ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Software Engineering and Security Group

Type Checking for Reliable APIs

Maria Kechagia and Diomidis Spinellis
{mkechagia, dds}@aueb.gr

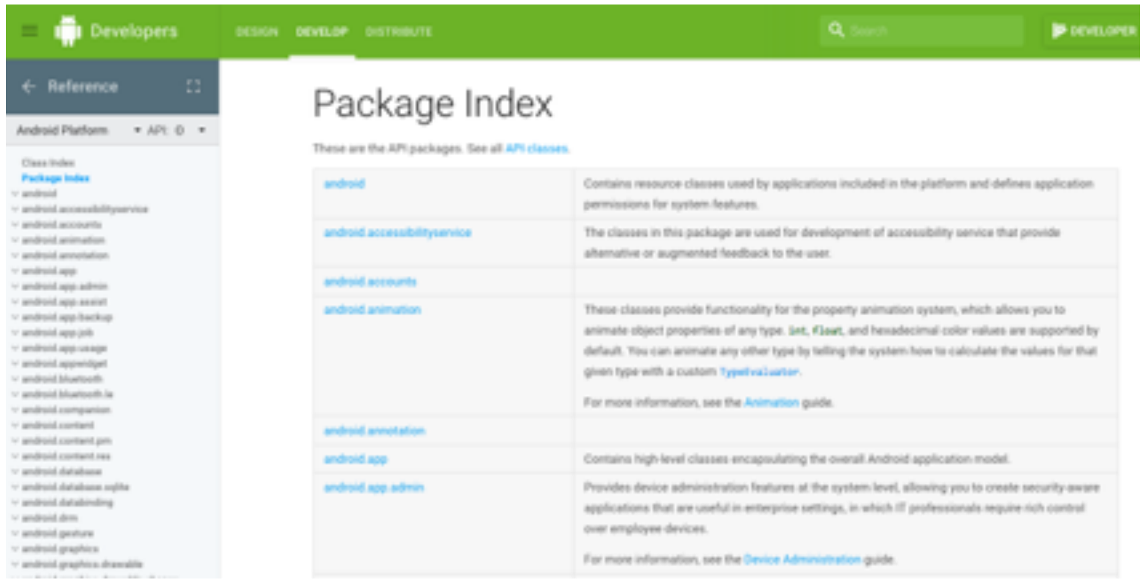
Athens University of Economics and Business

WAPI '17: 1st International Workshop on API Usage and Evolution

May 23rd 2017 - ICSE '17

Application Programming Interfaces (APIs)

- Bundles of interfaces that developers can use to build the main functionality of their client applications.



Developers DESIGN DEVELOP DISTRIBUTE

Reference

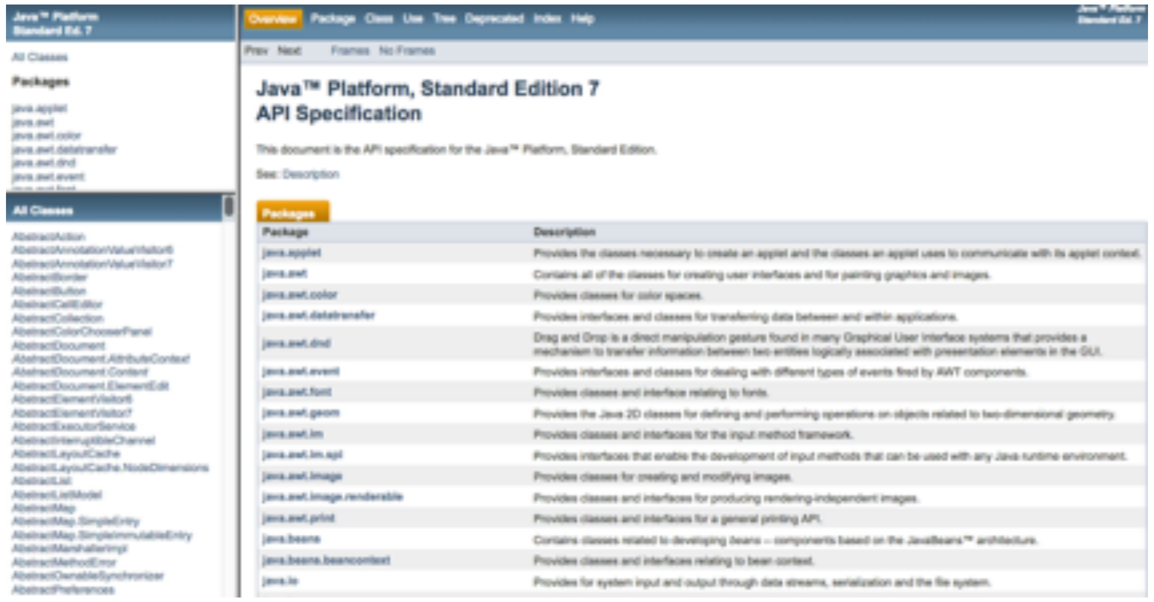
Android Platform API: O

Class Index
Package Index
android
android.accessibilityservice
android.accounts
android.animation
android.annotation
android.app
android.app.admin
android.app.assist
android.app.backup
android.app.job
android.app.usage
android.appwidget
android.bluetooth
android.bluetooth.le
android.companion
android.content
android.content.pm
android.content.res
android.database
android.database.sqlite
android.databinding
android.drm
android.gesture
android.graphics
android.graphics.drawable

Package Index

These are the API packages. See all API classes.

android	Contains resource classes used by applications included in the platform and defines application permissions for system features.
android.accessibilityservice	The classes in this package are used for development of accessibility service that provide alternative or augmented feedback to the user.
android.accounts	
android.animation	These classes provide functionality for the property animation system, which allows you to animate object properties of any type. int, float, and hexadecimal color values are supported by default. You can animate any other type by telling the system how to calculate the values for that given type with a custom TypeEvaluator. For more information, see the Animation guide.
android.annotation	
android.app	Contains high-level classes encapsulating the overall Android application model.
android.app.admin	Provides device administration features at the system level, allowing you to create security-aware applications that are useful in enterprise settings, in which IT professionals require rich control over employee devices. For more information, see the Device Administration guide.



Java™ Platform Standard Ed. 7

All Classes

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event

All Classes

AbstractAction
AbstractAnnotationValueVisitor6
AbstractAnnotationValueVisitor7
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChooserPanel
AbstractDocument
AbstractDocument.AttributeContext
AbstractDocument.Content
AbstractDocument.ElementEdit
AbstractElementVisitor6
AbstractElementVisitor7
AbstractExecutorService
AbstractInterruptibleChannel
AbstractLayoutCache
AbstractLayoutCache.NodeDimensions
AbstractList
AbstractListModel
AbstractMap
AbstractMap.SimpleEntry
AbstractMap.SimpleImmutableEntry
AbstractMarshallerImpl
AbstractMethodError
AbstractOwnableSynchronizer
AbstractPreferences

Overview Package Class Use Tree Deprecated Index Help

Prev Next Frames No Frames

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.
java.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.
java.awt.print	Provides classes and interfaces for a general printing API.
java.beans	Contains classes related to developing beans -- components based on the JavaBeans™ architecture.
java.beans.beancontext	Provides classes and interfaces relating to bean context.
java.io	Provides for system input and output through data streams, serialization and the file system.

Can we make APIs more reliable?

Type Checking

“*A type checker provides a compile-time guarantee that certain errors cannot occur. For example, Java’s type checker guarantees that a standard Java program cannot exit with a method-not-found exception. Unfortunately, standard type systems and checkers can’t help developers find and prevent all the errors that they care about in practice. Therefore, developers often reason manually about code correctness — a daunting task, especially in the face of incomplete or inconsistent documentation.*” [1]

[1] Werner Dietl, Stephanie Dietzel, Michael D. Ernst, Kivanç Muşlu, and Todd W. Schiller. 2011. Building and using pluggable type-checkers. In Proceedings of the 33rd International Conference on Software Engineering (ICSE '11). ACM, New York, NY, USA, 681-690.

Motivating Examples (1)

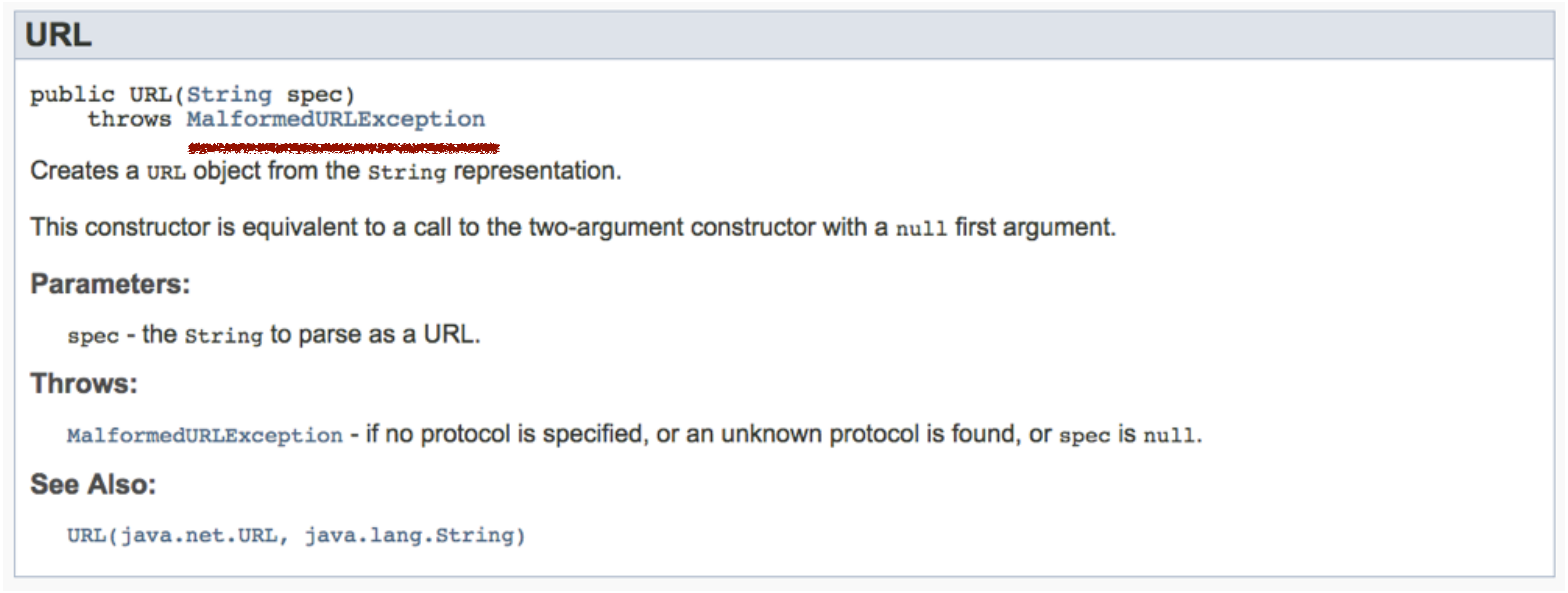
```
import java.net.MalformedURLException;
import java.net.URL;

public class URLReader {
    public static void main(String[] args) {
        try {
            /* URL signature:
             * URL(String spec) throws MalformedURLException
             */

            // Case 1: user input
            URL url1 = new URL(args[0]);

             // Case 2: constant url
            URL url2 = new URL("http://www.example.com");

            // ...
        } catch(MalformedURLException e) {
            System.err.println("Invalid URL");
            // Give some new URL or
           //use default URL ...
        }

        // ...
    }
}
```

input from the user, database etc.

static constant

Is the `MalformedURLException` necessary?

Java API



URL

```
public URL(String spec)
    throws MalformedURLException
```

Creates a `URL` object from the `String` representation.

This constructor is equivalent to a call to the two-argument constructor with a `null` first argument.

Parameters:

`spec` - the `String` to parse as a URL.

Throws:

`MalformedURLException` - if no protocol is specified, or an unknown protocol is found, or `spec` is `null`.

See Also:

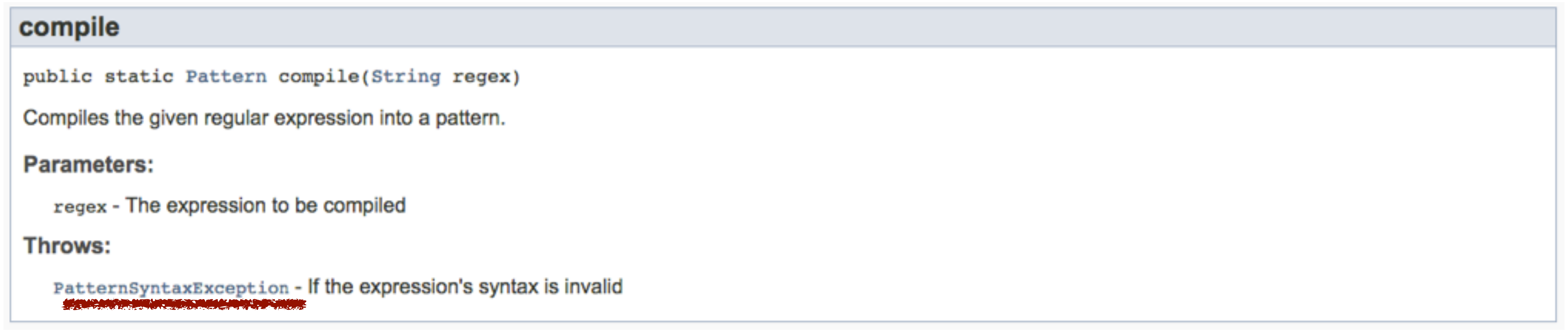
`URL(java.net.URL, java.lang.String)`

Motivating Examples (2)

input from the user;
any exception?

```
import java.util.regex.Pattern;

public class Parser {

    public static void main(String[] args) {
        // Case 4: User input
        Pattern pattern1 = Pattern.compile(args[0]);

       /* Pattern compile(String regex)
        *    throws PatternSyntaxException
        */

        // Case 5: Constant value
        Pattern pattern2 = Pattern.compile("^xy");

        // ...
    }
}
```

Java API



compile

```
public static Pattern compile(String regex)
```

Compiles the given regular expression into a pattern.

Parameters:

`regex` - The expression to be compiled

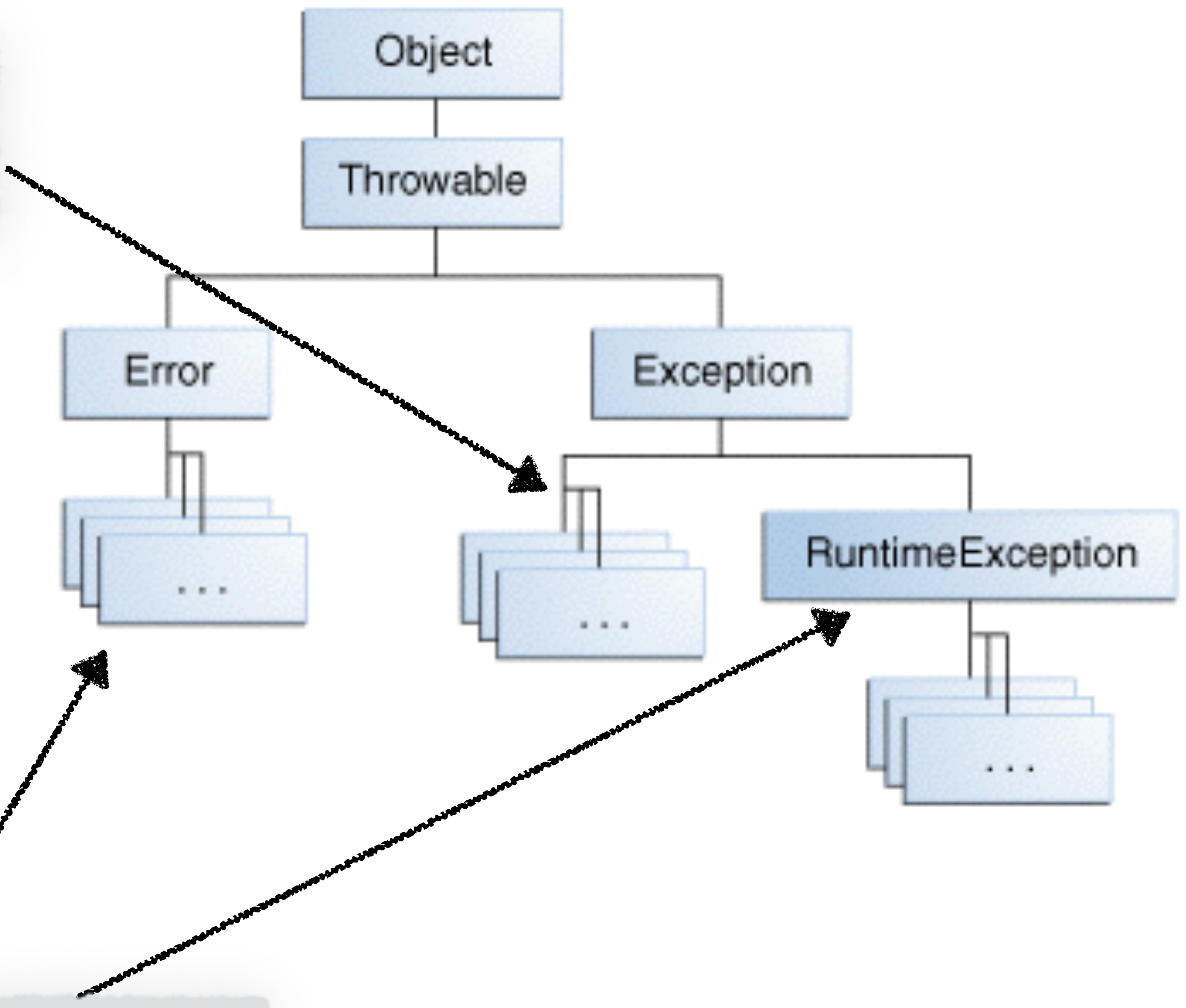
Throws:

`PatternSyntaxException` - If the expression's syntax is invalid

Java Exception Hierarchy

Checked Exceptions

(IOException, SQLException, **MalformedURLException**, XMLParseException, DataFormatException, TimeoutException etc.)



Unchecked Exceptions

(NullPointerException, IllegalArgumentException, ArrayIndexOutOfBoundsException, **PatternSyntaxException**, ArithmeticException etc.)

Solution

We propose to configure **at compile time** the checking associated with Application Programming Interfaces' methods that can receive possibly **malformed values** (e.g. erroneous user inputs and problematic retrieved records from databases) and thus cause **application execution failures**.

Malformed URL

```
import java.net.MalformedURLException;
import java.net.URL;

public class URLReader {
    public static void main(String[] args) {
        try {
            /* URL signature:
             * URL(String spec) throws MalformedURLException
             */

            // Case 1: user input
            URL url1 = new URL(args[0]);

            // ...
        } catch(MalformedURLException e) {
            System.err.println("Invalid URL");
            // Give some new URL or
           //use default URL ...
        }

        // Case 2: constant url
        URL url2 = new URL("http://www.example.com");

        // ...
    }
}
```

convert at compile time and throw unchecked exception

```
String u = "http://www.example.com/";
URL url3 = new URL(ThrowingUncheckedException.instance,
@WellformedURL u);
```

Malformed Pattern

```
import java.util.regex.InvalidPatternCheckedException;
import java.util.regex.Pattern;

public class Parser {

    public static void main(String[] args) {
         try {
            // Case 4: User input
            Pattern pattern1 = Pattern.compile(args[0]);

            // ...
         } catch(InvalidPatternCheckedException e) {
            System.err.println("Invalid pattern");
            // Give a new correct pattern ...
         }

        /* Pattern compile(String regex)
         *   throws PatternSyntaxException
         */

        // Case 5: Constant value
        Pattern pattern2 = Pattern.compile("^xy");

        // ...
    }
}
```

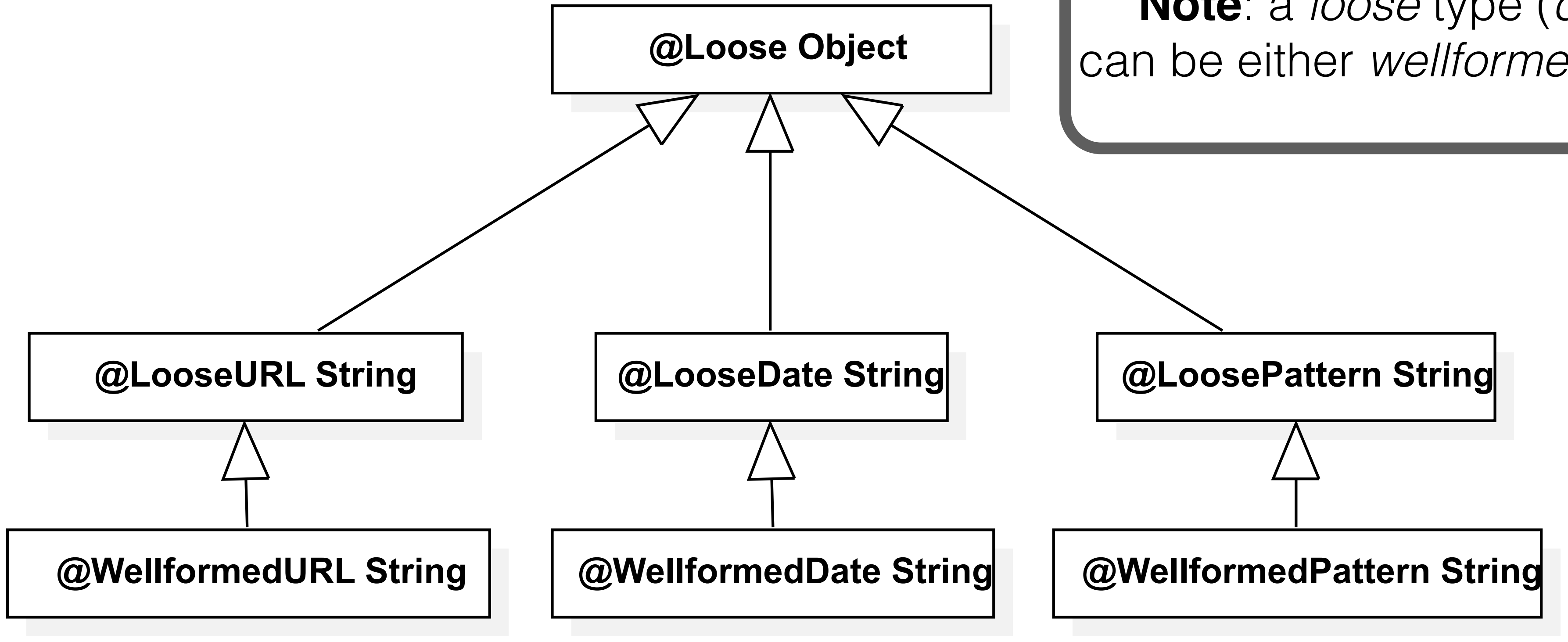
convert at compile time and throw checked exception

```
// Case 6: User input
Pattern pattern =
Pattern.compile(ThrowingCheckedE
xception.instance, args[0]);
```

The Type-Checker

- ✓ Extends the Java's built-in type-checker and prevents from errors due to invalid external inputs
- ✓ Improves developers productivity using checked exceptions only where it is necessary
- ✓ It uses type qualifiers (annotations)
- ✓ It is based on the Checker Framework

Type Qualifiers' Hierarchy



Note: a *loose* type (*default type*) can be either *wellformed* or *erroneous*

New Type Checker for malformed input values

1. **Type Qualifiers:**

   `@Wellformed T’, e.g. @Wellformed String url`
   `@Loose T, e.g. @Loose String url`

2. **Type Introduction Rules:**

   $\left.\begin{array}{l}\texttt{constant}\text{ values are all well-formed}\\ \texttt{@Wellformed T}\text{ types are all well-formed}\end{array}\right\}$ known at compile time

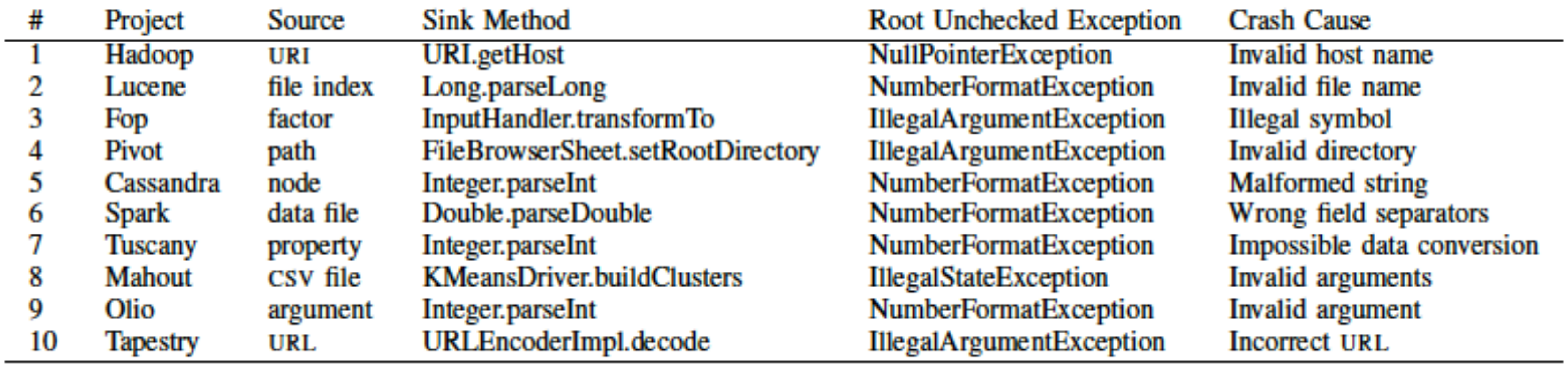
3. **Type Inference Rules:**

$$\frac{\Gamma \vdash t1 : T \quad \Gamma \vdash t2 : T}{\Gamma \vdash funct(t1, t2) : T} \qquad \frac{\Gamma \vdash t1 : T' \quad \Gamma \vdash t2 : T}{\Gamma \vdash funct(t1, t2) : T'} \qquad \frac{\Gamma \vdash t1 : T' \quad \Gamma \vdash t2 : T'}{\Gamma \vdash funct(t1, t2) : T'}$$

4. **At compile time:**

   - `f(@wellformed x)` throws **unchecked** exception
   - `f(x)`throws **checked** exception

Apache Projects



#	Project	Source	Sink Method	Root Unchecked Exception	Crash Cause
1	Hadoop	URI	URI.getHost	NullPointerException	Invalid host name
2	Lucene	file index	Long.parseLong	NumberFormatException	Invalid file name
3	Fop	factor	InputHandler.transformTo	IllegalArgumentException	Illegal symbol
4	Pivot	path	FileBrowserSheet.setRootDirectory	IllegalArgumentException	Invalid directory
5	Cassandra	node	Integer.parseInt	NumberFormatException	Malformed string
6	Spark	data file	Double.parseDouble	NumberFormatException	Wrong field separators
7	Tuscany	property	Integer.parseInt	NumberFormatException	Impossible data conversion
8	Mahout	CSV file	KMeansDriver.buildClusters	IllegalStateException	Invalid arguments
9	Olio	argument	Integer.parseInt	NumberFormatException	Invalid argument
10	Tapestry	URL	URLEncoderImpl.decode	IllegalArgumentException	Incorrect URL

Future Work

- Test the implementation of the type-checker
- Evaluate the type-checker on software projects
- Validate the results of the type-checker using stack traces from JIRA
- Implementation of useful checkers (URL, SQL, Regex, …)

Discussion Point

- The role of type systems and specialized type checking in API design.

Thank you!