



# Design Patterns for JMX and Application Manageability

Updated 28<sup>th</sup> June 2005



# Module Objectives



- When we have completed this module, you should be able to:
  - Apply design patterns while instrumenting your application for management
  - Know where and why to instrument your application

# List of Design Patterns

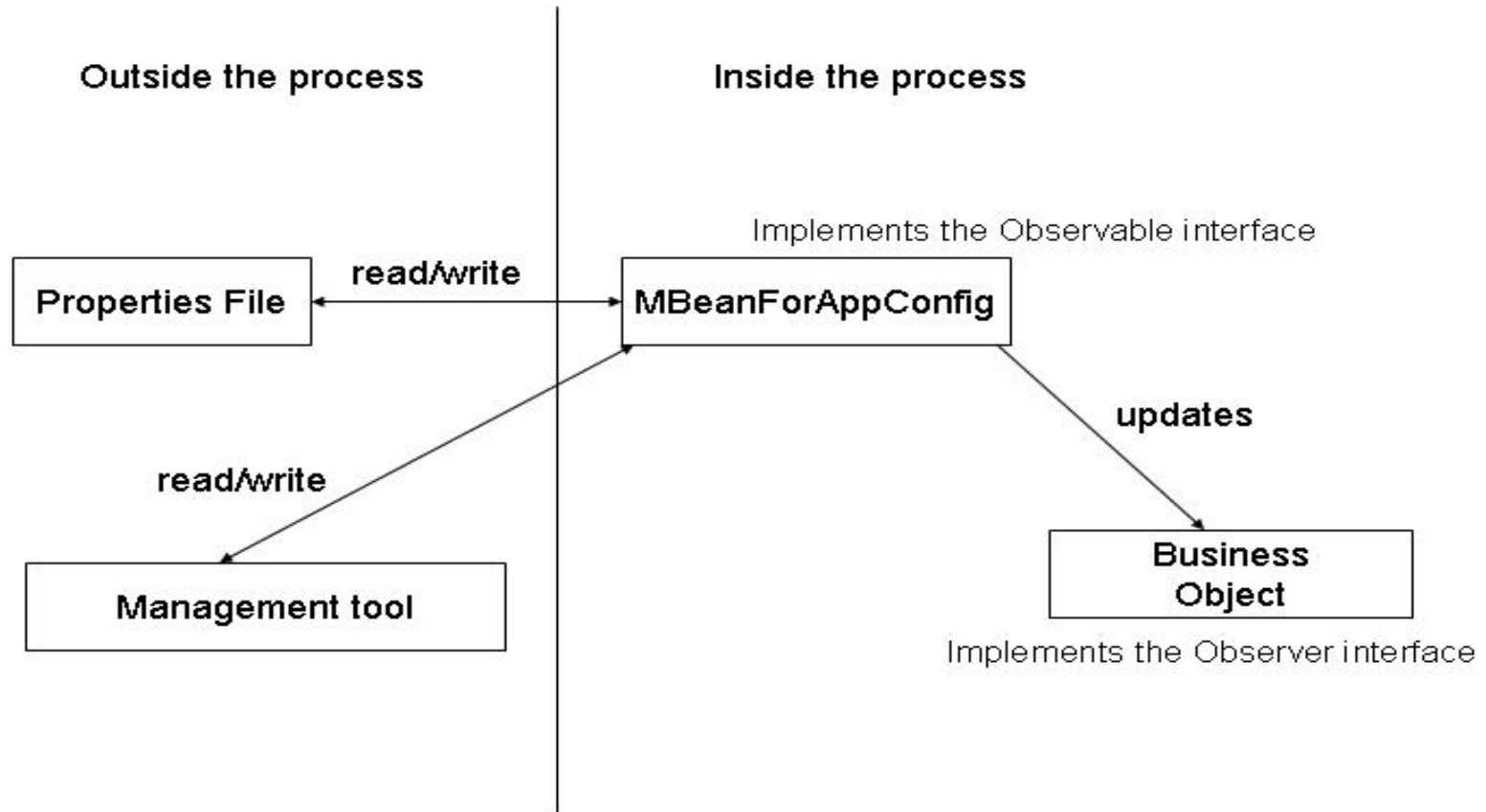
- Design Patterns for manageability
  - MBeanForAppConfig
  - MBeanWritesToLog
  - MBeanLivesForever
  - MBeanHelper
  - ManageabilityFacade
  - ManageabilityAspect
  - MBeanAggregator
  - PolicyMBean

# The MBeanForAppConfig Pattern



- One or more MBeans designed to contain the runtime configuration parameters for your application
- Constructed at initialization by reading from a properties file or other persistent data
- Allows those configuration parameters to be changed if necessary (e.g. the amount of logging)
- MBean browsers can view its content

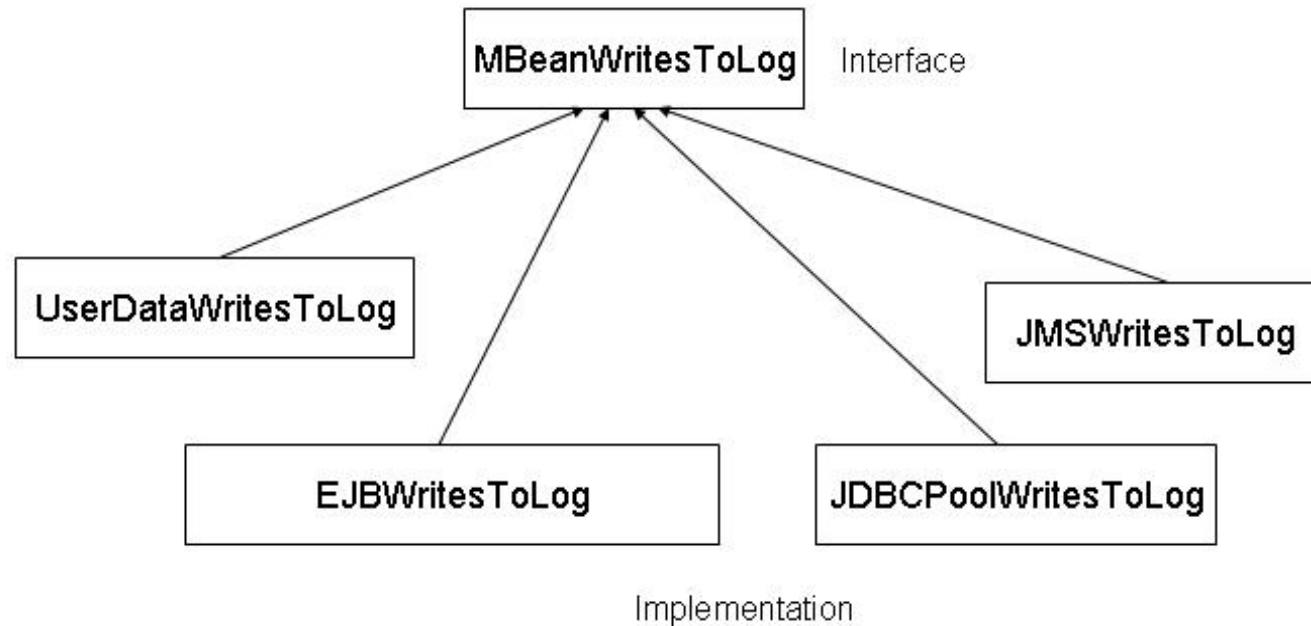
# MBeanForAppConfig Pattern



# The MBeanWritesToLog Pattern

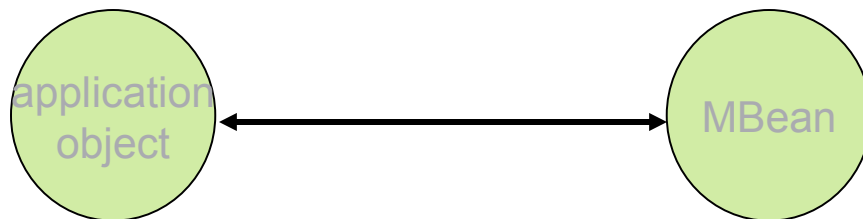
- MBeans are good places for counting quantities that are rapidly changing
- Requirement sometimes is to draw a time series of such data (e.g. # of users logged in, # of concurrent connections)
- Sending that data to a log file is a good approach
- The data can be processed in the log file offline from the MBean
- See Takiyu Liu's work at JavaOne 2003

# The MBeanWritesToLog Pattern



# Using the JMX model – decisions for developers

- Should a business object implement the Mbean interface?  
-or-
- Should the business object talk to a separate MBean object?
  
- Advantages and disadvantages?



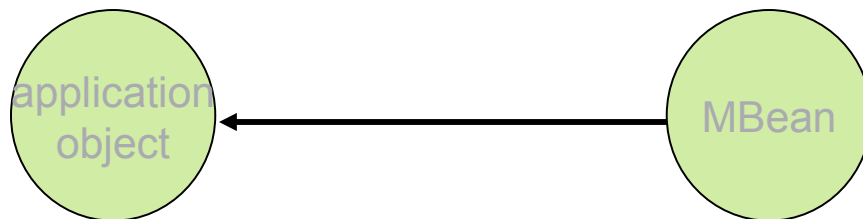
# Using an interface or separate object?



- Developer registers the new MBean with the MBean server
  - means every instance needs to register if we use the inheritance method
- The management interface should be separate from the business objects interfaces
- Use the MBean interface or connect the MBeans to the application objects
  - One-way or two-way references?
- Place code in the application object to feed data to the MBean
- Issue control requests from the MBean to the application object and provide appropriate responses
- One central MBean for all instances of a managed class – may be a good idea

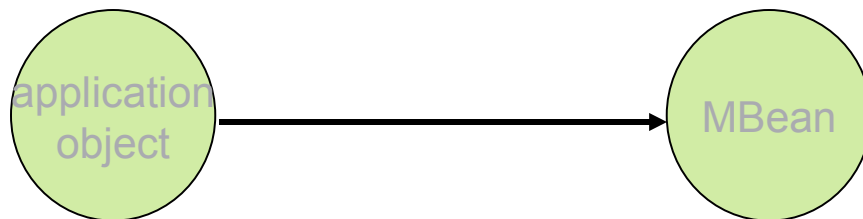
# Two Design models for JMX-enabling Applications

- Separation of concerns is the given
- Pull model
  - MBean polls the application components and obtains relevant state
  - The MBean must obtain a reference to the object
  - The MBean must pull the required information through method invocations and set its state accordingly



# Two Design models for JMX-enabling Applications

- Separation of concerns is given
- Push model
  - Application updates MBean on relevant state
  - The application must be able to locate the MBean in order to invoke it when necessary
  - The application may be responsible for instantiating the MBean and registering it with the MBean Server
  - The application can then update the MBean instance with relevant details when desired



# Recommendations



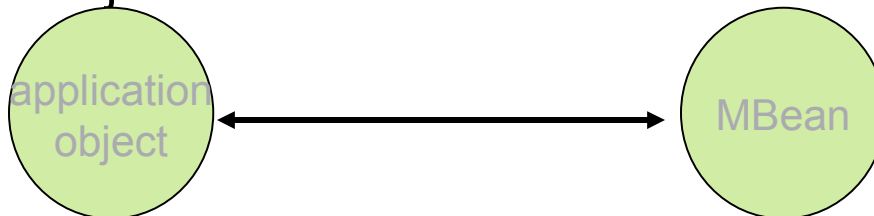
- Make your MBean as coarsely grained (large) as possible

But

- Do not mix completely different manageability issues into one MBean
  
- There should be very few MBeans needed for your application, ideally

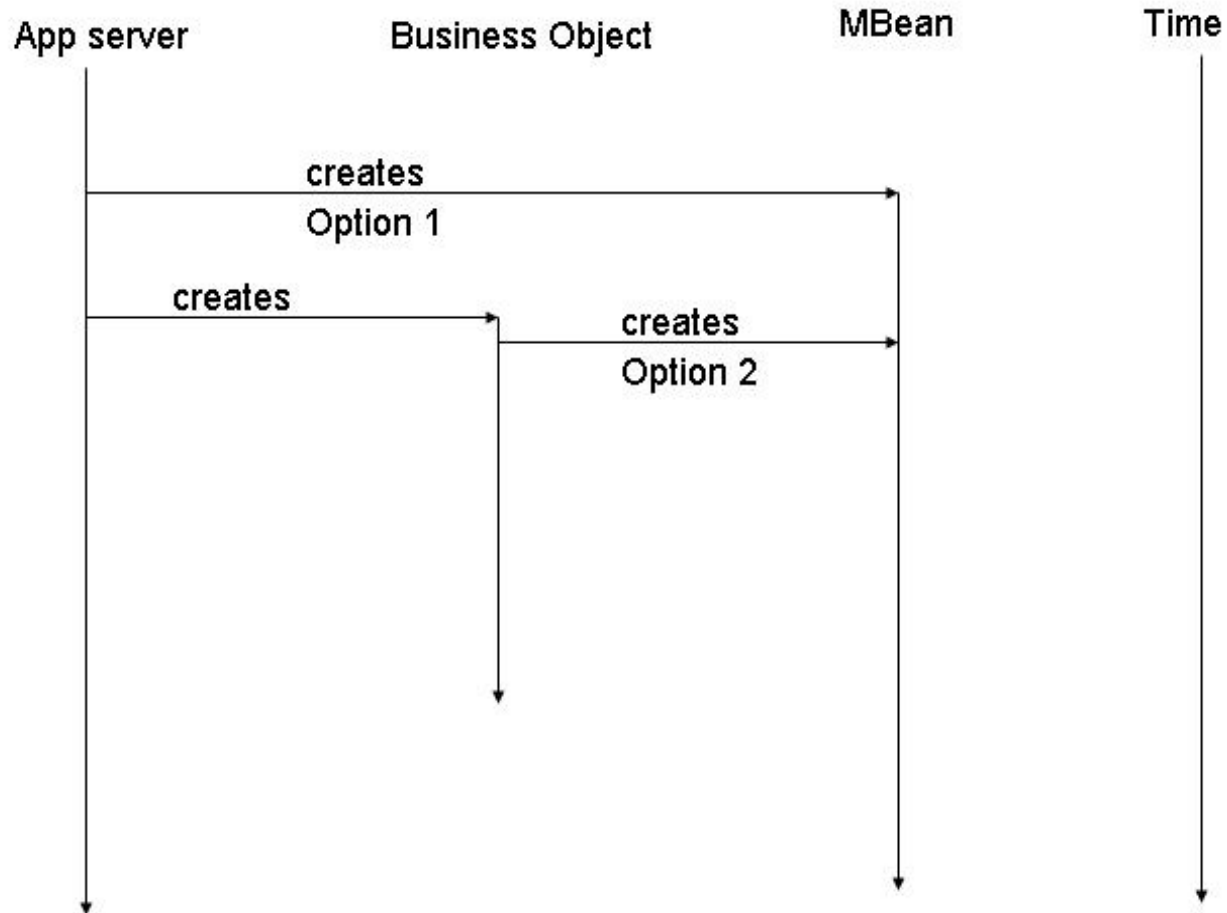
# Motivation for a JMX MBean Creational Pattern

- Which object creates the other object?
- (The answer depends on the lifetime we need to give the MBean – should it be alive past the lifetime of the application object?)
- MBean could create the application object
- Application object could create the MBean
- Application runtime server could create the MBean
- MBean could be in a separate process to the application object



# MBean LifeTime - 1 : MBeanLivesForever

## MBean Object Creation and Lifetime



# MBeanLivesForever Pattern

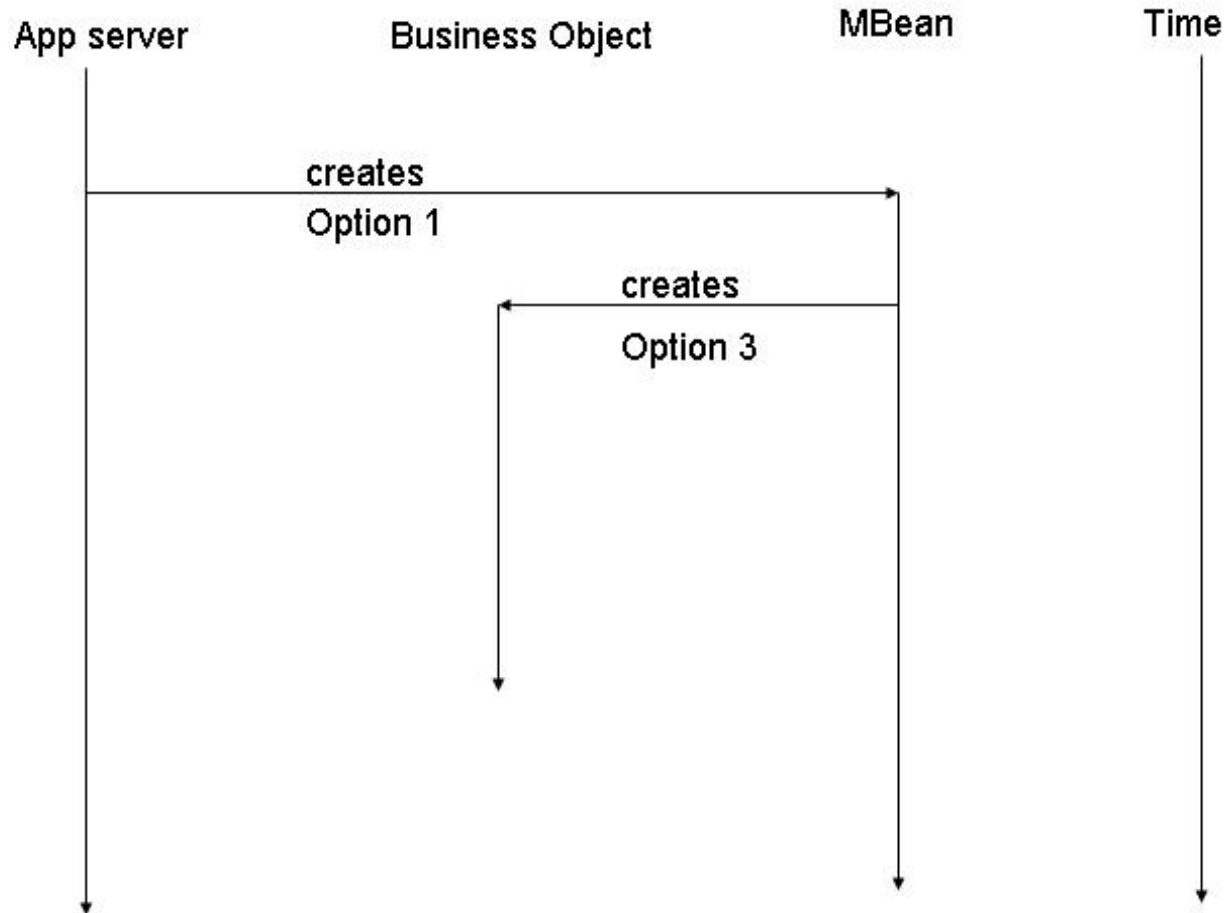
- Application servers/containers can create one or more MBeans for you (in a startup class)
- Important if the MBean needs to see the instantiation of other objects – like business objects
- These business objects are created here separately from the MBeans
- MBean is alive for the lifetime of the process



# MBean Lifetime – 2 : MBeanAsCreator



## MBean Object Creation and Lifetime



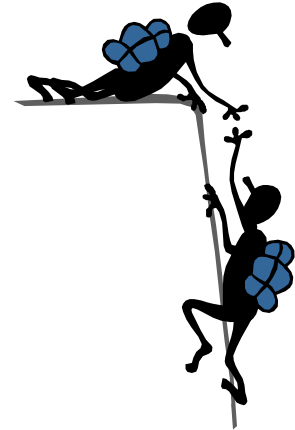
# MBeanAsCreator Pattern



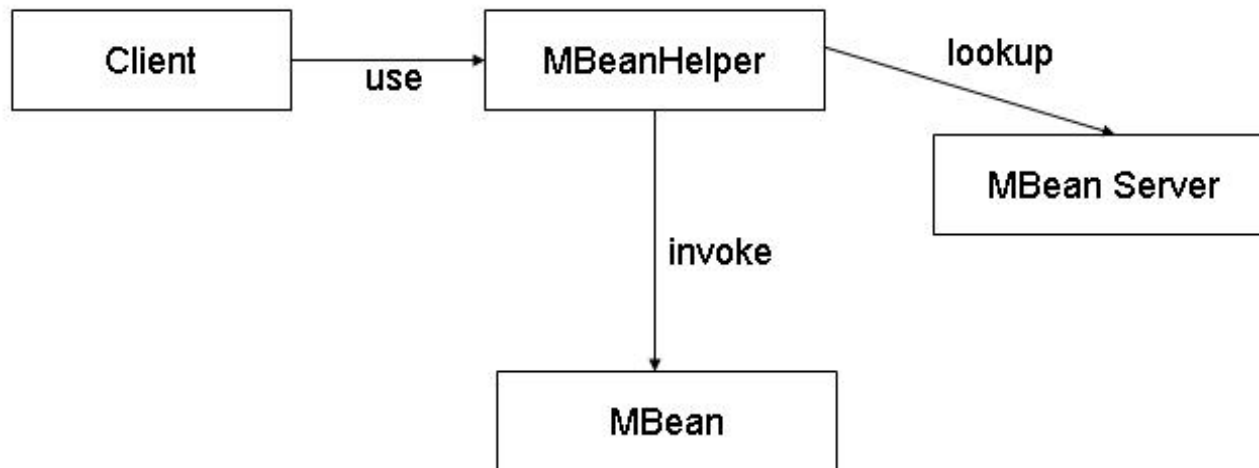
- The MBean is the *instantiation point* for business objects – MBean controls their lifespan
- MBean is alive for the lifetime of the process
- Easy capturing of business objects coming and going

# Motivation for an MBean Helper Pattern

- MBean Helper might take care of the mechanics of registration and lookup of the MBean (which are standard for every MBean)
- Details of strings for the MBean name and methods to be invoked are hidden from the business object



# The MBeanHelper Pattern



# The MBeanHelper Pattern



- The business object does not talk to the MBean object directly
- Instead they use a Proxy – the MBeanHelper, through which all communication flows to the MBean
- MBeanHelper takes care of the mechanics of registration and lookup of the MBean
- Details of the strings for MBean name and any invoked method name are hidden from the business object
- Better strong typing on method signature in MBeanHelper

# MBean Proxy

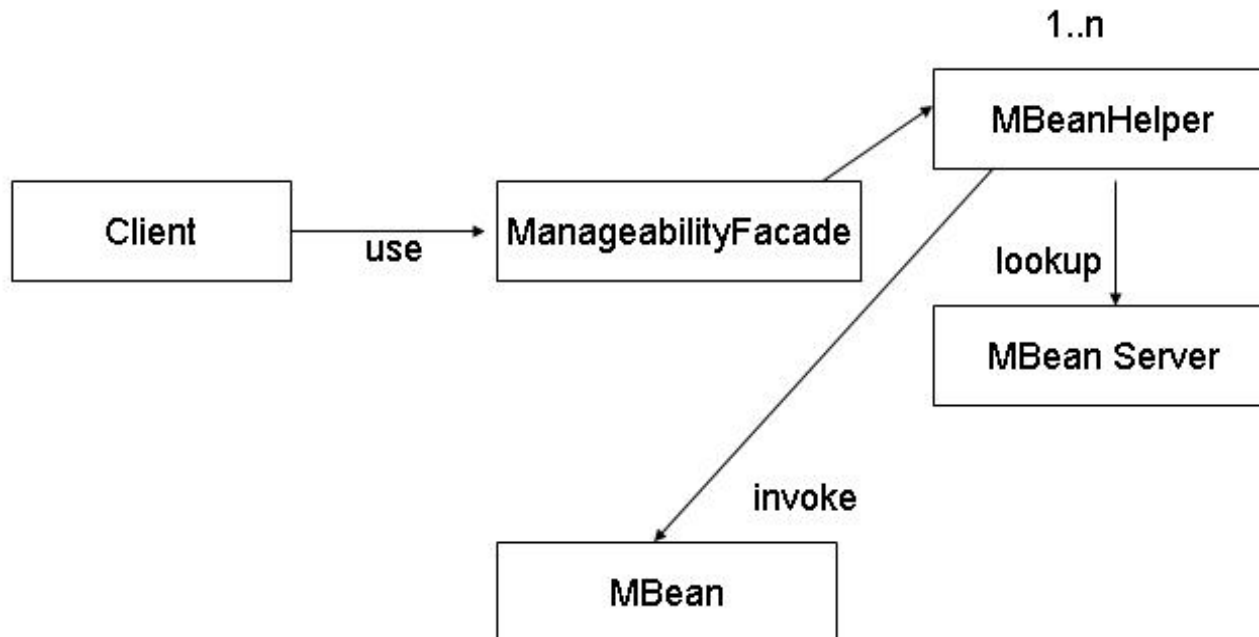
- Use the dynamic proxy in Java, `java.lang.reflect.Proxy`

## Benefits

- No more Strings in object and method names
- No need for one-to-one relationship between Helper and MBean
- MBeanRegistrar takes care of registering the bean with the MBeanServer
- MBeanProxy creates a proxy with the same interface as the MBean, ready for invocations
- Naming of MBean based on name of MBean interface



# The Manageability Façade Pattern



# The ManageabilityFacade Pattern

- Think of this Façade as a collection of all the MBeans in one centralized place, with just one method

ManageabilityFacade.update(origin, msgtype, data)

- Individual MBean names and methods are not important to the business objects that use the ManageabilityFacade
- The ManageabilityFacade decides which MBean gets updated
- Separation of business object concerns from manageability concerns
- Manageability implementation can vary without affecting the business logic



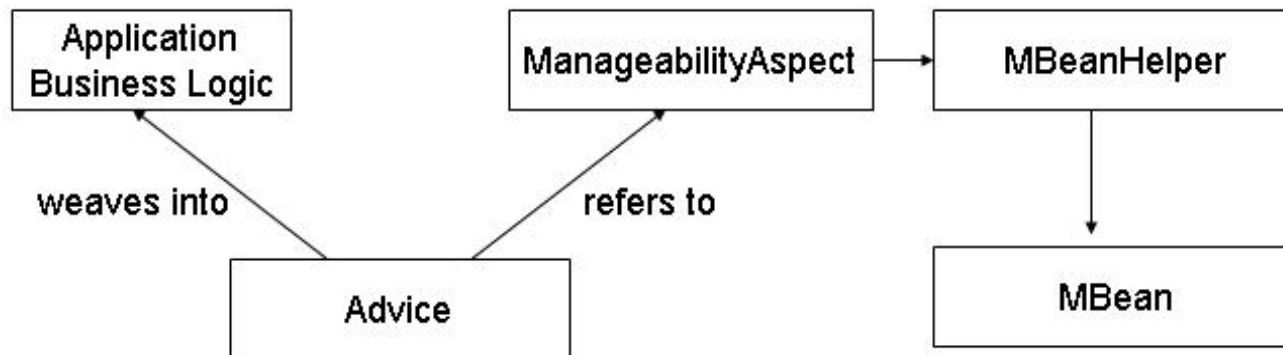
# ManageabilityFacade Pattern Example

- Management tool not aware of which business objects are active
- External interface hidden from the management tool
- Each business component invokes the ManagementFacade
- No knowledge on how the information will be treated
- Data can be filtered in the ManagementFacade to avoid data overload

# ManageabilityFacade Pattern Example

- Use AbstractFactory to create MBean
- Makes it possible to extend configuration of MBeans
- Use MBeanLivesForever Pattern when creating the MBean

# An Aspect for Manageability

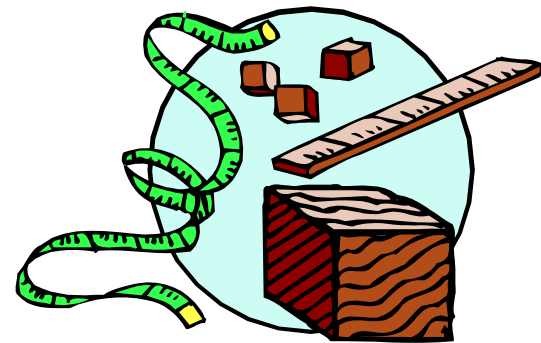


# The ManageabilityAspect Pattern

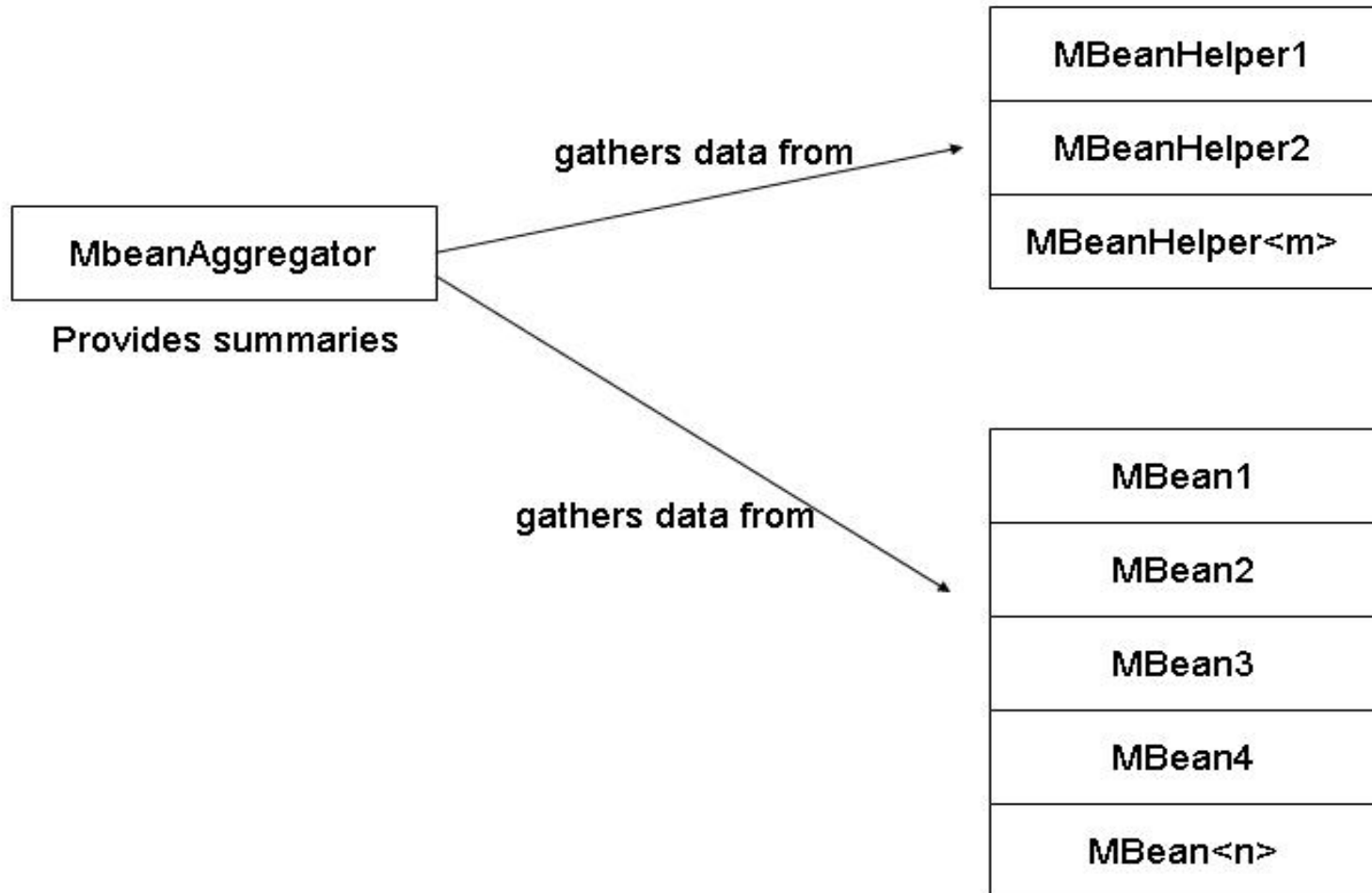
- Already accepted in the area of Logging and Monitoring (Laddad's book, JBoss)
- Now that we have centralized Manageability concerns into one Façade, why not weave it into the business logic as an Aspect?
- No code in the business logic for Manageability
- Can change the manageability implementation independently of the business logic

# Manageability Aspect Pattern Example

- Performance measurement
  - Have your advice active
  - Control which packages, classes or methods get measured from the management tool
  - Performance measurement in real-time
  - No extra code
- Security
  - Audit invocations
  - Audit data



# The MBeanAggregator Pattern



# The MBeanAggregator Pattern



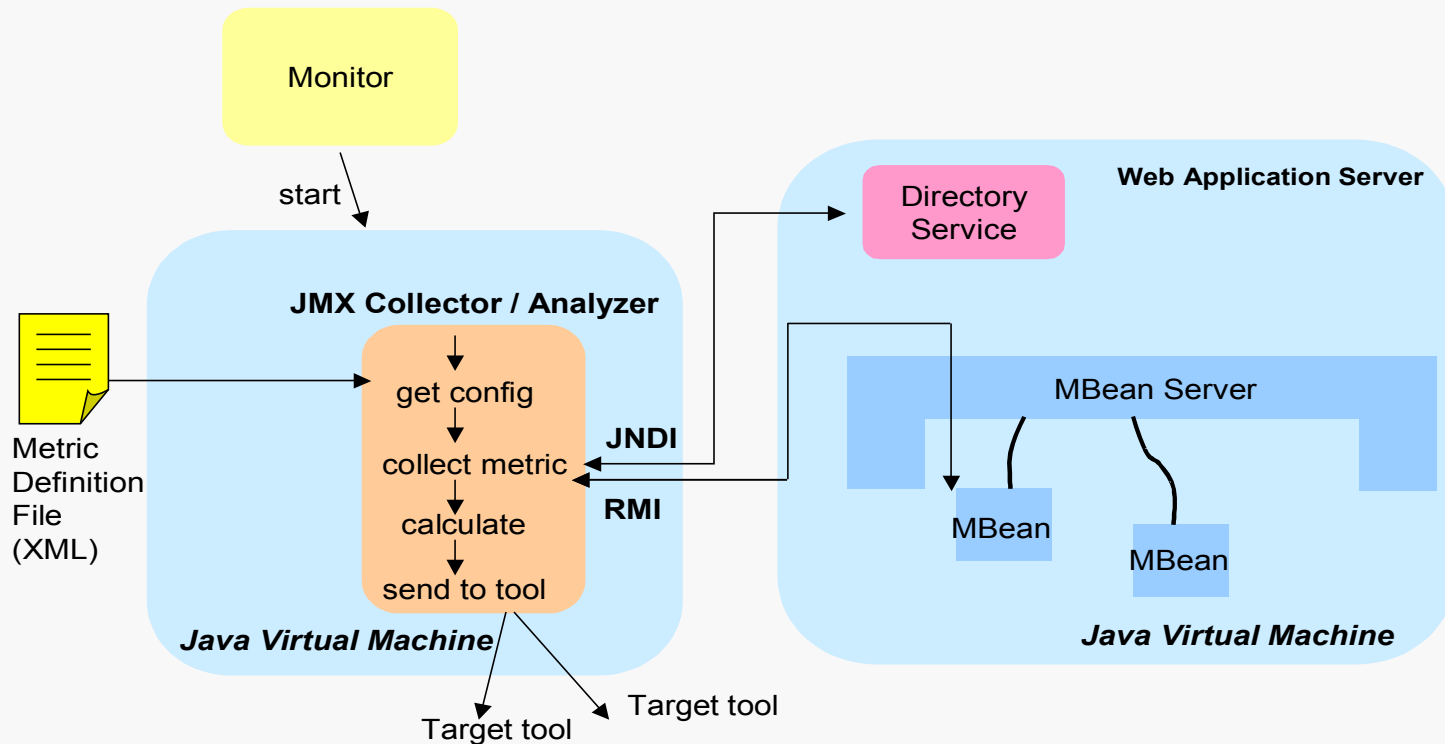
- There can be hundreds or even thousands of MBeans to deal with
- Management tools should not be polling large number of these MBeans or MBeanHelpers to get the important data they need.
- Collection points are needed in an Aggregator object – which may be an MBean itself.
- Fewer numbers of these will help performance and scalability
- They could cross application server boundaries



# A Guiding Principle for Application Manageability

- A Policy is that set of rules/measures which express the action to be taken on the occurrence of a management event
- **Manageability policies can change over time**
- These policies should therefore be **removed from the business application code** (and from the MBean code)
- Policies belong in a policy engine with a scripting language

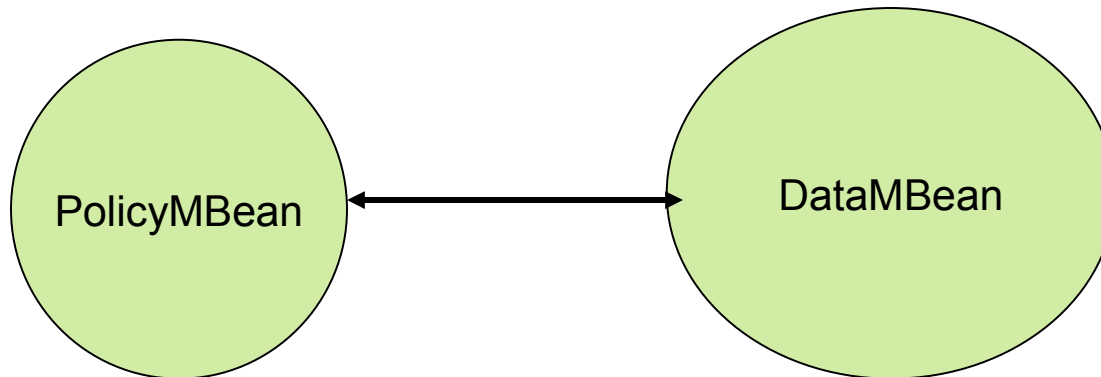
# MBeans Free of Management Policy - Separation



# The PolicyMBean Pattern



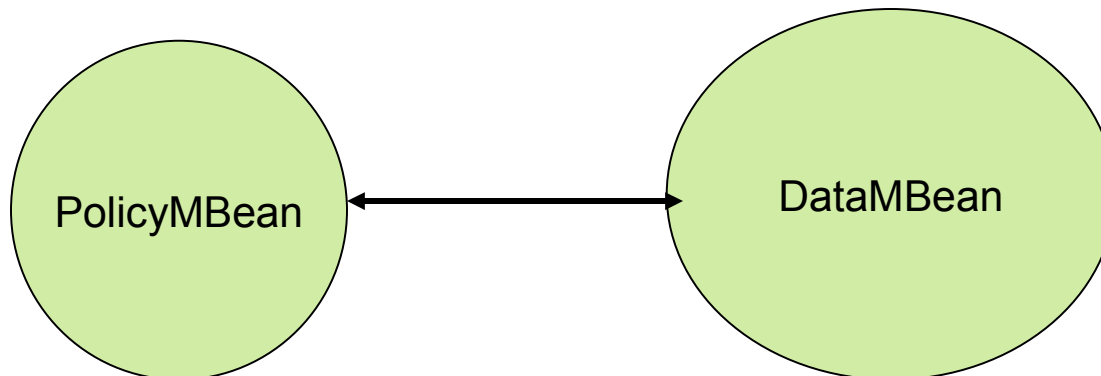
- A compromise on separation :If you are not prepared to remove the management policy decisions from the code, an MBean can help
- Separate the policy from the data collector MBean
- Keep all changeable decisions in the PolicyMBean



# The PolicyMBean Pattern



- External tools collect information from the DataMBean
- Changes to the policy (for data collection) only affect the PolicyMBean
- The PolicyMBean tells the DataMBean when to collect its data and what granularity to apply to measures



# Recommendations

- Profile your application with a performance tool, both for business logic and manageability logic effects on the response/throughput
- OpenView Transaction Analyzer does a good job on this
- Critical measures in the application may be the best place to place MBeans, but may also be the hot points for performance bottlenecks
- Gauge the effects of access or update to the MBean

## What have we seen in this module

- A set of design patterns for implementing JMX MBeans in our application
- Separation of business concerns from management concerns (and separation of these interfaces) is a good principle
- Extracting the policy from the business logic and from the management logic in the MBeans is a good principle in design
- Treat your MBeans as data containers, to be polled at intervals
- Profile your application to find its hotspots as a trigger for implementing manageability and to watch the effects of adding manageability code



**i n v e n t**