Use Chain of Responsibility when1) more than one object may handle a request and the handler isn't known a priori. The handler should be as curtained automatically.2) you want to issue a request to one of several objects without specifying the receiver explicitly.3) the set of objects that can handle a request should be specified dynamically.,Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.,As the name suggests the chain of responsibility pattern creates a chain of receiver objects for a request. This pattern decouples sender and receiver of a request based on type of request. This pattern comes under behavioral patterns.In this pattern normally each receiver contains reference to another receiver.,In null object pattern a null object replaces check of null object instance. instead of putting if check for a null value null object reflects a do nothing relationship. such null object can also be used to provide default behaviour in case data is not available.In null object pattern we create an abstract class specifying various operations to be done concrete classes extending this class and a null object class providing do nothing implemention of this class and will be used seemlessly where we need to check null value.,provide an object as a surrogate for the lack of an object of a given type.the null object pattern provides intelligent do nothing behavior hiding the details from its collaborators.,use the iterator pattern 1) to access an aggregate object's contents without exposing its internal representation.2) to support multiple traversals of aggregateobjects.3) to provide a uniform interface for traversing different aggregate structures(that is to support polymorphic iteration).,the iterator pattern allow us to:1)access contents of a collection without exposing its internal structure.2)support multiple simultaneous traversals of a collection.3)provide a uniform interface for traversing different collection.,use the command pattern when you want to parameterize objects by an action to perform as menultem objects did above.you can express such parameterization in a procedural language with a callback function that is a function that's registered somewhere to be called at a later point. commands are an object-oriented replacement for callbacks.specify queue and execute requests at different times. a command object can have a lifetime independent of the original request. if the receiver of a request can be represented in an address space-independent way then you can transfer a command object for the request to a different process and fulfill the request there. support undo. the command's execute operation can store state for reversing its effects in the command itself. the command interface must have an added unexecute operation that reverses the effects of a previous call to execute. executed commands are stored in a history list. unlimited-level undo and redo is achieved by traversing this list backwards and forwards calling unexecute and execute respectively. support logging changes so that they can be reapplied in case of a system crash. by augmenting the command interface with load and store operationsyou can keep a persistent log of changes. recovering from a crash involves reloading logged commands from disk and reexecuting them with the execute operation. structure a system around high-level operations built on primitives operations. such a structu re is common in informat ion systems that support transactions. a transaction encapsulates a set of changes to data. the command pattern offers a way to model transactions. commands have a common interface letting you invoke all transactions the same way. the pattern also makes it easy to extend the system with new transactions.,Encapsulate a request as an object thereby letting you parameterize clients with different requests queue or log requests and support undoable operations.,Command pattern is a data driven design pattern and falls under behavioral pattern category. A request is wrapped under an object as command and passed to invoker object. Invoker object looks for the appropriate object which can handle this command and passes the command to the corresponding object which executes the command,use the mediator pattern when1) a set of objects communicate in well-defined but complex ways .the resulting interdependencies are unstructured and difficult to understand.2) reusing an object is difficult because it refers to and communicates with many other objects.3) a behavior that's distributed between several classes should be customizable without a lot of subclassing.,define an object that encapsulates how a set of objects interact .mediator promotes loose coupling by keeping objects from referring to each other explicitly and itlets you vary their interaction independently.,Mediator pattern is used to reduce communication complexity between multiple objects or classes. This pattern provides a mediator class which normally handles all the communications between different classes and supports easy maintenance of the code by loose coupling. Mediator pattern falls under behavioral pattern category.,use the observer pattern in any of the following situations:1) when an abstractio n has two aspects one dependent on the other. Encapsulating these aspects in separate objects lets you vary and reuse them independently.2) when a change to one object requires changing others and you don't know how many objects need to be changed.3) when an object should be able to notify other objects without making assumptions about who these objects are. in other words you don't want these objects tightly coupled.,define a one-to-many dependency between objects so that when one object changes state all its dependents are notified and updated automatically.,Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified its depenedent objects are to be notified automatically. observer pattern falls under behavioral pattern category.,the template method pattern should be used 1) to implement the invariant parts of an algorithm once and leave it up to subclasses to implement the behavior that can vary.2) when common behavior among subclasses should be factored and localized in a common class to avoid code duplication. this is a good example of 'refactoring to generalize' as described by opdyke and johnson. you first identify the differences in the existing code and then separate the differences into new operations. finally you replace the differing code with a template method that calls one of these new operations.3) to control subclasses extensions. you ca n define a template method that calls 'hook' operations (see consequences) at specific points thereby permitting extensions only at those points.,Define the skeleton of an algorithm in an operation deferring some steps to subclasses. template method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.,In template pattern an abstract class exposes defined way(s)/template(s) to execute its methods. its subclasses can override the method implementation as per need but the invocation is to be in the same way as defined by an abstract class. this pattern comes under behavior pattern category.,use the interpreter pattern when there is a language to interpret and you can represent statements in the language as abstract syntax trees. the interpreter pattern work s best when 1) the grammar is simple. for complex grammars the cl ass hierarchy for the grammar becomes large and unmanageable. tools such as parser generators are a better alternative in such cases. they can interpret expressions without building abstract syntax trees which can save space and possibly time.2) efficiency is not a critical concern. the most efficient interpreters are usually not implemented by interpreting parse trees directly but by first translating them into another form. for example regular expressions are often transformed into state machines. but even then the translator can be implemented by the interpreter pattern so the pattern is still applicable.,the interpreter pattern is used exaustively in defining grammars,define a representation f or its grammar along with an interpreter that uses the representation to interpret sentences in the language.,Interpreter pattern provides a way to evaluate language grammar or expression. This type of pattern comes under behavioral pattern. This pattern involves implementing an expression interface which tells to interpret a particular context. This pattern is used in sql parsing symbol processing engine etc.,the strategy design pattern splits the behavior (there are many behaviors) of a class from the class itself.,use the strategy pattern when 1) many related classes differ only in their behavior. strategies provide a way to configure a class with one of many behaviors.2) you need different variants of an algorithm. for example you might define algorithms reflecting different space/time trade-offs. strategies can be used when these variants are implemented as a class hierarchy of algorithms.3)an algorithm uses data that clients shouldn't know about. use the strategy pattern to avoid exposing complex algorithm-specific data structures.4) a class defines many behaviors,Define a family of algorithms encapsulate each one and make them interchangeable. strategy lets the algorithm vary independently from clients that use it.,In strategy pattern a class behavior or its algorithm can be changed at run time. this type of design pattern comes under behavior pattern. in strategy pattern we create objects which represent various strategies and a context object whose behavior varies as per its strategy object. the strategy object changes the executing algorithm of the context object.,Allow an object to alter its behavior when its internal state changes. the object will appear to change its class.,In state pattern a class behavior changes based on its state. this type of design pattern comes under behavior pattern.in state pattern we create objects which represent various states and a context object whose behavior varies as its state object changes.,use the memento pattern when 1) a snapshot of (some portion of) an object's state must be saved so that it can be restored to that state later and 2) a direct interface to obtaining the state would expose implementation details and break the object's encapsulation.,undo and restore operations in most software.database transactions discussed earlier.,without violating encapsulation capture and externalize an object's internal state so that the object can be restored to this state later.,Memento pattern is used to restore state of an object to a previous state. Memento pattern falls under behavioral pattern category.,use the visitor pattern when 1) an object structure contains many classes of objects with differing interfaces and you want to perform operations on these objects that depend on their concrete classes.2) many distinct and unrelated operations need to be performed on object s in an object structure and you want to avoid 'polluting' their classes with these operations. visitor lets you keep related operations together by defining them in one class. when the object structure is shared by many applications use visitor to put operations in just those applications that need them.3) the classes defining the object structure rarely change but you often want to define new operations over the structure. changing the object structure classes requires redefining the interface to all visitors which is potentially costly. if the object structure classes change often then it's probably better to define the operations in those classes.,the visitor pattern is a great way to provide a flexible design for adding new visitors to extend existing functionality without changing existing code.,represent an operation to be performed on the elements of an object structure. visitor lets you define a new operation without changing the classes of the elements on which it operates.,In visitor pattern we use a visitor class which changes the executing algorithm of an element class. By this way execution algorithm of element can vary as and when visitor varies. This pattern comes under behavior pattern category. As per the pattern element object has to accept the visitor object so that visitor object handles the operation on the element object.,Use the Adapter pattern when 1)you want to use an existing class and its interface does not match the one you need.2)you want to create a reusable class that cooperates with unrelated orunforeseen classes- that is- classes that don't necessarily have compatible interfaces.3) (object adapter only) you need to use several existing subclasses but it's unpractical to adapt their interface by subclassing every one. An object adapter can adapt the interface of its parent class.,Convert the interfac e of a class into another interface clients expect. Adapter lets classes work together that couldn' t otherwise because of incompatible interfaces.,Adapter pattern works as a bridge between two incompatible interfaces. This type of design pattern comes under structural pattern as this pattern combines the capability of two independent interface.This pattern involves a single class which is responsible to join functionalities of independent or incompatible interfaces.,Use Decorator when 1)to add responsibilities to individual objects dynamically and transparently that is without affecting other objects.2)for responsibilities that can be withdrawn.3)when extension by subclassing is impractical. Sometimes a large number of independent extensions are possible and would produce an explosion of subclasses to support ever y combination. Or a class definition maybe hidden or otherwise unavailable for subclassing.,Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.,Decorator pattern allows a user to add new functionality to an existing object without altering its structure. This type of design pattern comes under structural pattern as this pattern acts as a wrapper to existing class.This pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact.,Use the Composite pattern when 1) you want to represent part-whole hierarchies of objects.2) you want clients to be able to ignore the difference between compositions of objects and individual objects. Clients will treat all objects in the composite structure uniformly.,Compose objects into tree structures to represent part-whole hierarchies. composite lets clients treat individual objects and compositions of objects uniformly.,Composite pattern is used where we need to treat a group of objects in similar way as a single object. Composite pattern composes objects in term of a tree structure to represent part as well as whole hierarchy. This type of design pattern comes under structural pattern as this pattern creates a tree structure of group of objects.This pattern creates a class that contains group of its own objects. This class provides ways to modify its group of same objects.,Use the Bridge pattern when 1) you want to avoid a permanent binding between an abstraction and its implementation.This might be the case- for example-when the implementation must be selected or switched at run-time.2) both the abstractions and their implementations should be extensible by subclassing. In this case- the Bridge pattern lets you combine the different abstractions and implementations and extend them independently. 3) changes in the implementation of an abstraction should have no impact on clients; that is their code should not have to be recompiled.,Decouple an abstraction from its implementation so that the two can vary independently.,Bridge is used when we need to decouple an abstraction from its implementation so that the two can vary independently. This type of design pattern comes under structural pattern as this pattern decouples implementation class and abstract class by providing a bridge structure between them.This pattern involves an interface which acts as a bridge which makes the functionality of concrete classes independent from interface implementer classes. Both types of classes can be altered structurally without affecting each other.,Use the Facade pattern when1)you want to provide a simple interface to a complex subsystem.subsystem often get more complex as they evolve.Most patterns when applied result in more and smaller classes.This makes the subsystem more reusable and easier to customize but it also becomes harder to use for clients that don't need to customize it.A facade can provide a simple default view of the subsystem that is good enough for most clients.Only clients needing more customizability will need to look beyond the facade2)there are many dependencies between clients and the implementation classes of an abstraction.Introduce a facade to decouple the subsystem from clients and other subsystems thereby promoting subsystem independenceportability3)you want to layer your subsystems. Use a facade to define an entry point to each subsystem level.If subsystems are dependent then you can simplify the dependencies between them by making them communicate other solely through their facades.,Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.,Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities.This pattern involves a single class which provides simplified methods required by client and delegates calls to methods of existing system classes.,The private class data design pattern seeks to reduce exposure of attributes by limiting their visibility. It reduces the number of class attributes by encapsulating them in single Data object. It allows the class designer to remove write privilege of attributes that are intended to be set only during construction, even from methods of the target class.,The flyweight pattern applies to a program using a huge number of objects that have part of their internal state in common where the other part of state can vary. The pattern is used when the larger part of the objects state can be made extrinsic (external to that object).,The Flyweight pattern's effectiveness depends heavily on how and where it's used.Apply the Flyweight pattern when all of the following are true:1)An application uses a large number of objects.2)Storage costs are high because of the sheer quantity of objects.3)Most object state can be made extrinsic.4)Many groups of objects may be replaced by relatively few shared objects once extrinsic state is removed.5)The application doesn't depend on object identity. Since flyweight objects may be shared identity test will return true for conceptually distinct objects.,Use sharing to support large numbers of fine-grained objects efficiently.,Flyweight pattern is primarily used to reduce the number of objects created and to decrease memory footprint and increase performance. This type of design pattern comes under structural pattern as this pattern provides ways to decrease object count thus improving the object structure of application.Flyweight pattern tries to reuse already existing similar kind objects by storing them and creates new object when no matching object is found.,Java RMI as has been explained implements a remote proxy. Security Proxies that controls access to objects can be found in many object oriented languages including java C# C+ +.,Proxy is applicable whenever there is a need for a more versatile or sophisticated reference to an object than a simple pointer. Here are several common situations in which the Proxy pattern is applicable:1. A remote proxy provides a local representative for an object in a different address space. NEXTSTEP [Add 94]uses the class NXProxy for this purpose. Coplien [Cop 92]calls this kind of proxy an 'Ambassador.'2. A virtual proxy creates expensive objects on demand. The ImageProxy described in the Motivation is an example of such a proxy.3. A protection proxy controls access to the original object. Protection proxies are useful when objects should have different access rights. For example KernelProxies in the Choices operating system [CIRM93] provide protected access to operating system objects.4. A smart reference i s a replacement f or a bare pointer that performs additional actions when an object is accessed. Typical uses include - counting the number of references to the real object so that it can be freed automatically when there are no more references (also called smart pointers[Ede92]).- loading a persistent object into memory when it's first referenced.- checking that the real object is locked before it's accessed to ensure that no other object can change it.,Provide a surrogate or placeholder for another object to control access to it.,In proxy pattern a class represents functionality of another class.This type of design pattern comes under structural pattern.In proxy pattern we create object having original object to interface its functionality to outer world.,1)when a class can't anticipate the type of the objects it is supposed to create.2)when a class wants its subclasses to be the ones to specific the type of a newly created object.,1)a class can't anticipate the class of objects it must create.2)a class wants its subclasses to specify the objects it creates.3)classes delegate responsibility to one of several helper subclasses- and you want to localize the knowledge of which helper subclass is the delegate.,Define an interface for creating an object but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.,An increasingly popular definition of factory method is: a static method of a class that returns an object of that class' type. But unlike a constructor the actual object it returns might be an instance of a subclass. Unlike a constructor an existing object might be reused instead of a new object created.,Abstract Factory and Factory Methods implemented as singletons.There are certain situations when the a factory should be unique. Having 2 factories might have undesired effects when objects are created. To ensure that a factory is unique it should be implemented as a singleton. By doing so we also avoid to instantiate the class before using it.,Use the Singleton pattern when 1)there must be exactly one instance of a class and it must be accessible to clients from a well-known access point.2) when the sole instance should be extensible by subclassing and clients should be able to use an extended instance without modifying their code.,Ensure a class only has one instance- and provide a global point of access to it.,Singleton pattern is one of the simplest design patterns in Java. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.This pattern involves a single class which is responsible to create an object while making sure that only single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class.,1)When the classes are instantiated at runtime.2)When the cost of creating an object is expensive or complicated.3)When you want to keep the number of classes in an application minimum.4)When the client application needs to be unaware of object creation and representation.,Use Prototype Pattern when a system should be independent of how its products are created- composed- and represented- and:1)Classes to be instantiated are specified at run-time.2)Avoiding the creation of a factory hierarchy is needed.3)It is more convenient to copy an existing instance than to create a new one.,Specify the kinds of objects to crea te using a prototypical instance- and create new objects by copying this prototype.,Prototype pattern refers to creating duplicate object while keeping performance in mind. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.,Basically we'll use an object pool whenever there are several clients who needs the same stateless resource which is expensive to create.,When a client asks for a Reusable objec,Reuse and share objects that are expensive to create.,The object pool pattern is a software creational design pattern that uses a set of initialized objects kept ready to use ? a 'pool' ? rather than allocating and destroying them on demand. A client of the pool will request an object from the pool and perform operations on the returned object. When the client has finished- it returns the object to the pool rather than destroying it; this can be done manually or automatically.Object pools are primarily used for performance: in some circumstances- object pools significantly improve performance. Object pools complicate object lifetime- as objects obtained from and returned to a pool are not actually created or destroyed at this time- and thus require care in implementation.,1)Builder Pattern is used when the creation algorithm of a complex object is independent from the parts that actually compose the object. 2)The system needs to allow different representations for the objects that are being built.,Use the Builder pattern when 1)The algorithm for creating a complex object should be independent of the parts that make up the object and how they're assembled.2)The construction process must allow different representations for the object that's constructed.,Separate the construction of a complex object from its representation so that the same construction process can create different representations.,The builder pattern is an object creation software design pattern. Unlike the abstract factory pattern and the factory method pattern whose intention is to enable polymorphism-the intention of the builder pattern is to find a solution to the telescoping constructor anti-pattern. The telescoping constructor anti-pattern occurs when the increase of object constructor parameter combination leads to an exponential list of constructors. Instead of using numerous constructors- the builder pattern uses another object- a builder-that receives each initialization parameter step by step and then returns the resulting constructed object at once.,Provide an interface for creating families of related or dependent objects without specifying their concrete classes.,Abstract Factory patterns work around a super-factory which creates other factories. This factory is also called as factory of factories. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. In Abstract Factory pattern an interface is responsible for creating a factory of related objects without explicitly specifying their classes. Each generated factory can give the objects as per the Factory pattern.,We should use the Abstract Factory design pattern when: the system needs to be independent from the way the products it works with are created. the system is or should be configured to work with multiple families of products. a family of products is designed to work only all together. the creation of a library of products is needed.for which is relevant only the interface.not the implementation.too.,When the system needs to be independent of how its object are created composed and represented. When the family of related objects has to be used together then this constraint needs to be enforced. When you want to provide a library of objects that does not show implementations and only reveals interfaces. When the system needs to be configured with one of a multiple family of objects.<br>