Sun Microsystems Laboratories

SML 97-0157

Modular System Building With Java™ Packages

Mick Jordan

Michael L. Van De Vanter

The Forest Project

8th Conference on Software Engineering Environments, Cottbus, Germany 8-9 April 1997

Background

- **Java programming**
  - **Java is a modern, mainstream programming language**
  - **very large systems are being built in Java**
  - **reuse of library classes will be a key productivity factor**
  - **geographically dispersed groups will collaborate via Intranet & Internet**
  - **a variety of relationships among groups will emerge**
- **System building tools for Java lack:**
  - **support for problems of scale**
  - **support for effective code reuse**
  - **support for distributed collaboration**
  - **modern technology (text files and *make* are obsolete)**

Goals for the Forest Project

- **Create a *scalable development model* for Java programming**
- **Build a *prototype* development environment: *JP***
- **Enhance *code reuse* by JP-compliant environments**
- **Extend the Java slogan:**

  “write once, *build anywhere*, run anywhere”

Working Hypothesis for the JP Development Environment

- **Modular system building is crucial**
  - **it must scale to very large systems**
  - **it must be compatible with current practice**
- **The Java *package mechanism* is a good start**
  - **most language properties are desirable**
  - **the rest can be supplied by an Integrated Development Environment**
- **Complexity must be reduced**
  - **abstract over inessential differences**
  - **automate much of what is now done manually**
- **Reliability must be increased**
  - **builds must be repeatable**
  - **information derived by tools must not be lost**

The Key Idea: Use Versioned Packages *as System Modules*

- **A versioned package (in the style of Vesta)**
  - **is a conventional package in the Java package namespace**
  - **is a first-class entity in the JP development environment**
  - **contains a *build script*: a parameterized program for building it**
  - **uses other package versions via `import` statements in the build script**
- **This concept simplifies development by unifying:**
  - **modular system construction**
  - **storage management**
  - **system building**
  - **configuration management**

Versioned Package Example

- **A versioned package**

```
COM.sun.labs.jvb.util
```

- **A package version**

```
COM.sun.labs.jvb.util.7
```

- **Source code for a class**

```
COM.sun.labs.jvb.util.7.Stack
```

- **Using the class in Java source code**

```
import COM.sun.labs.jvb.util.Stack
```

- **Using the package version in a JP build script**

```
import COM.sun.labs.jvb.util.7
```

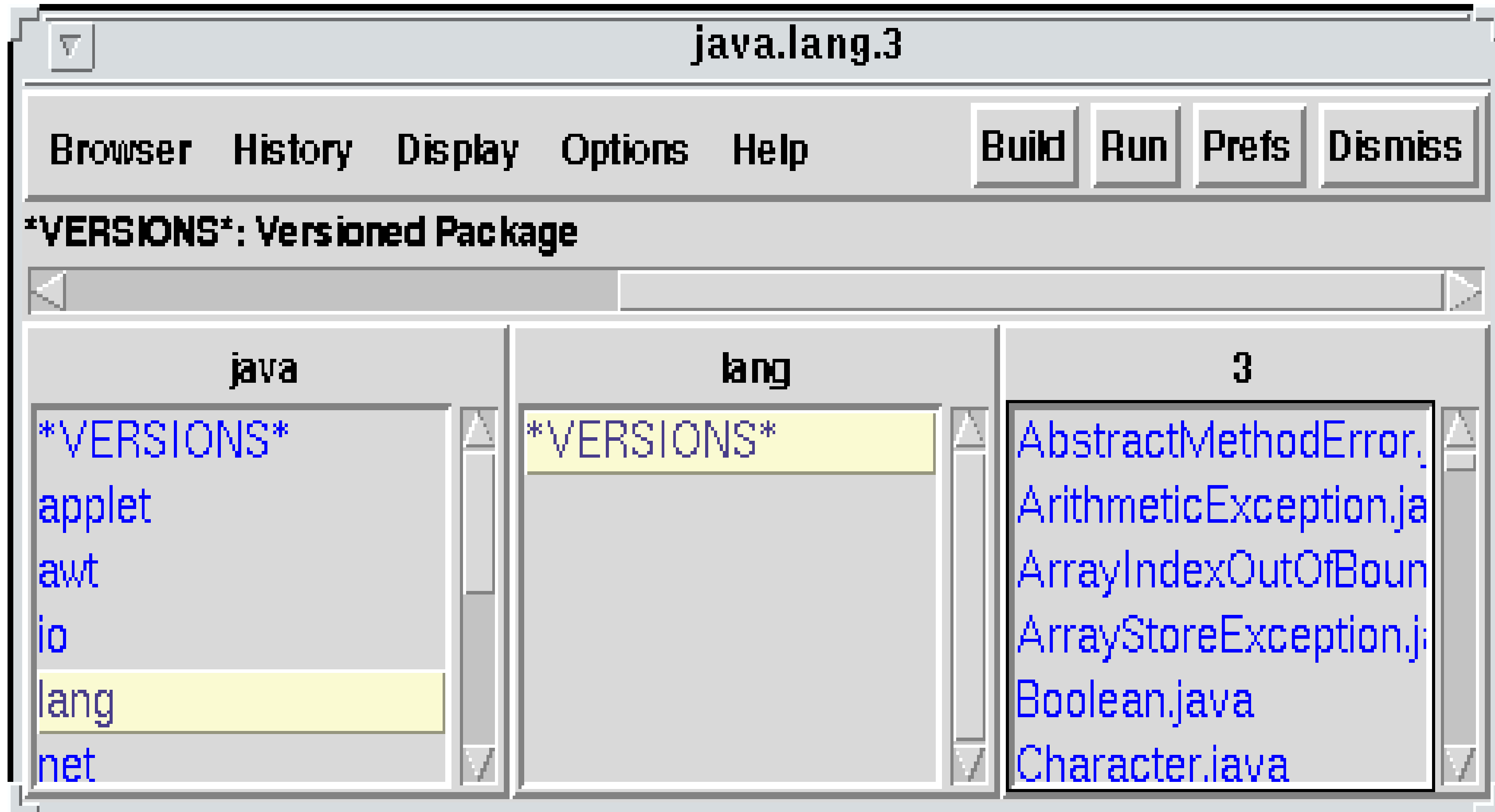
Versioned Packages and System Structure

- **The Java package mechanism offers some useful features**
  - **systematic, (potentially) global name management**
  - **access control within the language**
- **The Java package mechanism is weak for system structuring**
  - **name discipline not enforced**
  - **a package is not first-class (you can’t import a package, only its classes)**
  - **a package has no other semantics for compilation or runtime class loading**
- **Versioned package are first-class entities in JP**
  - **The unit of versioning/configurations, storage, and building**
  - **The unit of code sharing and replication across sites**

Versioned Packages and Naming Systems

- **Current tools require management of too many names**
  - **language entities**
  - **files and directories**
  - **build modules**
  - **versions and configurations**
- **Java packages *align* language (class) and storage (file) names**
- **JP packages unify them all**

Example: JP and Abstract Naming



Versioned Packages and Global Names

- **Global, location-independent, package names are important**
  - **replication**
  - **build anywhere**
- **Java names are potentially global**
  - **DNS-based names are officially encouraged, e.g. `COM.sun.labs.jvb.util`**
  - **global naming is not always followed**
- **But long names are awkward in Java**
  - **we want to write**

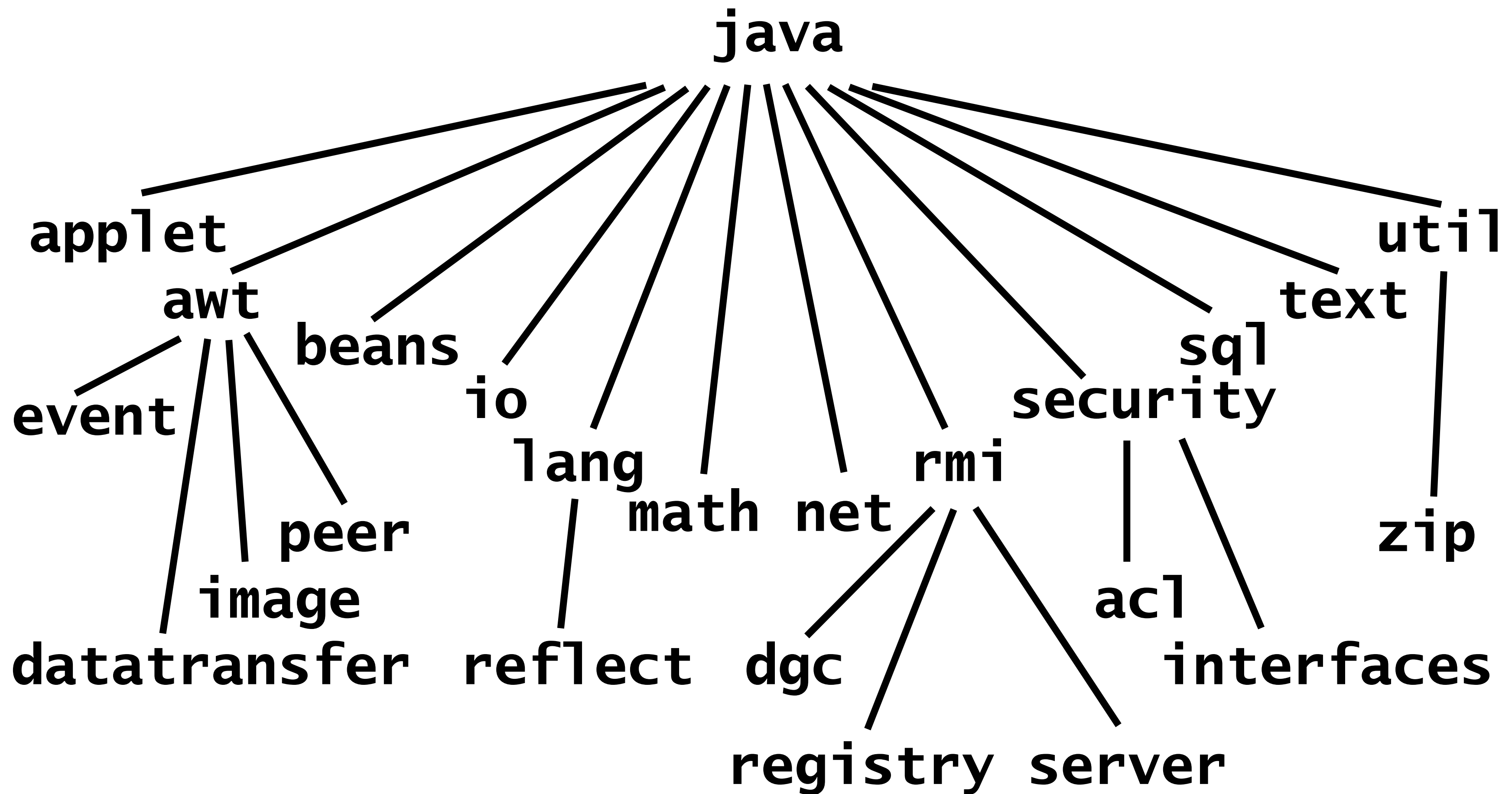
```
import COM.sun.labs.jvb as jvb;
class SpecialStack extends jvb.util.Stack {...}
```

  - **packages may not be imported in Java**
  - **renaming is not supported in Java**

Versioned Packages and Configuration Management

- ***umbrella* packages are used for aggregation**
  - **use other package versions via `import` in the build script**
  - **the build script `import` always refers to a precise, immutable version**
- **umbrellas typically contain no source code directly**
- **umbrellas are versioned**
- **umbrellas may recursively describe arbitrarily large systems**

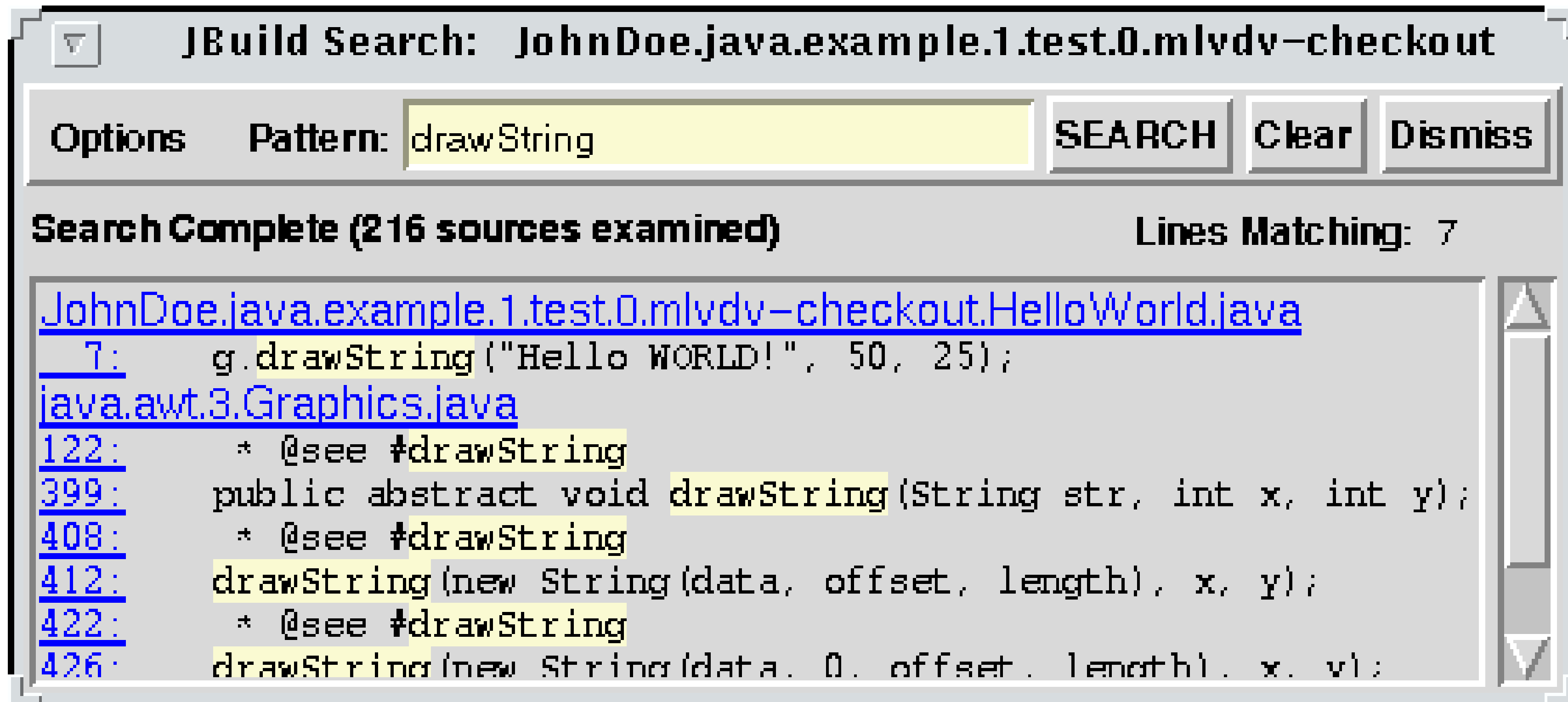
Package Hierarchy: JDK 1.1 Core Classes



Versioned Packages and System Building

- **Build scripts are Vesta-style mostly functional programs**
  - **compute over domain of source objects, class files, and other derived objects**
- **The build script interpreter is responsible for build avoidance**
  - **caches (memoizes) function evaluations**
  - **cache by *fingerprint*: a highly reliable abstract value**
- **Derived objects are unnamed and invisible unless needed**
- **Each build invocation creates a *package object*:**
  - **describes completely and precisely the sources and resources used**
  - **describes all results**
  - **available for use by tools in the environment**

Example: Build-directed Search in JP



Versioned Packages and Code Reuse

- **Package names are location-independent**
  - **assume global naming**
  - **check values abstractly via fingerprints**
- **Build scripts completely describe what must be built**
  - **libraries**
  - **compiler (in a versioned package, of course)**
  - **supports build-anywhere**
- **Build script `import` statements in JP are *abstract***
  - **package versions may stored locally, remotely, or mirrored**
  - **package versions may be available in source, binary, or binary-on-demand**

The JP Prototype Development Environment

- **Interim version (JP-lite)**
  - **in use for 1+ years**
  - **supported its own development**
  - **experience has been encouraging**
- **Next version (JP)**
  - **storage via orthogonal persistence for Java (OPJ) -- no files**
  - **add mechanisms for distributed development**
  - **explore collaboration models**

Conclusions

- **JP Synthesizes the Vesta approach and Java programming**
- **There is a good fit between**
  - **system modules for large system construction**
  - **versioned packages in the Java namespace**
  - **(but small language extensions would help)**
- **The Vesta approach has been shown to scale up**
- **The support environment is crucial**
  - **simplifies and automates much of the work**
  - **streamlines construction of higher-level programming tools**
  - **JP-compliant environments must be widely available**