Tim Ellison
1st October 2012

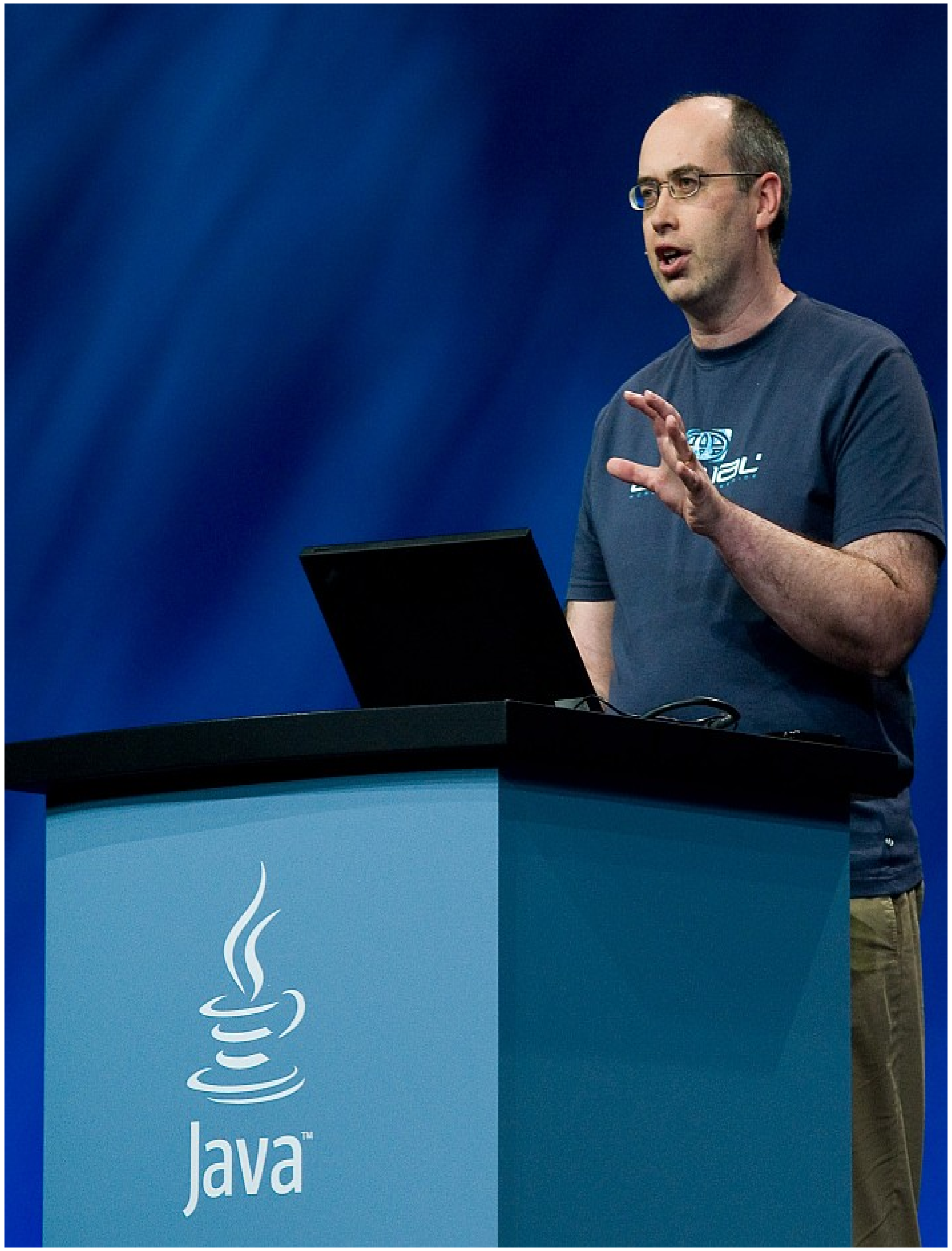
Java and Real-World Compatibility

CON5243

Important Disclaimers

Introduction to the speaker

- Over 20 years experience developing and deploying Smalltalk and Java SDKs
- Recent work focus:
  - IBM JDK architecture
  - Class library engineering
  - Open source participation
- My contact information:
  tim_ellison@uk.ibm.com



Agenda

- Why is compatibility important, and what do we mean by “real world” compatibility?
- Different types of compatibility : what are they? why are they important? how do we deal with them?
- Protecting yourself from incompatible changes in Java.
- Ways to find and address compatibility issues in your own code.

The Java promise

- Benefits of Java SE as a universal computing platform
  - Leverage investment in secure, portable, high-performance, rich function runtime
  - Available across a wide variety of heterogeneous environments
  - Broad ecosystem of tools, books, conferences, developers, services, etc.

- Java SE is specification based
  - There are a number of different specifications
    - Java Language Specification
    - Java Virtual Machine Specification
    - Java Platform API Specification
    - ...etc

JSR-133: Java™ Memory Model and Thread Specification

August 24, 2004, 4:42pm

This document is the JSR-133 specification, the *Java™ Memory Model and Thread Specification*

Java™ Platform, Standard Edition 7 API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
java.awt.image	Provides classes for creating and modifying images.

- These are detailed specifications for implementers and users, to
  - ensure consistency in behavior across implementations
  - describe how the language is interpreted, binaries are combined, the memory model executed, etc

- Java has evolved through collaboration on specifications and competition on implementation

The Java compatibility promise

- As a Java developer you are awarded certain assurances by sticking to the rules
    - often described as "write once run anywhere"
  - Java is therefore the 'platform' on which you program
    - you are not programming specifically for Linux or 64-bit or NFS or …
- Implementers try to optimize the Java platform to exploit the 'physical' platform
- Strong specifications give rise to a broad ecosystem
  - multiple JVMs, multiple class libraries
  - goals of different implementations
    - portability, research, production deployment, size, development support, ...
- The Technology Compatibility Kit (TCK)
  - ensures that implementations comply with the specifications
- Implementers use additional test suites to ensure implementation correctness, robustness, performance, etc

"Except for timing dependencies or other non-determinisms and given sufficient time and sufficient memory space, a program written in the Java programming language should compute the same result on all machines and in all implementations."

http://java.sun.com/docs/books/jls/third_edition/html/binaryComp.html#13.2

Real-world compatibility

- *Real-world* compatibility goes beyond the language, runtime, and specifications
- An application's dependency on the Java platform goes beyond the specifications, e.g.:
  - list of, and content of system properties
  - command line options
  - output format: version strings, verbose GC output, etc.
  - ...etc.
- WORA is the Java philosophy that gives you portability across implementations, platforms, etc.
  - the code you wrote continues to work across different Java implementations different hardware platforms, vendors, etc.
  - This is one specific aspect of compatibility
  - WORA is viable for a subset of applications, typically migration help is required
- Compatibility ensures the code you wrote continues to work as Java evolves
  - you can pick up bug fixes and enhancements in the platform
  - take advantage of security, performance, new places to run your code
  - some compatibility trade-offs are made as the platform evolves
  - there are a number of tricks and tips for ensuring compatibility and dealing with breakage
- This talk will look at some areas and examples of intentional and unintentional compatibility breakages

Types of Compatibility

- Functional compatibility : does running your program have the same effect ?
- Source compatibility : does your source code result in the same set of binaries ?
- Binary compatibility : do your binaries successfully link to the new platform binaries ?

Functional compatibility

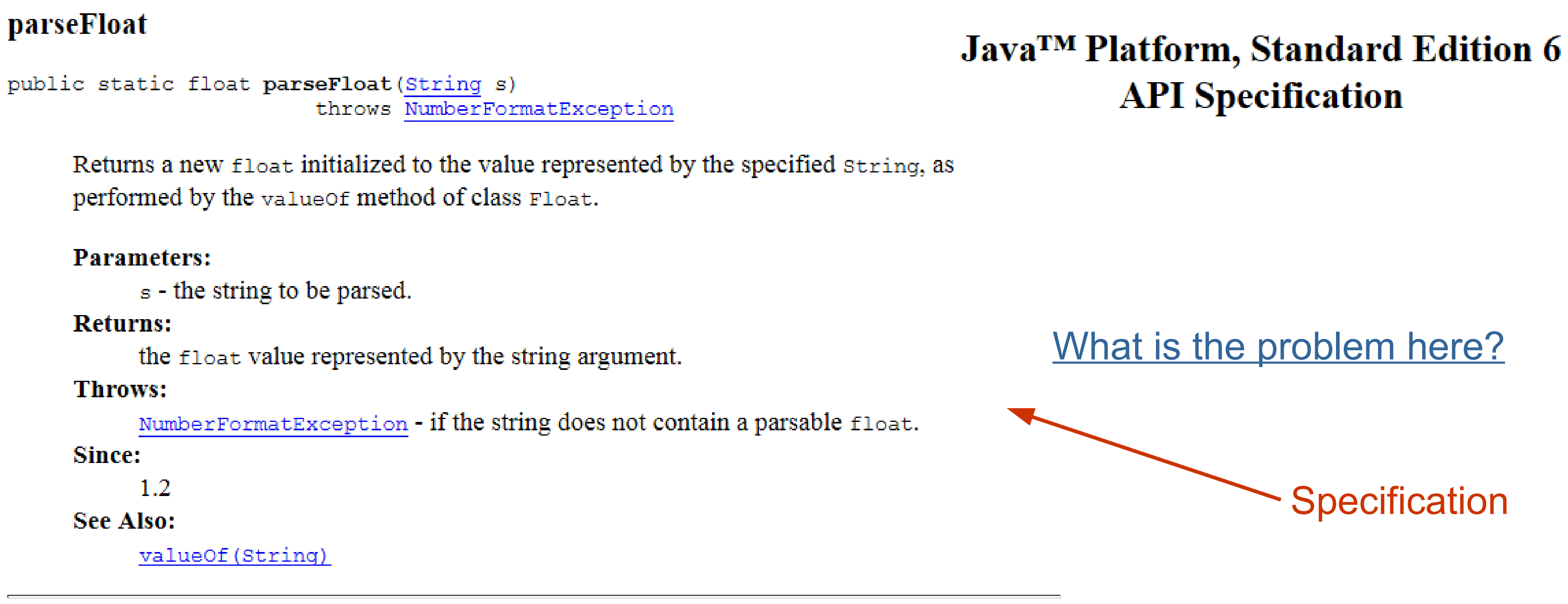
- The functional contract of Java is primarily defined by the specifications
- Some aspects of an implementations behavior are intentionally omitted from the spec
  - they are known or expected to behave differently in different implementations
  - omission does not overly constrain the implementation for future evolution
- Having application dependencies upon unspecified runtime behavior can lead to breakage
  - For example,
    - Hashing iteration order
    - Walking the stack trace looking for a method
    - Parsing message output from the launcher
    - ...etc.
- The platform is very good about preserving functional compatibility defined in the specifications
  - it would be bad if programs compiled and ran on the new platform, but did something different!
- When differences are found that are not compliant with the spec, either
  - the implementation is modified to fit the spec – often chosen for new code
  - the spec is changed to fit the implementation – often chosen for established code

Functional changes in the platform are not uncommon

- RFE 6693236: Verification of Version 51.0 Class Files
- RFE 6463998: Spec for java.lang.Float.parseFloat(String) and parseDouble(String) Updated to Document Exception
- RFE : java.lang.Character.isLowerCase/isUpperCase Methods Are Updated to Comply with the Specified Unicode Definition
- RFE 5045147: Inserting an Invalid Element into a TreeMap Throws an NPE
- RFE 5063507: Formatter.format() Now Throws FormatFlagsConversionMismatchException
- RFE 6621689: The Behavior for Several java.nio.channels.DatagramChannel Methods have Changed
- RFE 4640544: Updated Socket Channel Functionality
- RFE 7042594: Spec for java.awt.Color Method Documents Potential Exception
- RFE 6315717: The MouseEvent.getButton() method may return values outside of the [0-3] range
- RFE 6802853: Invoking Windows.setBackground may result in an UnsupportedOperationException exception
- RFE 7023011: Toolkit.getPrintJob(Frame, String, Properties) now throws NullPointerException
- RFE 4714232: The sun.awt.exception.handler System Property has Been Replaced with Official API
- RFE : Handling Certain Color Spaces, as Indicated in the JPEG Spec are Now Optional
- RFE 4700857: Separation of User Locale and User Interface Locale
- RFE 5108776: Reliable Event Handling has been Added to the JMX API
- RFE Out-of-the-Box JMX Management Has a New Keyword for ReadWrite Access
- … and more

java.lang
Class Float

Simple example



parseFloat

```
public static float parseFloat(String s)
                        throws NumberFormatException
```

Java™ Platform, Standard Edition 6 API Specification

Returns a new `float` initialized to the value represented by the specified `String`, as performed by the `valueOf` method of class `Float`.

Parameters:
`s` - the string to be parsed.
Returns:
the `float` value represented by the string argument.
Throws:
`NumberFormatException` - if the string does not contain a parsable `float`.
Since:
1.2
See Also:
`valueOf(String)`

What is the problem here?

Specification

Application

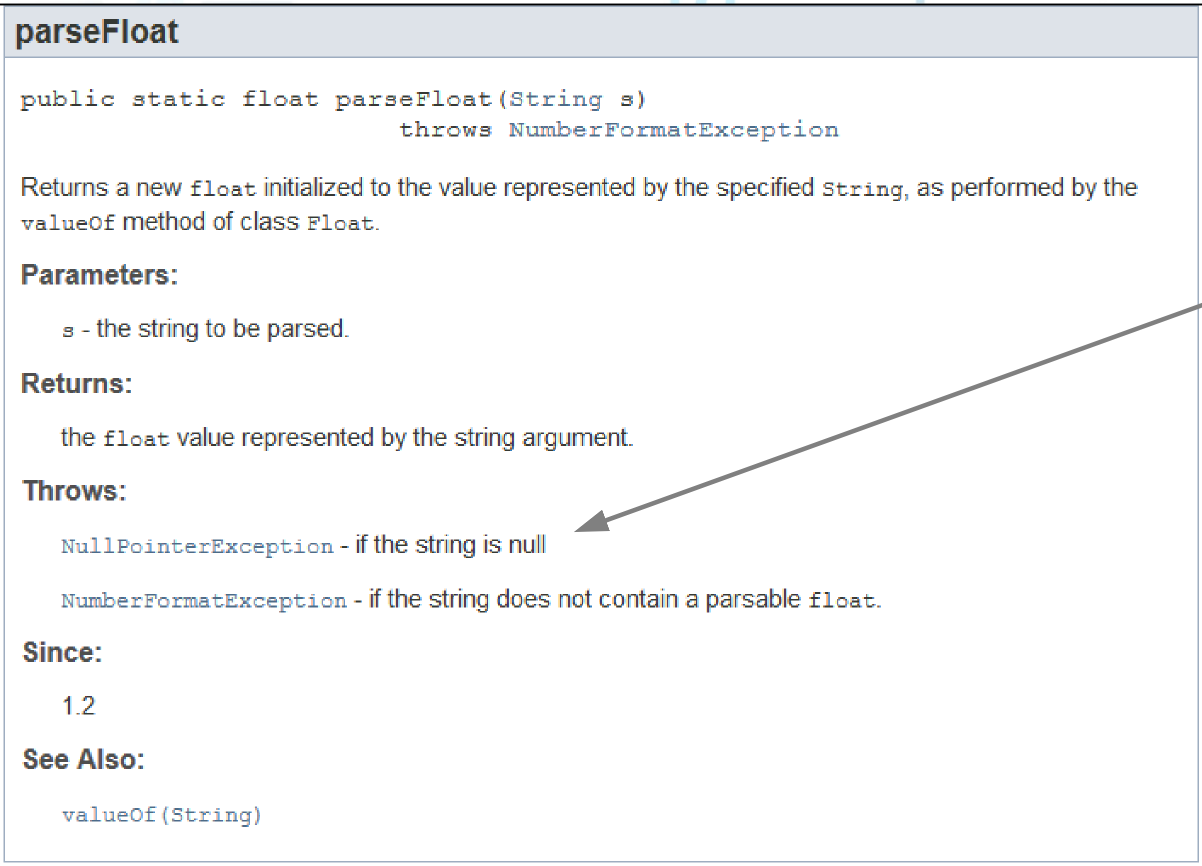
```
public class Parser {

    public static void main(String[] args) {
        System.out.println(Float.parseFloat(null));
    }
}
```

Result

```
Exception in thread "main" java.lang.NullPointerException
at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:989)
at java.lang.Float.parseFloat(Float.java:422)
at Parser.main(Parser.java:5)
```

java.lang

Class Float



parseFloat

```
public static float parseFloat(String s)
                        throws NumberFormatException
```

Returns a new `float` initialized to the value represented by the specified `String`, as performed by the `valueOf` method of class `Float`.

Parameters:

`s` - the string to be parsed.

Returns:

the `float` value represented by the string argument.

Throws:

`NullPointerException` - if the string is null

`NumberFormatException` - if the string does not contain a parsable `float`.

Since:

1.2

See Also:

`valueOf(String)`

Java™ Platform, Standard Edition 7 API Specification

In this case, the decision was made to change the spec to fit the implementation.

Current behaviour is well established, and not unreasonable.

Simply changing a comment can affect compatibility!

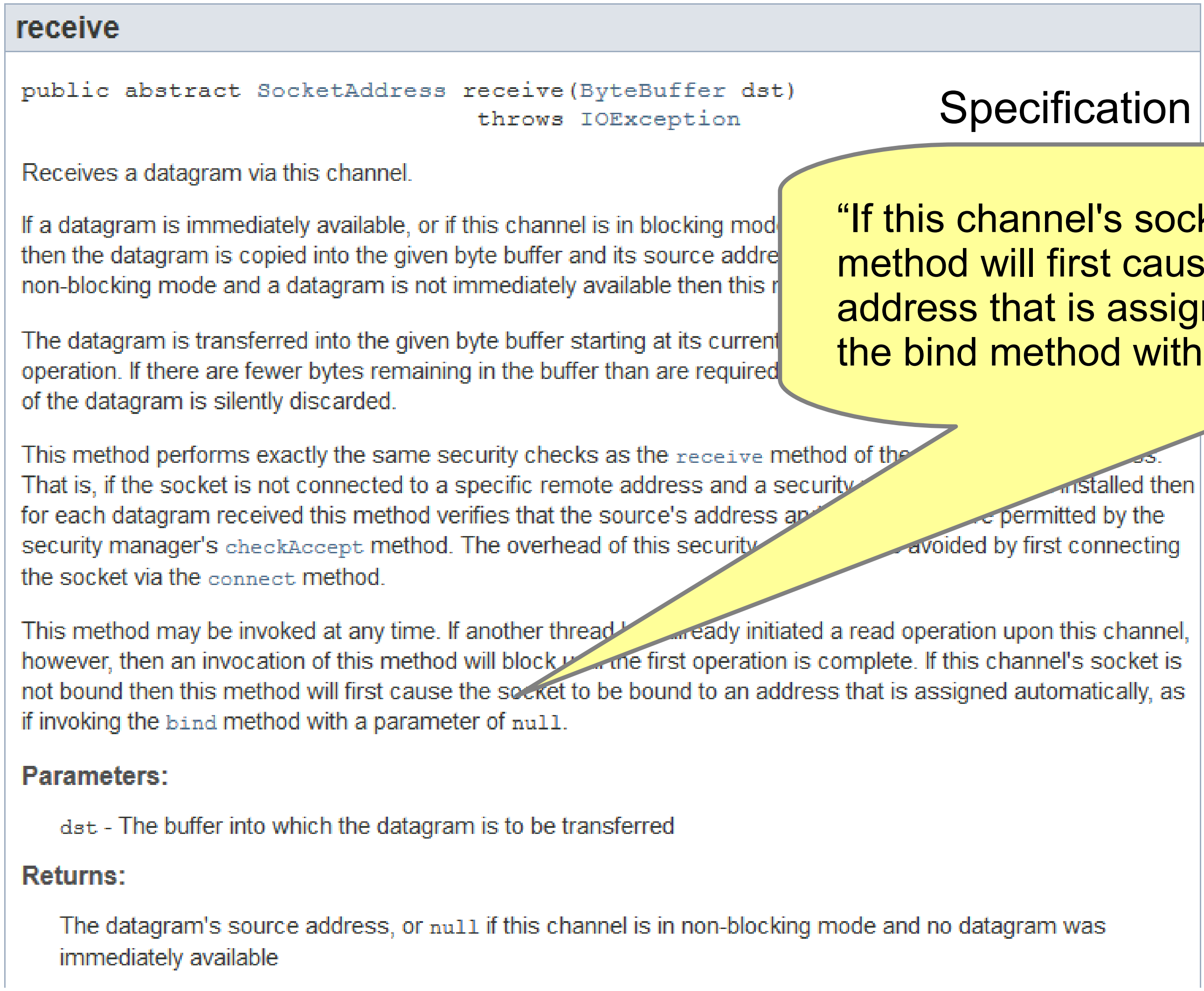
```
public class Parser {

    public static void main(String[] args) {
        System.out.println(Float.parseFloat(null));
    }
}
```

```
Exception in thread "main" java.lang.NullPointerException
at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:989)
at java.lang.Float.parseFloat(Float.java:422)
at Parser.main(Parser.java:5)
```

java.nio.channels

Class DatagramChannel

Further example



receive

```
public abstract SocketAddress receive(ByteBuffer dst)
                               throws IOException
```

Receives a datagram via this channel.

If a datagram is immediately available, or if this channel is in blocking mod
then the datagram is copied into the given byte buffer and its source addre
non-blocking mode and a datagram is not immediately available then this r

The datagram is transferred into the given byte buffer starting at its current
operation. If there are fewer bytes remaining in the buffer than are required
of the datagram is silently discarded.

This method performs exactly the same security checks as the `receive` method of the s.
That is, if the socket is not connected to a specific remote address and a security nstalled then
for each datagram received this method verifies that the source's address an e permitted by the
security manager's `checkAccept` method. The overhead of this security avoided by first connecting
the socket via the `connect` method.

This method may be invoked at any time. If another thread ready initiated a read operation upon this channel,
however, then an invocation of this method will block u the first operation is complete. If this channel's socket is
not bound then this method will first cause the socket to be bound to an address that is assigned automatically, as
if invoking the `bind` method with a parameter of `null`.

Parameters:

`dst` - The buffer into which the datagram is to be transferred

Returns:

The datagram's source address, or `null` if this channel is in non-blocking mode and no datagram was immediately available

Specification enhanced to say:

"If this channel's socket is not bound then this method will first cause the socket to be bound to an address that is assigned automatically, as if invoking the bind method with a parameter of null."

Experience now shows the original design choice was poor, but fixing it is a 'breaking' change.

`DatagramChannel`'s methods `#send`, `#receive`, and `#connect` behavior were changed when invoked on an unbound socket. These methods used to return `null`, but now do an implicit bind and continue...

To restore the previous behavior, the `sun.nio.ch.bugLevel` property can be set to the value of "1.4", "1.5", or "1.6"

More subtle behavioural changes

- Not all functional and behavioural changes are so clearly seen...

- Side effects from changing the VM specification
  - class file format changed in Java 5 to allow for class literals in the constant pool.
  - so code such as

    ```
        if (foo instanceof YourClass.class)
    ```

    doesn't cause **`YourClass`** to be loaded in Java 5.0 onwards
    - maybe **`YourClass`** has a static intializer that does something (fails early etc) when loaded
    - code compiled with 1.4.2 will still work, but show the old class init behavior

- Side effects from optimizations
  - simple optimizations by the class library developers or JIT developers may break your assumptions
    - eg. internal hashtable ordering breaks compatiblity of iteration ordering
    - jitted code can re-arrange some variable read and store within the bounds of the memory model

- Side effects from differences in implementation
  - **`Class.getName()`** returns an interned string vs. new string in Oracle's implementation- then it is used in an identity hash table assuming it was interned later
  - locks used and ordering of lock in class loading e.g. class loader lock -> deadlock

Functional compatibility in the real world

- Functional compatibility is tested using the technology compatibility test suite
- Functional fidelity is limited to the claims made in the specifications
- As the platform evolves functional compatibility breakages are dealt with by:
  - Changing the implementation to match the spec
  - Changing the spec to match the implementation
  - Providing some backwards compatibility options
- Changes to existing function are only considered at major platform release boundaries
  - … and then only after careful consideration
  - Highly unlikely in update and maintenance releases
  - Reflects the 'cost' of moving onto new updates
- Changes can be hard to spot and may result in different behavior only on edge-cases
  - Requires extensive testing of the application to mitigate the effects
- Resolution requires understanding the intent of the application

Types of Compatibility

- Functional compatibility : does running your program have the same effect ?
- Source compatibility : does your source code result in the same set of binaries ?
- Binary compatibility : do your binaries successfully link to the new platform binaries ?

Source compatibility

- Does your source code result in the 'same' set of binaries as the platform changes?

- Most obviously, does the source still compile on the later versions of Java?
  – language changes may invalidate previously acceptable programs
  – may make previously invalid programs valid – but not such an interesting set

- Main requirement is, if I compile this source again on a different/newer Java implementation does the compiler resolve your application's name references to the same or equivalent things in the new binary?

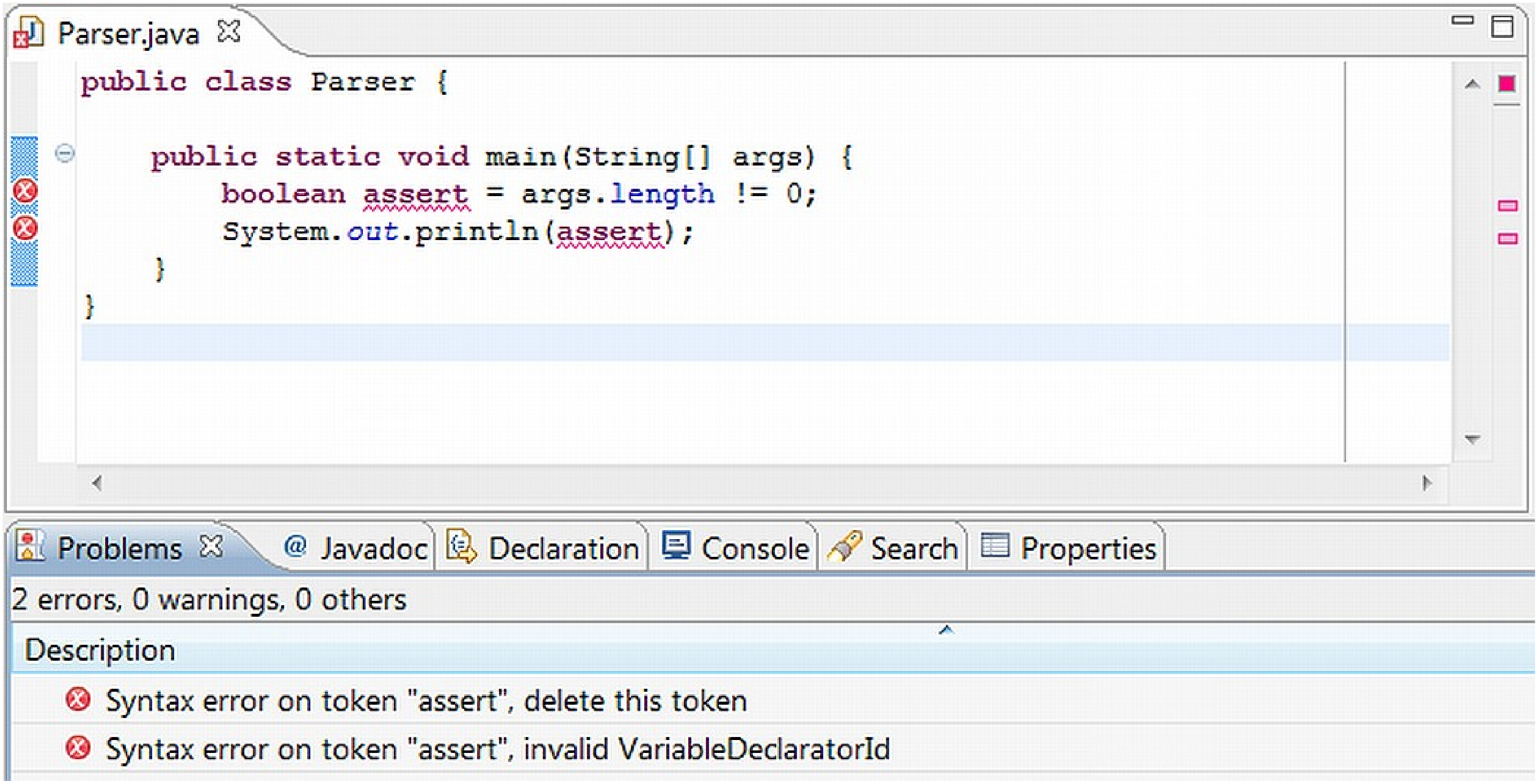
- Why would that not happen?
  – introduction of new language keywords (strictfp, assert, enum)
  – introduction of new types may cause ambiguity where non existed in prior releases

Simple example

```
public class Parser {

    public static void main(String[] args) {
        boolean assert = args.length != 0;
        System.out.println(assert);
    }
}
```

```
java version "1.4.2"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2)
Classic VM (build 1.4.2, J2RE 1.4.2 IBM Windows 32 build cn142sr1aifx-20060318
(142SR1a + 98226) (JIT enabled: jitc))

false
```

Compiling the same source on Java 5.0 ...



```
public class Parser {

    public static void main(String[] args) {
        boolean assert = args.length != 0;
        System.out.println(assert);
    }
}
```

2 errors, 0 warnings, 0 others

Description
Syntax error on token "assert", delete this token
Syntax error on token "assert", invalid VariableDeclaratorId

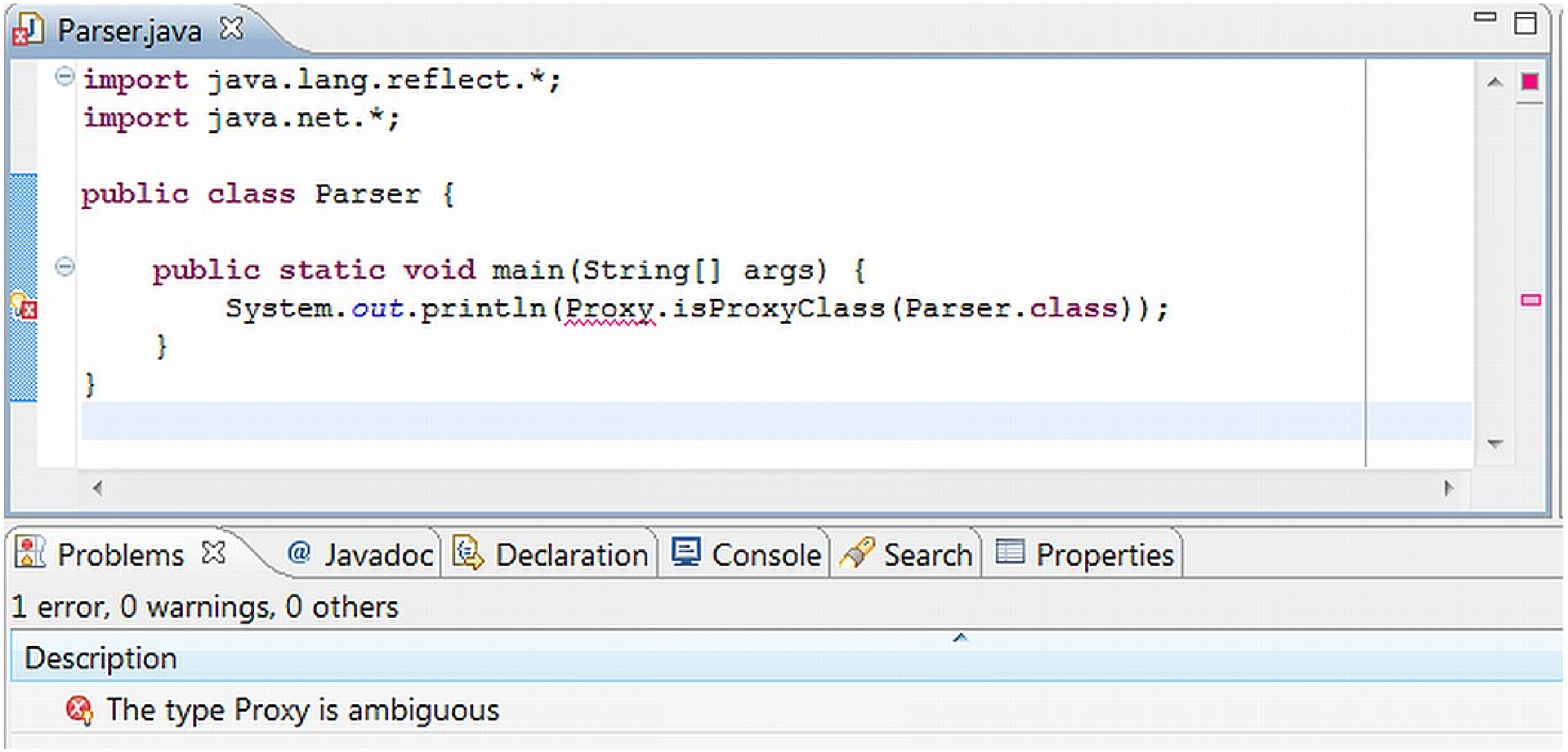
"**`assert`**" was a legal variable name in Java 1.4.2, but became a keyword of the Java language in Java 5.0

An example of naming issues in the JDK

```
import java.lang.reflect.*;
import java.net.*;

public class Parser {

    public static void main(String[] args) {
        System.out.println(Proxy.isProxyClass(Parser.class));
    }
}
```

```
java version "1.4.2"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2)
Classic VM (build 1.4.2, J2RE 1.4.2 IBM Windows 32 build cn142sr1aifx-20060318
(142SR1a + 98226) (JIT enabled: jitc))

false
```

Compiling the same source on Java 5.0 ...



```
import java.lang.reflect.*;
import java.net.*;

public class Parser {

    public static void main(String[] args) {
        System.out.println(Proxy.isProxyClass(Parser.class));
    }
}
```

1 error, 0 warnings, 0 others

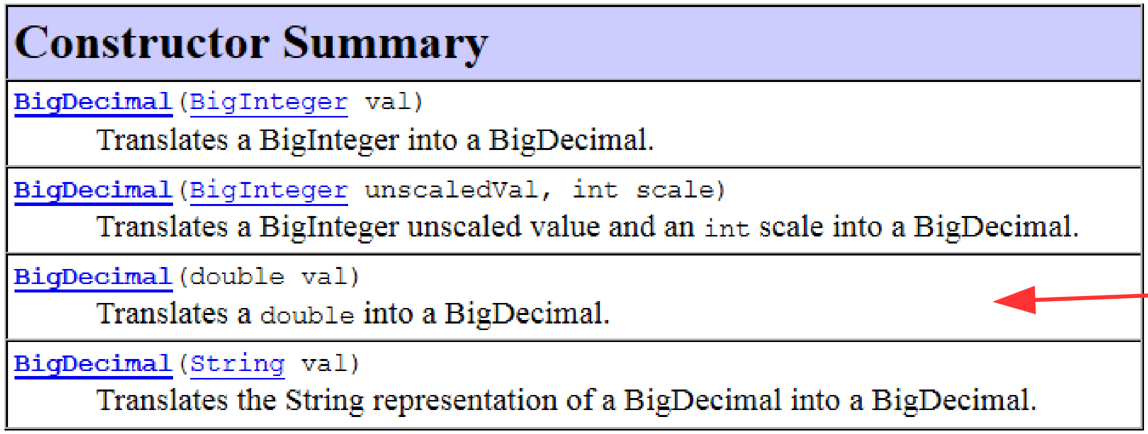
The type Proxy is ambiguous

A second type with the same name was introduced in Java 5.0.
This may happen in your application code too.

Method overloading

```
import java.math.BigDecimal;

public class Parser {

    public static void main(String[] args) {
        System.out.println(new BigDecimal(1));
    }
}
```

Actual parameter `(int)1` undergoes widening primitive conversion



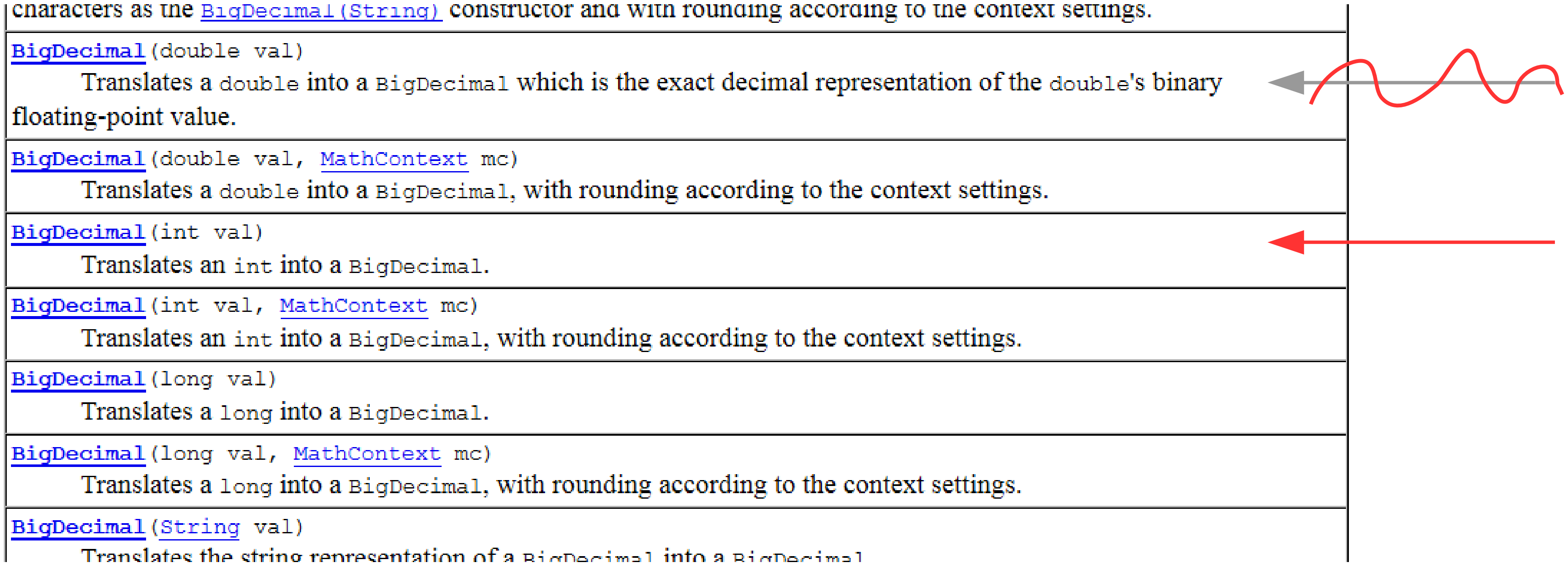
Constructor Summary
`BigDecimal(BigInteger val)` Translates a BigInteger into a BigDecimal.
`BigDecimal(BigInteger unscaledVal, int scale)` Translates a BigInteger unscaled value and an `int` scale into a BigDecimal.
`BigDecimal(double val)` Translates a `double` into a BigDecimal.
`BigDecimal(String val)` Translates the String representation of a BigDecimal into a BigDecimal.

Matched constructor

Compiling the same source on Java 5.0 ...

```
import java.math.BigDecimal;

public class Parser {

    public static void main(String[] args) {
        System.out.println(new BigDecimal(1));
    }
}
```

Adding a more specific constructor means the same program now compiles against a different method...



characters as the BigDecimal(String) constructor and with rounding according to the context settings.
BigDecimal(double val)   Translates a double into a BigDecimal which is the exact decimal representation of the double's binary floating-point value.
BigDecimal(double val, MathContext mc)   Translates a double into a BigDecimal, with rounding according to the context settings.
BigDecimal(int val)   Translates an int into a BigDecimal.
BigDecimal(int val, MathContext mc)   Translates an int into a BigDecimal, with rounding according to the context settings.
BigDecimal(long val)   Translates a long into a BigDecimal.
BigDecimal(long val, MathContext mc)   Translates a long into a BigDecimal, with rounding according to the context settings.
BigDecimal(String val)   Translates the string representation of a BigDecimal into a BigDecimal.

Source compatibility in the real world

- Java's update releases will maintain source compatibility, but major releases may break source compatibility with sufficient, considered justification
  - General policy:
    - maintenance releases dot-dot releases do not introduce new language features or APIs
    - functionality and major releases are upwards but not downwards source compatible
    - deprecated APIs are only there to support backwards compatibility, and programs are advised to move off these APIs

- Some code is reachable in the platform (e.g. **`com.sun.*`**) but does not form part of the Java platform specification, so comes with no compatibility or WORA promises.
  - looking ahead to modularity, these may be permanently hidden by module scope visibility

- The same source may not compile correctly due to
  - changes to the language
  - changes to the set of types referrable during compilation
- Some of these problems can be resolved by 'mechanical' transform of source code
  - e.g. type and variable renaming

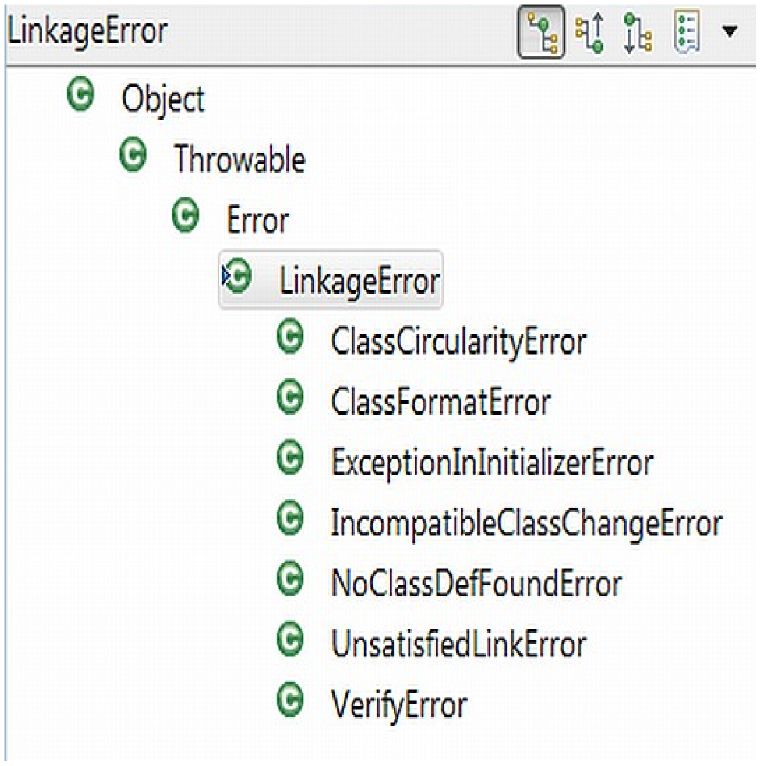
Types of Compatibility

- Functional compatibility : does running your program have the same effect ?
- Source compatibility : does your source code result in the same set of binaries ?
- Binary compatibility : do your binaries successfully link to the new platform binaries ?

Binary compatibility

"A change to a type is binary compatible with (equivalently, does not break binary compatibility with) pre-existing binaries if pre-existing binaries that previously linked without error will continue to link without error. "
http://docs.oracle.com/javase/specs/jls/se7/html/jls-13.html

- What assurances are given for your existing binary program files?
- Maybe your applications are third-party libraries
  – you want to ensure that new versions of your program are drop-in replacements for the current version
  – important to understand as it is often infeasible or impossible to recompile the whole application stack
- Java implementer: how can Java evolve and ensure we continue to run the billions of lines of Java code out there?
- Each Java version goal is to be upwards binary compatible with earlier versions
  – code compiled for Java 5 will work on Java 6 and Java 7 etc, with caveats (to be discussed)



```
LinkageError
  Object
    Throwable
      Error
        LinkageError
          ClassCircularityError
          ClassFormatError
          ExceptionInInitializerError
          IncompatibleClassChangeError
          NoClassDefFoundError
          UnsatisfiedLinkError
          VerifyError
```

Binary file version loading compatibility

```
$ java Test
The java class could not be loaded. java.lang.UnsupportedClassVersionError: Test
(Unsupported major.minor version 50.0)
```

- VM checks class file format generated by the compiler

Java version	Version accepted
1.0.2	45.0 → 45.3
1.1.x	45.0 → 45.66535
1.*n*	45 → (44+*n*).0

e.g. 1.6 45.0 -> 50.0

- Tools like javac can emit specific class file format for different sources
  - e.g., javac -target 1.4
  - of course, your code must only use the APIs that are available in the earlier runtime...
    - Use javac -source and -bootclasspath options

- Byte code optimizers and obfuscators may produce class files that violate class file format spec

Evolving Java APIs in the platform and applications (while preserving linkage)

- Some modifications are binary compatibility preserving
  - Adding a new API package, type, field, method
  - Adding an element to an annotation type with a default value
  - Adding or deleting unchecked exceptions thrown
  - Add, delete, or change static or instance initializers
  - Change abstract classes to non-abstract classes
  - Change final classes to non-final classes
  - Change body of method or constructor
  - Increase method access; that is, from protected access to public access
  - ...and more
- Others break binary compatibility
  - Deleting an API package, type, field, method
  - Changing a method name
  - Changing the method return type
  - Re-ordering type parameters
  - Adding a public or protected method to an interface
  - Changing a type from non-abstract to abstract
  - Adding checked exceptions thrown
  - ...and more

http://wiki.eclipse.org/Evolving_Java-based_APIs

Evolving Java APIs

- Some modifications are binary compatibility preserving
  – Adding a new API pa
  – Adding an element
  – Adding or deleting
  – Add, delete, or cha
  – Change abstract cl
  – Change final classe
  – Change body of me
  – Increase method a
  – ...and more

- Others break binary co
  – Deleting an API package, ty
  – Changing a method name
  – Changing the method return type
  – Re-ordering type parameters
  – Adding a public or protected method to an interface
  – Changing a type from non-abstract to abstract
  – Adding checked exceptions thrown
  – ...and more

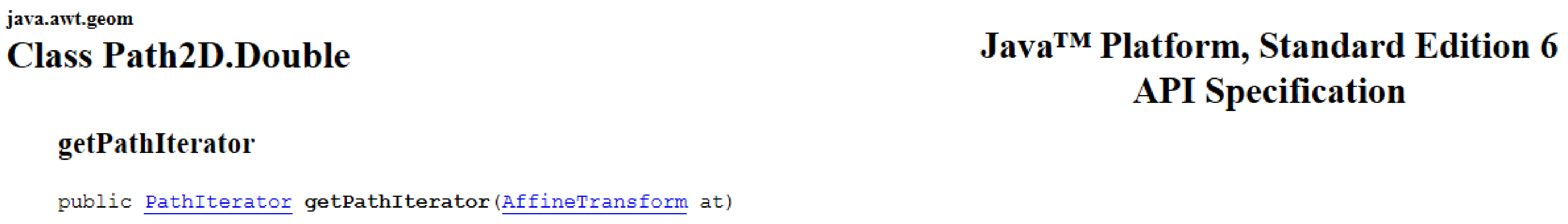
Virtual extension methods proposed for Java 8

```
interface Collection<T> {
   ...
   void forEach(Block<T> block)
      default Collections.<T>forEach;
}
```

Allows interfaces to evolve while preserving binary compatibility. Provides a default for implementing classes that do not implement the extension method.

http://wiki.eclipse.org/Evolving_Java-based_APIs

Example: modest, considered, binary breaking changes



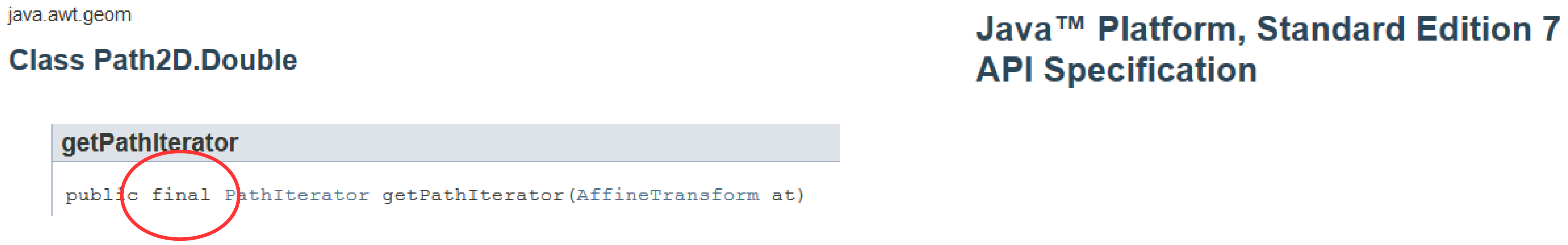
java.awt.geom

Class Path2D.Double

Java™ Platform, Standard Edition 6 API Specification

getPathIterator

```
public PathIterator getPathIterator(AffineTransform at)
```



java.awt.geom

Class Path2D.Double

Java™ Platform, Standard Edition 7 API Specification

getPathIterator

```
public final PathIterator getPathIterator(AffineTransform at)
```

`Path2D.Double` was never intended to be overridden. The spec was changed in Java 7.0 causing types that try to override to fail to link.

Serialization compatibility

- Serialization is used for object persistence and interchange (e.g. RMI)

- The binary representation of a serialized object contains a serialVersionUID embodying a type version identifier.

- Classes writing and reading instance representations must be compatible
  - If the class is `Serializable` then you must consider the impact of Class changes
  - Even internal, private methods and fields may affect the compatibility of serialized objects

- The specification tells you the list of compatible and incompatible changes
  - See `http://docs.oracle.com/javase/6/docs/platform/serialization/spec/version.html`

- General rule:
  - Take control of the version identifier by specifying it explicitly
    - `private static final long serialVersionUID = 1L;`
  - Update the version identifier when you are no longer compatible

Final thoughts...

Evolving Java code safely

- As an API producer you should consider the implications of changes on compatibility
- As an API consumer there are steps you can take to minimize impact of JDK compatibility changes

- API docs and compiler warnings are your friend

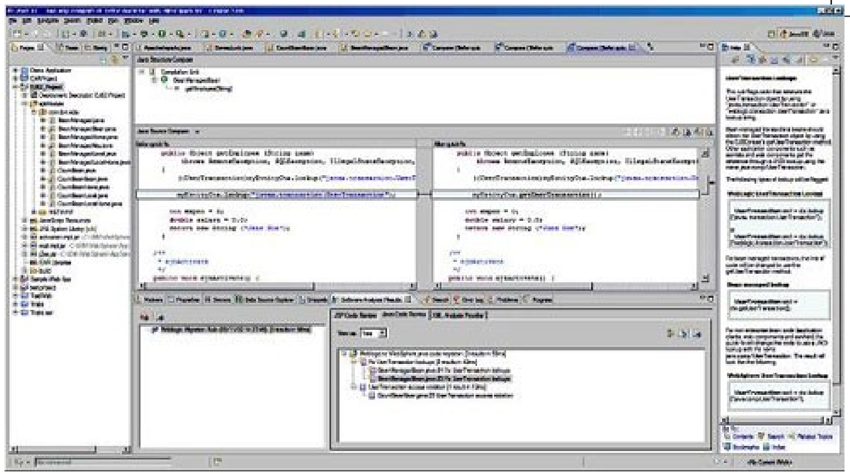
- Addressing compatibility and complexity
  – access to early builds in OpenJDK to help uncover issues
  – forum for direct discussion on proposals with designers
  – ability to submit fixes and workarounds, and get feedback on issues you may encounter
  – not a place to ask for help with your application, this is platform issues

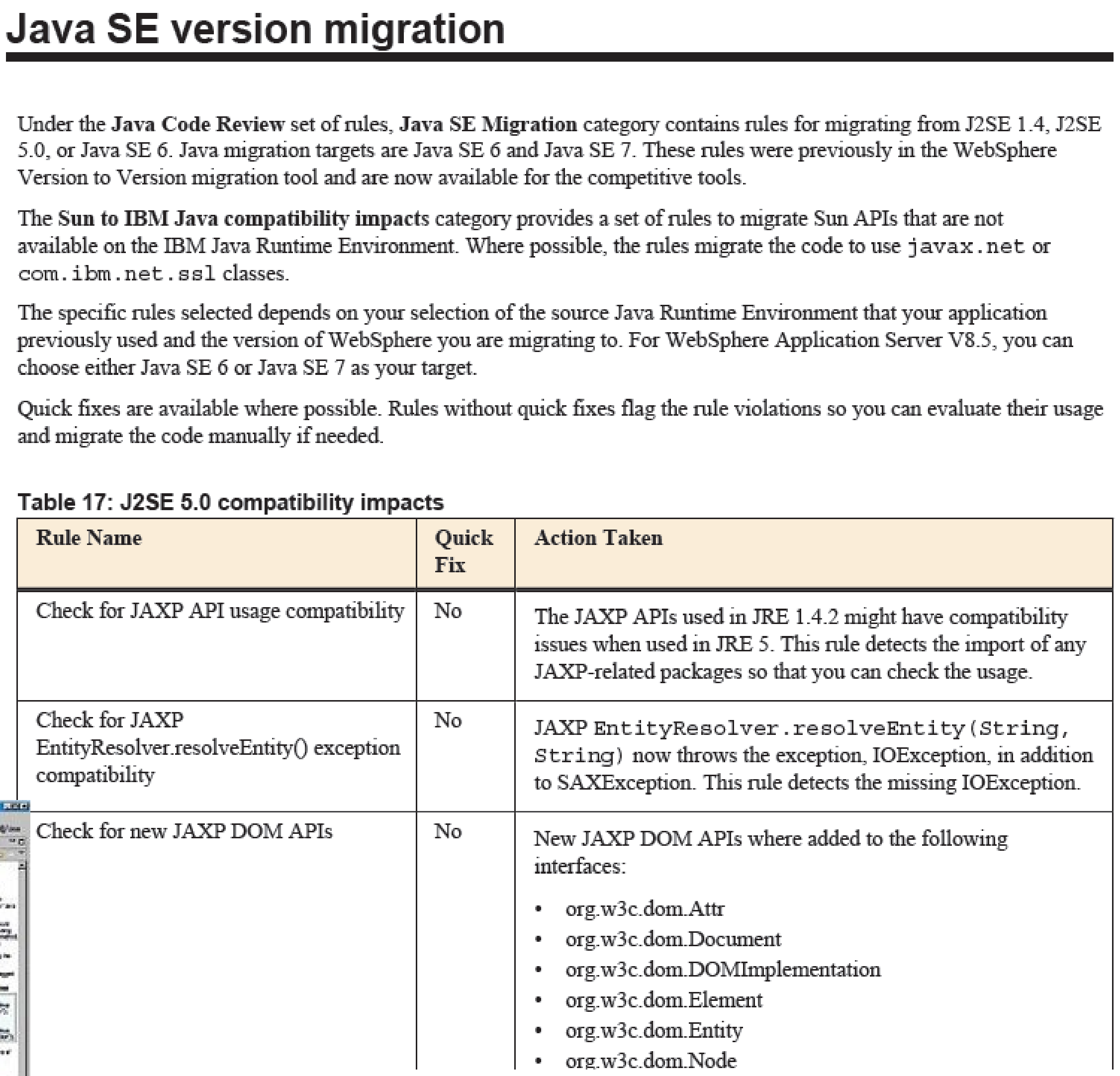
- Modularity will be a real test of application compliance
  – well-behaved applications will see benefits from a modular runtime
  – poorly behaved applications will have to run in a compatibility mode

IBM WebSphere Application Server Migration Toolkit

- Free download
- Contains rules covering functional, source, and binary compatibility impact
- Analyzes your source and offers advice and quick fixes

websphere migration toolkit





Java SE version migration

Under the **Java Code Review** set of rules, **Java SE Migration** category contains rules for migrating from J2SE 1.4, J2SE 5.0, or Java SE 6. Java migration targets are Java SE 6 and Java SE 7. These rules were previously in the WebSphere Version to Version migration tool and are now available for the competitive tools.

The **Sun to IBM Java compatibility impacts** category provides a set of rules to migrate Sun APIs that are not available on the IBM Java Runtime Environment. Where possible, the rules migrate the code to use `javax.net` or `com.ibm.net.ssl` classes.

The specific rules selected depends on your selection of the source Java Runtime Environment that your application previously used and the version of WebSphere you are migrating to. For WebSphere Application Server V8.5, you can choose either Java SE 6 or Java SE 7 as your target.

Quick fixes are available where possible. Rules without quick fixes flag the rule violations so you can evaluate their usage and migrate the code manually if needed.

Table 17: J2SE 5.0 compatibility impacts

Rule Name	Quick Fix	Action Taken
Check for JAXP API usage compatibility	No	The JAXP APIs used in JRE 1.4.2 might have compatibility issues when used in JRE 5. This rule detects the import of any JAXP-related packages so that you can check the usage.
Check for JAXP EntityResolver.resolveEntity() exception compatibility	No	JAXP `EntityResolver.resolveEntity(String, String)` now throws the exception, IOException, in addition to SAXException. This rule detects the missing IOException.
Check for new JAXP DOM APIs	No	New JAXP DOM APIs where added to the following interfaces: • org.w3c.dom.Attr • org.w3c.dom.Document • org.w3c.dom.DOMImplementation • org.w3c.dom.Element • org.w3c.dom.Entity • org.w3c.dom.Node

References

- **Get Products and Technologies:**
  - IBM Java Runtimes and SDKs:
    - https://www.ibm.com/developerworks/java/jdk/
  - IBM Monitoring and Diagnostic Tools for Java:
    - https://www.ibm.com/developerworks/java/jdk/tools/
  - IBM WebSphere Application Server Migration Toolkit
    - http://www.ibm.com/developerworks/websphere/downloads/migtoolkit/

- **Learn:**
  - IBM Java InfoCenter:
    - http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/index.jsp

- **Discuss:**
  - IBM Java Runtimes and SDKs Forum:
    - http://www.ibm.com/developerworks/forums/forum.jspa?forumID=367&start=0

Copyright and Trademarks