# J2SE 5.0 Language Features

**Sang Shin**
Java Technology Architect
Sun Microsystems, Inc.

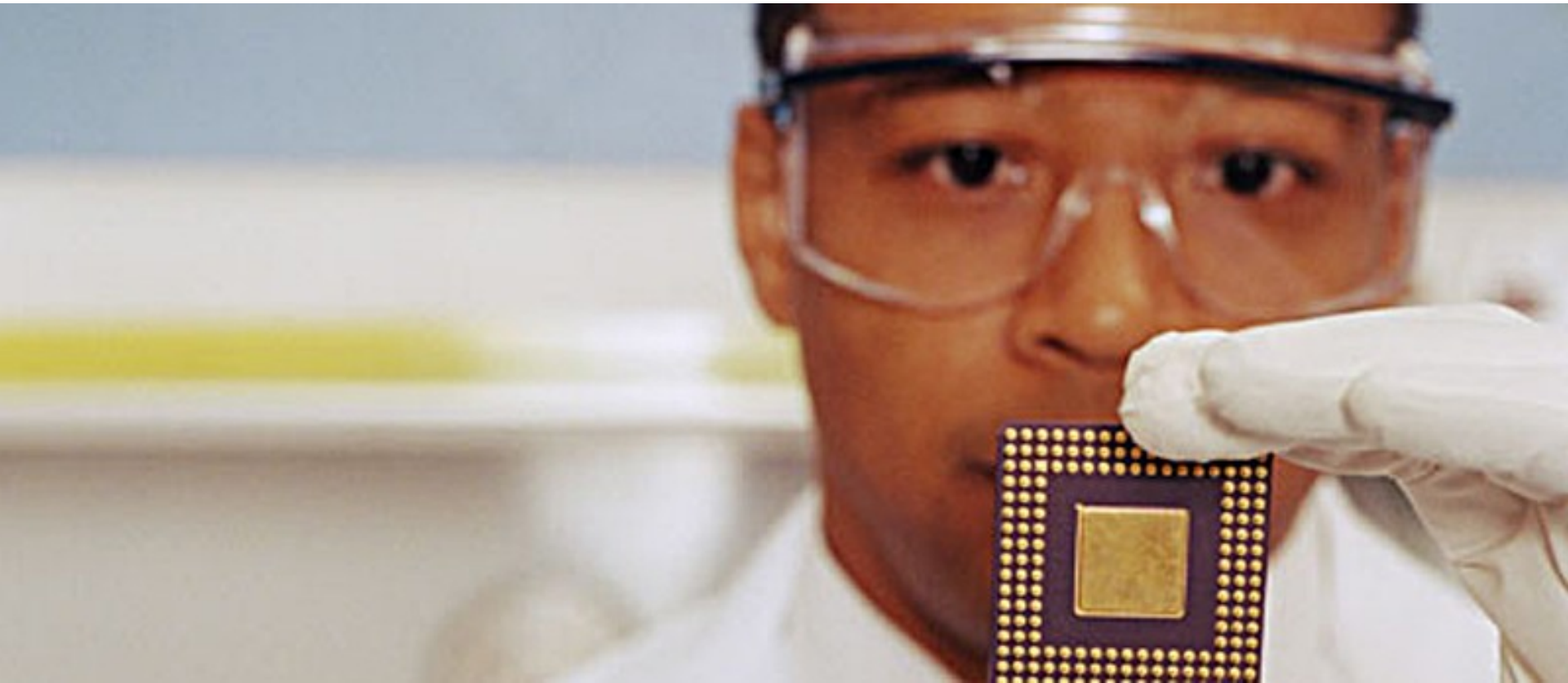
# Agenda

- J2SE 5.0 Design Themes
- Language Changes
  - Generics & Metadata
- Library API Changes
  - Concurrency utilities
- Virtual Machine
- Monitoring & Management
- Next Release: Mustang

# J2SE 5.0 Design Themes

- Focus on quality, stability, compatibility
  - Many enterprise software already run over J2SE 5.0
- Support a wide range of application styles
  - “from desktop to data center”
- Big emphasis on scalability
  - exploit big heaps, big I/O, big everything
- Continuing to deliver great new features
  - Maintaining portability and compatibility
- Ease of development
  - Faster, cheaper, more reliable

# Language Changes



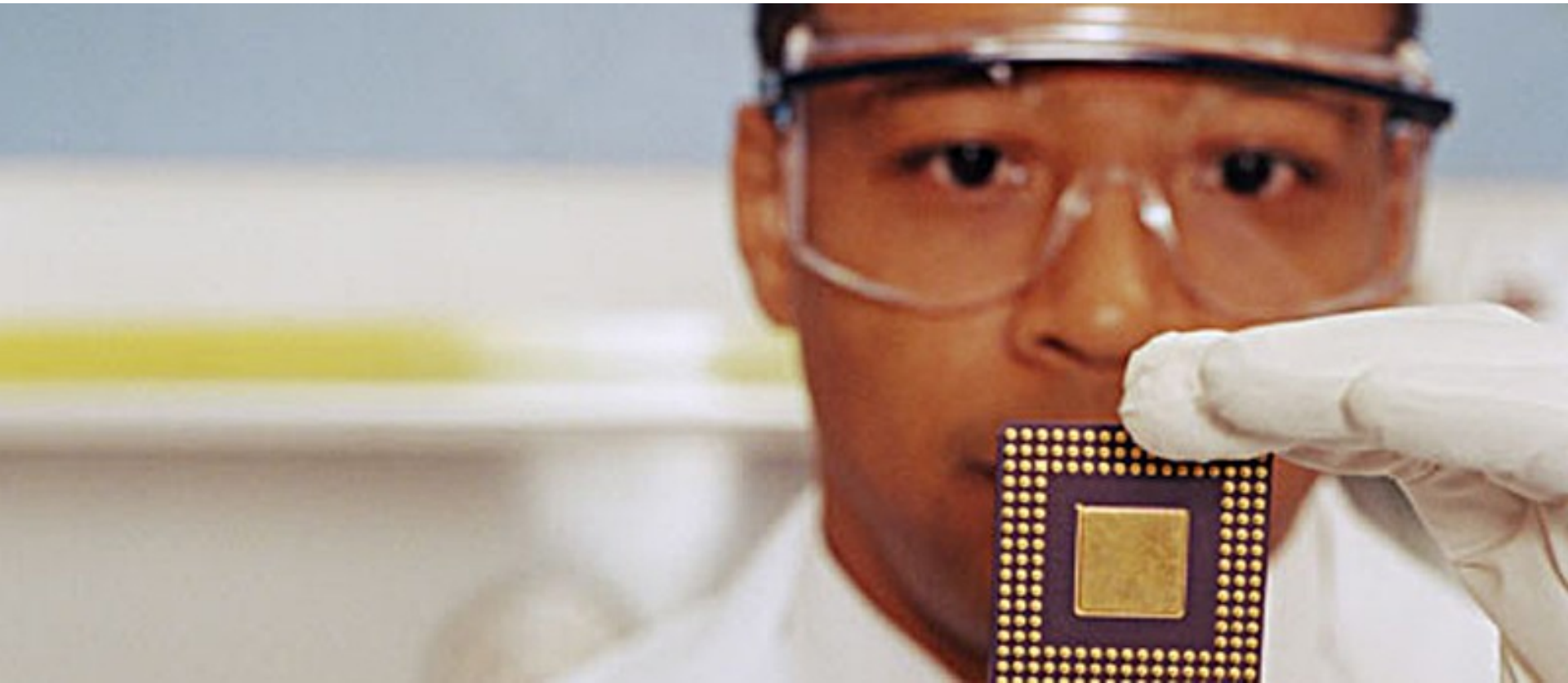
# Java Language Changes

- JDK 1.0
  - Initial language, very popular
- JDK1.1
  - Inner classes, new event model
- JDK 1.2, 1.3
  - No changes at language level
- JDK 1.4
  - Assertions (minor change)
- JDK 5.0
  - Biggest changes to language since release 1.0

# Seven Major New Features

- Generics
- Autoboxing/Unboxing
- Enhanced for loop (“foreach”)
- Type-safe enumerations
- Varargs
- Static import
- Metadata

# Autoboxing & Unboxing



# Autoboxing/Unboxing of Primitive Types

- Problem: (pre-J2SE 5.0)
  - Conversion between primitive types and wrapper types (and vice-versa)
  - You need manually convert a primitive type to a wrapper type before adding it to a collection

```
int i = 22;
List l = new LinkedList();
l.add(new Integer(i));
```

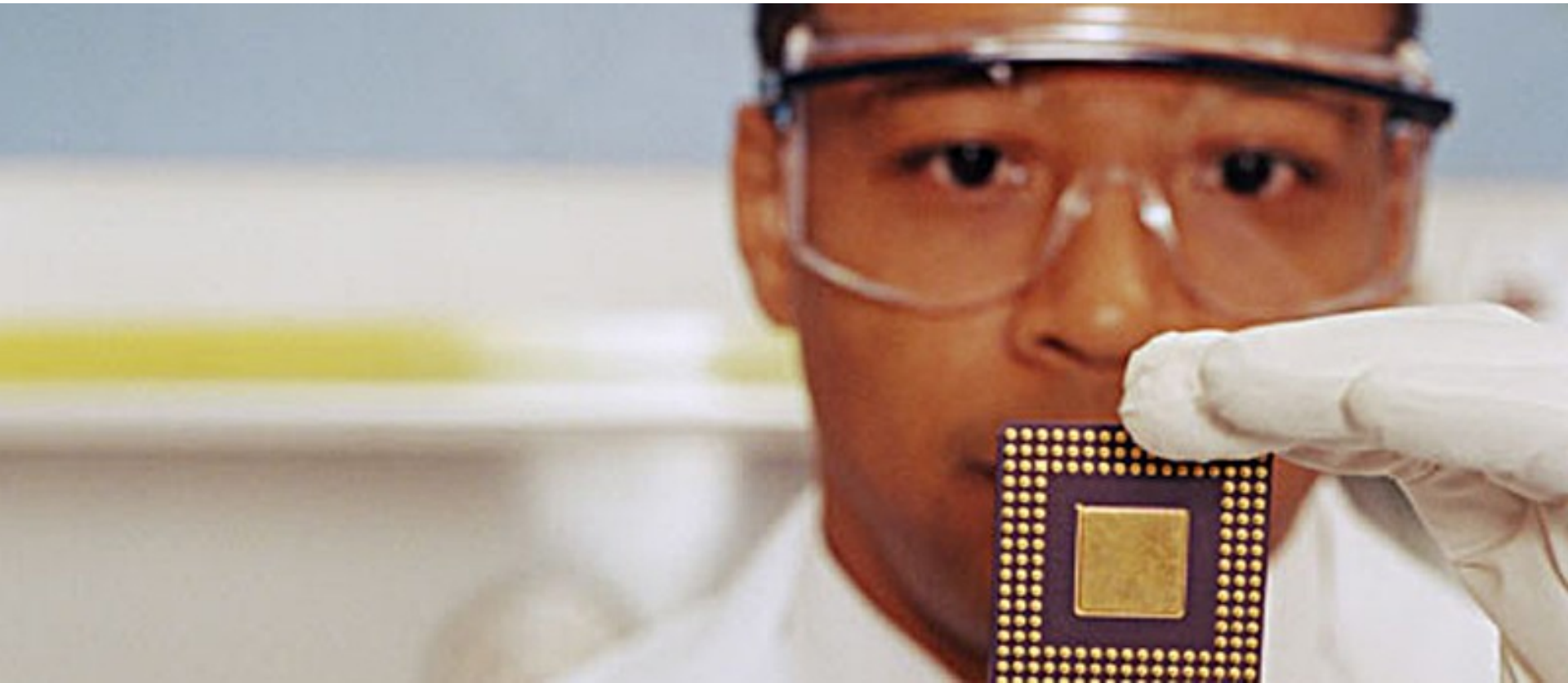
# Autoboxing/Unboxing of Primitive Types

- Solution: Let the compiler do it

```
Byte byteObj = 22;      // Autoboxing conversion
int i = byteObj         // Unboxing conversion

ArrayList<Integer> al = new ArrayList<Integer>();
al.add(22);  //  Autoboxing conversion
```

# Enhanced for Loop



# Enhanced for Loop (foreach)

- Problem: (pre-J2SE 5.0)
  - Iterating over collections is tricky
  - Often, iterator only used to get an element
  - Iterator is error prone
    (Can occur three times in a for loop)
- Solution: Let the compiler do it
  - New for loop syntax
    ```
    for (variable : collection)
    ```
  - Works for Collections and arrays

# Enhanced for Loop Example

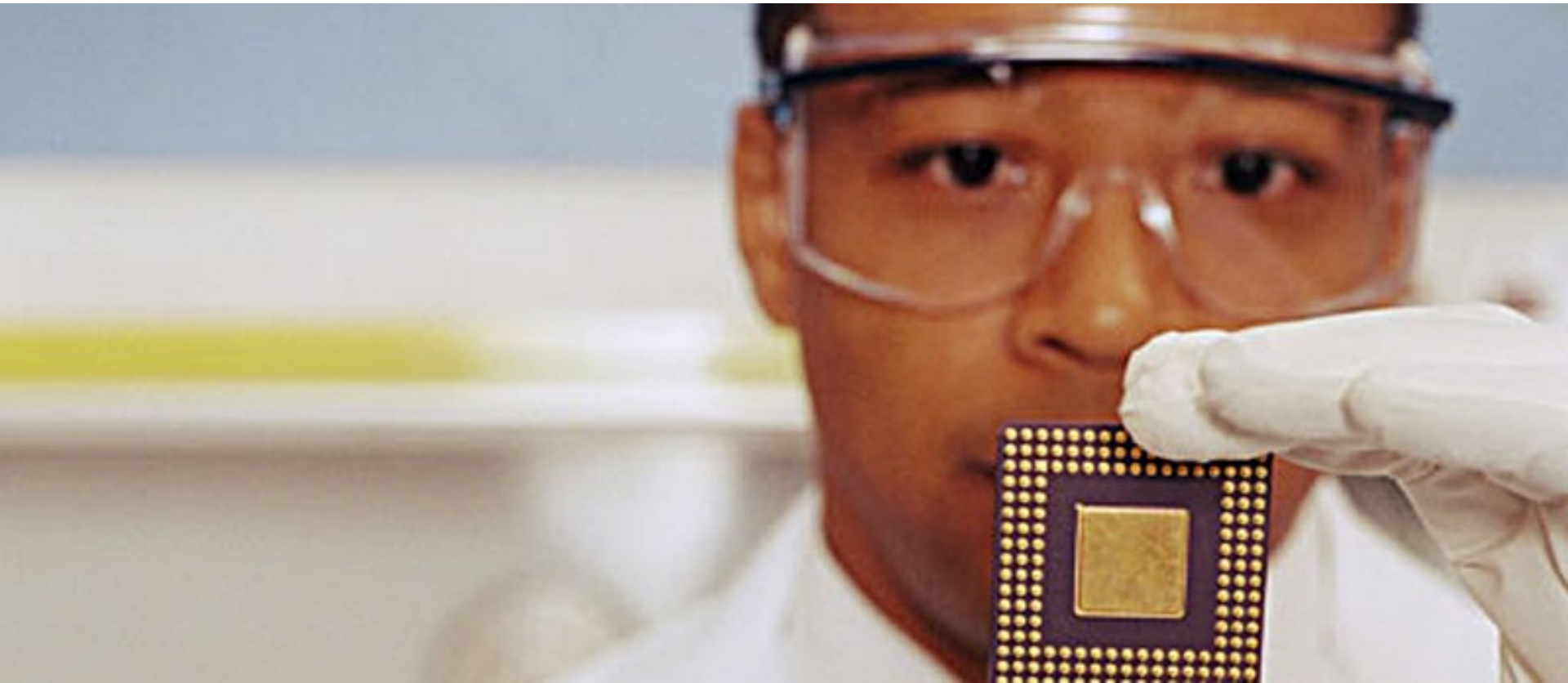
- Old code

```
void cancelAll(Collection c) {
  for (Iterator i = c.iterator(); i.hasNext(); ){
    TimerTask task = (TimerTask)i.next();
    task.cancel();
  }
}
```

- New Code

```
void cancelAll(Collection<TimerTask> c) {
  for (TimerTask task : c)
    task.cancel();
}
```

# Type-safe Enumerations



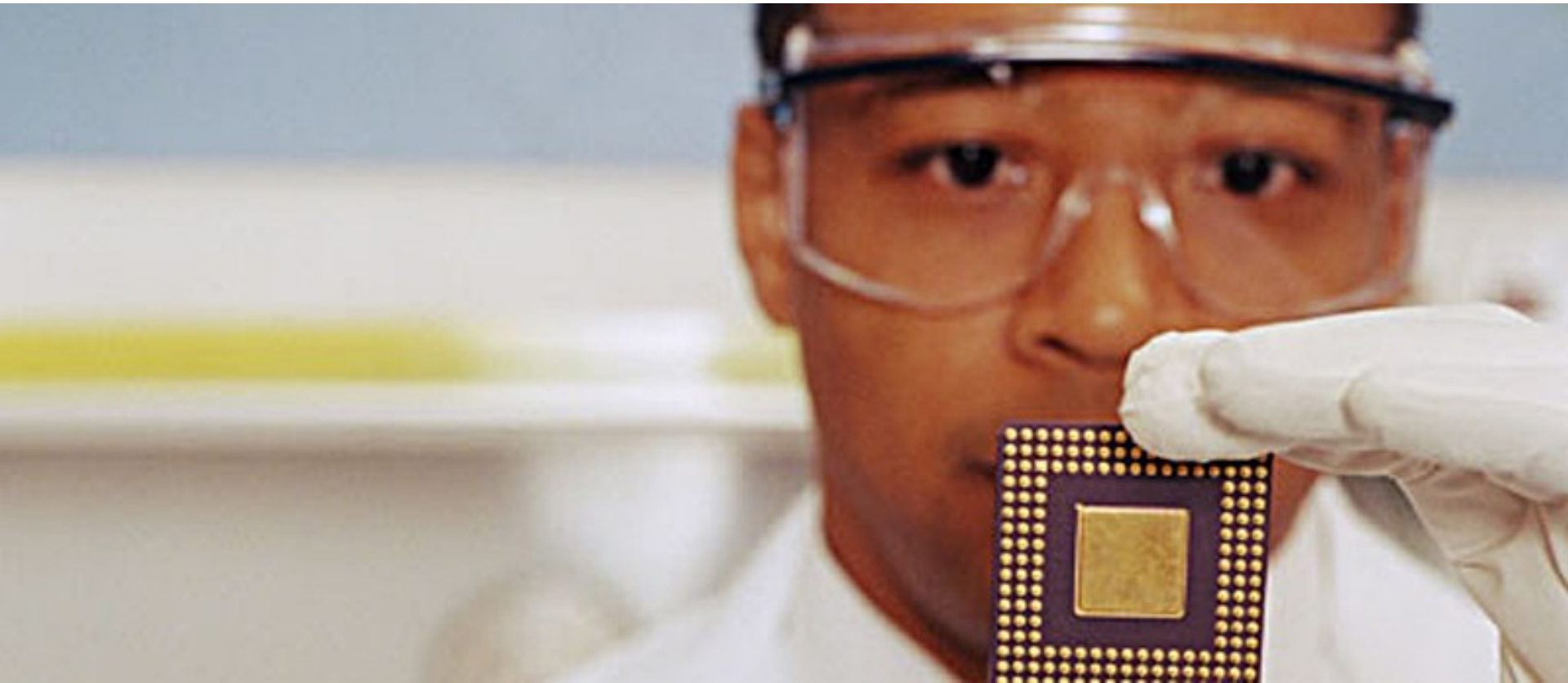
# Type-safe Enumerations

- Problem: (pre-J2SE 5.0) Previously, if you wanted to define an enumeration you either:
  - Defined a bunch of integer constants
  - Followed one of the various “type-safe enum patterns”
- Issues of using Integer constants
  - public static final int SEASON_WINTER = 0;
  - Public static final int SEASON_SUMMER = 1;
  - Not type safe (any integer will pass)
  - No namespace (SEASON_*)
  - Brittleness (how do add value in-between?)
  - Printed values uninformative (prints just int values)

# Type-safe Enumerations

- Issues of using “type-safe enum patterns”
  - Verbose
  - Do not work well with switch statements
- Solution: New type of class declaration
  - enum type has public, self-typed members for each enum constant
  - New keyword, enum

# Varargs



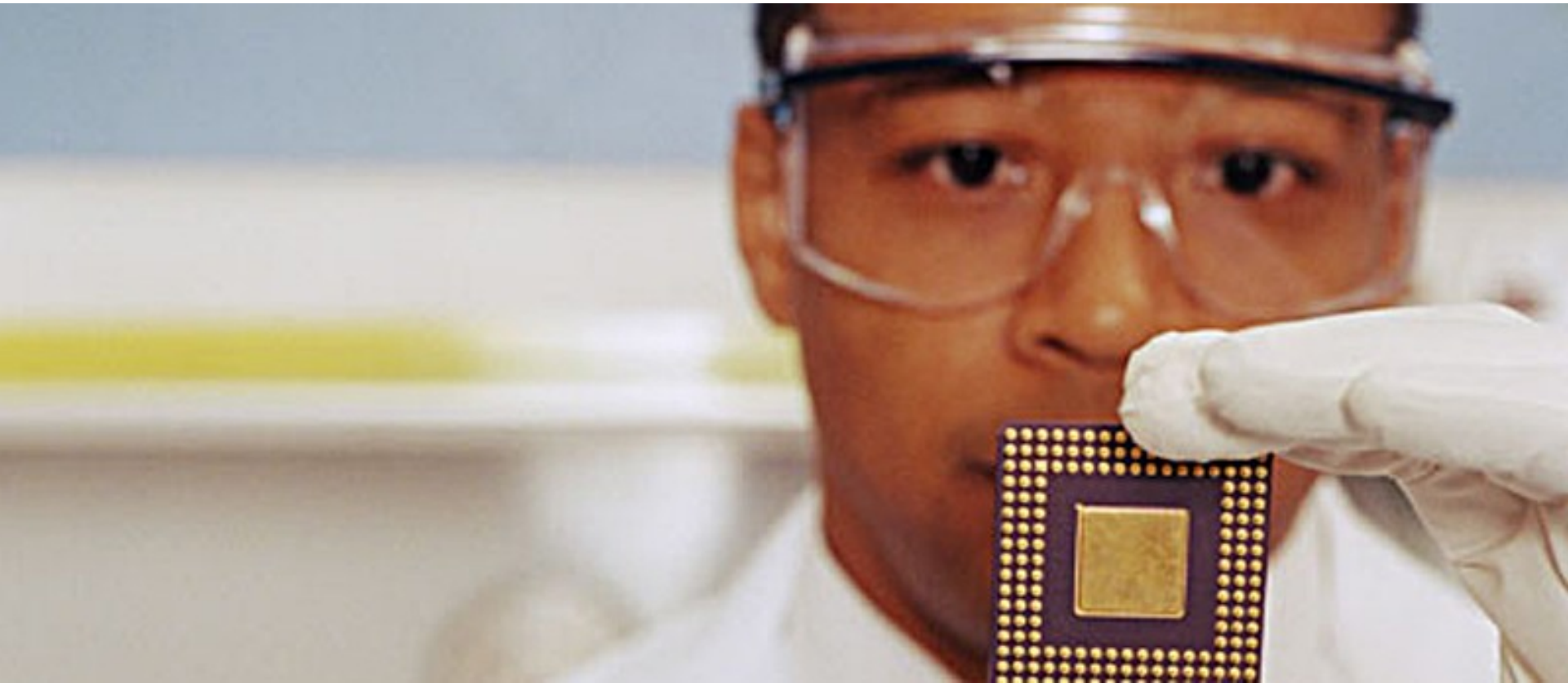
# Varargs

- Problem: (in pre-J2SE 5.0)
  - To have a method that takes a variable number of parameters
  - Can be done with an array, but caller has to create it first
  - Look at java.text.MessageFormat
- Solution: Let the compiler do it for you
  - public static String format (String fmt, Object... args);
  - Java now supports printf(...)

# Varargs examples

- APIs have been modified so that methods accept variable-length argument lists where appropriate
  - Class.getMethod
  - Method.invoke
  - Constructor.newInstance
  - Proxy.getProxyClass
  - MessageFormat.format
- New APIs do this too
  - System.out.printf("%d + %d = %d\n", a, b, a+b);

# Static Imports

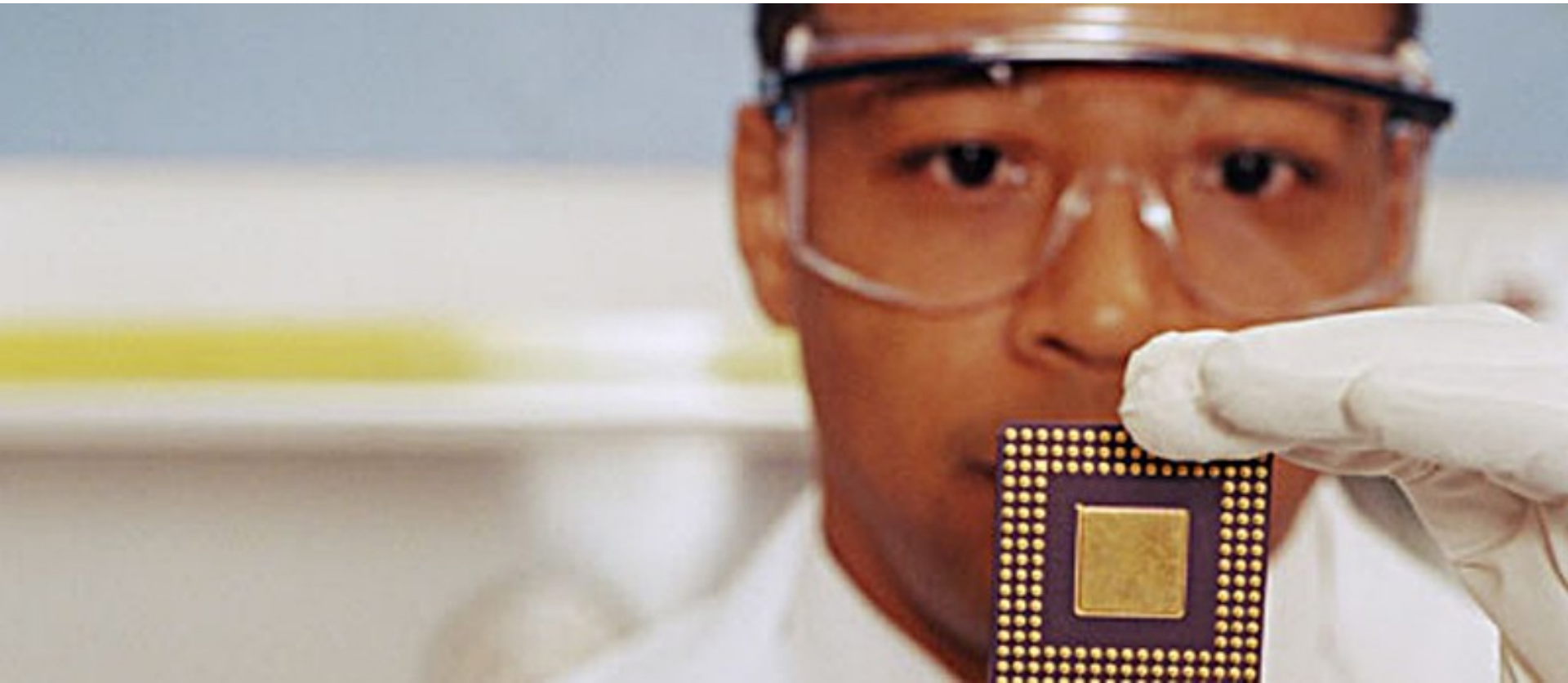


# Static Imports

- Problem: (pre-J2SE 5.0)
  - Having to fully qualify every static referenced from external classes
- Solution: New import syntax
  - `import static TypeName.Identifier;`
  - `import static Typename.*;`
  - Also works for static methods and enums

    e.g `Math.sin(x)` becomes `sin(x)`

# Formatted I/O



# Simple Formatted I/O & Scanner

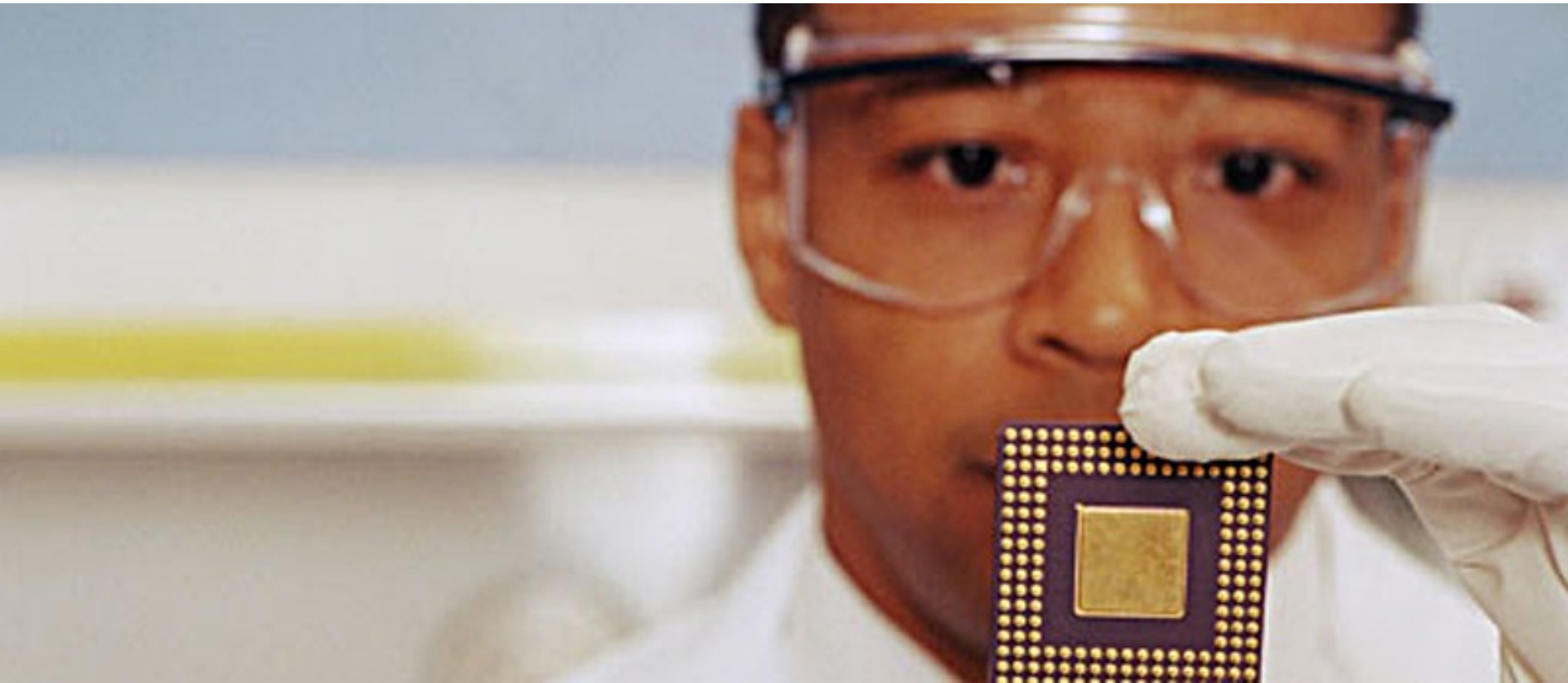
- Printf is popular with C/C++ developers
  - Powerful, easy to use
- Finally adding printf to J2SE 5.0 (using varargs)

```
out.printf("%-12s is %2d long", name, l);
out.printf("value = %2.2F", value);
```

- Also a simple scanning API: convert text into primitives or Strings

```
Scanner s = new Scanner(System.in);
int n = s.nextInt();
```

# Virtual Machine



# Class Data Sharing

- Improved startup time
  - > especially for small applications
  - > up to 30% faster
- Reduced memory footprint
- During JRE installation, a set of classes are saved into a file, called a "shared archive"
- During subsequent JVM invocations, the shared archive is memory-mapped in
- -Xshare:on, -Xshare:off, -Xshare:auto, -Xshare:dump

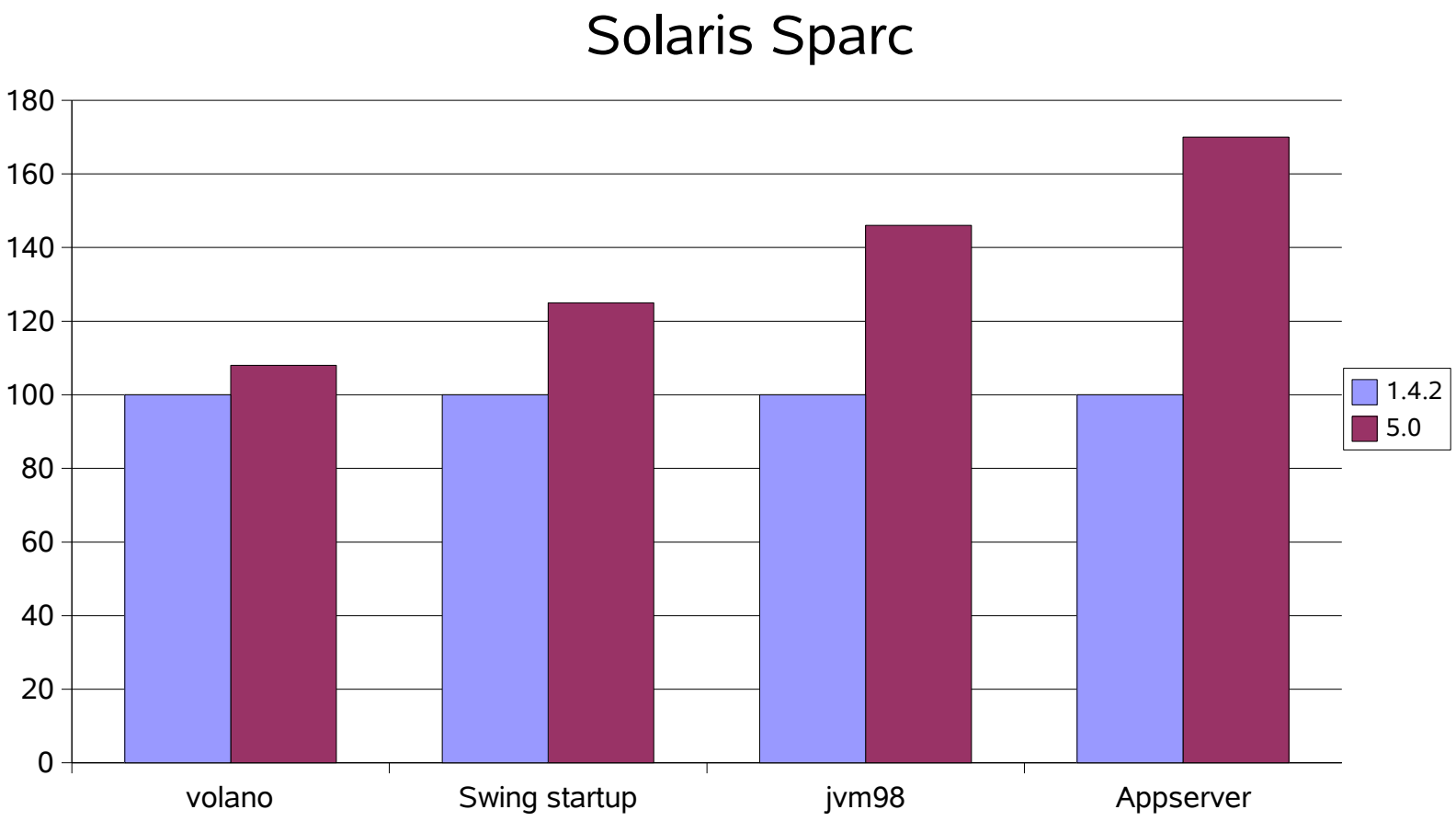
# Server Class Machine

- Auto-detected
  - Application will use Java HotSpot Server VM
  - Server VM starts slower but runs faster than Client VM
- 2 CPU, 2GB memory (except windows)
  - Uses server compiler
  - Uses parallel garbage collector
  - Initial heap size is 1/64 of physical memory up to 1GB
  - Max heap size is 1/4 of physical memory up to 1GB

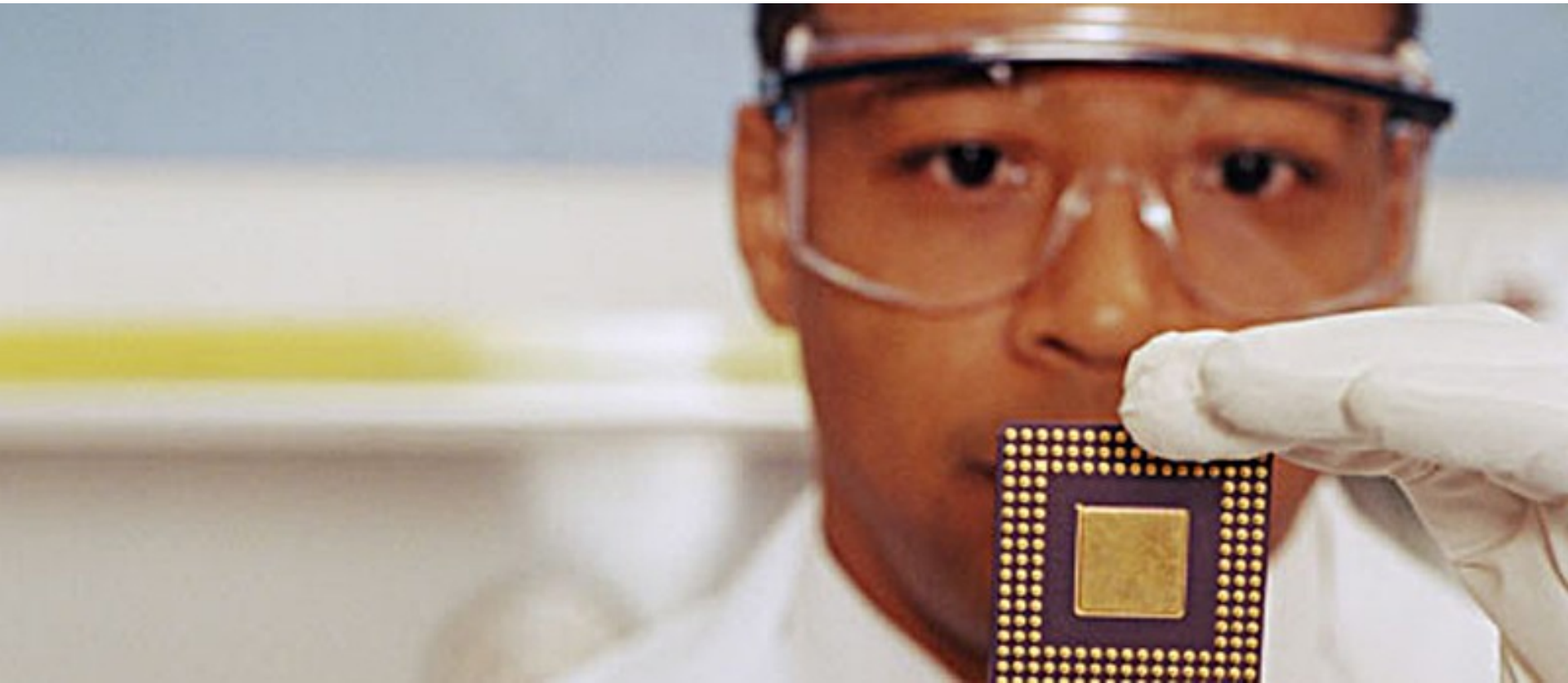
# JVM Self Tuning (Ergonomics)

- Maximum pause time goal
  - -XX:MaxGCPauseMillis=<nnn>
  - This is a hint, not a guarantee
  - GC will adjust parameters to try and meet goal
  - Can adversely effect application throughput
- Throughput goal
  - -XX:GCTimeRatio=<nnn>
  - GC Time : Application time = 1 / (1 + nnn)
  - e.g. -XX:GCTimeRatio=19  (5% of time in GC)

# Performance Improvement



# Monitoring & Management



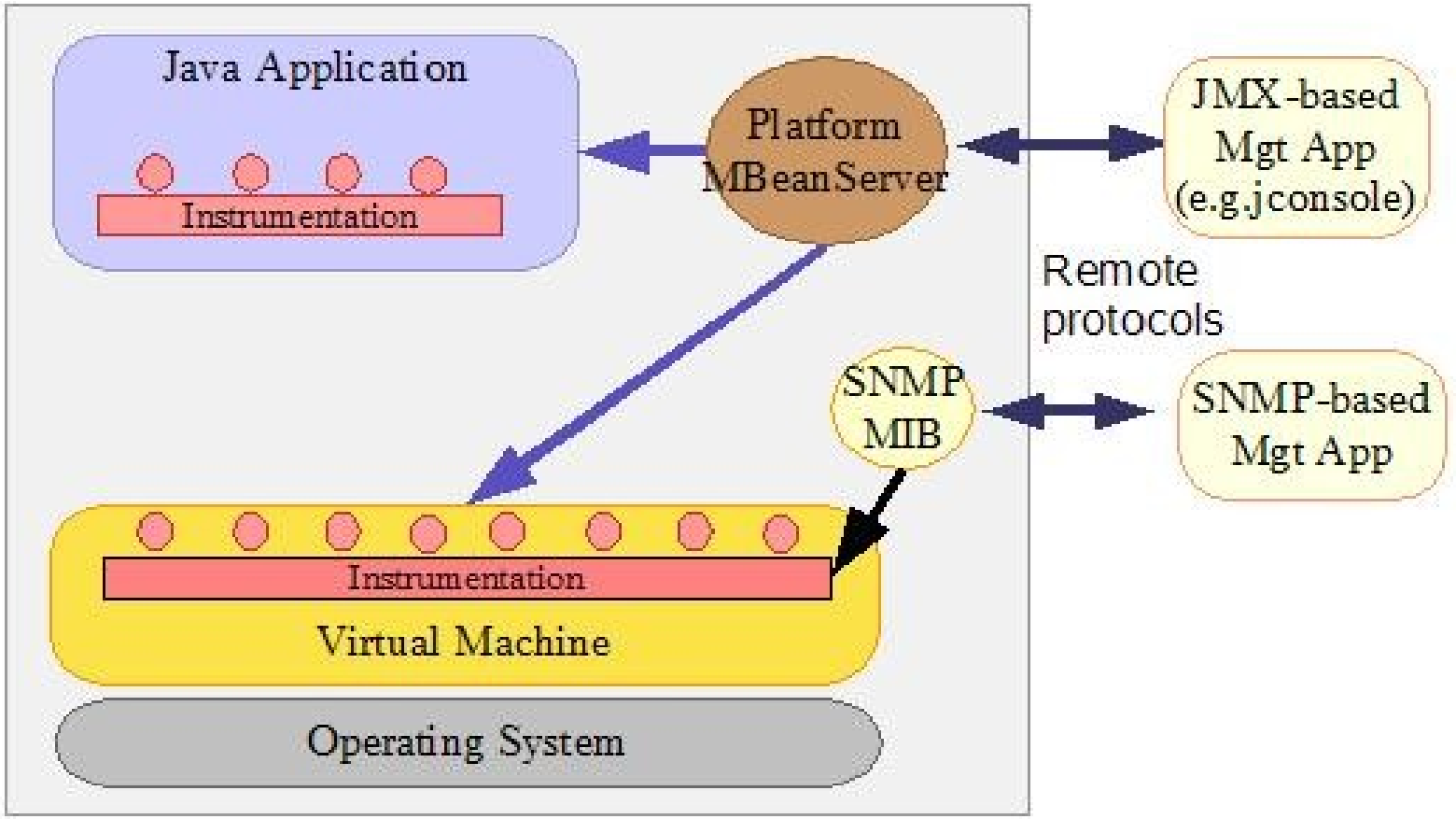
# Monitoring & Management

- Key component of RAS in the Java platform (Reliability, Availability, Serviceability)
- Features
  - JVM instrumentation and integrated JMX
  - Monitoring and management APIs
  - Tools

# JVM TI (JVM Tool Interface)

- New native programming interface for use by development and monitoring tools
  - Replaces JVMPI (JVM Profiler Interface) and JVMDI (JVM Debugger Interface)
- Improved performance analysis
- Java Platform Debugger Architecture uses JVM TI and provides higher-level interface
- Supports bytecode level instrumentation
  - Provides the ability to alter the Java virtual machine bytecode instructions which comprise the target program

# J2SE 5.0 Monitoring & Management



# Integrated JMX (JSR-003): MBean

- An MBean is a managed object that follows the design patterns conforming to the JMX specification
- An MBean can represent a device, an application, or any resource that needs to be managed
- The management interface of an MBean comprises a set of readable and/or writable attributes and a set of invokable operations
- MBeans can also emit notifications when predefined events occur

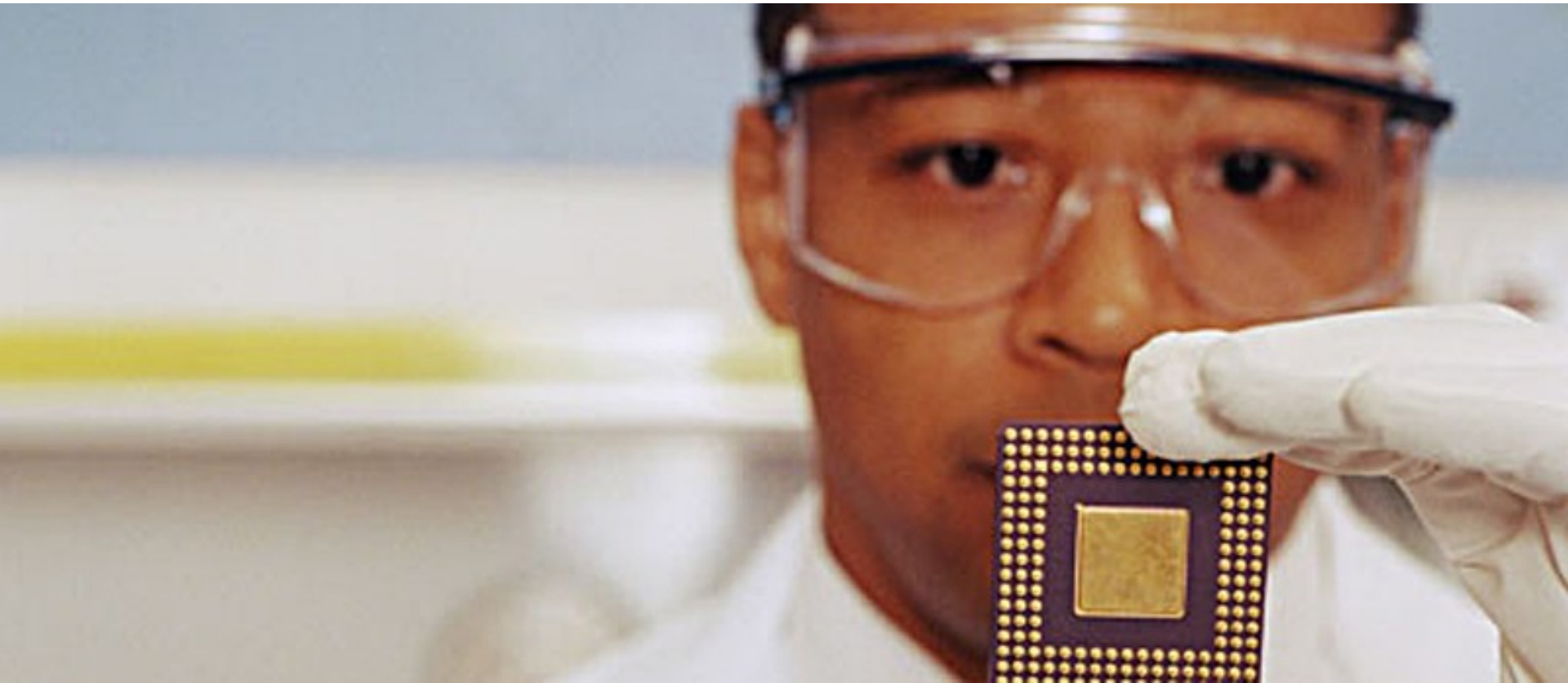
# Platform Beans (MXBean's)

- **Provides** API access to
  - number of classes loaded,
  - threads running
  - Thread state
  - contention stats
  - stack traces
  - GC statistics
  - memory consumption, low memory detection
  - VM uptime, system properties, input arguments
  - On-demand deadlock detection

# JConsole

- JMX-compliant GUI tool that connects to a running JVM, which started with the management agent
- To start an application with the management agent for local monitoring, set the com.sun.management.jmxremote system property when you start the application
  - JDK_HOME/bin/java -Dcom.sun.management.jmxremote -jar JDK_HOME/demo/jfc/Java2D/Java2Demo.jar
- To start JConsole
  - JDK_HOME/bin/jconsole

# JConsole Demo



# Thank You!

**Sang Shin**
Java Technology Architect
Sun Microsystems, Inc.