UNLOCK
OPPORTUNITY
What will you open?

Java SE Beyond Basics: Generics, Annotation, Concurrency, and JMX

Sang Shin
sang.shin@sun.com
www.javapassion.com
Sun Microsystems Inc.

solaris Java

SUN TECH DAYS 2006-2007
A Worldwide Developer Conference

Agenda

- Generics
- Annotation
- Concurrency
- JMX (Management & Monitoring)
- Performance
- Summary

Hands-on Labs

- Generics
  - www.javapassion.com/handsonlabs/1110_tigergenerics.zip
- Annotation
  - www.javapassion.com/handsonlabs/1111_javase5generics.zip
- Concurrency
  - www.javapassion.com/handsonlabs/1108_javase5concurrency.zip
- JMX (Management & Monitoring)
  - www.netbeans.org/kb/articles/jmx-tutorial.html

Generics

Sub-topics of Generics

- What is and why use Generics?
- Usage of Generics
- Generics and sub-typing
- Wildcard
- Type erasure
- Interoperability
- Creating your own Generic class

Generics: What is it? How do define it? How to use it? Why use it?

What is Generics?

- Generics provides abstraction over Types
  - Classes, Interfaces and Methods can be Parameterized by Types (in the same way a Java type is parameterized by an instance of it)
- Generics makes type safe code possible
  - If it compiles without any errors or warnings, then it must not raise any unexpected ClassCastException during runtime
- Generics provides increased readability
  - Once you get used to it

Definition of a Generic Class: LinkedList<E>

- Definitions: LinkedList<E> has a type parameter E that represents the type of the elements stored in the linked list

```
public class LinkedList<E>
    extends AbstractSequentialList<E>
    implements List<E>, Queue<E>, Cloneable, java.io.Serializable{
    private transient Entry<E> header = new Entry<E>(null, null, null);
    private transient int size = 0;

    public E getFirst() {
            if (size==0) throw new NoSuchElementException();
            return header.next.element;
    }
```

Usage of Generic Class: LinkedList<Integer>

- Usage: Replace type parameter <E> with concrete type argument, like <Integer> or <String> or <MyType>
  - LinkedList<Integer> can store only Integer or sub-type of Integer as elements

```
LinkedList<Integer> li =
             new LinkedList<Integer>();
li.add(new Integer(0));
Integer i = li.iterator().next();
```

Example: Definition and Usage of Parameterized List interface

```
// Definition of the Generic'ized
// List interface
//
interface List<E>{
   void add(E x);
  Iterator<E> iterator();
   ...
}

// Usage of List interface with
// concrete type parameter,String
//
List<String> ls = new ArrayList<String>(10);
```

Type parameter

Type argument

Why Generics? Non-genericized Code is not Type Safe

```
// Suppose you want to maintain String
// entries in a Vector.  By mistake,
// you add an Integer element.  Compiler
// does not detect this. This is not
// type safe code.

Vector v = new Vector();
v.add(new String(“valid string”)); // intended
v.add(new Integer(4));            // unintended

// ClassCastException occurs during runtime
String s = (String)v.get(0);
```

Why Generics?

- Problem: Collection element types
  - Compiler is unable to verify types of the elements
  - Assignment must have type casting
  - ClassCastException can occur during runtime
- Solution: Generics
  - Tell the compiler the type of the collection
  - Let the compiler do the casting
  - Example: Compiler will check if you are adding Integer type entry to a String type collection
    - Compile time detection of type mismatch

Generics:
Usage of Generics

Using Generic Classes: Example 1

- Instantiate a generic class to create type specific object
- In J2SE 5.0, all collection classes are rewritten to be generic classes

```
// Create a Vector of String type
Vector<String> vs = new Vector<String>();
vs.add(new Integer(5)); // Compile error!
vs.add(new String("hello"));
String s = vs.get(0);    // No casting needed
```

Using Generic Classes: Example 2

- Generic class can have multiple type parameters
- Type argument can be a custom type

```
// Create HashMap with two type parameters
HashMap<String, Mammal> map =
  new HashMap<String, Mammal>();
map.put(“wombat”, new Mammal("wombat"));

Mammal w = map.get(“wombat”);
```

Generics: Sub-typing

Generics and Sub-typing

- You can do this (using pre-J2SE 5.0 Java)
  - Object o = new Integer(5);
- You can even do this (using pre-J2SE 5.0 Java)
  - Object[] or = new Integer[5];
- So you would expect to be able to do this (Well, you can't do this!!!)
  - ArrayList<Object> ao = new ArrayList<Integer>();
  - This is counter-intuitive at the first glance

Generics and Sub-typing

- Why this compile error? It is because if it is allowed, ClassCastException can occur during runtime – this is not type-safe

```
ArrayList<Integer> ai = new ArrayList<Integer>();
ArrayList<Object> ao = ai; // If it is allowed at compile time,
ao.add(new Object());
Integer i = ai.get(0); // This would result in
                       // runtime ClassCastException
```

- So there is no inheritance relationship between type arguments of a generic class

Generics and Sub-typing

- The following code work
  - ArrayList<Integer> ai = new ArrayList<Integer>();
  - List<Integer> li2 = new ArrayList<Integer>();
  - Collection<Integer> ci = new ArrayList<Integer>();
  - Collection<String> cs = new Vector<String>(4);
- Inheritance relationship between generic classes themselves still exists

Generics and Sub-typing

- The following code work
  - ArrayList<Number> an = new ArrayList<Number>();
  - an.add(new Integer(5));   // OK
  - an.add(new Long(1000L));   // OK
  - an.add(new String(“hello”)); // compile error
- Entries in a collection maintain inheritance relationship

Generics:
Wild card

Why Wildcards? Problem

- Consider the problem of writing a routine that prints out all the elements in a collection
- Here's how you might write it in an older version of the language (i.e., a pre-5.0 release):

```
static void printCollection(Collection c) {
    Iterator i = c.iterator();
    for (k = 0; k < c.size(); k++) {
        System.out.println(i.next());
    }
}
```

Why Wildcards? Problem

- And here is a naive attempt at writing it using generics (and the new for loop syntax): Well.. You can't do this!

```
static void printCollection(Collection<Object> c) {
  for (Object o : c)
    System.out.println(o);
}

public static void main(String[] args) {
  Collection<String> cs = new Vector<String>();
  printCollection(cs); // Compile error
  List<Integer> li = new ArrayList<Integer>(10);
  printCollection(li); // Compile error
}
```

Why Wildcards? Solution

- Use Wildcard type argument <?>
- Collection<?> means Collection of unknown type
- Accessing entries of Collection of unknown type with Object type is safe

```
static void printCollection(Collection<?> c) {
  for (Object o : c)
    System.out.println(o);
}

public static void main(String[] args) {
  Collection<String> cs = new Vector<String>();
  printCollection(cs); // No Compile error
  List<Integer> li = new ArrayList<Integer>(10);
  printCollection(li); // No Compile error
}
```

More on Wildcards

- You cannot access entries of Collection of unknown type other than Object type

```
static void printCollection(Collection<?> c) {
  for (String o : c) // Compile error
    System.out.println(o);
}

public static void main(String[] args) {
  Collection<String> cs = new Vector<String>();
  printCollection(cs); // No Compile error
  List<Integer> li = new ArrayList<Integer>(10);
  printCollection(li); // No Compile error
}
```

More on Wildcards

- It isn't safe to add arbitrary objects to it however, since we don't know what the element type of c stands for, we cannot add objects to it.

```
static void printCollection(Collection<?> c) {
  c.add(new Object()); // Compile time error
  c.add(new String()); // Compile time error
}

public static void main(String[] args) {
  Collection<String> cs = new Vector<String>();
  printCollection(cs); // No Compile error
  List<Integer> li = new ArrayList<Integer>(10);
  printCollection(li); // No Compile error
}
```

Bounded Wildcard

- If you want to bound the unknown type to be a subtype of another type, use Bounded Wildcard

```
static void printCollection(
           Collection<? extends Number> c) {
  for (Object o : c)
    System.out.println(o);
}

public static void main(String[] args) {
  Collection<String> cs = new Vector<String>();
  printCollection(cs); // Compile error
  List<Integer> li = new ArrayList<Integer>(10);
  printCollection(li); // No Compile error
}
```

Generics:
Raw Type & Type Erasure

Raw Type

- Generic type instantiated with no type arguments
- Pre-J2SE 5.0 classes continue to function over J2SE 5.0 JVM as raw type

```
// Generic type instantiated with type argument
List<String> ls = new LinkedList<String>();

// Generic type instantiated with no type
// argument – This is Raw type
List lraw = new LinkedList();
```

Type Erasure

- All generic type information is removed in the resulting byte-code after compilation
- So generic type information does not exist during runtime
- After compilation, they all share same class
  - The class that represents ArrayList<String>, ArrayList<Integer> is the same class that represents ArrayList

Type Erasure Example Code: True or False?

```
ArrayList<Integer> ai = new ArrayList<Integer>();
ArrayList<String> as = new ArrayList<String>();
Boolean b1 = (ai.getClass() == as.getClass());
System.out.println("Do ArrayList<Integer> and ArrayList<String> share same class? " + b1);
```

Type-safe Code Again

- The compiler guarantees that either:
  - the code it generates will be type-correct at run time, or
  - it will output a warning (using Raw type) at compile time – in this case, you are responsible to make sure the warning is a benign one
- What is “type-safe code” again?
  - If your code compiles without any compile errors and without warnings (or with warnings on safe operations), then you will never get a ClassCastException during runtime

Generics: Interoperability

What Happens to the following Code?

```
import java.util.LinkedList;
import java.util.List;

public class GenericsInteroperability {

    public static void main(String[] args) {

        List<String> ls = new LinkedList<String>();
        List lraw = ls;
        lraw.add(new Integer(4));
        String s = ls.iterator().next();
    }

}
```

Compilation and Running

- Compilation results in a warning message
  - GenericsInteroperability.java uses unchecked or unsafe operations.
- Running the code
  - ClassCastException

Generics: Creating Your Own Generic Class

Defining Your Own Generic Class

```
public class Pair<F, S> {
   F first;  S second;

   public Pair(F f, S s) {
      first = f;  second = s;
   }

   public void setFirst(F f){
      first = f;
   }

   public F getFirst(){
      return first;
   }

   public void setSecond(S s){
      second = s;
   }

   public S getSecond(){
      return second;
   }
}
```

Using Your Own Generic Class

```
public class MyOwnGenericClass {

    public static void main(String[] args) {

        // Create an instance of Pair <F, S> class.  Let's call it p1.
        Number n1 = new Integer(5);
        String s1 = new String("Sun");
        Pair<Number,String> p1 = new Pair<Number,String>(n1, s1);
        System.out.println("first of p1 (right after creation) = " + p1.getFirst());
        System.out.println("second of p2  (right after creation) = " + p1.getSecond());

        // Set internal variables of p1.
        p1.setFirst(new Long(6L));
        p1.setSecond(new String("rises"));
        System.out.println("first of p1(after setting values) = " + p1.getFirst());
        System.out.println("second of p1 (after setting values) = " + p1.getSecond());
    }

}
```

Annotation

Sub-topics of Annotations

- What is and Why annotation?
- How to define and use Annotations?
- 3 different kinds of Annotations
- Meta-Annotations

How Annotation Are Used?

- Annotations are used to affect the way programs are treated by tools and libraries
- Annotations are used by tools to produce derived files
  - Tools: Compiler, IDE, Runtime tools
  - Derived files : New Java code, deployment descriptor, class files

Ad-hoc Annotation-like Examples in pre-J2SE 5.0 Platform

- Ad-hoc Annotation-like examples in pre-J2SE 5.0 platform
  - Transient
  - Serializable interface
  - @deprecated
  - javadoc comments
  - Xdoclet
- J2SE 5.0 Annotation provides a standard, general purpose, more powerful annotation scheme

Why Annotation?

- Enables “declarative programming” style
  - Less coding since tool will generate the boliler plate code from annotations in the source code
  - Easier to change
- Eliminates the need for maintaining "side files" that must be kept up to date with changes in source files
  - Information is kept in the source file
  - example) Eliminate the need of deployment descriptor

Annotation:
How do you define & use annotations?

How to “Define” Annotation Type?

- Annotation type definitions are similar to normal Java interface definitions
  - An at-sign (@) precedes the interface keyword
  - Each method declaration defines an element of the annotation type
  - Method declarations must not have any parameters or a throws clause
  - Return types are restricted to primitives, String, Class, enums, annotations, and arrays of the preceding types
  - Methods can have default values

Example: Annotation Type Definition

```
/**
 * Describes the Request-For-Enhancement(RFE) that led
 * to the presence of the annotated API element.
 */
public @interface RequestForEnhancement {
    int    id();
    String synopsis();
    String engineer() default "[unassigned]";
    String date() default "[unimplemented]";
}
```

How To “Use” Annotation

- Once an annotation type is defined, you can use it to annotate declarations
  - class, method, field declarations
- An annotation is a special kind of modifier, and can be used anywhere that other modifiers (such as public, static, or final) can be used
  - By convention, annotations precede other modifiers
  - Annotations consist of an at-sign (@) followed by an annotation type and a parenthesized list of element-value pairs

Example: Usage of Annotation

```
@RequestForEnhancement(
    id       = 2868724,
    synopsis = "Enable time-travel",
    engineer = "Mr. Peabody",
    date     = "4/1/3007"
)
public static void travelThroughTime(Date destination) {
  ... }
```

It is annotating travelThroughTime method

Annotation:
3 Types of Annotations (in terms of Sophistication)

3 Different Kinds of Annotations

- Marker annotation
- Single value annotation
- Normal annotation

Marker Annotation

- An annotation type with no elements
  - Simplest annotation
- Definition

```
/**
 * Indicates that the specification of the annotated API element
 * is preliminary and subject to change.
 */
public @interface Preliminary { }
```

- Usage – No need to have ()

```
@Preliminary
public class TimeTravel { ... }
```

Single Value Annotation

- An annotation type with a single element
  - The element should be named “value”
- Definition

```
/**
 * Associates a copyright notice with the annotated API element.
 */
public @interface Copyright {
    String value();
}
```

- Usage – can omit the element name and equals sign (=)

```
@Copyright("2002 Yoyodyne Propulsion Systems")
public class SomeClass { ... }
```

Normal Annotation

- We already have seen an example
- Definition

```
public @interface RequestForEnhancement {
    int    id();
    String synopsis();
    String engineer() default "[unassigned]";
    String date();    default "[unimplemented]";
}
```

- Usage

```
@RequestForEnhancement(
    id       = 2868724,
    synopsis = "Enable time-travel",
    engineer = "Mr. Peabody",
    date     = "4/1/3007"
)
public static void travelThroughTime(Date destination) { ... }
```

Annotation:
Meta-Annotations

@Retention Meta-Annotation

- How long annotation information is kept
- Enum RetentionPolicy
  - > SOURCE - SOURCE indicates information will be placed i n the source file but will not be available from the class files
  - > CLASS (Default)- CLASS indicates that information will be placed in the class file, but will not be available at runtime through reflection
  - > RUNTIME - RUNTIME indicates that information will be stored in the class file and made available at runtime through reflective APIs

@Target Meta-Annotation

- Restrictions on use of this annotation
- Enum ElementType
  - TYPE, FIELD, METHOD, PARAMETER, CONSTRUCTOR, LOCAL_VARIABLE, ANNOTATION_TYPE, PACKAGE

Example: Definition and Usage of an Annotation with Meta Annotation

Definition of Accessor annotation

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.CLASS)
public @interface Accessor {
  String variableName();
  String variableType() default "String";
}
```

Usage Example of the Accessor annotation

```
@Accessor(variableName = "name")
public String myVariable;
```

Reflection

- Check if MyClass is annotated with @Name annotation

```
boolean isName =
  MyClass.class.isAnnotationPresent(Name.class);
```

Reflection

- Get annotation value of the @Copyright annotation

```
String copyright = MyClass.class.getAnnotation
   (Copyright.class).value();
```

- Get annotation values of @Author annotation

```
Name author =
   MyClass.class.getAnnotation(Author.class).value()
String first = author.first();
String last = author.last();
```

Concurrency

Concurrency Utilities: JSR-166

- Enables development of simple yet powerful multi-threaded applications
  - Like Collection provides rich data structure handling capability
- Beat C performance in high-end server applications
  - Fine-grained locking, multi-read single write lock
- Provide richer set of concurrency building blocks
  - wait(), notify() and synchronized are too primitive
- Enhance scalability, performance, readability and thread safety of Java applications

Why Use Concurrency Utilities?

- Reduced programming effort
- Increased performance
- Increased reliability
  - Eliminate threading hazards such as deadlock, starvation, race conditions, or excessive context switching are eliminated
- Improved maintainability
- Increased productivity

Concurrency Utilities

- Task Scheduling Framework
- Callable's and Future's
- Synchronizers
- Concurrent Collections
- Atomic Variables
- Locks
- Nanosecond-granularity timing

Concurrency: Task Scheduling Framework

Task Scheduling Framework

- Executor/ExercuteService/Executors framework supports
  - standardizing invocation
  - scheduling
  - execution
  - control of asynchronous tasks according to a set of execution policies
- Executor is an interface
- ExecutorService extends Executor
- Executors is factory class for creating various kinds of ExercutorService implementations

Executor Interface

- Executor interface provides a way of de-coupling task submission from the execution
  - execution: mechanics of how each task will be run, including details of thread use, scheduling
- Example

```
Executor executor = getSomeKindofExecutor();
 executor.execute(new RunnableTask1());
 executor.execute(new RunnableTask2());
```

- Many Executor implementations impose some sort of limitation on how and when tasks are scheduled

Executor and ExecutorService

ExecutorService adds lifecycle management

```
public interface Executor {
  void execute(Runnable command);
}

public interface ExecutorService extends Executor {
  void shutdown();
  List<Runnable> shutdownNow();
  boolean isShutdown();
  boolean isTerminated();
  boolean awaitTermination(long timeout,
                          TimeUnit unit);

  // other convenience methods for submitting tasks
}
```

Creating ExecutorService From Executors

```
public class Executors {
  static ExecutorService
      newSingleThreadedExecutor();

  static ExecutorService
      newFixedThreadPool(int n);

  static ExecutorService
      newCachedThreadPool(int n);

  static ScheduledExecutorService
      newScheduledThreadPool(int n);

  // additional versions specifying ThreadFactory
  // additional utility methods
}
```

pre-J2SE 5.0 Code

Web Server—poor resource management

```
class WebServer {

  public static void main(String[] args) {
    ServerSocket socket = new ServerSocket(80);

    while (true) {
      final Socket connection = socket.accept();
      Runnable r = new Runnable() {
        public void run() {
          handleRequest(connection);
        }
      };
      // Don't do this!
      new Thread(r).start();
    }
  }
}
```

Executors Example

Web Server—better resource management

```
class WebServer {
  Executor pool =
    Executors.newFixedThreadPool(7);

  public static void main(String[] args) {
    ServerSocket socket = new ServerSocket(80);

    while (true) {
      final Socket connection = socket.accept();
      Runnable r = new Runnable() {
        public void run() {
          handleRequest(connection);
        }
      };
      pool.execute(r);
    }
  }
}
```

Concurrency:
Callables and Futures

Callable's and Future's: Problem (pre-J2SE 5.0)

- If a new thread (callable thread) is started in an application, there is currently no way to return a result from that thread to the thread (calling thread) that started it without the use of a shared variable and appropriate synchronization
  - This is complex and makes code harder to understand and maintain

Callables and Futures

- Callable thread (Callee) implements Callable interface
  - Implement call() method rather than run()
- Calling thread (Caller) submits Callable object to Executor and then moves on
  - Through submit() not execute()
  - The submit() returns a Future object
- Calling thread (Caller) then retrieves the result using get() method of Future object
  - If result is ready, it is returned
  - If result is not ready, calling thread will block

Build CallableExample (This is Callee)

```
class CallableExample
      implements Callable<String> {

  public String call() {
    String result = “The work is ended”;

    /*  Do some work and create a result  */

    return result;
  }
}
```

Future Example (Caller)

```
ExecutorService es =
  Executors.newSingleThreadExecutor();

Future<String> f =
  es.submit(new CallableExample());

/*  Do some work in parallel  */

try {
  String callableResult = f.get();
} catch (InterruptedException ie) {
  /*  Handle  */
} catch (ExecutionException ee) {
  /*  Handle  */
}
```

Concurrency:
Synchronizers: Semaphore

Semaphores

- Typically used to restrict access to fixed size pool of resources
- New Semaphore object is created with same count as number of resources
- Thread trying to access resource calls **`aquire()`**
  - Returns immediately if semaphore count > 0
  - Blocks if count is zero until **`release()`** is called by different thread
  - **`aquire()`** and **`release()`** are thread safe atomic operations

Semaphore Example

```
private Semaphore available;
private Resource[] resources;
private boolean[] used;

public Resource(int poolSize) {
  available = new Semaphore(poolSize);
  /*  Initialise resource pool  */
}
public Resource getResource() {
  try { available.aquire() } catch (IE) {}
  /*  Acquire resource  */
}
public void returnResource(Resource r) {
  /*  Return resource to pool  */
  available.release();
}
```

Concurrency:
Concurrent Collections

BlockingQueue Interface

- Provides thread safe way for multiple threads to manipulate collection
- ArrayBlockingQueue is simplest concrete implementation
- Full set of methods
  - `put()`
  - `offer()` [non-blocking]
  - `peek()`[non-blocking]
  - `take()`
  - `poll()` [non-blocking and fixed time blocking]

Blocking Queue Example: Logger placing log messages

```
private ArrayBlockingQueue messageQueue =
  new ArrayBlockingQueue<String>(10);

Logger logger = new Logger(messageQueue);

public void run() {
  String someMessage;
  try {
    while (true) {
      /*  Do some processing  */

      /*  Blocks if no space available  */
      messageQueue.put(someMessage);
    }
  } catch (InterruptedException ie) { }
}
```

Blocking Queue Example: Log Reader reading log messages

```
private BlockingQueue<String> msgQueue;

public LogReader(BlockingQueue<String> mq){
  msgQueue = mq;
}

public void run() {
  try {
    while (true) {
      String message = msgQueue.take();
      /*  Log message  */
    }
  } catch (InterruptedException ie) {
    /*  Handle  */
  }
}
```

Concurrency:
Atomic Variables

Atomics

- java.util.concurrent.atomic
  - Small toolkit of classes that support lock-free thread-safe programming on single variables

```
AtomicInteger balance = new AtomicInteger(0);

public int deposit(integer amount) {
  return balance.addAndGet(amount);
}
```

Concurrency:
Locks

Locks

- Lock interface
  - More extensive locking operations than synchronized block
  - Caution: No automatic unlocking like synchronized block – use try/finally to unlock
  - Advantage: Non-blocking access is possible using `tryLock()`
- ReentrantLock
  - Concrete implementation of Lock
  - Holding thread can call `lock()` multiple times and not block
  - Useful for recursive code

ReadWriteLock

- Has two locks controlling read and write access
  - Multiple threads can acquire the read lock if no threads have a write lock
  - If a thread has a read lock, others can acquire read lock but nobody can acquire write lock
  - If a thread has a write lock, nobody can have read/write lock
  - Methods to access locks

```
rwl.readLock().lock();
rwl.writeLock().lock();
```

ReadWrite Lock Example

```
class ReadWriteMap {
    final Map<String, Data> m = new TreeMap<String, Data>();
    final ReentrantReadWriteLock rwl =
                         new ReentrantReadWriteLock();
    final Lock r = rwl.readLock();
    final Lock w = rwl.writeLock();
    public Data get(String key) {
        r.lock();
        try { return m.get(key) }
         finally { r.unlock(); }
    }
    public Data put(String key, Data value) {
        w.lock();
        try { return m.put(key, value); }
        finally { w.unlock(); }
    }
    public void clear() {
        w.lock();
        try { m.clear(); }
        finally { w.unlock(); }
    }
}
```

JMX (Java Management Extension)

JMX Introduction

- Overview of JMX
- Instrument you Application
- Accessing your instrumentation remotely
- What's coming in JDK 6

What is JMX?

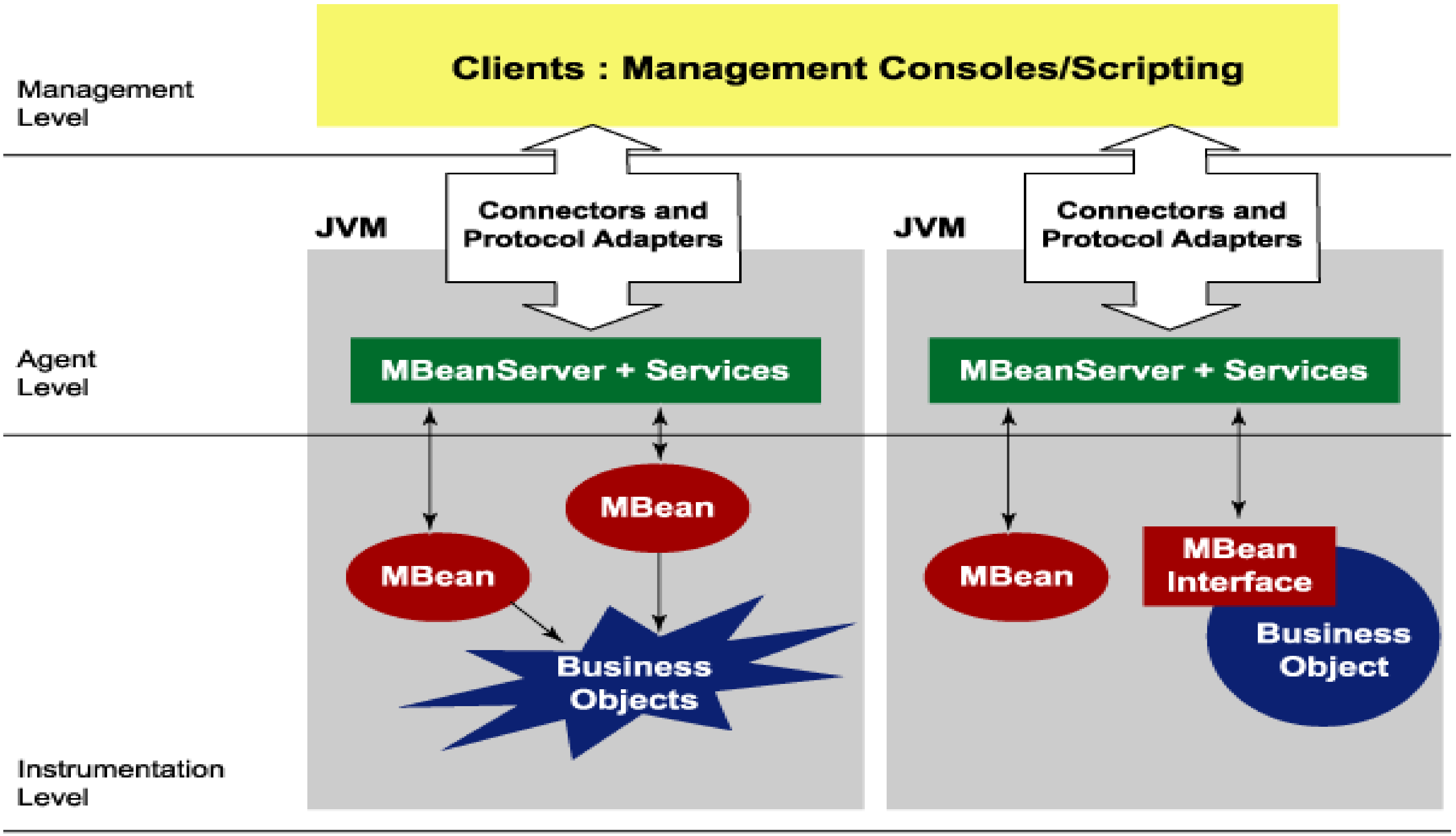
- Standard API for developing observable applications – JSR 3 and JSR 160
- Provides access to information such as
  - > Number of classes loaded
  - > Virtual machine uptime
  - > Operating system information
- Applications can use JMX for
  - > Management – changing configuration settings
  - > Monitoring – getting statistics and notifications
- Mandatory in J2SE 5.0 and J2EE 1.4

JMX:
Architecture

JMX Architecture

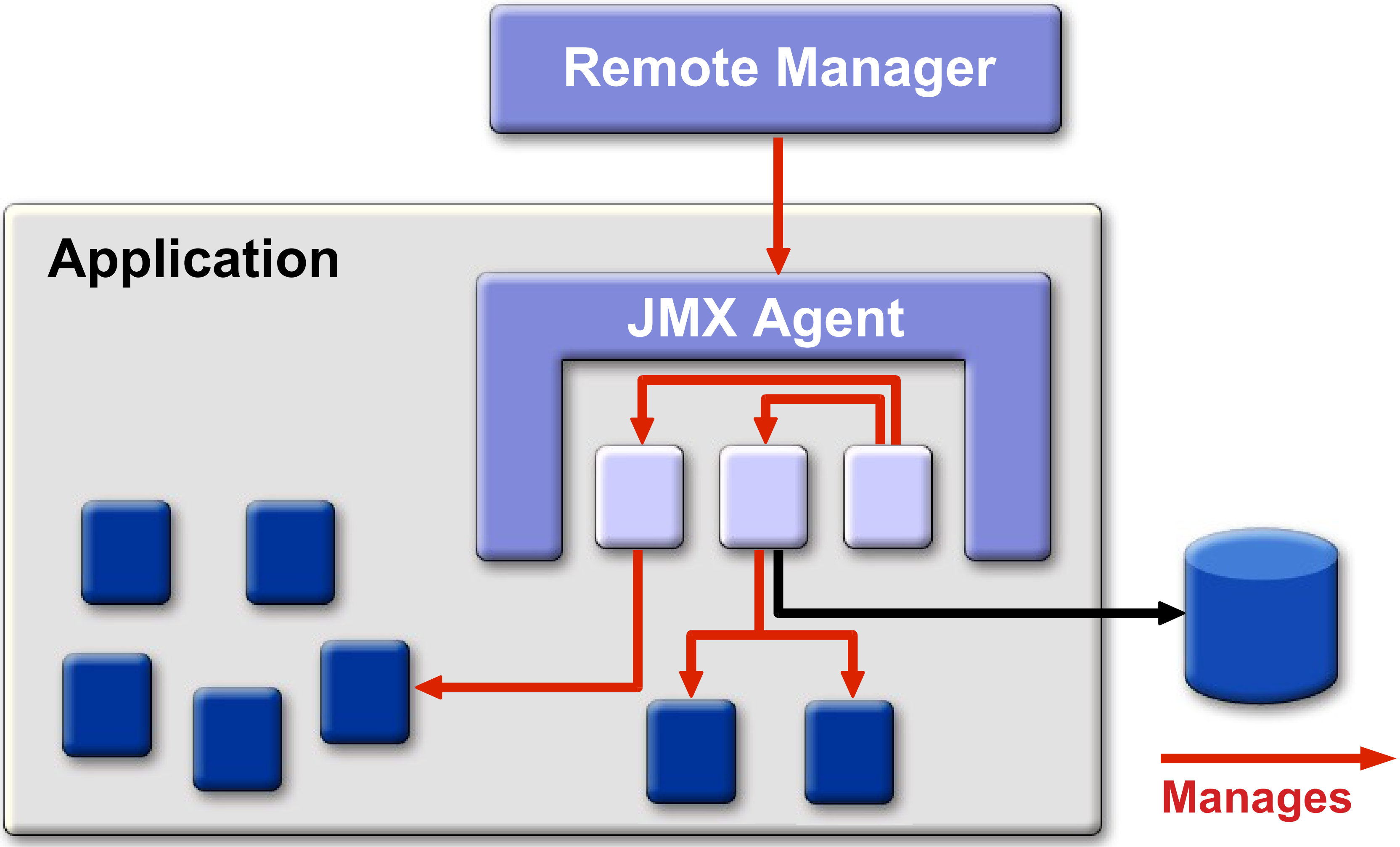
- Instrumentation Level
  - > MBeans instruments resources, exposing attributes and operations
- Agent Level
  - > MBean Server
  - > Predefined services
- Remote Management
  - > Protocol Adaptors and Standard Connectors enables remote Manager Applications

JMX Architecture



Courtsey: Borislav Iordanov

JMX Architecture

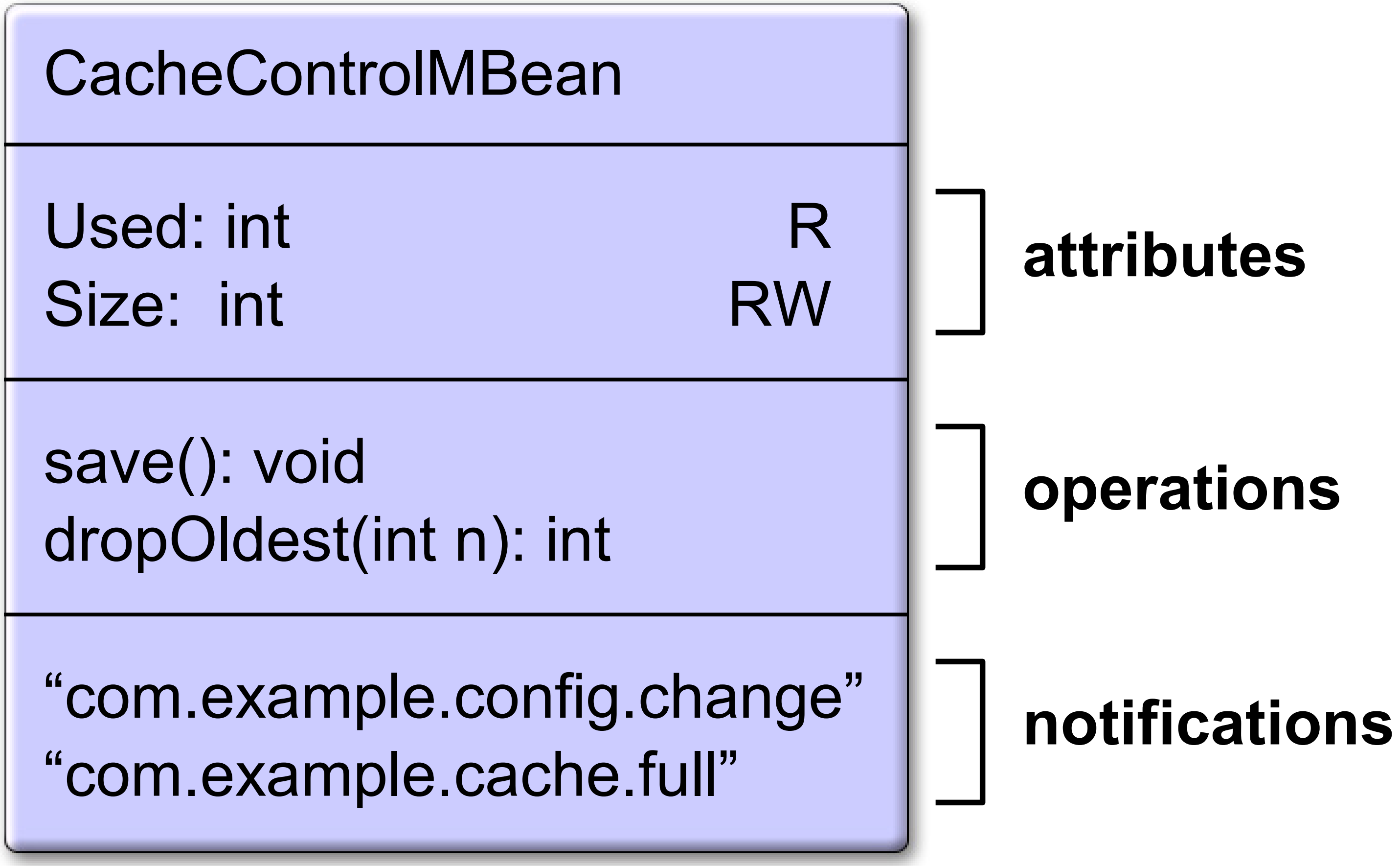


JMX:
MBean

Managed Beans(MBeans)

- A MBean is a named *managed object* representing a *resource*
  - An application configuration setting
  - Device
  - Etc.
- A MBean can have
  - Attributes that can be read and/or written
  - Operations that can be invoked
  - Notifications that the MBean can broadcast

A MBean Example



CacheControlMBean

Used: int R
Size: int RW

save(): void
dropOldest(int n): int

“com.example.config.change”
“com.example.cache.full”

attributes

operations

notifications

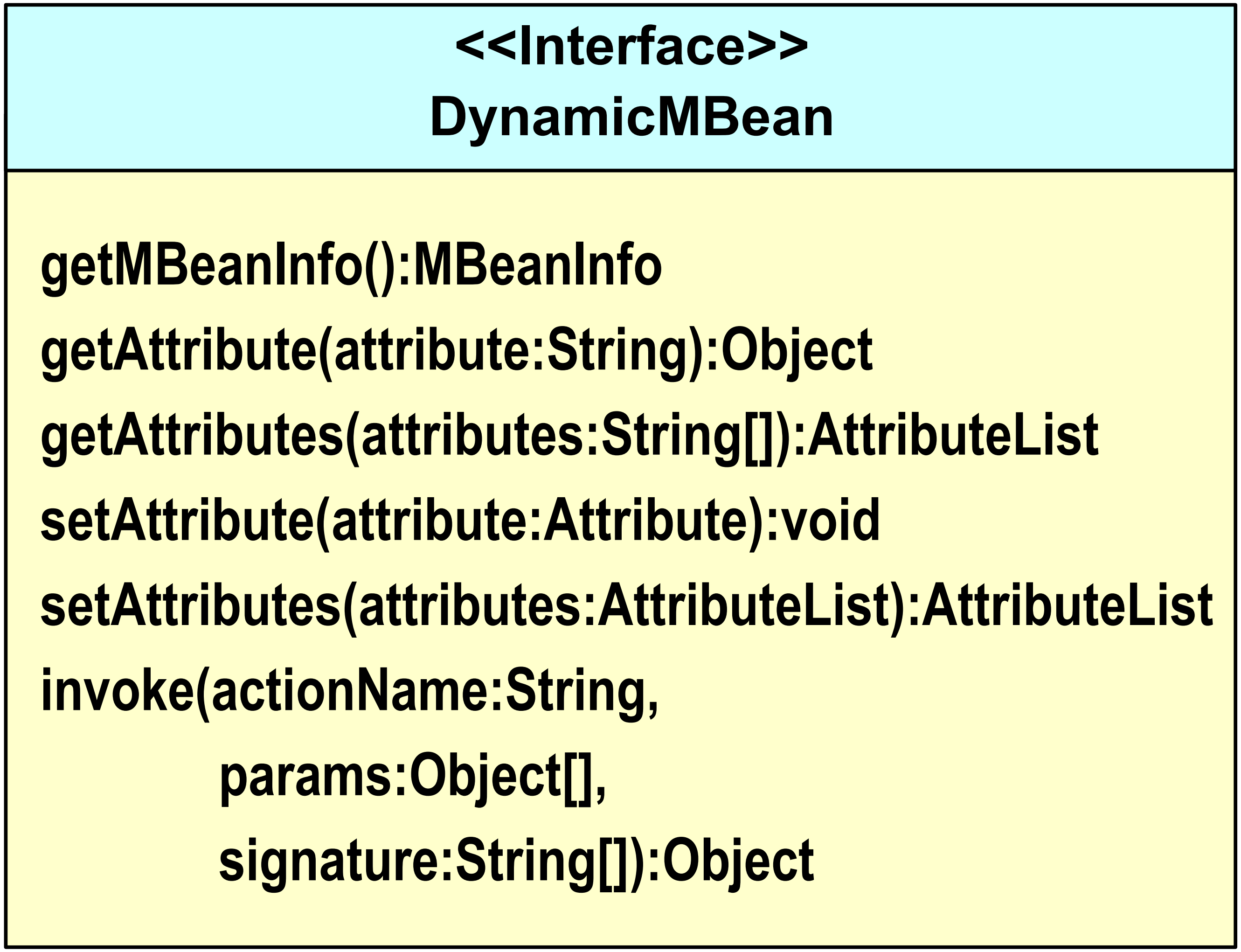
Standard MBean

- Standard MBean is the simplest model to use
  - Quickest and Easiest way to instrument static manageable resources
- Steps to create a standard MBean
  - Create an Java interface call FredMBean
  - Follows JavaBeans naming convention
  - Implement the interface in a class call Fred
- An instance of Fred is the MBean

Dynamic MBean

- Expose attributes and operations at Runtime
- Provides more flexible instrumentations
- Step to create Dynamic MBeans
  - > Implements DynamicMBeans interface
  - > Method returns all Attributes & Operations
- The same capability as Standard MBeans from Agent's perspective

DynamicMBean Interface



<<Interface>>
DynamicMBean

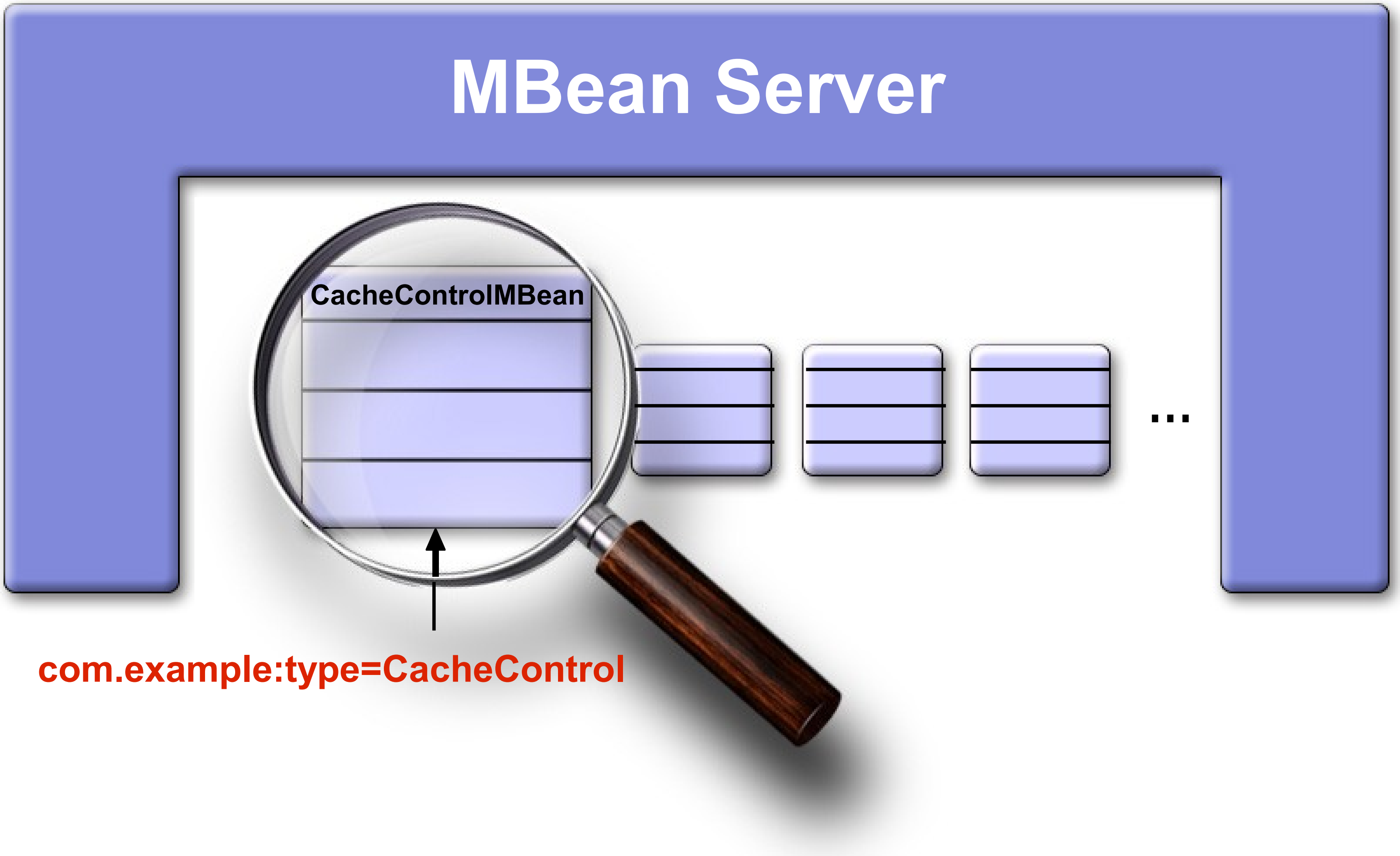
```
getMBeanInfo():MBeanInfo
getAttribute(attribute:String):Object
getAttributes(attributes:String[]):AttributeList
setAttribute(attribute:Attribute):void
setAttributes(attributes:AttributeList):AttributeList
invoke(actionName:String,
          params:Object[],
          signature:String[]):Object
```

JMX Notification

- JMX notifications consists of the following
  - **`NotificationEmitter`** – event generator, typically your MBean
  - **`NotificationListener`** – event listener
  - **`Notification`** – the event
  - **`NotificationBroadcasterSupport`** – helper class
- Register with MBean server to receive events

JMX:
MBean Server

MBean Server

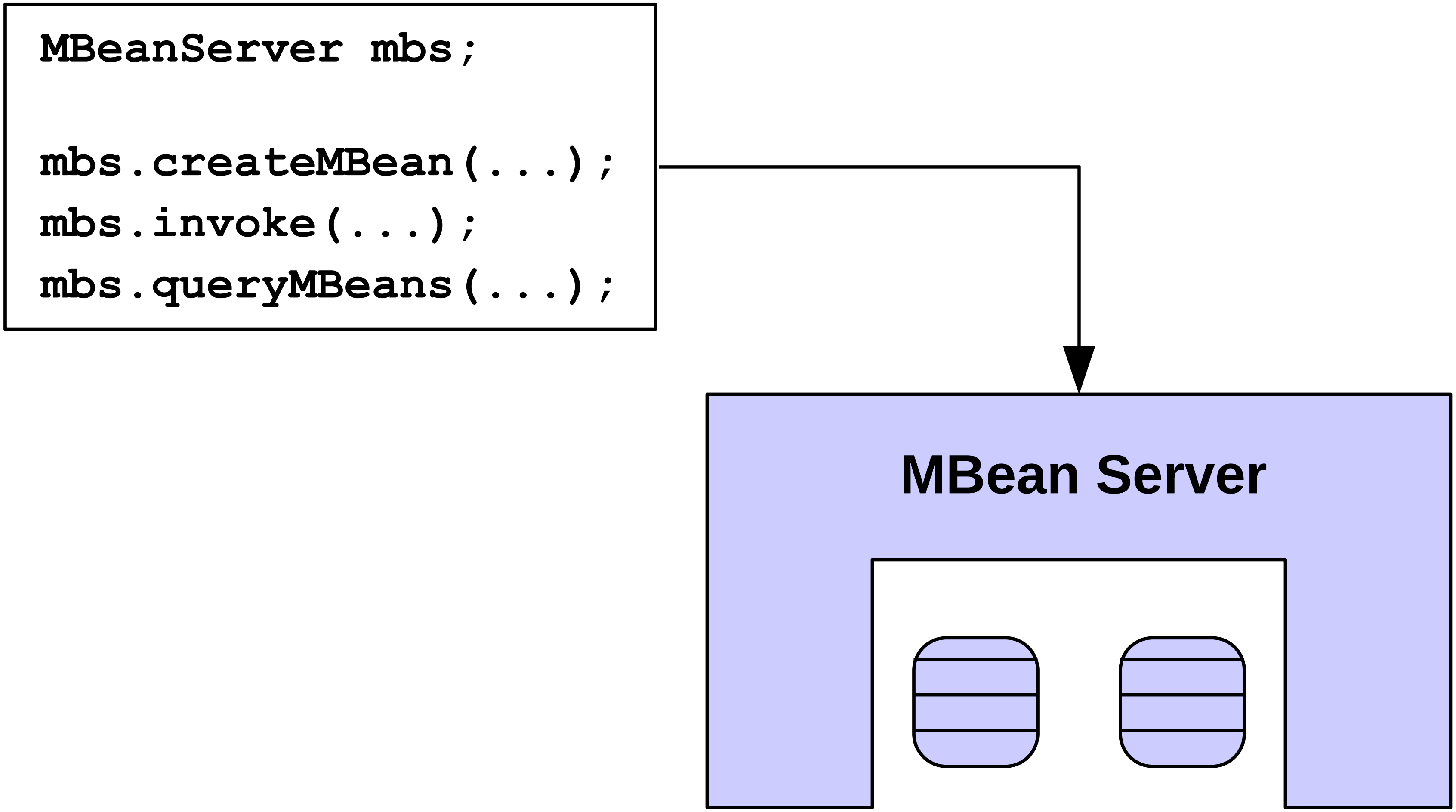


MBean Server

- To be useful, an MBean must be registered in an MBean Server
- Usually, the only access to MBeans is through the MBean Server
- You can have more than one MBean Server per Java™ Virtual Machine (JVM™ machine)
- But usually, as of Java SE 5, everyone uses the Platform MBean Server
  - > java.lang.management.ManagementFactory. getPlatformMBeanServer()

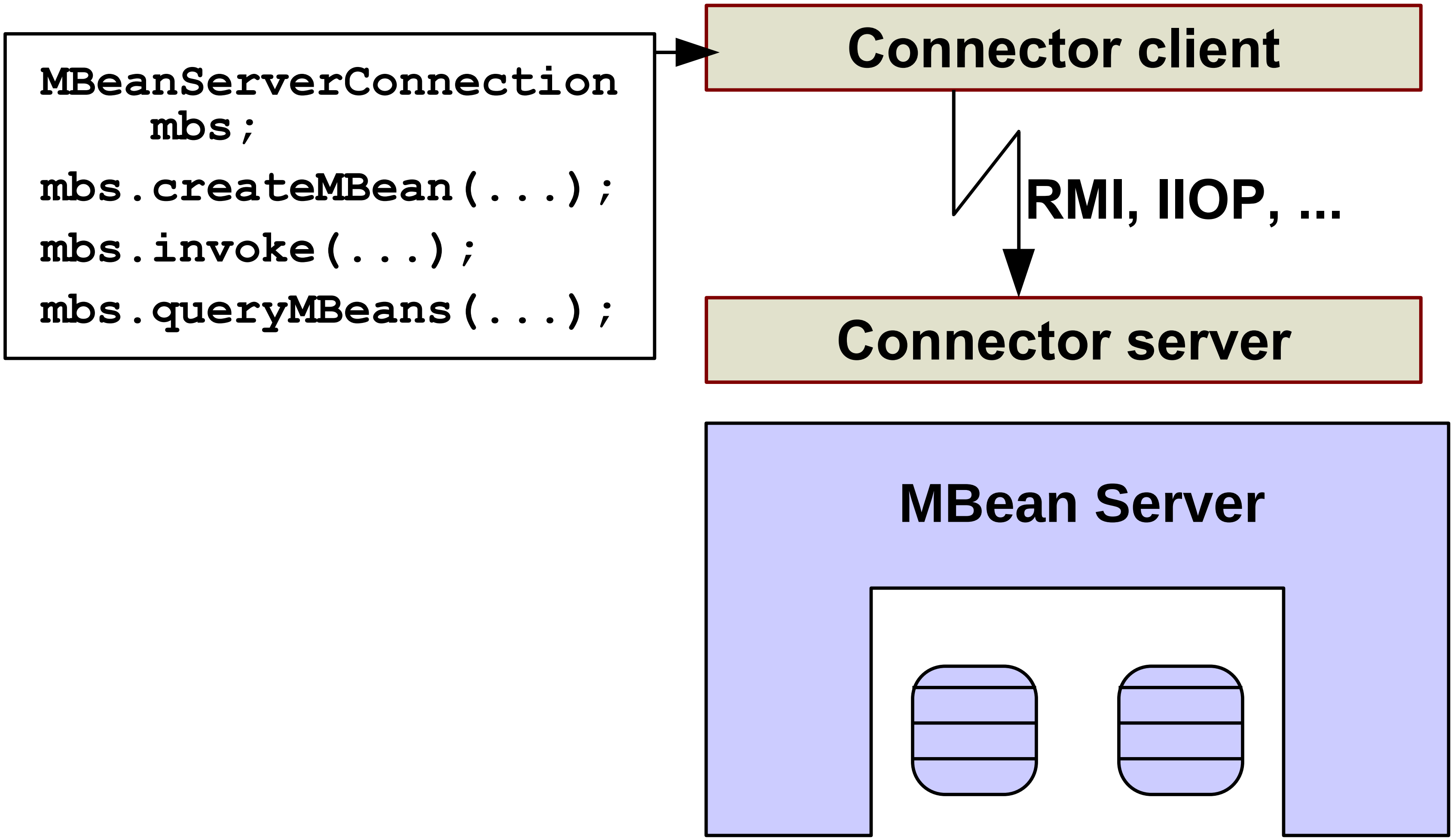
JMX:
Client Types

MBean Server: Local Clients



```
MBeanServer mbs;

mbs.createMBean(...);
mbs.invoke(...);
mbs.queryMBeans(...);
```

MBean Server: Connector Client

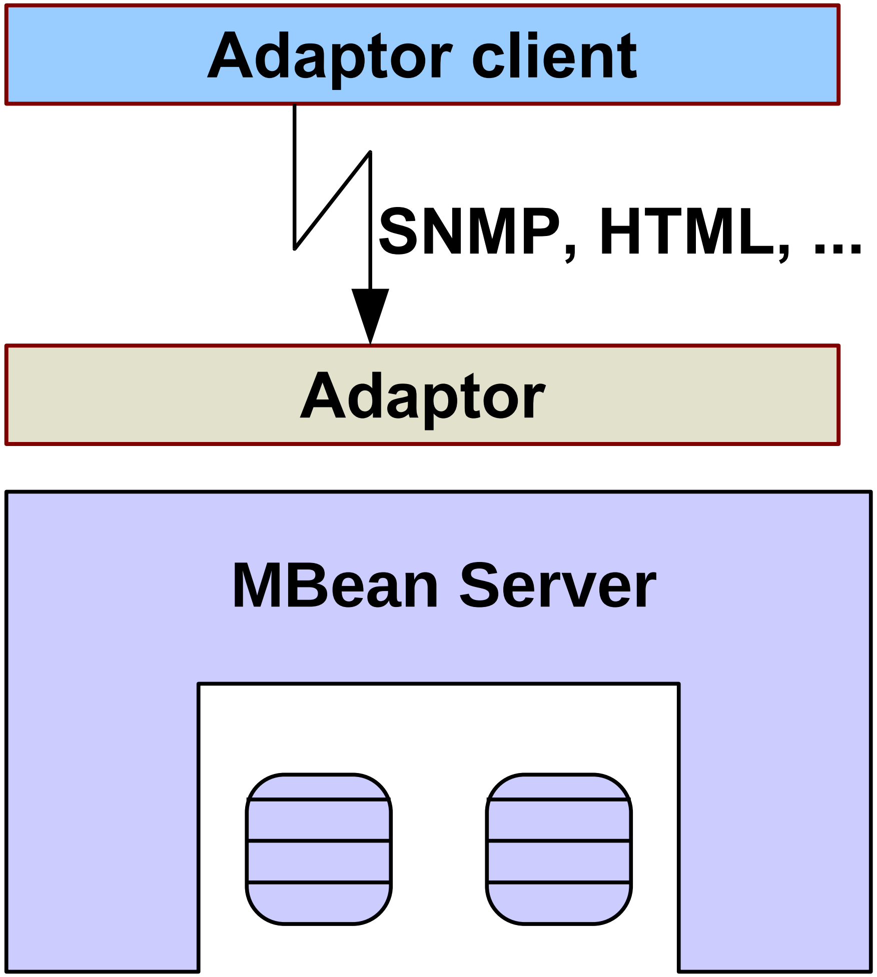


```
MBeanServerConnection
    mbs;
mbs.createMBean(...);
mbs.invoke(...);
mbs.queryMBeans(...);
```

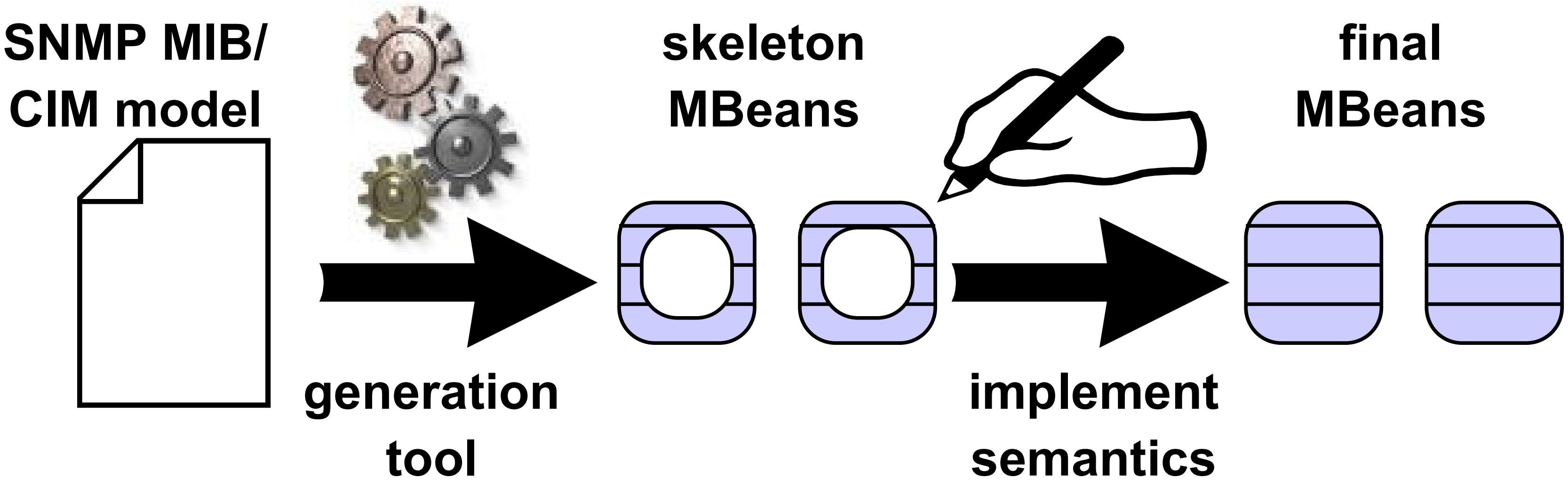
MBean Server: Connector

- Connectors defined by the JMX Remote API (JSR 160)
  - Unrelated to the J2EE™ Connector Architecture
- Java SE architecture includes RMI and RMI/IIOP connectors
- JSR 160 also defines a purpose-built protocol, JMXMP
- Future work: a SOAP-based connector for the Web Services world (JSR 262)

MBean Server: Adaptor Client



Mapping SNMP or CIM to JMX API



- Generation not currently standard
  - proprietary solutions exist (Sun's is JDMK)
- Implementing semantics may mean mapping to another, “native” JMX API model
- Automated reverse mapping from JMX API to SNMP or CIM gives poor results

JMX:
JMX API Services

JMX API Services

- JMX API includes a number of pre-defined services
  - Services are themselves MBeans
- Monitoring service (thresholding)
  - javax.management.monitor
- Relation service (relations between MBeans)
  - javax.management.relation
- Timer service
  - javax.management.timer
- M-let service
  - javax.management.loading

JMX:
Steps of instrumenting Your Application

Steps for Instrumenting Your App

- Create MBean's
  - Define an MBean interface
  - Add attributes and operations
  - Add notifications
  - Implement MBean interface
- Create JMX agent
  - Provides a method to create and register your MBeans.
  - Provides access to the MBean server
- Run the application with JConsole

JMX:
Demo – Running Anagram application with JMX support

Demo Scenario

- Anagram game is managed via JMX
  - Manage and monitor number of seconds it takes a user to provide a right answer
  - Monitor number of times a user has provided solutions
  - Subscribe event notification

Java SE Beyond Basics: Generics, Annotation, JMX

Sang Shin
sang.shin@sun.com
www.javapassion.com
Sun Microsystems Inc.

What will you open?

SUN TECH DAYS 2006-2007
A Worldwide Developer Conference

Instrument ClickCounter
(Standard MBean)

- Create ClickCounterStdMBean interface
- Create Mbean ClickCounterStd implementing ClickCounterStdMBean, and extending ClickFrame as well
- Get System MBean Server
- Register ClickCounterStd
- We’re done!

Accessing the JMX Agent

- J2SE 5.0 and later releases
  - **`java -Dcom.sun.management.jmxremote Main`**
  - See **http://java.sun.com/j2se/1.5.0/docs/guide/management/agent.html** for more options
- Start jconsole
- In JavaSE 6.0, you can attach jconsole without settng **`com.sun.management.jmxremote`** property
- One click monitoring with NetBeans IDE

Standard Mbean Interface

```
public interface ClickCounterStdMBean {
  public void reset();
  public int getDisplayNumber();
  public void setDisplayNumber(int inNumber);
  public int getCountNumber();
}
```

Implements Standard MBean

```
public class ClickCounterStd
        extends ClickFrame
        implements ClickCounterStdMBean {

    public void reset() {
            getModel().reset();
            updateLabel();
    }

    public int getDisplayNumber() {
            returngetModel().getDisplayNumber();
    }
    . . . . . .
}
```

Registering a MBean

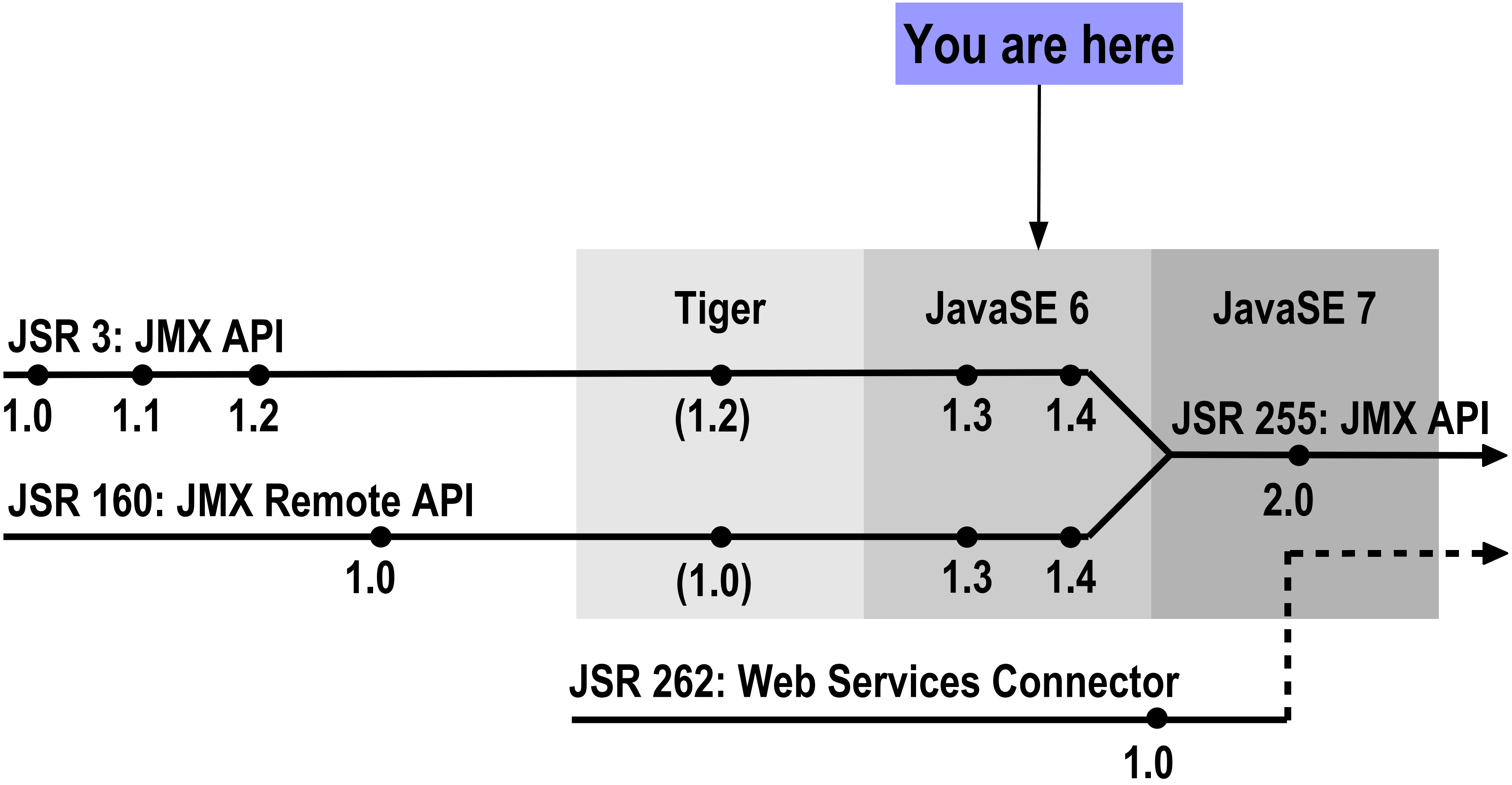
```
MBeanServer mbs = ManagementFactory
      .getPlatformMBeanServer();

ObjectName name = new ObjectName(
    “shen.joey.demo.ClickCount:type=ClickCounterStd”);


ClickFrame counter = new ClickCounterStd();

mbs.registerMBean(counter, name);
```

JMX: Roadmap

What's coming next?

- JSR 3 defined the JMX API
  - Updated in Java Platform, Standard Edition 6
- JSR 160 defined the JMX Remote API
  - Also updated in JavaSE 6.0
- JSR 255 merges and updates JSRs 3 and 160
  - It will produce JMX API version 2.0 in Java SE 7

JMX API Versions



http://jdk6.dev.java.net/

MXBeans: Problem Statement (1)

- An MBean interface can include arbitrary Java programming language types

```
public interface ThreadMBean {
  public ThreadInfo getThreadInfo();
}
public class ThreadInfo {
  public String getName();
  public long getBlockedCount();
  public boolean isSuspended();
  ...
}
```

- When values must be grouped automatically

MXBeans: Problem Statement (2)

- An MBean interface can include arbitrary Java programming language types

```
public interface ThreadMBean {
  public ThreadInfo getThreadInfo();
}
```

- Client must have these classes
- What about generic clients like jconsole?
- What about versioning?

MXBeans (1)

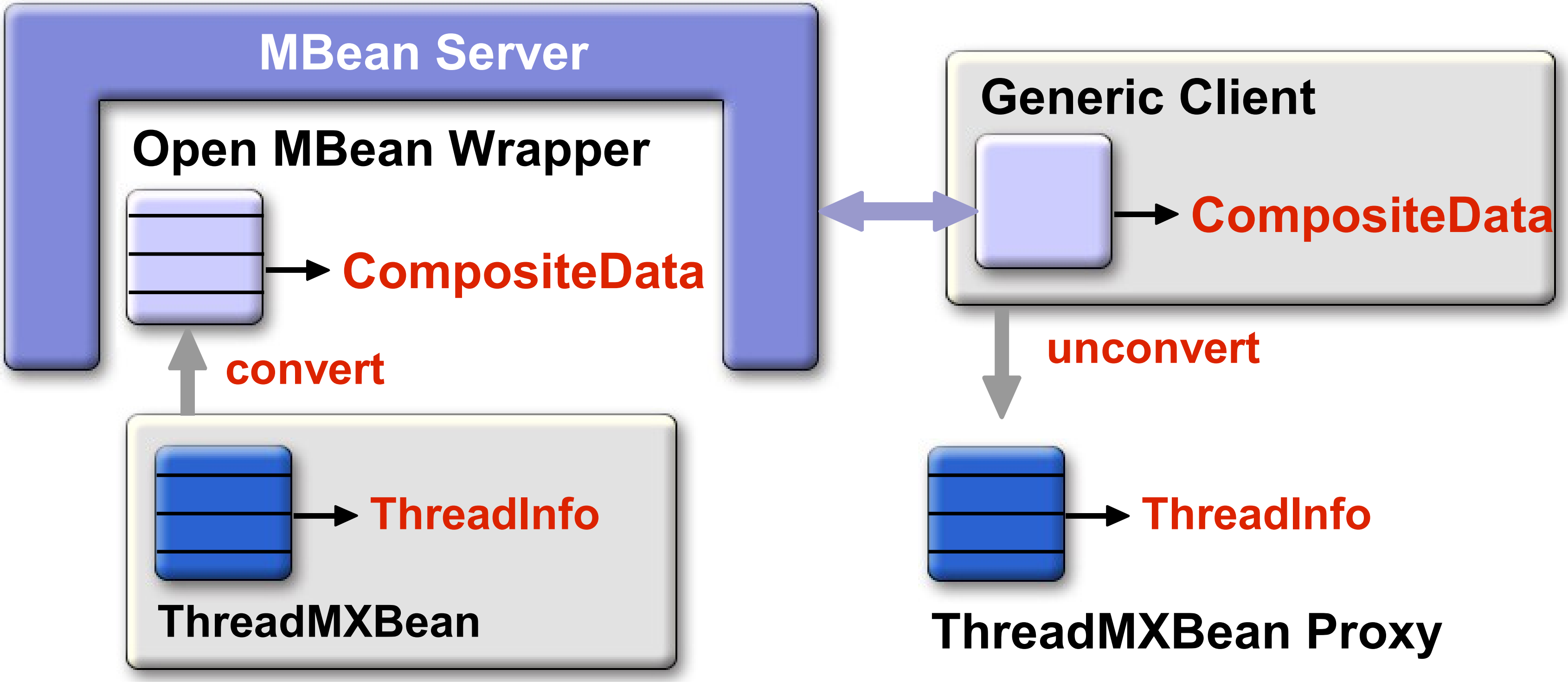
- MXBeans were designed for the instrumentation of the VM itself (JSR 174)
  - Already exist in java.lang.management
  - User-defined MXBeans are new in Mustang
- Management interface still a bean interface
- Can reference arbitrary types,
  with some restrictions

MXBeans (2)

```
public interface ThreadMXBean {
      public ThreadInfo getThreadInfo();
}
public class ThreadMXBeanImpl implements ThreadMXBean {
  // Do not need Something/SomethingMXBean naming
  public ThreadInfo getThreadInfo() {
    return new ThreadInfo(...);
  }
}
ThreadMXBean mxbean = new ThreadMXBeanImpl();
ObjectName name =
  new ObjectName("java.lang:type=Threading");

mbs.registerMBean(mxbean, name);
```

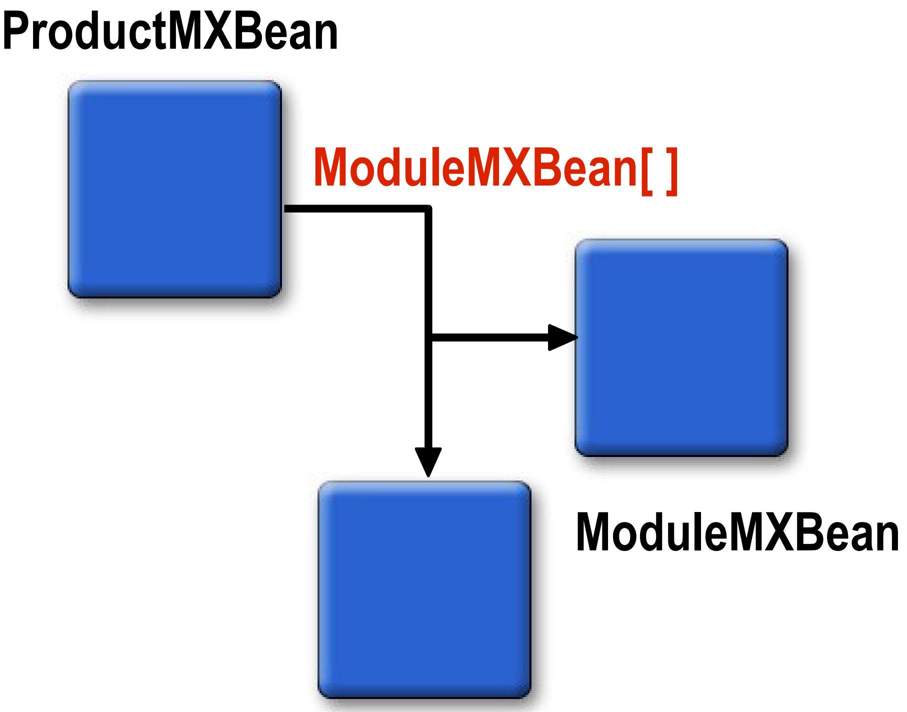
MXBeans (3)

- Generic client can access as Open MBean
- Model-aware client can make ThreadMXBean proxy

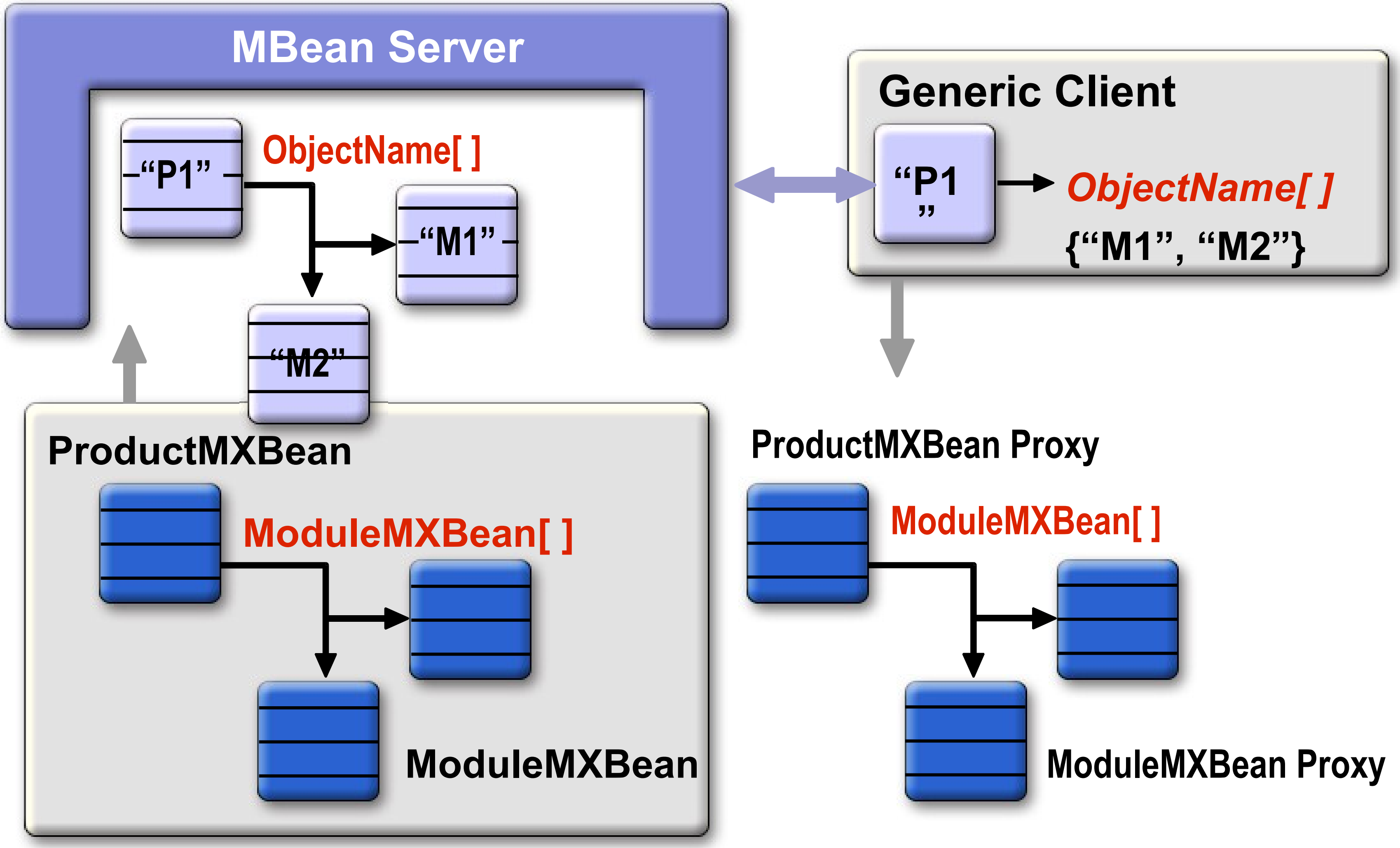


MXBean References (1)

```
public interface ProductMXBean {
        ModuleMXBean[] getModules();
}
```

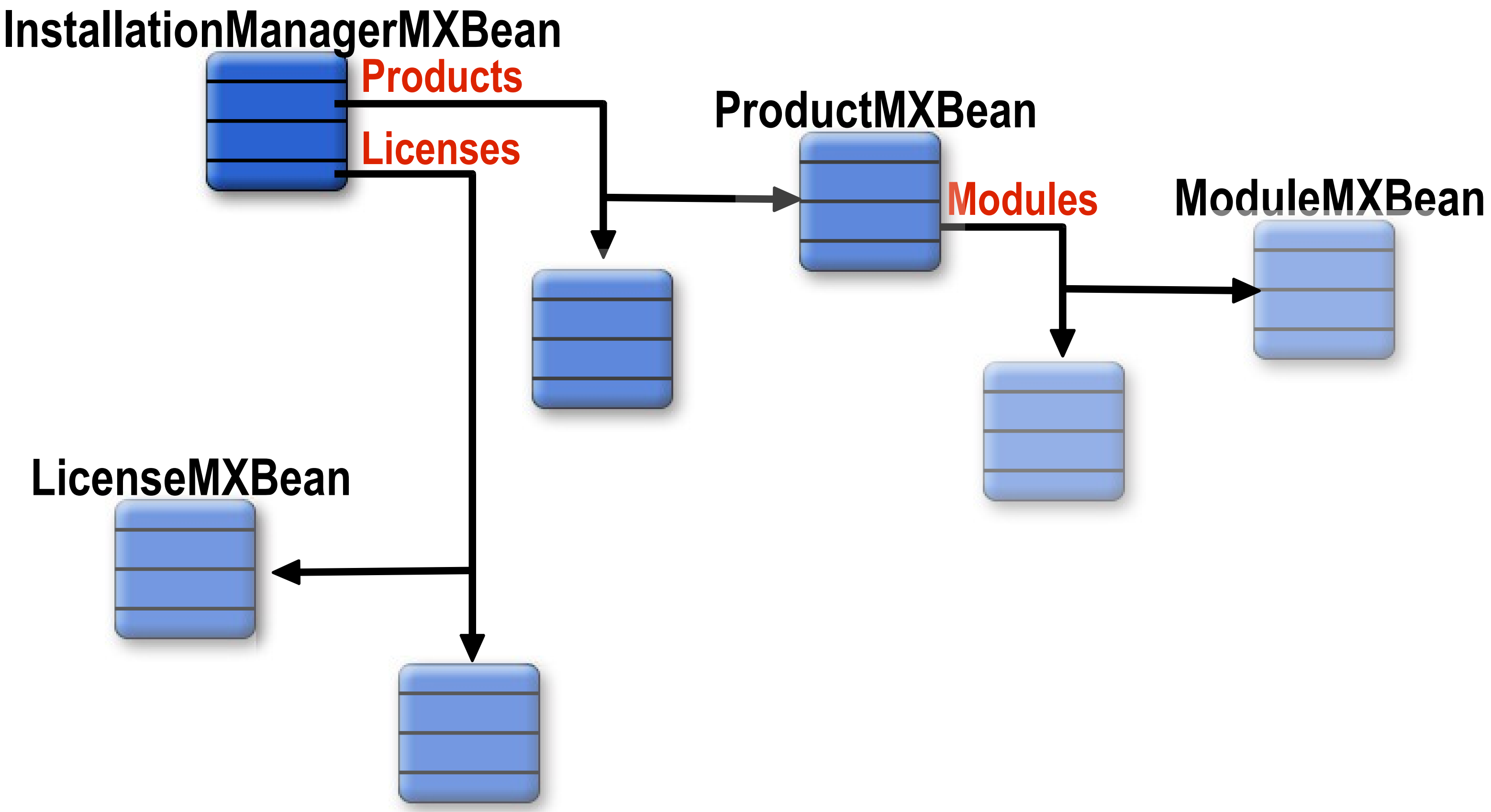


MXBean References (2)

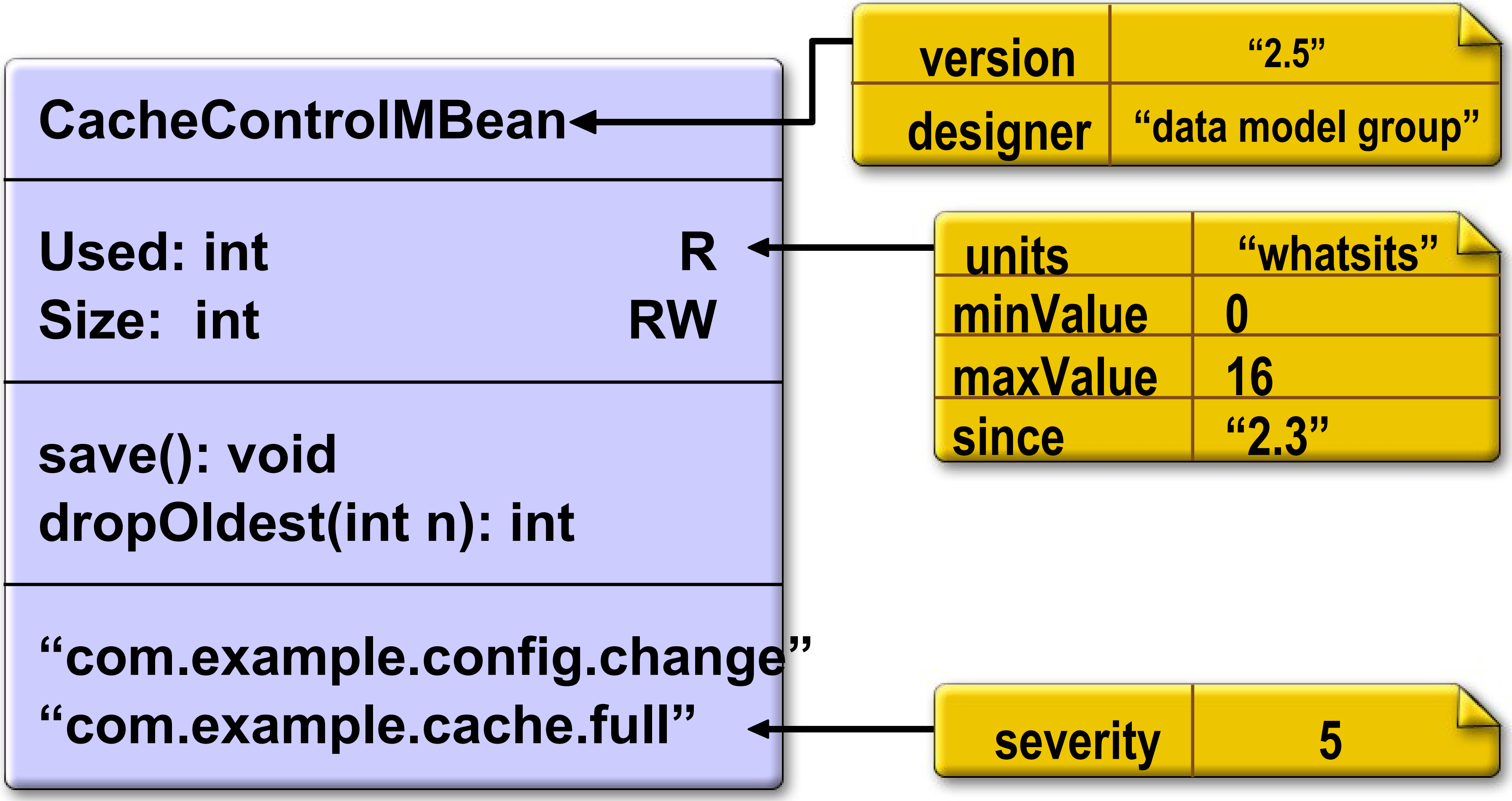


MXBean References (3)

Navigating From a Starting Point



Descriptors



CacheControlMBean

Used: int R
Size: int RW

save(): void
dropOldest(int n): int

“com.example.config.change”
“com.example.cache.full”

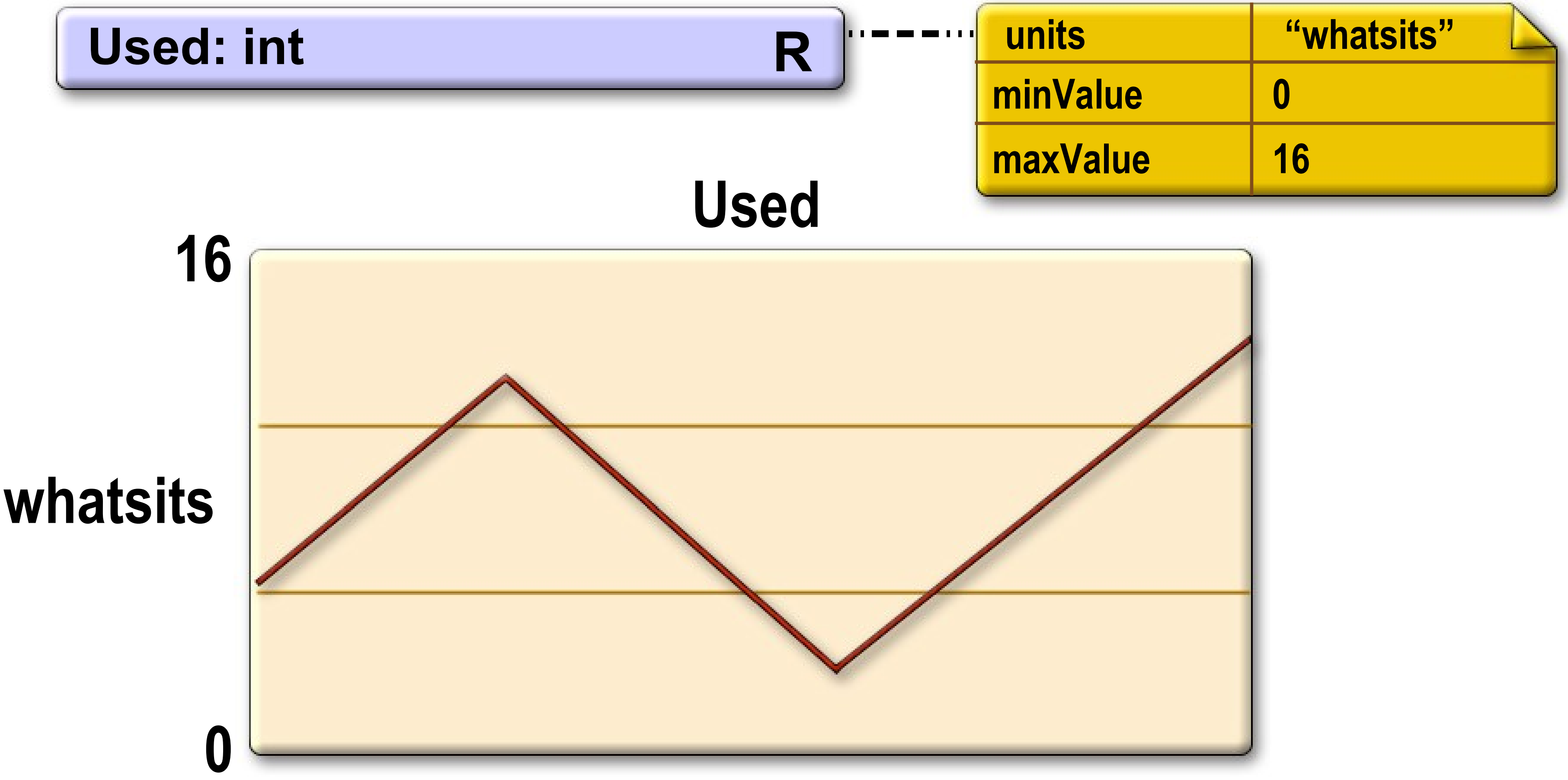
version	“2.5”
designer	“data model group”

units	“whatsits”
minValue	0
maxValue	16
since	“2.3”

severity	5

Descriptors and Generic Clients

(like jconsole)



Used: int R

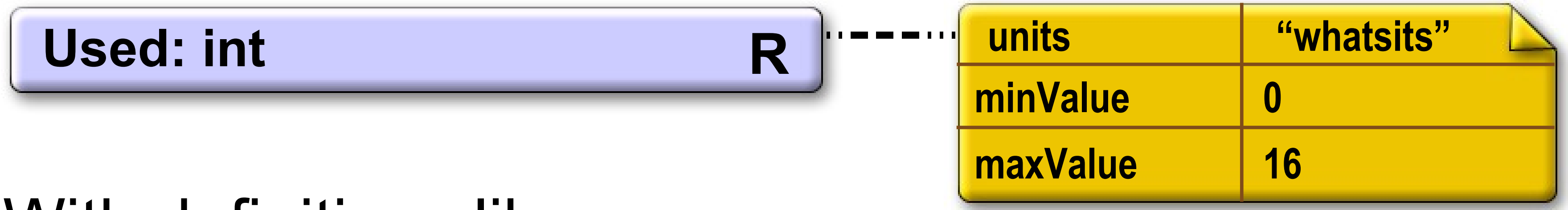
units	“whatsits”
minValue	0
maxValue	16

Descriptor Details

- Classes MBeanInfo, MBeanAttributeInfo, etc., now have an optional Descriptor
- Every attribute, operation, notification can have its own Descriptor
- Descriptor is set of (key,value) pairs
- Some keys have conventional meanings
- Users can add their own keys
- Descriptors have always existed in Model MBeans

Descriptor Annotations

```
public interface CacheControlMBean {
   @Units("whatsits") @Range(minValue=0, maxValue=16)
   public int getUsed();
}
```



Used: int R

units	"whatsits"
minValue	0
maxValue	16

With definitions like:

```
public @interface Range {
   @DescriptorKey("minValue")
   public int minValue();
   @DescriptorKey("maxValue")
   public int maxValue();
}
```

Some Other Mustang changes

- Generified at last!
  - Set<ObjectName> queryNames(...)
- More-general ObjectName wildcards
  - domain:type=Dir,path="/root/*"
- Simpler Notification use
  - NotificationBroadcasterSupport(MBeanNotificationInfo[])
  - class StandardEmitterMBean extends StandardMBean
- Monitor attributes of complex type
  - MonitorMBean.setObservedAttribute("ThreadInfo.size")

http://jdk6.dev.java.net

JMX Summary

- JMX core concept
  - > MBean, MXBean, MBeanServer
- Instrument App with JMX
  - > Create MBean
  - > Register to MBeanServer
- Coming Next
  - > MXBean
  - > Descriptor

Performance

GC Algorithms

- Reference Counting
  - It’s straightforward and easy
  - Need support of compilers
  - Circularly referenced detection

- Tracing collectors
  - Root Objects: Local Variable, Static Variable, Registers

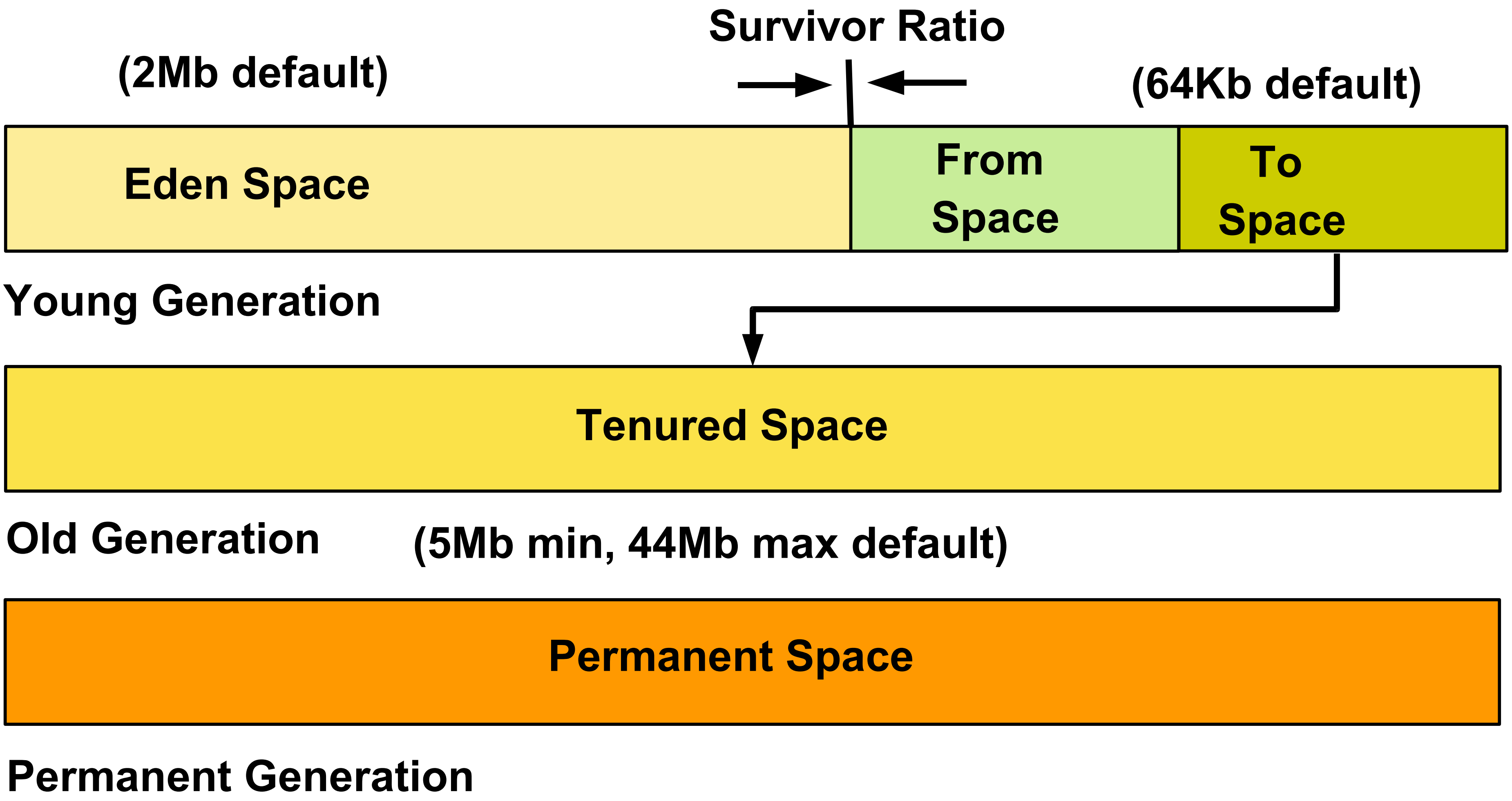
Mark-Sweep Collector

- Behavior
  - Stop the world
  - Mark all the reachable objects
  - Exam all the heap, Sweep the unreachable objects
- Advantage
  - Simple to implement
  - Dose not depend on compilers
- Limitation
  - Every allocated object is visited
  - Heap fragment

Copying Collector

- Advantage
  - > Visit only live objects
  - > Compact in the new space
  - > Greatly reduces the cost of object allocation
  - > Easy reclaim
- Limitation
  - > Larger memory footprint
  - > Overhead of copying
  - > Long live objects
  - > Adjust reference address

HotSpot Heap Layout



Survivor Ratio

(2Mb default)

(64Kb default)

Eden Space

From Space

To Space

Young Generation

Tenured Space

Old Generation

(5Mb min, 44Mb max default)

Permanent Space

Permanent Generation

General Tuning Advice

- Allocate as much memory as possible to VM (As long as pause time is not the problem)
  - > 64M default is often too small
- Set -Xms and -Xmx the same
  - > Increase predictability, improve startup time
- Set Eden/Tenured ratio
  - > Eden < 50% (Not for throughput and Concurrent collectors)
  - > NewRatio=2 seems to be good
- Disable explicit GC
  - > -XX:+DisableExplicitGC

Throughput Collector

- When there are a large number of processors
- Parallel version of the young generation collector
- -XX:+UseParallelGC to enable
- J2SE 1.5 will automatically choose throughput collector
- on Server Machines(2+ Processors, 2G+ Memory)
- -XX:MaxGCPauseMillis=<nnn>
- -XX:GCTimeRatio=<Apps time/GC time>

Concurrent Collector

- For the sack of low pause time
- Applications which have a large set of long-lived data running on more than one processor
- Parallel in Young generation collecting, Concurrent in Tenured generation collecting
- -XX:+UseConcMarkSweepGC to enable

Resources

- http://java.sun.com/j2se/1.5.0/docs/guide/jmx/tutorial/tutorialTOC.html
- http://java.sun.com/docs/performance/
- http://java.sun.com/j2se/1.5.0/docs/guide/concurrency/index.html