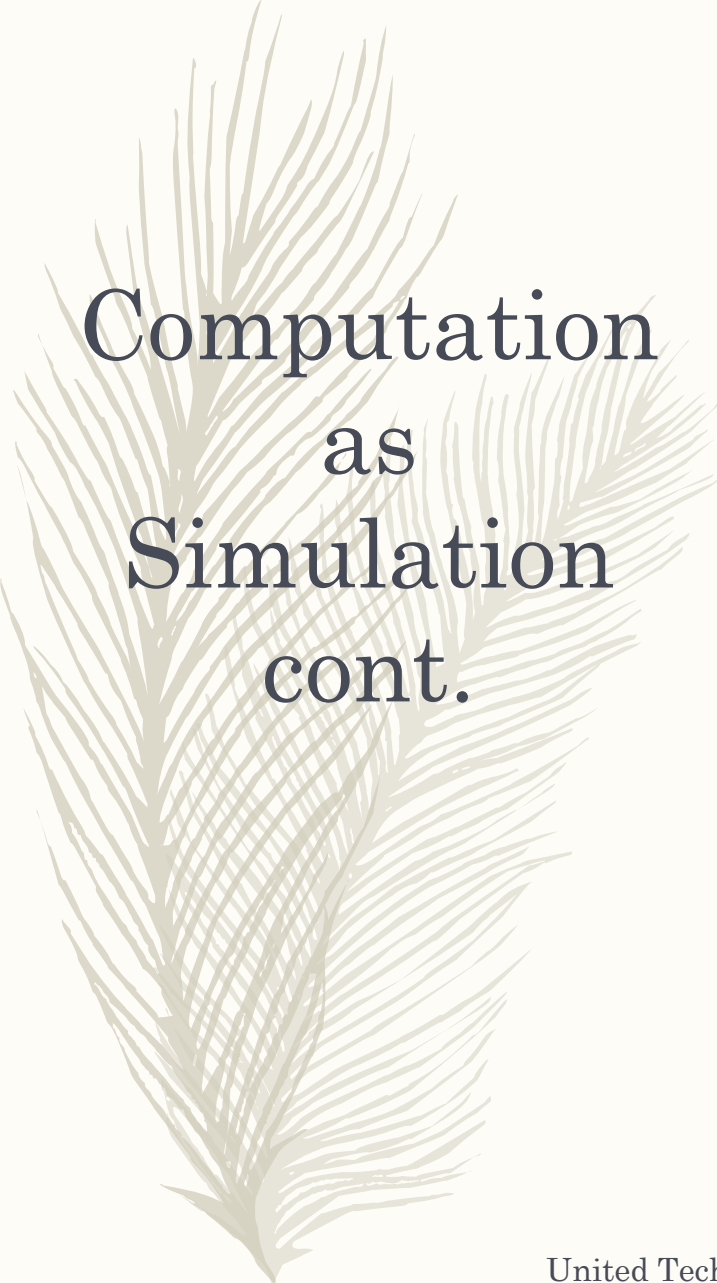


Figure Visualization of imperative programming.



Computation as Simulation cont.

- ❖ In contrast, in the object-oriented framework we never mention *memory addresses, variables, assignments*, or any of the conventional programming terms. Instead, we speak of *objects, messages*, and *responsibility* for some action.
- ❖ *Instead of a bit-grinding processor...plundering data structures, we have a universe of well-behaved objects that courteously ask each other to carry out their various desires [Ingalls 1981].*
- ❖ *The object-oriented programming as “**animistic**”: a process of creating a host of helpers that form a community and assist the programmer in the solution of a problem [Actor 1987].*
- ❖ This view of programming as creating a “universe” is in many ways like a style of computer simulation called “*discrete event-driven simulation*” where the user creates computer models of the various elements of the simulation, describes how they will interact with one another, and sets them moving.
- ❖ *In the object-oriented program, the user describes what the various entities in the universe for the program are, and how they will interact with one another, and finally sets them in motion.*
- ❖ *Thus, in object-oriented programming, we have the view that **computation is simulation** [Kay 1977].*



Coping with Complexity

- ❖ When computing was in its infancy, most programs were written in assembly language, by a single individual, and would not be considered large by today's standards.
- ❖ Even so, as programs became more complex, programmers found that they had a difficult time remembering all the information they needed to know in order to develop or debug their software.
- ❖ Which values were contained in what registers? Did a new identifier name conflict with any other previously defined name? What variables needed to be initialized before control could be transferred to another section of code?
- ❖ The introduction of higher-level languages, such as Fortran, Cobol and Algol, solved some difficulties (such as automatic management of local variables, and implicit matching of arguments to parameters), while simultaneously raising people's expectations of what a computer could do in a manner that only introduced yet new problems.
- ❖ As programmers attempted to solve ever more complex problems using a computer, tasks exceeding in size the grasp of even the best programmers became the norm. Thus, teams of programmers working together to undertake major programming efforts became commonplace.

Coping with Complexity cont.: The Nonlinear Behavior of Complexity

- ❖ As programming projects became larger, an interesting phenomenon was observed.
- ❖ A task that would take one programmer two months to perform could not be accomplished by two programmers working for one month.
- ❖ In Fred Brooks's memorable phrase, *“the bearing of a child takes nine months, no matter how many women are assigned to the task”*.
- ❖ The reason for this nonlinear behavior was complexity - in particular, the interconnections between software components were complicated, and large amounts of information had to be communicated among various members of the programming team.
- ❖ Brooks further said: *“Since software construction is inherently a systems effort - an exercise in complex interrelationships - communication effort is great, and it quickly dominates the decrease in individual task time brought about by partitioning. Adding more men then lengthens, not shortens, the schedule”*.
- ❖ What brings about this complexity? It is not simply the sheer size of the tasks undertaken, because size by itself would not be a hindrance to partitioning each into several pieces.
- ❖ The unique feature of software systems developed using conventional techniques that makes them among the most complex systems developed by people is their *high degree of interconnectedness*. Interconnectedness means the dependence of one portion of code on another section of code.
- ❖ Consider that any portion of a software system must be performing an essential task, or it would not be there. Now, if this task is useful to the other parts of the program, there must be some communication of information either into or out of the component under consideration. Because of this, a complete understanding of what is going on requires a knowledge both portion of code we are considering and the code that uses it. In short, an individual section of code cannot be understood in isolation.



Summary

- ❖ *Object-oriented programming is not simply a few new features added to programming languages. Rather, it is a new way of thinking about the process of decomposing problems and developing programming solutions.*
- ❖ *Object-oriented programming views a program as a collection of loosely connected agents, termed objects. Each object is responsible for specific tasks. It is by the interaction of objects that computation proceeds. In a certain sense, therefore, programming is nothing more or less than the simulation of a model universe.*
- ❖ *An object is an encapsulation of state (data values) and behavior (operations). Thus, an object is in many ways like special purpose computer.*
- ❖ *The behavior of objects is dictated by the object class. Every object is an instance of some class. All instances of the same class will behave in a similar fashion (that is, invoke the same method) in response to a similar request.*
- ❖ *An object will exhibit its behavior by invoking a method (like executing a procedure) in response to a message. The interpretation of the message (that is, the specific method used) is decided by the object and may differ from one class of objects to another.*



Summary cont.

- ❖ *Classes can be linked to each other by means of the notion of inheritance. Using inheritance, classes are organized into a hierarchical inheritance tree. Data and behavior associated with classes higher in the tree can also be accessed and used by classes lower in the tree. Such classes are said to inherit their behavior from the parent classes.*
- ❖ *Designing an object-oriented program is like organizing a community of individuals. Each member of the community is given certain responsibilities. The achievement of the goals for the community come about through the work of each member, and the interactions of members with each other.*
- ❖ *By reducing the interdependency among software components, object-oriented programming permits the development of reusable software systems. Such components can be created and tested as independent units, in isolation from other portions of a software application.*
- ❖ *Reusable software components permit the programmer to deal with problems on a higher level of abstraction. We can define and manipulate objects simply in terms of the messages they understand and a description of the tasks they perform, ignoring implementation details.*



Abstraction

- ❖ An analogy from the world atlas. Zooming in reveals finer details. Zooming out reveals a different worldview at every step.
- ❖ Consider another analogy of automobiles. A layman driver's perspective is different than that of the mechanic. A mechanic's perspective is different than that of an expert design engineer.
- ❖ *Abstraction is the purposeful suppression, or hiding, of some details of a process or artifact, in order to bring out more clearly other aspects, details, or structure.*
- ❖ *Information hiding is the purposeful omission of details in the development of an abstract representation.*



Layers of Abstraction

- ❖ In a typical program written in the object-oriented style there are many important levels of abstraction.
- ❖ The higher-level abstractions are part of what makes an object-oriented program object-oriented.
- ❖ At the highest level, a program is viewed as a “community” of objects that must interact with each other in order to achieve their common goal.
- ❖ The notion of community has two different forms in OOP: a *community of programmers* and a *community of objects* they create.
- ❖ The ideas of abstraction and information hiding are applicable to both levels.
- ❖ Each object in this community provides a service that is used by other members of the organization.
- ❖ At this highest level of abstraction the important features to emphasize are the *lines of communication and cooperation*, and the way in which the members must *interact with each other*.
- ❖ A higher level of abstraction (than this) is not available in all OOP programs or languages.
- ❖ However, in languages such as java similar types of objects are grouped together to form singular units in *namespaces and packages* etc. The unit allows certain names to be exposed to the world outside the unit, while other features remain hidden inside the unit.



Layers of Abstraction cont.

- ❖ The next two levels of abstraction deal with the interactions between two individual objects.
- ❖ Often, we speak of objects as providing a service to other objects.
- ❖ We build on this intuition by describing *communication as an interaction* between a *client* and a *server*.
- ❖ A server means something an object that provides a service (and not a web server) in this context.
- ❖ The two layers of abstraction refer to the two views of this relationship; the *view from the client side* and the *view from the server side*.
- ❖ In a good object-oriented design, the services that the server provides without reference to any actions that the client may perform in using those services. One can think of this as being like a billboard advertisement as shown in figure.
- ❖ This level of abstraction is represented by an *interface*, a class-like mechanism that defines *behavior without describing an implementation*.

Layers of Abstraction cont.

Services Offered:

void push (Object val);	\$1
Object top ();	\$1
void pop ();	\$0.75

Joe's Data Structure Warehouse

"For All your Data Structure Needs!"

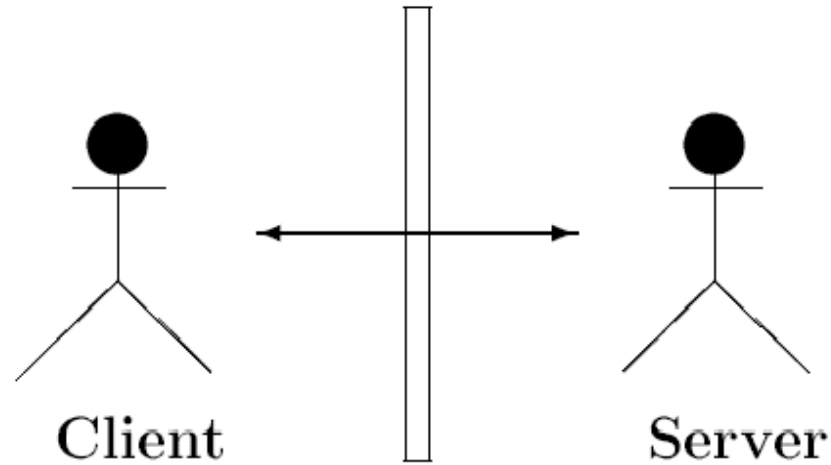


Fig: communication between client and servers

```
interface Stack {  
    public void push (Object val);  
    public Object top() throws EmptyStackException;  
    public void pop () throws EmptyStackException;  
}
```

Figure: Billboard Advertisement



Layers of Abstraction cont.

- ❖ The next level of abstraction looks at the same boundary but from the *server side*.
- ❖ This level considers a *concrete implementation* of the abstract behavior.
- ❖ For example, there are any number of data structures (array, linked list etc.) that can be used to satisfy the requirements of a Stack.
- ❖ Concerns at this level deal with the way in which the *services* are being *realized*.

```
public class LinkedList implements Stack ...{  
    public void pop() throws EmptyStackException{...}  
    ...  
}
```



Layers of Abstraction cont.

- ❖ Finally, the last level of abstraction considers *a single task in isolation*; that is, a single method.
- ❖ Concerns at this level of abstraction deal with the precise *sequence of operations* used to perform just this *one activity*.
- ❖ For example, we might investigate the technique used to perform *the removal of the most recent element placed into a stack*.
- ❖ In fact, programmers are often called upon to quickly move back and forth between different levels of abstraction.

```
public class LinkedList implements Stack...{  
    ...  
    public void pop() throws EmptyStackException{  
        if(isEmpty())  
            throw new EmptyStackException();  
        removeFirst();//delete first element of list  
    }  
    ...  
}
```



Finding a Right Level of Abstraction

- ❖ In early stages of software development a critical problem is finding the right level of abstraction.
- ❖ A common error is to dwell on the lowest levels, worrying about the implementation details of various key components, rather than striving to ensure that the high-level organizational structure promotes a clean separation of concerns.
- ❖ The programmer or, in larger projects, the design team must walk a fine line in trying to identify the right level of abstraction at any one point of time.
- ❖ *One does not want to ignore or throw away too much detail about a problem, but also one must not keep so much detail that important issues become obscured.*

Other Forms of Abstraction

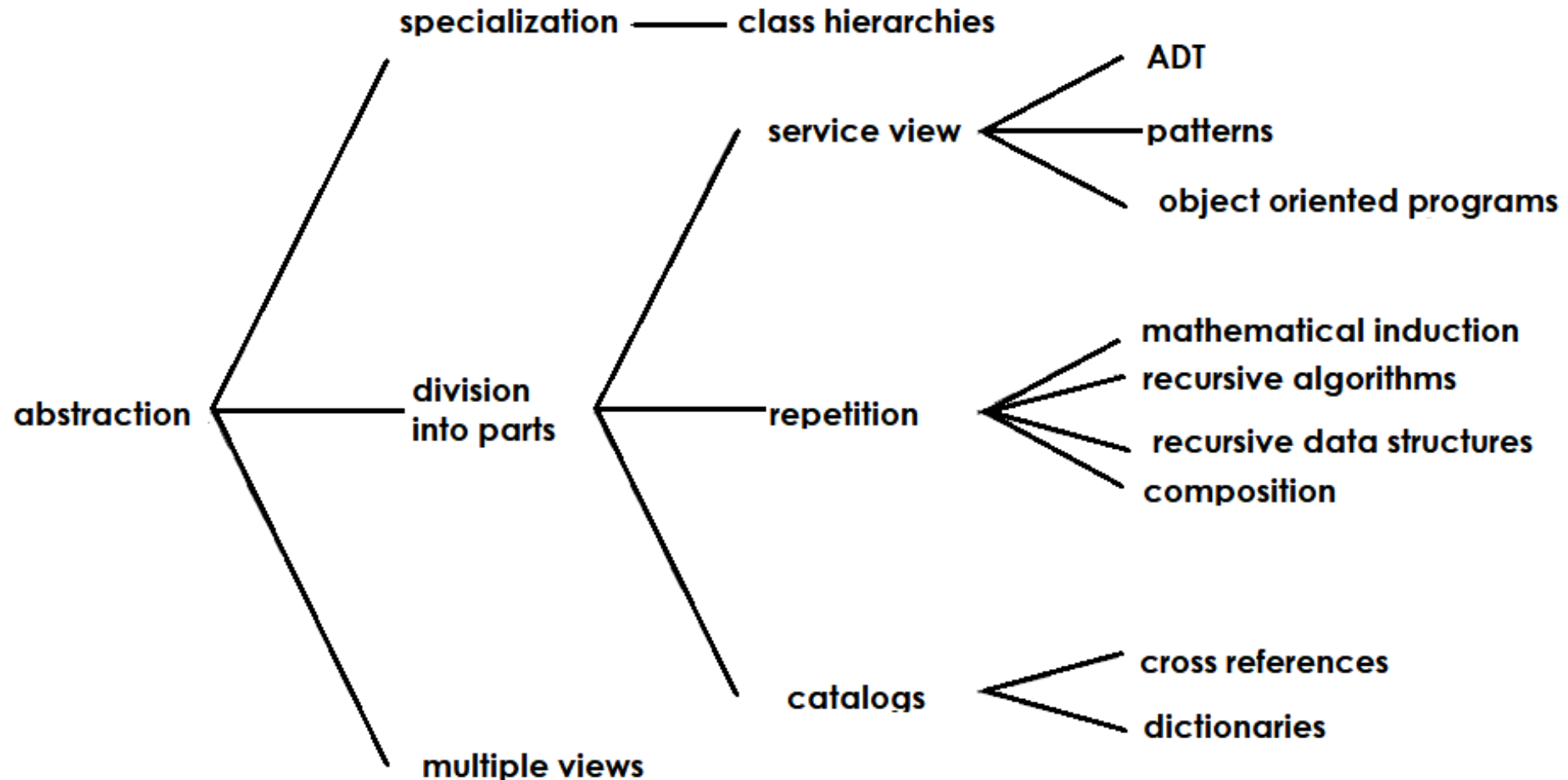



Figure: some techniques for handling complexities, with examples



Other Forms of Abstraction cont.

- ❖ Abstraction is used to help understand a complex system.
- ❖ A common technique is to divide a layer into constituent parts.
- ❖ This is the approach we used when we described an automobile as being composed of the engine, the transmission, the body and the wheels.
- ❖ The next level of understanding is then achieved by examining each of these parts in turn. This is nothing more than the application of the old maxim divide and conquer .
- ❖ Another form is the idea of layers of specialization.
- ❖ An understanding of an automobile is based, in part, on knowledge that it is a wheeled vehicle, which is in turn a means of transportation.
- ❖ There is other information we know about wheeled vehicles, and that knowledge is applicable to both an automobile and a bicycle.
- ❖ There is other knowledge we have about various means of transportation, and that information is also applicable to pack horses as well as bicycles.
- ❖ Yet another form of abstraction is to provide multiple views of the same artifact.
- ❖ Each of the views can emphasize certain detail and suppress others, and thus bring out different features of the same object.
- ❖ A layman's view of a car, for example, is very different from the view required by a mechanic.



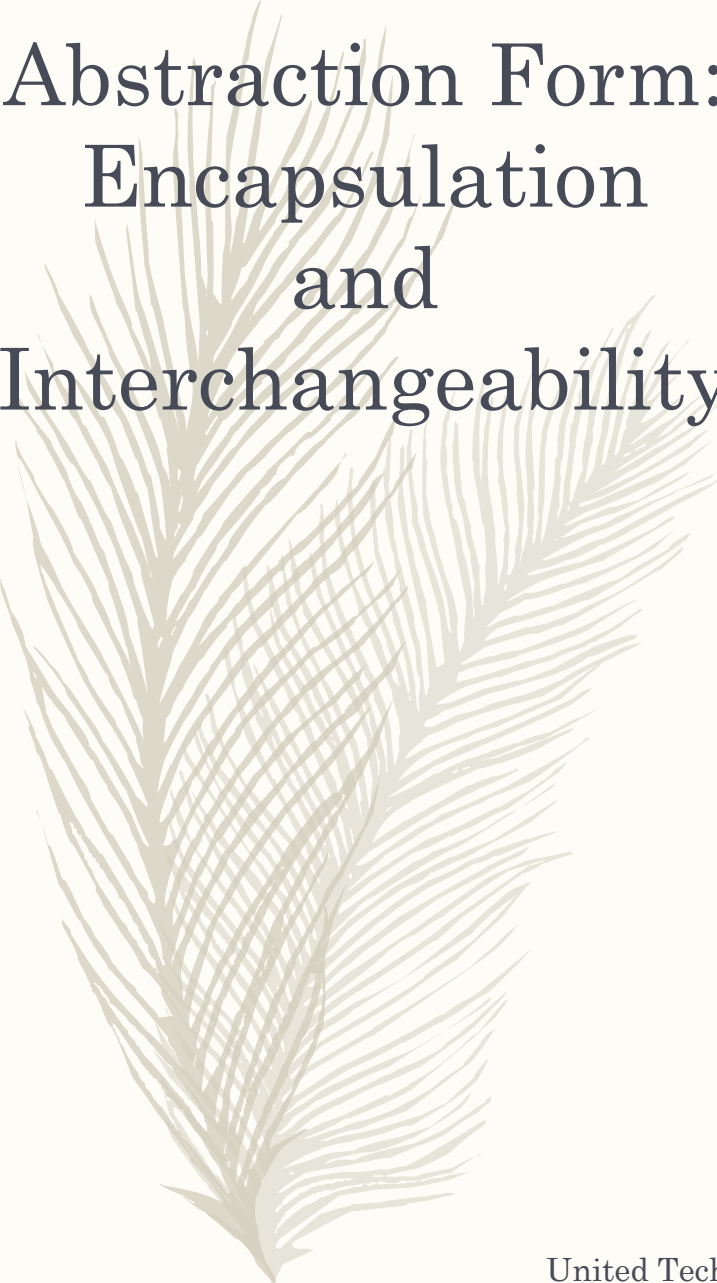
Is-a and Has-a Abstraction

- ❖ *The ideas of division into parts and division into specializations represent the two most important forms of abstraction used in object-oriented programming. These are commonly known as is-a and has-a abstraction.*
- ❖ *The idea of division into parts is has-a abstraction. The meaning of this term is easy to understand; a car “has-a” engine, and it “has-a” transmission, and so on.*
- ❖ *The concept of specialization is referred to as “is-a” abstraction. Again, the term comes from the English sentences that can be used to illustrate the relationships. A bicycle “is-a” wheeled vehicle, which in turn “is-a” means of transportation.*
- ❖ *Both is-a and has-a abstractions will reappear in later chapters and be tied to specific programming language features.*



Abstraction Form: Division into Parts

- ❖ The ideas of division into parts and division into specializations represent the two most important forms of abstraction used in object-oriented programming. These are commonly known as is-a and has-a abstraction.
- ❖ The idea of division into parts is has-a abstraction. The meaning of this term is easy to understand; a car has-a“ engine, and it has-a" transmission, and so on.
- ❖ The concept of specialization is referred to as is-a“ abstraction. Again, the term comes from the English sentences that can be used to illustrate the relationships.
- ❖ A bicycle “is-a” wheeled vehicle, which in turn “is-a” means of transportation.
- ❖ Another example might be organizing information about motion in a human body.
- ❖ At one level we are simply concerned with mechanics, and we consider the body as composed of bone for rigidity, muscles for movement, eyes and ears for sensing, the nervous system for transferring information and skin to bind it all together. At the next level of detail we might ask how the muscles work and consider issues such as cell structure and chemical actions. But chemical actions are governed by their molecular structure.
- ❖ And to understand molecules we break them into their individual atoms.
- ❖ Any explanation must be phrased at the right level of abstraction; trying to explain how a person can walk, for example, by understanding the atomic level details is almost certainly difficult, if not impossible.



Abstraction Form: Encapsulation and Interchangeability

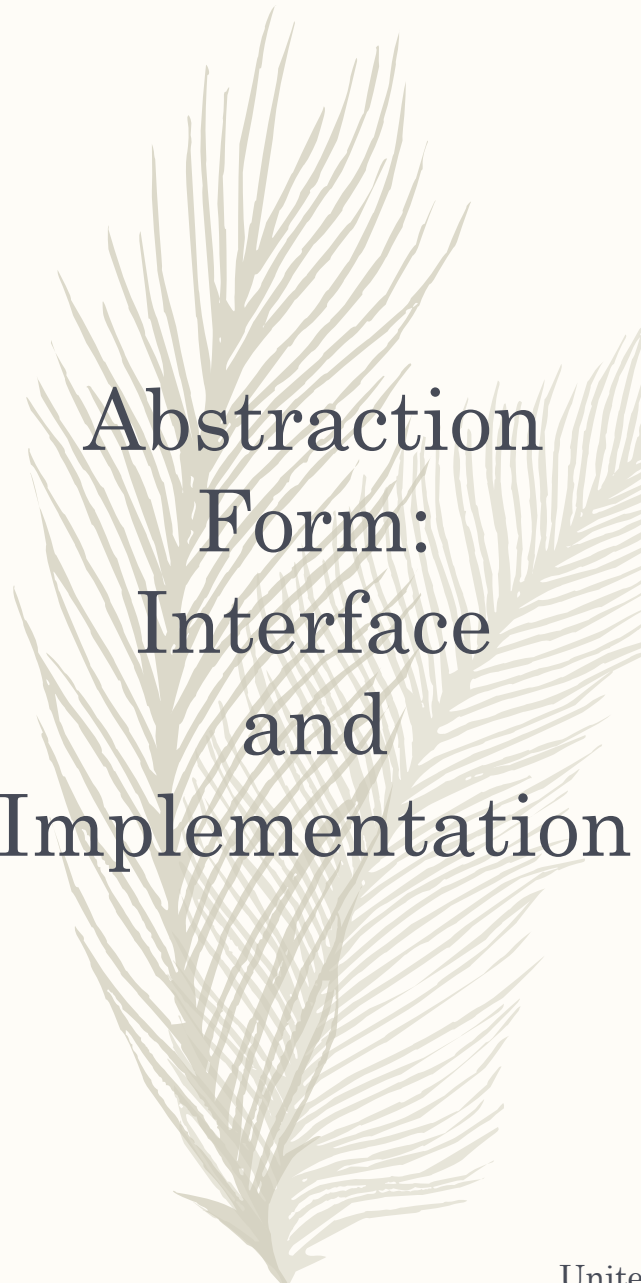
- ❖ A key step in the creation of large systems is the division into components.
- ❖ Suppose instead of writing software, we are part of a team working to create a new automobile.
- ❖ By separating the automobile into the parts engine and transmission , it is possible to assign people to work on the two aspects more or less independently of each other.
- ❖ We use the term encapsulation to mean that there is a strict division between the inner and the outer view.
- ❖ Those members of the team working on the engine need only an abstract outside, as it were view of the transmission, while those actually working on the transmission need the more detailed inside view.
- ❖ An important benefit of encapsulation is that it permits us to consider the possibility of interchangeability.
- ❖ When we divide a system into parts, a desirable goal is that the interaction between the parts is kept to a minimum.

Encapsulation and Interchangeability cont.

- ❖ For example, by encapsulating the behavior of the engine from that of a transmission we permit the ability to exchange one type of engine with another without incurring an undue impact on the other portions of the system.
- ❖ *For these ideas to be applicable to software systems, we need a way to discuss the task that a software component performs and separate this from the way in which the component fulfills this responsibility.*


Catalog

- When the number of components in a system becomes large it is often useful to organize the items by means of a catalog.
- We use many different forms of catalog in everyday life. Examples include a telephone directory, a dictionary, or an internet search engine. Similarly, there are a variety of different catalogs used in software. One example is a simple list of classes.
- Another catalog might be the list of methods defined by a class. A reference book that describes the classes found in the Java standard library is a very useful form of catalog. In each of these cases the idea is to provide the user a mechanism to quickly locate a single part (be it class, object, or method) from a larger collection of items.



Abstraction Form: Interface and Implementation

- ❖ In software we use the terms interface and implementation to describe the distinction between the what aspects of a task, and the how features; between the outside view, and the inside view.
- ❖ An interface describes what a system is designed to do.
- ❖ This is the view that users of the abstraction must understand.
- ❖ The interface says nothing about how the assigned task is being performed.
- ❖ So to work, an interface is matched with an implementation that completes the abstraction.
- ❖ The designers of an engine will deal with the interface to the transmission, while the designers of the transmission must complete an implementation of this interface.
- ❖ Similarly, a key step along the path to developing complex computer systems will be the division of a task into component parts.
- ❖ These parts can then be developed by different members of a team. Each component will have two faces, the interface that it shows to the outside world, and an implementation that it uses to fulfill the requirements of the interface.
- ❖ The division between interface and implementation not only makes it easier to understand a design at a high level (since the description of an interface is much simpler than the description of any specific implementation), but also make possible the interchangeability of software components (as I can use any implementation that satisfies the specifications given by the interface).



Abstraction Form: The Service View

- ❖ The idea that an interface describes the service provided by a software component without describing the techniques used to implement the service is at the heart of a much more general approach to managing the understanding of complex software systems.
- ❖ Each member of the community is providing a service that is used by other members of the group.
- ❖ No member could solve the problem on their own, and it is only by working together that the desired outcome is achieved.



Abstraction Form: Composition

- ❖ The idea is to begin with a few primitive forms and add rules for combining forms to create new forms.
- ❖ The key insight in composition is to permit the combination mechanism to be used both on the new forms as well as the original primitive forms.
- ❖ We begin with the primitive types, such as int and Boolean.
- ❖ The idea of a class then permits the user to create new types.
- ❖ These new types can include data fields constructed out of previous types, either primitive or user-defined.
- ❖ Since classes can build on previously defined classes, very complex structure can be constructed piece by piece.

```
class Box { //a box is a new data type  
  
    ...  
  
    private int value ;//built out of existing data type  
  
}
```

Composition cont.

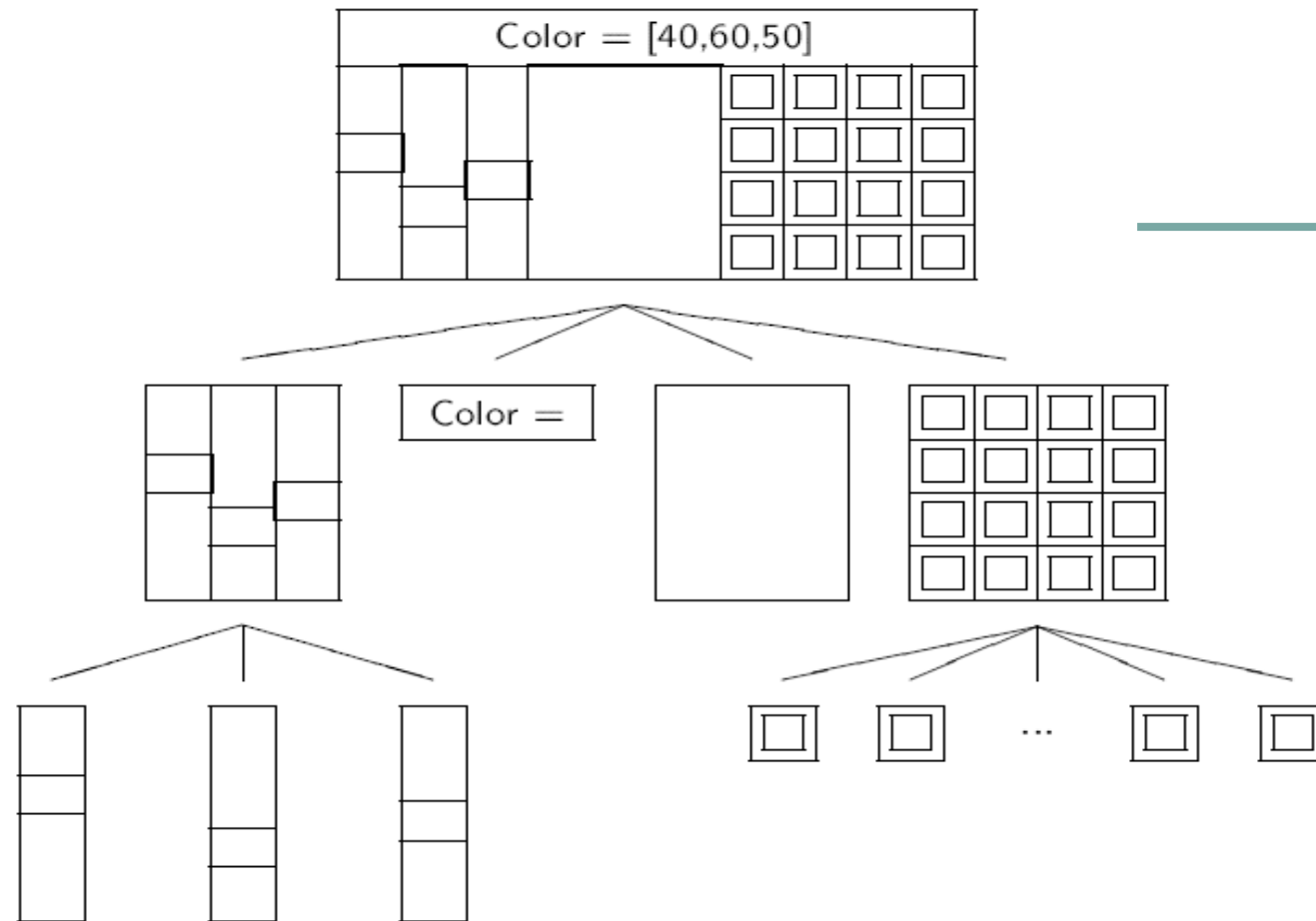
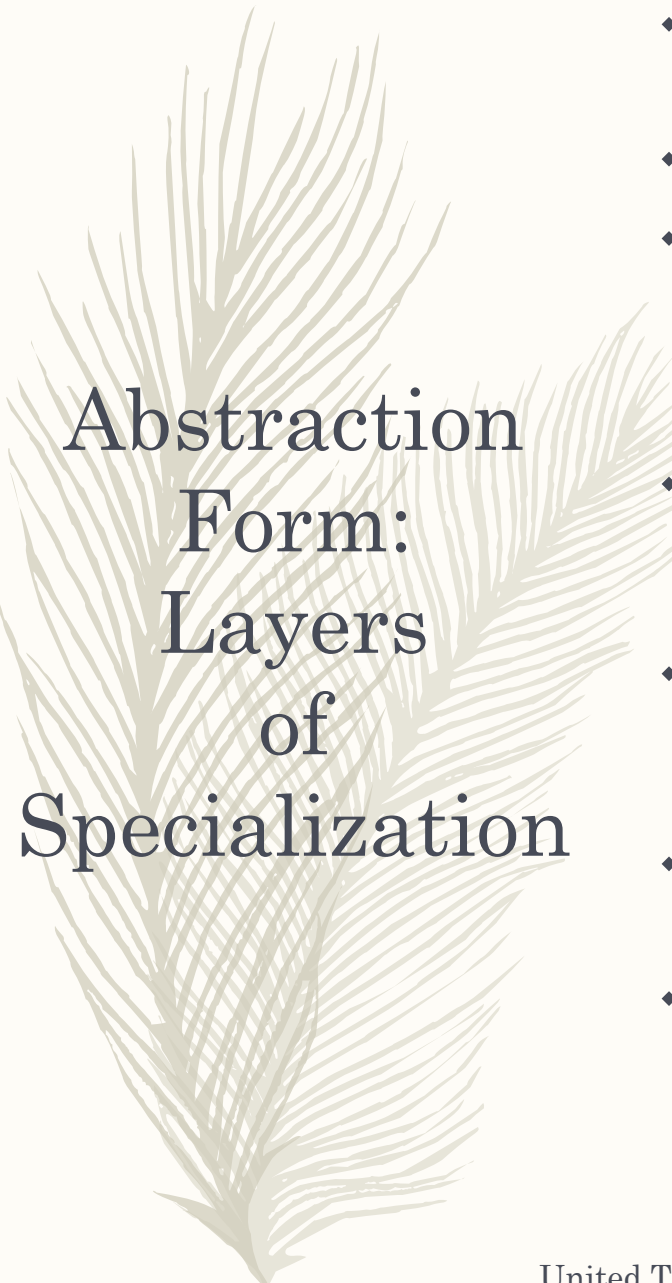


Figure Composition in the Creation of User Interfaces



Abstraction Form: Layers of Specialization

- ❖ Yet another approach to dealing with complexity is to structure abstraction using layers of specialization.
- ❖ This is sometimes referred to as a taxonomy.
- ❖ For example, in biology we divide living things into animals and plants. Living things are then divided into vertebrates and invertebrates. Vertebrates eventually includes mammals, which can be divided into (among other categories) cats and dogs, and so on.
- ❖ The key difference between this and the earlier abstraction is that the more specialized layers of abstraction (for example, a cat) is indeed a representative of the more general layer of abstraction (for example, an animal).
- ❖ This was not true when, in an earlier example, we descended from the characterization of a muscle to the description of different chemical interactions.
- ❖ These two different types of relations are sometimes described using the heuristic keywords “is-a” and “has-a”.
- ❖ The first relationship, that of parts to a whole, is a has-a relation, as in the sentence “a car has an engine”. In contrast, the specialization relation is described using is-a, as in “a cat is a mammal”.

Layers of Specialization

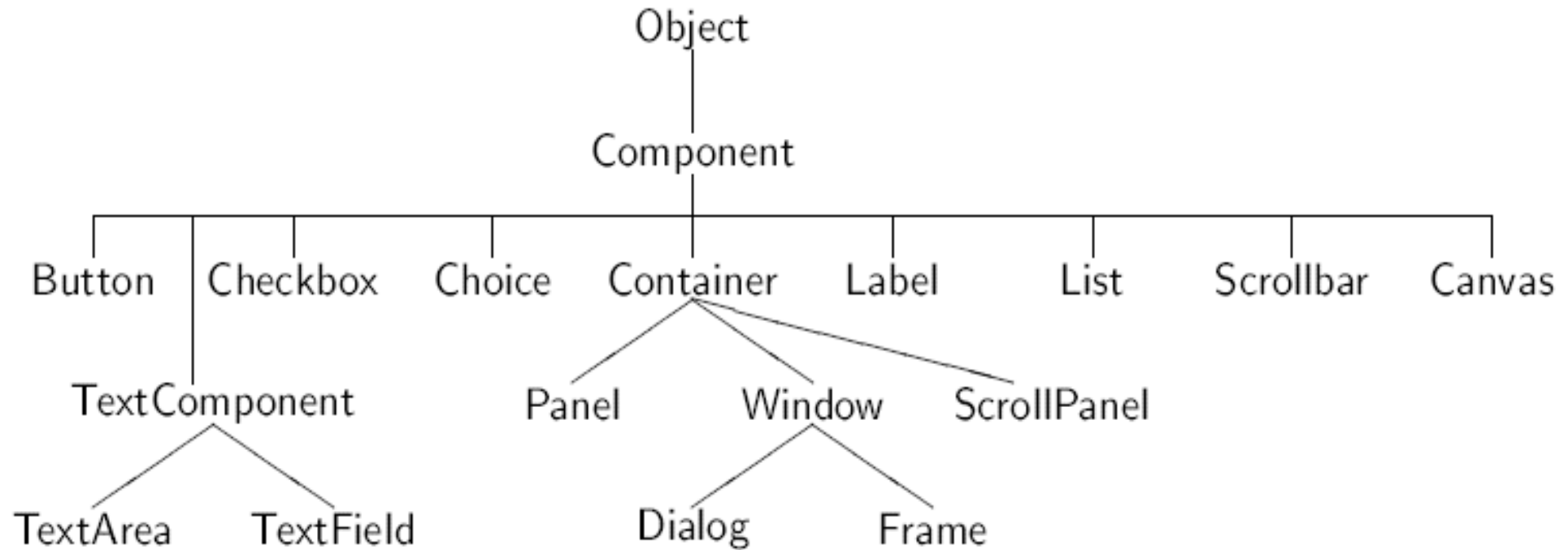


Figure The AWT class hierarchy

Non-Standard Behavior

- Phyl and his friends remind us that there are almost never generalizations without their being exceptions. A platypus (such as phyl) is a mammal that lays eggs. Thus, while we might associate the tidbit of knowledge “gives birth to live young” with the category Mammal , we then need to amend this with the caveat “lays eggs” when we descend to the category Platypus.
- *Object-oriented languages will also need a mechanism to override information inherited from a more general category.*
- We will explore this in more detail once we have developed the idea of class hierarchies.

Abstraction Form: Patterns

- ❖ When faced with a new problem, most people will first look to previous problems they have solved that seem to have characteristics in common with the new task.
- ❖ These previous problems can be used as a model, and the new problem attacked in a similar fashion, making changes as necessary to fit the different circumstances.
- ❖ This insight lies behind the idea of a software pattern.
- ❖ A pattern is nothing more than an attempt to document a proven solution to a problem so that future problems can be more easily handled in a similar fashion.
- ❖ In the object-oriented world this idea has been used largely to describe patterns of interaction between the various members of an object community.
- ❖ Example: *Network proxy*.
- ❖ When the proxy receives a request for data or action, it bundles the request as a package, transmits the package over the network, receives the response, unpackages the response and hands it back to the client. In this fashion the client is completely unaware of the details of the network protocol.
- ❖ The description of the pattern has captured certain salient points of the interaction (the need to hide the communication protocol from the client) while omitting many other aspects of the interaction (for example, the particular information being communicated between client and server).





Summary of Abstraction

- ❖ *People deal with complex artifacts and situations every day.*
- ❖ *Thus, while many readers may not yet have created complex computer programs, they nevertheless will have experience in using the tools that computer scientists employ in managing complexity.*
- ❖ *The most basic tool is abstraction, the purposeful suppression of detail in order to emphasize a few basic features.*
- ❖ *Information hiding describes the part of abstraction in which we intentionally choose to ignore some features so that we can concentrate on others.*
- ❖ *Abstraction is often combined with a division into components.*
- ❖ *For example, we divided the automobile into the engine and the transmission.*
- ❖ *Components are carefully chosen so that they encapsulate certain key features and interact with other components through a simple and fixed interface .*
- ❖ *The division into components means we can divide a large task into smaller problems that can then be worked on more-or-less independently of each other.*



Summary of Abstraction cont.

- ❖ *It is the responsibility of a developer of a component to provide an implementation that satisfies the requirements of the interface.*
- ❖ *A point of view that turns out to be very useful in developing complex software system is the concept of a service provider.*
- ❖ *A software component is providing a service to other components with which it interacts.*
- ❖ *In real life we often characterize members of the communities in which we operate by the services they provide.*
- ❖ *A delivery person is charged with transporting flowers from a florist to a recipient.*
- ❖ *Thus this metaphor allows one to think about a large software system in the same way that we think about situations in our everyday lives.*
- ❖ *Another form of abstraction is a taxonomy, in object-oriented languages more often termed an inheritance hierarchy.*
- ❖ *The layers are more detailed representatives of a general category.*
- ❖ *An example of this type of system is a biological division into categories such as Living Thing-Animal- Mammal-Cat.*



Summary of Abstraction cont.

- ❖ *Each level is a more specialized version of the previous.*
- ❖ *This division simplifies understanding, since knowledge of more general levels is applicable to many more specific categories.*
- ❖ *When applied to software this technique also simplifies the creation of new components, since if a new component can be related to an existing category all the functionality of the older category can be used for free.*
- ❖ *Thus, for example, by saying that a new component represents a Frame in the Java library we immediately get features such as a menu bar, as well as the ability to move and resize the window.*
- ❖ *Finally, a particular tool that has become popular in recent years is the pattern.*
- ❖ *A pattern is simply a generalized description of a solution to a problem that has been observed to occur in many places and in many forms.*
- ❖ *The pattern described how the problem can be addressed, and the reasons both for adopting the solution and for considering other alternatives.*