Java SE 6 Top 10 Features

Sang Shin
Michèle Garoche
http://www.javapassion.com
"Learn with Passion!"

The JDK 6 Top 10

1. Scripting
2. Web Services
3. Database (JDBC 4.0, Java DB)
4. More Desktop APIs
5. Monitoring and Management
6. Compiler Access
7. Pluggable Annotations
8. Desktop Deployment
9. Security
10. Quality, Compatibility, Stability

1. Scripting

Motivation for Scripting Support

- Provides developers an opportunity to leverage the advantages of different languages in the same application
- Extends scripting languages using the powerful Java technology libraries
  - Reuse of code modules in other programming languages
- Produces an environment in which developers and end users can collaborate to create more useful, dynamic applications
  - By delivering Java applications that can be

Scripting

- Scripting for the Java Platform (JSR 223)
  - Mechanism for configuring script engines into Java SE
  - APIs for mixing script fragments into Java applications
- A JavaScript engine is included in Sun's implementation of Java SE 6
  - Mozilla Rhino engine
- Conformant scripting engines
  - scripting.java.net

Scripting – Developer Example

```
// create a ScriptEngineManager
ScriptEngineManager m = new ScriptEngineManager();

// get an instance of JavaScript script engine
ScriptEngine engine = m.getEngineByName("js");

// evaluate a script
engine.eval("alert(\"Hello World!\")");
```

Demo: Scripting over Java SE

- Running ScriptPad sample app
- Executing JavaScript code
- Invoking Java methods from JavaScript code
- You can try this yourself
  - This sample application comes with JDK 6 as ready to open NetBeans projects

2. Web Services

Web Services Support on Java SE 6 Platform

- JAX-WS
- Data binding using JAXB 2.0
- Updates to the JAXP, which includes StaX
- Standards supported
  - SOAP 1.2
  - WS-I Basic Profile 1.1
  - XML-binary Optimized Packaging (XOP) and SOAP Message Transmission Optimization Mechanism (MTOM)
  - Representational State Transfer (REST)
  - Totally on XML schema

API Support

- Java SE 6 provides support for the JAX-WS web services stack.
  - For the client side: Service class for creating proxy
  - For the server side: Endpoint class for publication

Server-Side Programming Model

1. Write a Plain Old Java Object (POJO) implementing the service.
2. Add **`@WebService`** to it.
3. Optionally, inject a **`WebServiceContext`**
4. Publish the Web service endpoint through **`Endpoint.publish()`** method
   - WSDL is automatically generated at runtime
5. Point your clients at the Web Services Description Language (WSDL), for example:
   - **`http://myserver/myapp/MyService?WSDL`**.

Publishing Endpoint

- The **`publish`** methods can be used to start publishing an endpoint, at which point it starts accepting incoming requests.
- The **`stop`** method can be used to stop accepting incoming requests and take the endpoint down
- Publish using the HTTP server embedded in Java SE 6.
- Supports reasonable defaults for threading.
- Creates WSDL and publishes it at runtime:
  - **`http://localhost/calculator?WSDL`**

Publishing an Endpoint

```
@WebService
public class Calculator {
    @Resource
    WebServiceContext context;
    public int add(int a, int b) {
        return a+b;
    }
}
// Create and publish an endpoint
Calculator calculator = new Calculator();
Endpoint endpoint = Endpoint.publish
      ("http://localhost/calculator",calculator);
```

Client-side Programming

1. Point a tool at the WSDL for the service
2. Generate annotated classes and interfaces through a tool
3. Call new on the service class.
4. Get a proxy using a **`getxxxPort`** method.
5. Invoke any remote operations.

Example: Java SE-based Client

```
// Create a Service object
CalculatorService svc = new
    CalculatorService();

// Create a proxy from the Service object
Calculator proxy =
    svc.getCalculatorPort();

// Invoke a Web service operation
int answer = proxy.add(35, 7);
```

Demo: Web Services over Java SE

- Build and run EBay Web service and client from JDK 6 samples
- You can try this yourself
  - This sample applications come with JDK 6 as ready to open NetBeans projects

3. Database

JDBC 4.0 Support

- Updated the developer APIs (JDBC 4.0)
  - Exception handling improvement
    - New subclasses of SQLException
  - Enhanced BLOB/CLOB functionality
    - SetClob(), createClob()
  - SQLXML Data Type (from SQL 2003)
    - XML is a first-class data type – no longer need to use CLOBs to access XML data element

Java DB

- Java DB based on Apache Derby
  - JDBC conformant all-Java relational database
  - Bundled and pre-configured in JDK

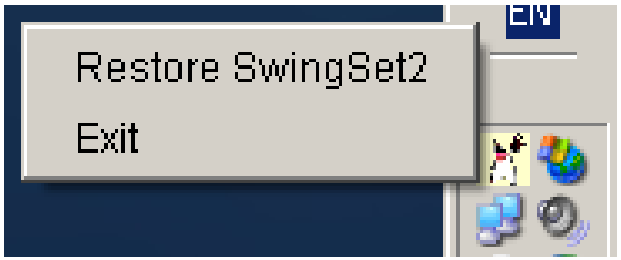
4. Desktop APIs

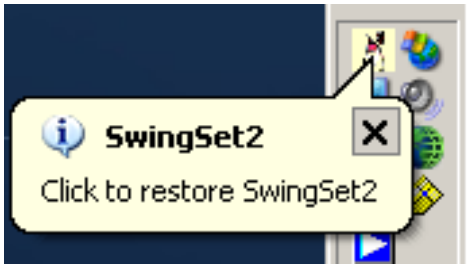
4. Desktop APIs

- AWT improvements
  - Tray icon
  - Splash screen
  - Desktop class
  - Dialog Modality enhancements and API
  - Text printing
- Swing improvement
  - GroupLayout – basis for NetBeans GUI Builder (Matisse)
  - JTable sorting and filtering
  - SwingWorker

Tray Icon

- Lets you access the system tray in your Java application
  - *SystemTray*
  - *TrayIcon*
- Give you the ability to add graphics, popup menus, and floating tip functionality to the system tray





Tray Icon: Usage

```
// Construct a TrayIcon
TrayIcon trayIcon = new TrayIcon(image, "Tray Demo",
                                 popupMenu);

// Add event listener
trayIcon.addActionListener(actionListener);

// Add the tray icon to the System tray
SystemTray.getSystemTray().add(trayIcon);
```

Splash Screen: Overview

- Before Java SE 6, Java runtime needs to be fully loaded and initialized before a visual image can be displayed
- Allows displaying a splash screen for the application instantly—before the Java runtime software starts!
  - GIF, PNG, and JPEG images supported
  - Transparency, translucency, and animation supported
  - Closed automatically when first top-level window displays

Splash Screen: Usage

- Display from command line

```
java -splash:image.gif TheApp
```

- Display from MANIFEST.MF (in a jar file)

```
Splashscreen-Image: image.gif
```

- Painting - You can change the image shown after the splash screen is loaded, but before the application starts.

```
SplashScreen splash =
    SplashScreen.getSplashScreen();
Graphics2D g = splash.createGraphics();
// your painting code here
splash.update();
```

Desktop Class

- New class: **`java.awt.Desktop`**
  - Has an enumeration of actions that may be supported for a file or URI
  - BROWSE, EDIT, MAIL, OPEN, and PRINT
- Things you can do in your Java apps
  - Launch the host system's default browser with a specific Uniform Resource Identifier (URI)
  - Launch the host system's default email client
  - Launch applications to open, edit, or print files associated with those applications
- Depends on platform capabilities to work:
  - **`Desktop.isDesktopSupported()`**

Demo: Desktop API

- Build and run sample applications
  - Tray icon
  - Splash screen
  - Desktop class
- You can try this yourself
  - www.javapassion.com/handsonlabs/javase6features

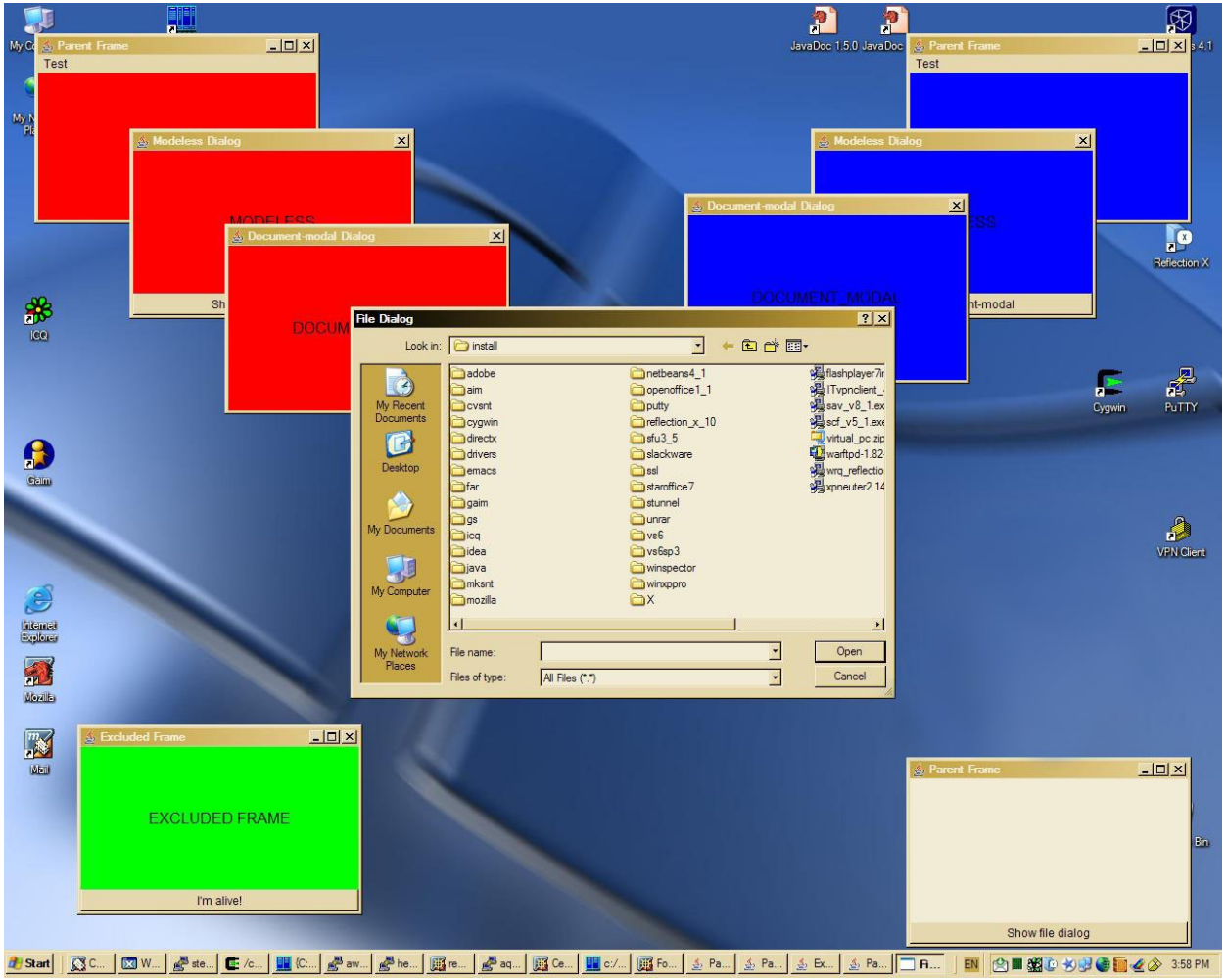
Dialog Modality Enhancement

- New modality model is introduced
  - This new model allows the developer to scope, or limit, a dialog box's modality blocking, based on the modality type that the developer chooses
  - Allows windows and dialog boxes to be truly parentless
  - Solves the problem of interacting with JavaHelp in J2SE 1.5 when modal dialog box is on the front

Modality Types

- modeless
  - does not block any other window
- document-modal
  - blocks input to all top-level windows from the same document
- application-modal
  - blocks all windows from the same application
- toolkit-modal
  - blocks all windows that run in the same toolkit

New Dialog Modality API



Text Printing

- Easily print a Swing text component:
  - Prints the entire contents of the text component
  - Does not have to be visible
  - `javax.swing.text.JTextComponent.print();`
- Reformats for printed page
- Optionally displays print dialog and progress box
- Supports optional header/footer
- Will not split lines in half!

Demo: Desktop API

- Build and run sample applications
  - Dialog Modality enhancements and API
  - Text printing
  - JTable sorting and filtering

SwingWorker

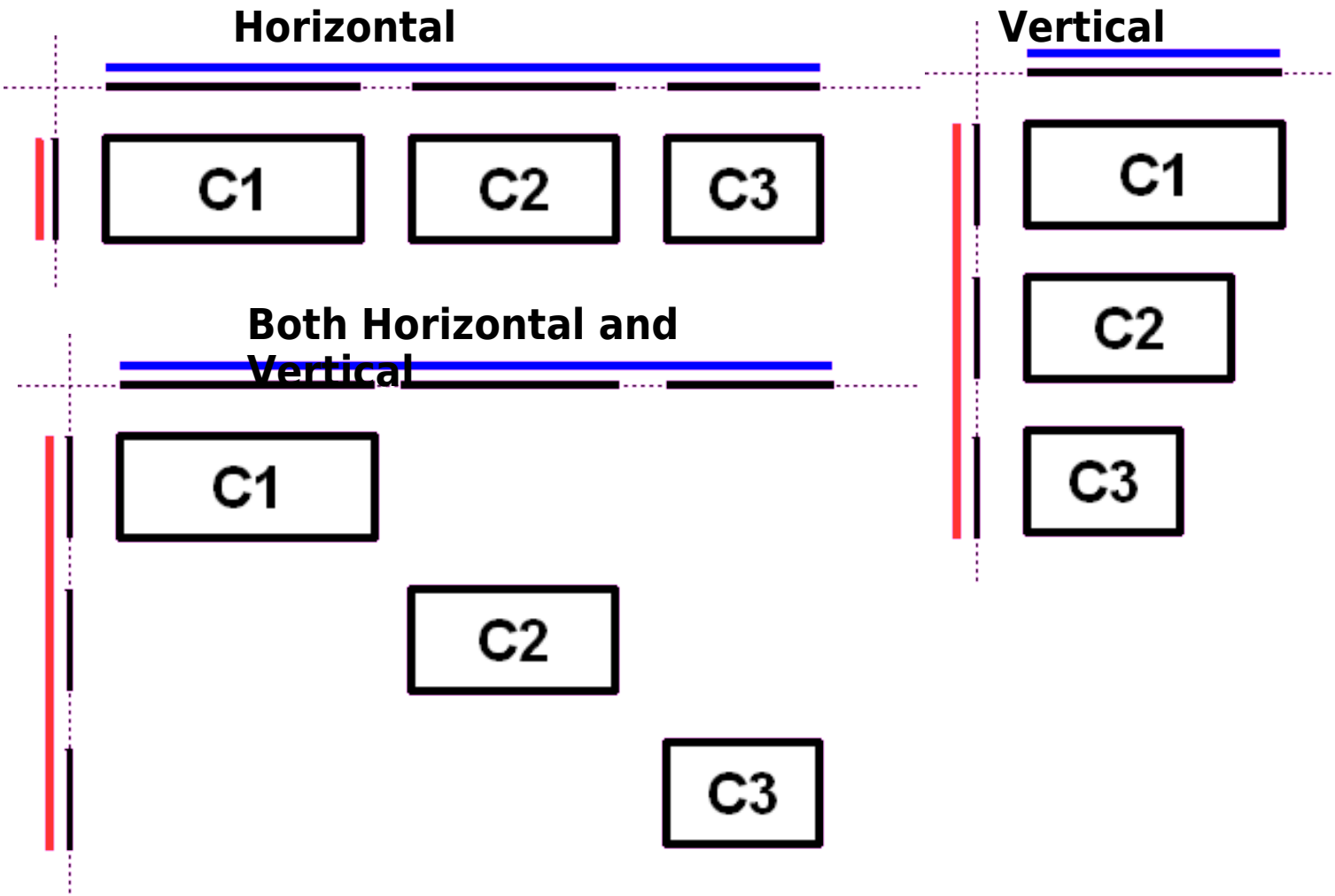
Easing multi-threaded applications with Swing

- Makes it easy to offload work to separate th
- Makes use of concurrency package
- Makes it more generic
- Supports partial results
- Supports PropertyChangeListener
- More information:
  - java.sun.com/docs/books/tutorial/uiswing/concur

`javax.swing.GroupLayout` Class

- New layout manager to support new Matisse GUI builder
  - NetBeans™ IDE ships with Matisse
  - Can also use `GroupLayout` in J2SE 1.5 software using stand-alone library
- More capabilities for relative positioning of components
- Works with horizontal and vertical layout separately

Using `GroupLayout` Class



JTable Sorting and Filtering

- Add sorting to your JTable with one method call:
  - `setAutoCreateRowSorter(true)`
- Specify your own comparators
- Supports secondary and tertiary sort columns
- Can specify a filter to limit what is shown:
  - Regular expression, number, and date implementations provided

Demo: NetBeans GUI Builder

- Build and run ContactEditor GUI
- You can try this yourself
  - www.javapassion.com/handsonlabs/nbguibuilder/

5. Monitoring & Management

Potential Problems That Can Be Detected

- Memory leaks
- Thread deadlocks
- Dirty references
- Infinite loops

Monitoring and Management

- *jps*: lists JVM's
- *jconsole*: can connect to applications that did not start up with the JMX agent
- *jmap*: takes a detailed 'photograph' of what's going on in memory at any one point in time
- *jhat*: forensic expert that will help you interpret the result of *jmap*
- *jstack*: takes a 'photograph' of all the threads and what they are up to in their own stack frames

Demo: jconsole, jps, jmap, jhat, jstack

- Run a sample Java application
- Use the tools
  - Use *jps* to see process ids of all Java processes
  - Use *jconsole* to connect it
  - Use *jmap* to capture snapshot of heap of a Java process
  - Use *jhat* to interpret it
  - Use *jstack* to thread-dump on a live process
- You can try this
  - www.javapassion.com/handsonlabs/javase6tools/

Demo: Memory Leak Detection via NetBeans

- Find out exactly where memory leaking code in your Java application is located
- You can try this
  - www.javapassion.com/handsonlabs/nbprofilermemory/

6. Compiler Access

Compiler Access

- Opens up programmatic access to *javac* for in-process compilation of dynamically generated Java code
- Really aimed at people who create tools for Java development and for frameworks
  - JavaServer Pages (JSP) or PHP construction kit engines that need to generate a bunch of classes on demand
  - Average developers will benefit indirectly from faster performing tool
    - Jasper JSP engine runs JSP TCK 3.5x faster

7. Pluggable Annotations

Pluggable Annotations

- JSR 175 of JDK 5 standardized how annotations are declared in Java code but annotation processing details were relegated as an implementation detail
- JSR 269 of JDK 6, Pluggable Annotation Processing API, standardizes annotation processing as well
  - The annotation processors act as plug-ins to the compiler, hence "pluggable annotation processing”

7. Pluggable Annotations

- Allow developers to define new annotations...

```
@ForReview
public void myMethod() {...}
```

- ...and APIs to define components that process them...

```
import javax.annotation.processing.*;

public class ForReviewProcessor extends AbstractProcessor {..}
```

- ...and integrate them with the Java Compiler

```
javac -processor ForReviewProcessor MyCode.java
```

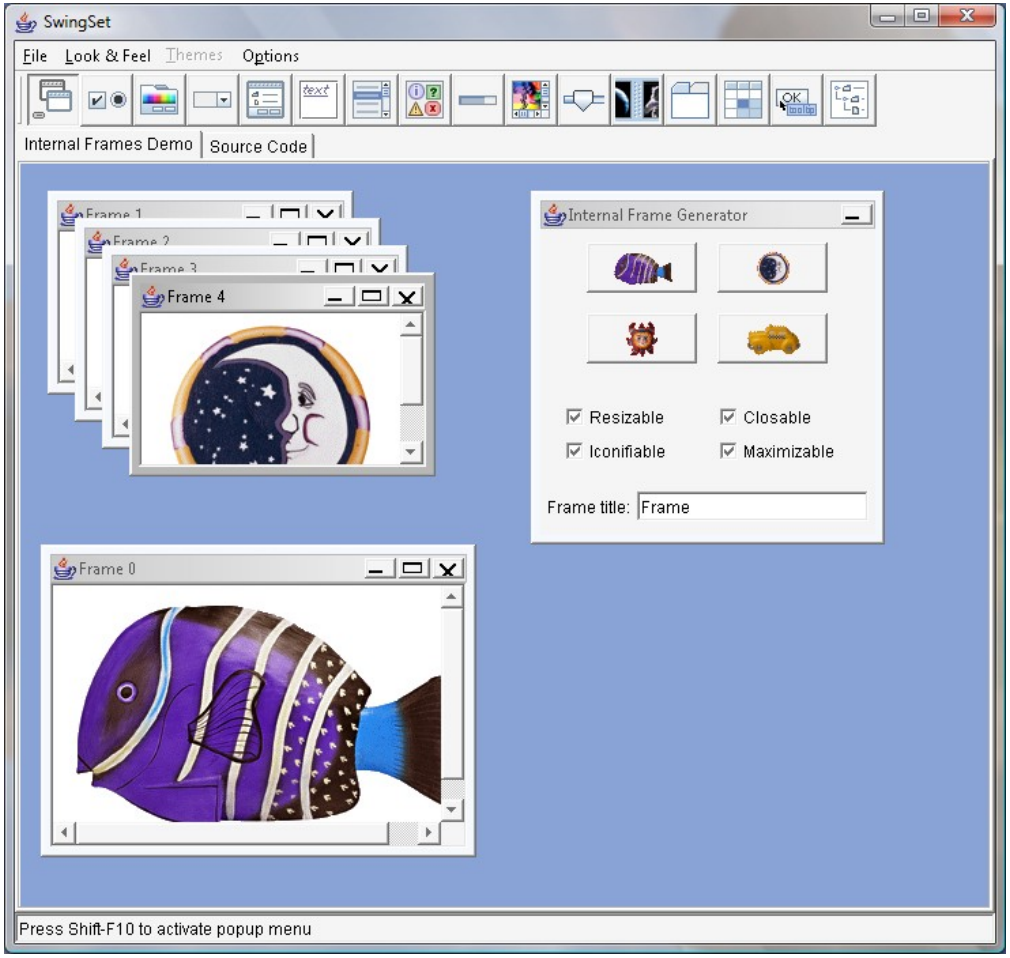
8. Desktop Deployment

8. Desktop Deployment

- We improved actual performance
  - graphics hardware acceleration on Windows
- ...and perceived performance
  - true double buffering
- We improved the native look & feels
  - Updated Swing Look&Feel Windows/Unix
  - LCD text rendering
- We revamped Java Web Start and JRE installations
  - no more scary security dialog

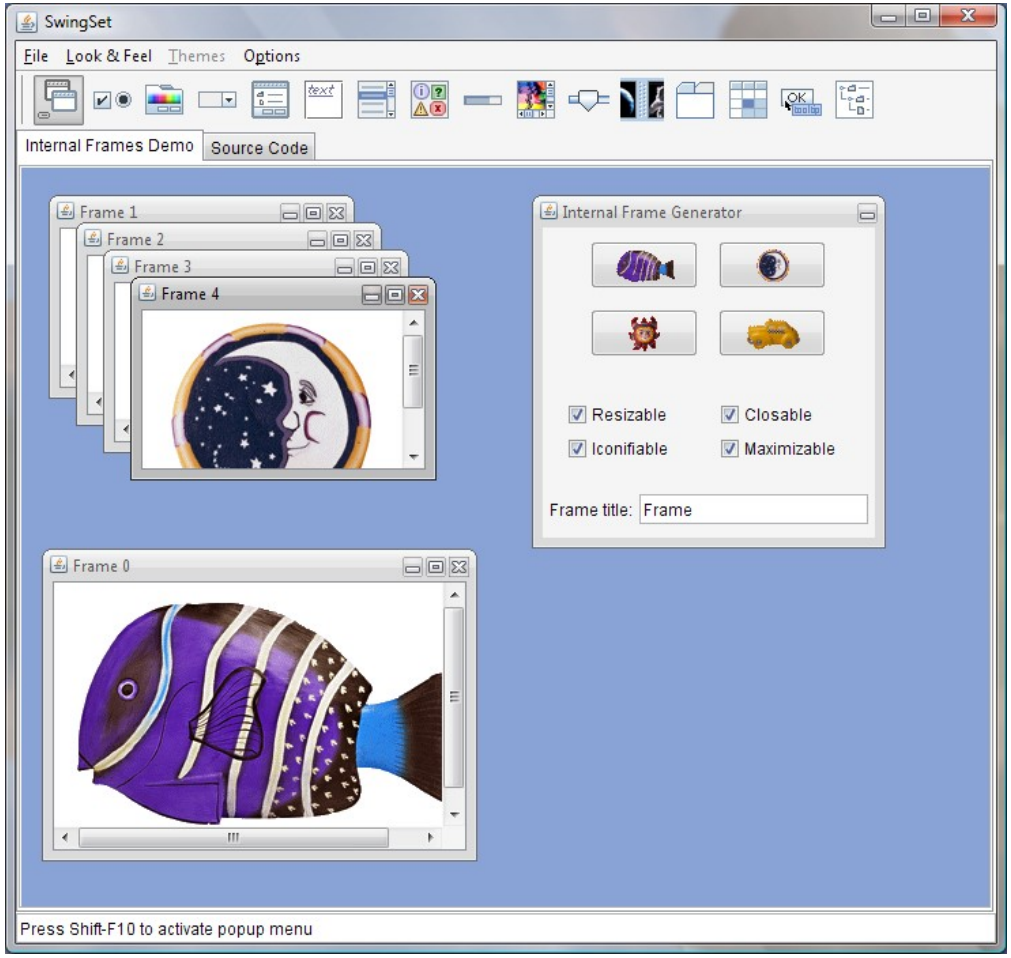
Windows Look and Feel Improvements

SwingSet on Vista with 5.0



Windows Look and Feel Improvements

SwingSet on Vista with 6



9. Security

9. Security

- We added important new APIs
  - XML Digital Signature (XMLDSig) API (JSR 105)
  - Smart Card I/O API (JSR 268)
- Improved authentication schemes
  - JAAS-based authentication using LDAP
  - Native Platform Java GSSAPI (Generic Security Services Application Programming Interface) integration

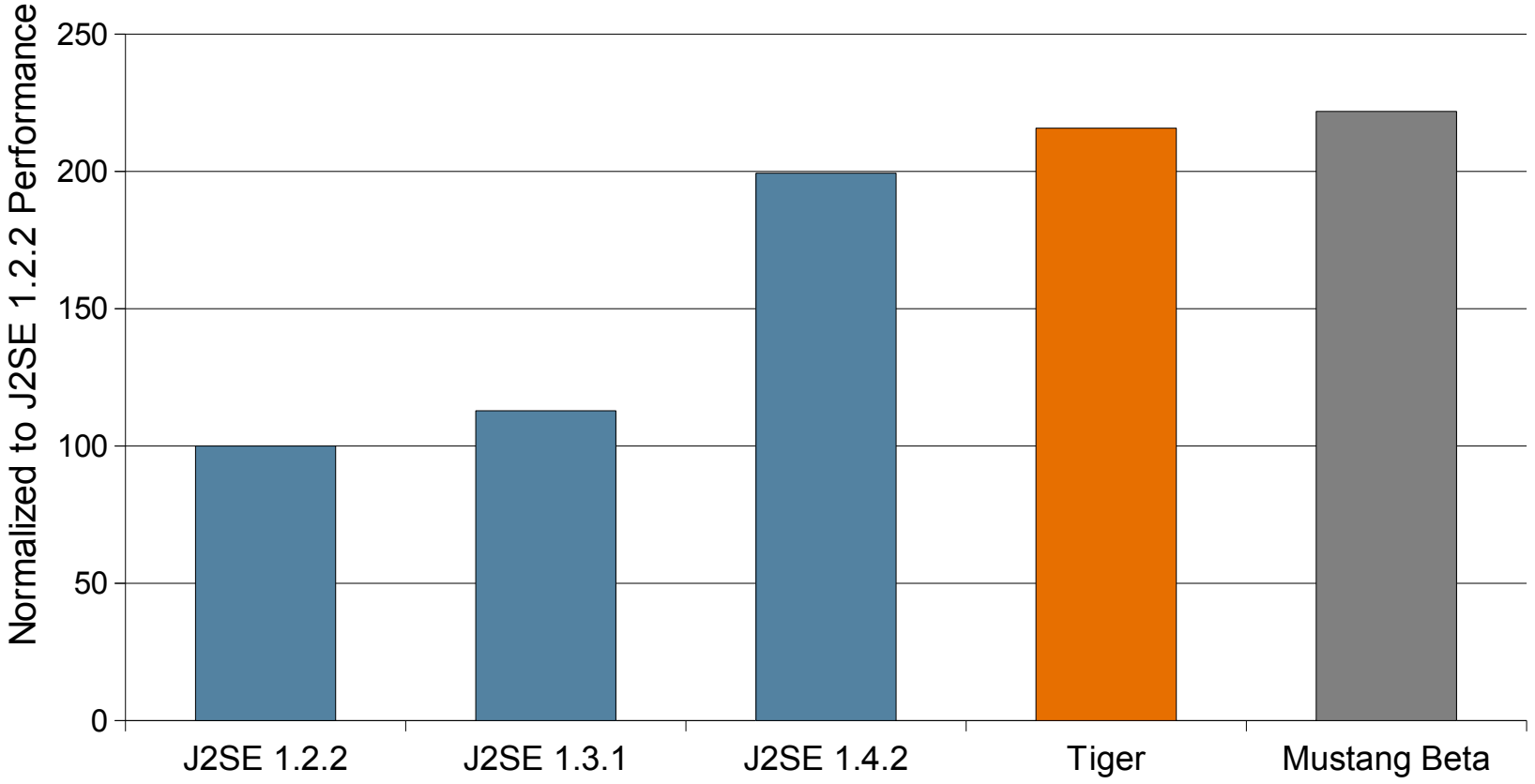
10. Quality, Stability, Compatibility

10. Quality, Stability, Compatibility

- We are still running the Big App tests
- We now have 80,000+ JCK tests
- We've had good uptake of weekly builds
- We ran a Regression Challenge

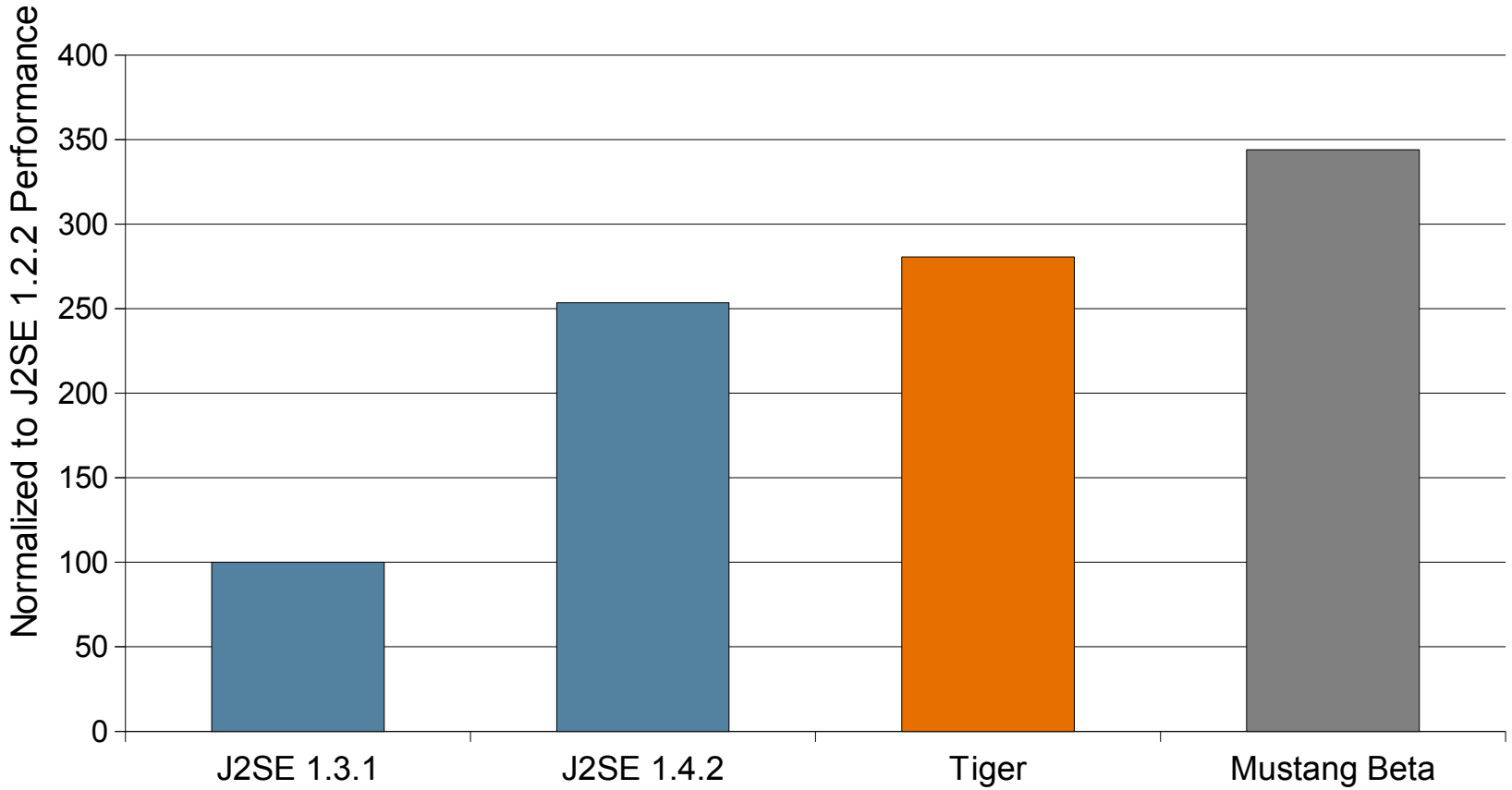
10. Performance Improvement

Client Benchmark: SwingMark



10. Performance Improvement

Server Benchmark: SPECjbb2000



Why Java SE 6?

Running Apps on Java SE 6

- Applications run faster on the desktop and servers
- New 'Dynamic Attach' diagnostics simplify troubleshooting
- Expanded Solaris DTrace support provides additional value on Solaris
- Improved 'native' look and feel across Solaris, Linux, and Windows
- First Java platform with full support for Windows Vista

Building Apps on Java SE 6

- JavaScript integrated and included with the platform
- Scripting languages framework extends support for Ruby, Python, and other languages
- Complete light-weight platform for web services, right out of the box
- Simplified GUI design and expanded native platform support
- Full JDBC4 implementation providing improved XML support for Databases
- Java DB included with the JDK, a free to use and deploy Java Database
- Full support by NetBeans IDE 5.5, 5.5.1 and 6.0

Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps
"Learn with Passion!"