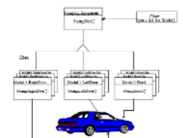
ACADEMIC AND RESEARCH PURPOSES ONLY

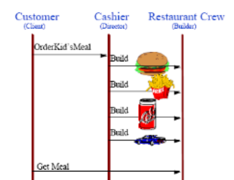
Design Patterns
(Quick Reference)

Creational	Structural	Behavioral	Architectural	Design
1. Abstract Factory	1. Adapter	1. Chain of Responsibility	1. Layers	1. Whole-Part
2. Factory Method	2. Bridge	2. Command	2. Pipes and Filters	2. Master-Slave
3. Builder	3. Composite	3. Interpreter	3. Blackboard	3. Proxy
4. Prototype	4. Decorator	4. Iterator	4. Broker	4. Command Processor
5. Singleton	5. Facade	5. Mediator	5. Model-View-Controller	5. View Handler
	6. Flyweight	6. Memento	6. Presentation-Abstraction-Control	6. Forwarder-Receiver
	7. Proxy	7. Observer	7. Microkernel	7. Client-Dispatcher-Server
		8. State	8. Reflection	8. Publisher-Subscriber
		9. Strategy		
		10. Template Method		
		11. Visitor		

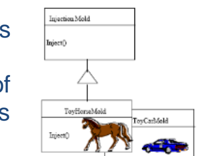
Creational



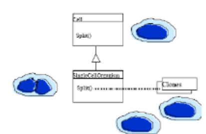
1. Abstract Factory: The purpose of the *Abstract Factory* is to provide an interface for creating families of related objects without specifying concrete classes.
Ex: Sheet metal stamping equipment is an example of an *Abstract Factory* for creating auto body parts. Using rollers to change the dies, the concrete class can be changed. The possible concrete classes are hoods, trunks, roofs, left and right front fenders, etc. The master parts list ensures that classes will be compatible. Note that an *Abstract Factory* is a collection of *Factory Methods*



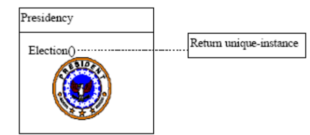
2. Builder: The *Builder* pattern separates the construction of a complex object from its representation, so the same construction process can create different representations.
Ex: Fast food restaurants use a *Builder* to construct their children's meals. There can be variation in the contents (the main course, the drink, or the toy), but the process for building a children's meal remains the same. Note that the *Builder* returns a finished product, whereas the *Abstract Factory* returns a collection of related parts



3. Factory Method: The *Factory Method* defines an interface for creating objects, but lets subclasses decide which classes to instantiate.
Ex: In injection molding, manufacturers process plastic molding powder and inject the plastic into molds of desired shapes. Like the *Factory Method*, the subclasses (in this case the molds) determine which classes to instantiate. In the example, the ToyHorseMold class is being instantiated.

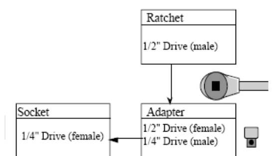


4. Prototype: The *Prototype* pattern specifies the kind of objects to instantiate using a prototypical instance.
Ex: The mitotic division of a cell results in two cells of identical genotype. This cell "cloning" is an example of the *Prototype* pattern in that the original cell takes an active role in creating a new instance of itself.

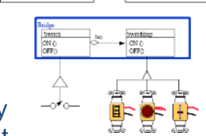


5. Singleton: The *Singleton* pattern ensures that a class has only one instance and provides a global point of reference to that instance.
Ex: The office of the Presidency of the United States is an example of a *Singleton*, since there can be at most one active president at any given time. Regardless of who holds the office, the title "The President of the United States" is a global point of reference to the individual.

Structural

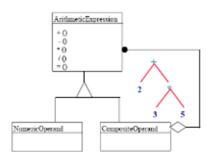


1. Adapter: The *Adapter* pattern allows otherwise incompatible classes to work together by converting the interface of one class to an interface expected by the clients.
Ex: A 1/2" drive ratchet will not ordinarily work with a 1/4" drive socket. Using an *Adapter*, the female end interfaces with the 1/2" drive ratchet, and the male end interfaces with the 1/4" socket.

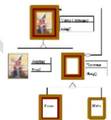


2. Bridge: The *Bridge* pattern decouples an abstraction from its implementation, so that the two can vary independently. Note that the schematics of house wiring state only where switches will be located, not what

type of switch it will be.
Ex: A switch is a device for turning lights, ceiling fans, garbage disposals on or off. The actual implementation of the switch is decoupled from the abstract switch. This decoupling of the abstraction and implementation is an example of the *Bridge* Pattern.

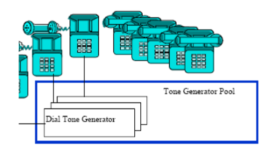


3. Composite: The *Composite* composes objects into tree structures, and lets clients treat individual objects and compositions uniformly.
Ex: Arithmetic expressions can be expressed as trees where an operand can be a number or an arithmetic expression. Since the individual objects, and compositions of individual objects are treated uniformly, an arithmetic expression is an example of the *Composite* pattern.

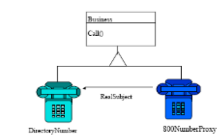


4. Decorator: The *Decorator* attaches additional responsibilities to an object dynamically.
Ex: Paintings can be hung on a wall with or without frames. Often, paintings will be matted and framed before hanging. The painting, frame, and matting form a visual component

5. Façade: The *Facade* defines a unified, higher level interface to a subsystem, that makes it easier to use.
Ex: When ordering from a catalog, consumers do not have direct contact with the Order Fulfillment, Billing, and Shipping departments. The Customer Service Representative acts as a *Facade*, or unified interface, to each of the departments involved in the transaction.

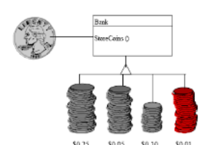


6. Flyweight: The *Flyweight* uses sharing to support large numbers of objects efficiently.
Ex: Most telephone subscribers are unaware that the pool of Tone generators (such as dial tone, busy tone or reorder tone) is much smaller than the number of subscribers. The pool of Dial Tone generators is an example of a *Flyweight.*

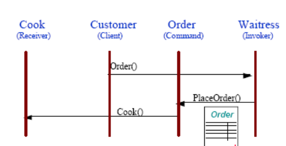


7. Proxy: The *Proxy* provides a surrogate or place holder to provide access to an object.
Ex: An 800 (8xx) number is a proxy for the real directory number. Callers will dial the 800 number, just as they would dial the actual directory number.

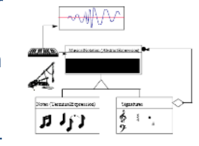
Behavioral



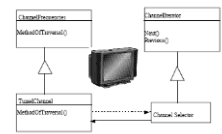
1. Chain of Responsibility: The *Chain of Responsibility* pattern avoids coupling the sender of a request to the receiver.
Ex: A mechanical sorting bank uses a single slot for all coins. As each coin is dropped, a *Chain of Responsibility* determines which tube accommodates each coin. If a tube cannot accommodate the coin, the coin is passed on until a tube can accept the coin.



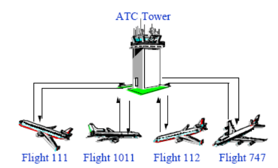
2. Command: The *Command* pattern allows requests to be encapsulated as objects, thereby allowing clients to be parameterized with different requests.
Ex: The “check” at a restaurant is used to encapsulate the customer’s order. The waitress takes the order, but it is the cook that carries out the *Command*. Since the pad of “checks” can be used by different restaurants, they can support many different commands



3. Interpreter: The *Interpreter* pattern defines a grammatical representation for a language, and an interpreter to interpret the grammar.
Ex: Musical notation provides a grammatical representation for the pitch and duration of sounds. When musicians play from a score, they are *Interpreters,* interpreting that grammar.



4. Iterator: The *Iterator* provides ways to access elements of an aggregate object sequentially without exposing the underlying structure of the object.
Ex: The channel selector on modern day television sets is an example of an *Iterator.* Rather than a dial containing all possible channels, or one button per tuned channel, today’s television sets have a Next and Previous button for tuning channels. The “Channel Surfer” doesn’t need to know if Channel 3 or Channel 4 comes after Channel 2.

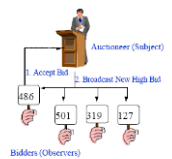


5. Mediator: The *Mediator* defines an object that controls how a set of objects interact. Loose coupling between colleague objects is achieved by having colleagues communicate with the Mediator, rather than one another.
Ex: The control tower at an airport provides a central point of communication for aircraft in the terminal area. Constraints on terminal area airspace are maintained by the tower. With the centralized communication and constraint maintenance, the tower behaves as a *Mediator.*

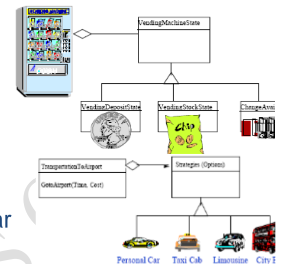
6. Memento: The *Memento* captures and externalizes an object’s internal state, so the object can be restored to that state later.
Ex: Most people are particular about the radio station that they listen to in the car. When there

ACADEMIC AND RESEARCH PURPOSES ONLY

is more than one driver, (Father, Mother, Child), the radio station is likely to have changed with the driver. The preset buttons serve as mementos, allowing the radio to be restored to the desired tuning with one button push.

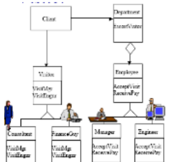


7. Observer: The *Observer* defines a one-to-many relationship, so that when one object changes state, the others are notified and updated automatically.
Ex: When a bidder at an auction accepts a bid, he or she raises a numbered paddle which identifies the bidder. The bid price then changes and all *Observers* must be notified of the change. The auctioneer then broadcasts the new bid to the bidders.

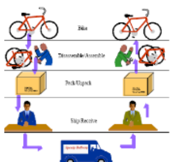


8. State: The *State* pattern allows an object to change its behavior when it's internal state changes.
Ex: The behavior of a vending machine varies based on its *State.* The currency on deposit, the product inventory, the selection, and the currency in the change bank are all part of the vending machine's *State.*

9. Strategy: A *Strategy* defines a set of algorithms that can be used interchangeably.
Ex: There are many modes of transportation to and from an airport. Choosing a particular transportation *Strategy* involves making tradeoffs between cost, time, and convenience.

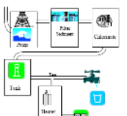


10. Template Method: The *Template Method* defines a skeleton of an algorithm in an operation and defers some steps to subclasses.
Ex: Subdivision developers often use a *Template Method* to produce a variety of home models from a limited number of floor plans. The basic floor plans are a skeleton, and the differentiation is deferred until later in the building process.



11. Visitor: The *Visitor* pattern represents an operation to be performed on the elements of an object structure, without changing the classes on which is operates.
Ex: An outside consultant coming into a department, to meet with everyone in the department on a one-on one basis is an example of *Visitor.* While the visitor is escorted to each cubicle, she does nothing. Once she arrives at a cubicle, she springs into action interviewing the employee (sending messages and obtaining results).

Architectural patterns



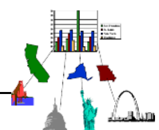
1. Layers: The purpose of *Layers* is to structure applications that can be decomposed into different subtasks, with each subtask at an appropriate level of abstraction.
Ex: The *Layers* pattern helps structure applications that can be decomposed into groups of subtasks at different layers of abstraction. Consider shipping a bike. The person at the shipping or receiving end of the process, or the driver may not know how to assemble or disassemble a bike. Then again, they don't need to know how.

2. Pipes and Filters: *Pipes and Filters* provides a structure for systems that process a stream of data.
Ex: Public water systems demonstrate *Pipes and Filters.* The water flowing through pipes is the input to a filter. Pipes are also used as the output from a filter. This example shows that name correspondence between the example and pattern can produce a viable example.

3. Blackboard: The *Blackboard* pattern is used to assemble knowledge from a variety of sources when no deterministic solution strategy is known.
Ex: Watch just about any police show on television and notice that the crimes cannot be solved immediately. There is often blackboard with forensic evidence, ballistic reports, crime scene data, etc. posted. When all of the experts add their pieces of the puzzle, the police are able to solve the crime!

4. Broker: The *Broker* pattern is used to structure distributed systems with decoupled components that interact by remote service invocations. The Broker component is responsible for coordinating communication between clients and servers.
Ex: A travel agent is a broker for travel related services. A customer dealing with a travel agent can book passage on ships, planes and trains, reserve a hotel room and rental car, and book tours. The customer deals only with the travel agent, although several companies are involved. Note that there are often complex codes on the itinerary. The broker and travel services companies understand these codes. The customer often does not.

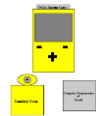
5. Model-View-Controller: The *Model-View-Controller* pattern divides an interactive application into a Model (or core data), view (display) and controller.
Ex: Drawing different graphs pie-chart, line graph etc. (views) of data (model) using configuration (control) in any database application



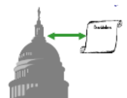
ACADEMIC AND RESEARCH PURPOSES

6. Presentation-Abstraction-Controller: The *Presentation-Abstraction-Controller* pattern defines a structure for interactive systems in the form of a hierarchy of cooperating agents. Each agent is responsible for a specific aspect of the application's functionality.
Ex: Polls will often gather information from all over the country, and display results as national totals, regional totals, state totals and city totals. The information is gathered at each site, and then reported to a central site.

7. Microkernel: The *Microkernel* pattern separates the minimal functional core from extended functionality.
Ex: Nintendo makes a Gameboy hand-held video game. The Gameboy unit provides the core functionality, such as displays and input/output for numerous games and the Gameboy camera. A game cartridge or the Gameboy camera can be inserted to allow the user to play.

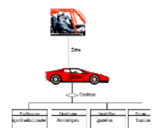


8. Reflection: The *Reflection* pattern provides a mechanism for changing structure and behavior of a system dynamically. A system is structured as a base level, and meta level. The meta level contains information about system properties. The base level exhibits the application structure and behavior. Changes to meta level information affect subsequent base level structure and behavior.
Ex: *Reflection* provides a mechanism for changing structure and behavior dynamically. Consider a constitution. Changing the behavior of a legislative body is done by changing the constitution. The constitution can be considered to be a metaobject, while the legislative body is the base object.

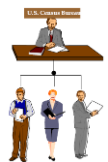


Design Level

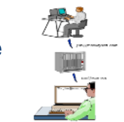
1. Whole-Part: The *Whole-Part* pattern helps aggregate components that form a semantic unit.
Ex: A car encapsulates many parts that a driver cannot access directly. While driving, the fuel injector will supply fuel to the cylinder, the distributor will provide spark to the sparkplug in that cylinder, the sparkplug will fire, and the piston will be forced down, thereby turning the cam. The driver has no direct access to any of these parts.



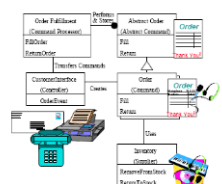
2. Master-Slave: A master component distributes work to identical slave components and computes a final result from the results when the slaves return
Ex: When tasked with taking the census, the Census Bureau utilizes census takers. The census takers are semantically the same, (a census taker for one area could just a easily work a different area), and the results are tabulated by the Census Bureau. Any "clients" of the Census Bureau deal directly with the Bureau, and not with the individual census takers.



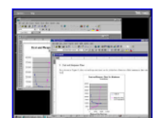
3. Proxy: The *Proxy* provides a surrogate or place holder to provide access to an object.
Ex: Many organizations offer a proxy e-mail address. When someone sends mail to a proxy address, he or she may have no idea that it is a proxy address. The proxy then forwards the message to the "real" e-mail address.



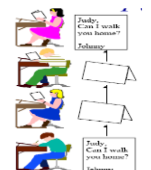
4. Command Processor: The *Command Processor* pattern separates the request for a service from its execution.
Ex: In catalog sales, orders can be received by Mail, Phone, or FAX. When a request is received, an order is generated. The order will cause items to be removed from stock in order to fill the order. If an order is returned, the item will be returned to stock, and the account will be credited. If a web site based order form is implemented, the basic order processing will remain intact.



5. View Handler: The *View Handler* pattern helps manage all views provided by a system.
Ex: Although window managers are common to software, the visual nature of windows allows one to examine behavior without looking at source code. The window manager opens, manipulates, and disposes of views. It also coordinates dependencies between views, and coordinates their updates.



6. Forwarder-Receiver: The *Forwarder-Receiver* pattern introduces a peer-to-peer communication model that decouples the forwarders and receivers from the underlying communication mechanisms.
Ex: School children often use the *Forwarder-Receiver* pattern. Johnny will write a message to Judy, fold it, pass it to Mary, and ask her to pass it to Judy. Mary will pass it to Billy, and ask him to pass it to Judy. Billy will pass it to Judy, who will unfold the message and read it.



7. Client-Dispatcher-Server: The *Client-Dispatcher-Sever* introduces an intermediate layer between clients and servers to provide location transparency and hide connection details.
Ex: When a client calls a company to talk with an individual, a receptionist often answers the call, and then routes it to the appropriate person. The client cannot tell if the person is in the same building, or at a remote site. Furthermore, the client does not know if the phone rings on the sales manager's desk, or if the receptionist puts him on hold, and yells, "Hey Larry, pick up the phone!"

8. Publisher-Subscriber: The *Publisher-Subscriber* pattern helps to keep the state of cooperating agents synchronized. A publisher notifies any number of subscribers of changes in state.

ACADEMIC AND RESEARCH PURPOSES ONLY

Ex: To communicate to others, one can post or publish a notice on a bulletin board. The subscribers (readers) of the bulletin board can vary dynamically. State information, such as having extra tickets can be communicated to unknown individuals when it is not feasible to poll people explicitly to determine if they require the tickets.

ACADEMIC AND RESEARCH PURPOSES ONLY