Java 8:
Create The Future

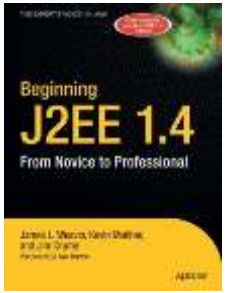
Jim Weaver
Java Technology Ambassador
@JavaFXpert
james.weaver@oracle.com

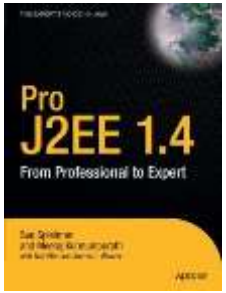
Program Agenda

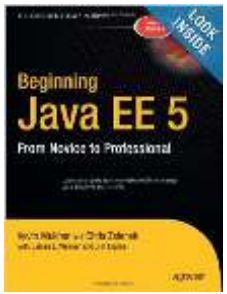
- Java SE 8: Enhancing the Core Java Platform
- Java ME 8: Building The Internet of Things
- NetBeans 8: The IDE For Java 8
- Where Next?

About the presenter

Author of several Java/JavaFX books







About the presenter

Co-leader of IoT & JavaFX communities at java.net

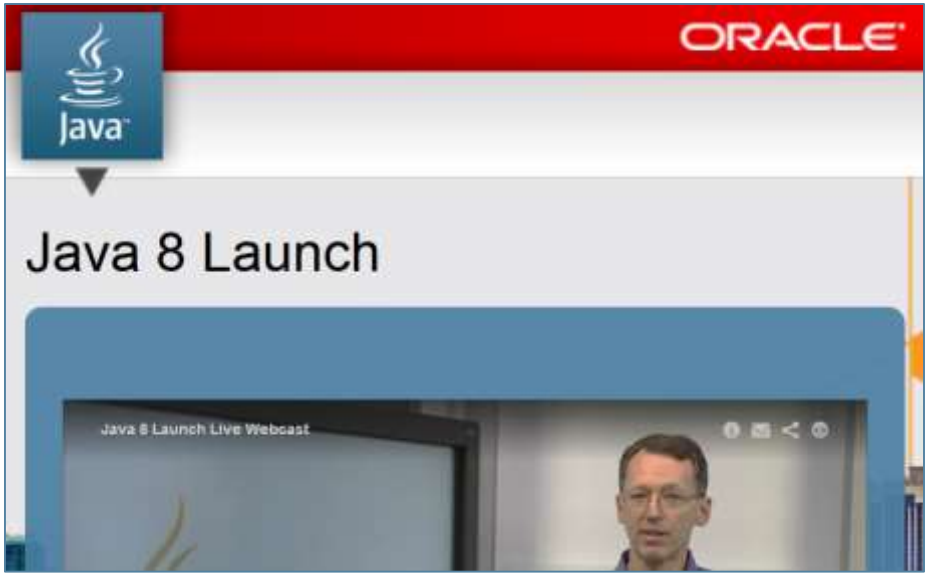
iotcommunity.net

JavaFX™ community

javafxcommunity.com

Java 8 Launch Event & Resource Videos

oracle.com/java8



IoT Developer Challenge

- Show the world what you can do with Java + IoT for a chance to win a trip to JavaOne for you and two team members.

oracle.com/java8

Key Dates:
Submissions begin March 3rd, 2014
Submission deadline is May 30th, 2014
Winners announced June 30th, 2014
JavaOne 2014 from Sept. 28 to Oct. 2, 2014

Java SE 8: Enhancing The Core Java Platform

A simple lambda example – event handling

Handling the Zoom Gesture (`ZoomEvent`)

```
ImageView rect = new ImageView(gesturesImg);

rect.setOnZoomStarted(e -> {
  curZoomFactor = rect.getScaleX();
});

rect.setOnZoom(e -> {
  rect.setScaleX(e.getTotalZoomFactor()
                 * curZoomFactor);
  rect.setScaleY(e.getTotalZoomFactor()
                 * curZoomFactor);
});
```

Lambda Expressions

Why Do We Need Them?

- Almost all machines now are multi-core, multi-processor, or both
- We need to make it simpler to write multi-threaded Java code
  - Java has always had the concept of threads
  - Even using the concurrency utilities and fork-join framework this is hard
- Let's add better library code
  - This is good, but sometimes we need to change the language too
- The answer: Lambda expressions and the streams API

The Problem: External Iteration

```
List<Student> students = ...
double highestScore = 0.0;
for (Student s : students) {
  if (s.gradYear == 2011) {
    if (s.score > highestScore) {
      highestScore = s.score;
    }
  }
}
```

- Client controls iteration
- *Inherently serial:* iterate from beginning to end
- Not thread-safe because business logic is stateful (mutable accumulator variable)

Internal Iteration With Inner Classes

```
List<Student> students = ...
double highestScore = students.stream().
  filter(new Predicate<Student>() {
    public boolean test(Student s) {
      return s.getGradYear() == 2011;
    }
  }).
  mapToDouble(new ToDoubleFunction<Student>() {
    public Double applyAsDouble(Student s) {
      return s.getScore();
    }
  }).
  max();
```

- Iteration, filtering and accumulation are handled by the library
- Not inherently serial – traversal *may* be done in parallel
- Traversal *may* be done lazily – so one pass, rather than three
- Thread safe – client logic is stateless
- High barrier to use
  - Syntactically ugly

Internal Iteration With Lambdas

```
SomeList<Student> students = ...
double highestScore = students.stream().
    filter(Student s -> s.getGradYear() == 2011).
    mapToDouble(Student s -> s.getScore()).
    max();
```

- More readable
- More abstract
- Less error-prone
- No reliance on mutable state
- Easier to make parallel

Lambda Expressions

Some Details

- Lambda expressions represent anonymous functions
  - Like a method, has a typed argument list, a return type, a set of thrown exceptions, and a body
  - Not associated with a class
- We now have parameterized behaviour, not just values

```
double highestScore = students.stream().
        filter(Student s -> s.getGradYear() == 2011)
        mapToDouble(Student s -> s.getScore())
        max();
```

What

How

Lambda Expressions

Functional Interfaces

- Definition
  - A *functional interface* is an interface with only one abstract method
  - However, the interface may have more than one method
- Identified structurally
  - Type is inferred from the context
  - Works for both assignment and method parameter contexts
- The type of a Lambda expression is a *functional interface*
  - Instances of functional interfaces are created with Lambda expressions
  - `@FunctionalInterface` annotation

Internal Iteration With Lambdas

```
filter(Student s -> s.getGradYear() == 2011)
```

- The type of this lambda is the **`Predicate`** functional interface
- The lambda expression is an implementation of the **`test()`** method.

Interface Predicate<T>

Method Summary

All Methods	Static Methods	Instance Methods	Abstract Methods	Default Methods

Modifier and Type	Method and Description
boolean	`test(T t)` Evaluates this predicate on the given argument.

Type Inferrence

- The compiler can often infer parameter types in a Lambda expression
- Inferrence uses the target functional interface's method signature
- Fully statically typed (no dynamic typing sneaking in)
  - More typing with less typing

```
static T void sort(List<T> l, Comparator<? super T> c);
```

```
List<String> ls = getList();
Collections.sort(ls, (String x, String y) -> x.length() - y.length());
```

```
Collections.sort(ls, (x, y) -> x.length() - y.length());
```

Lambda Epressions

Local Variable Capture & Lexical Scoping

- Lambda expressions can refer to *effectively final* local variables from the enclosing scope
  - This means a variable behaves as if it is marked final (even if it is not)
  - The variable is assigned once
- Lambda expressions are anonymous functions
  - They are not associated with an object
  - **`this`** will refer to the object in the surrounding scope

Method References

If a lambda only calls a method …

```
void createUI() {
  Button button = new Button("Guess What I'm Thinking");
  button.setOnAction(e -> guess(e));  // lambda only calls a method
...
}

void guess(ActionEvent event) {
  Button button = (Button) event.getSource();
  button.setText("You love Lambda!");
}
```

Method References

If a lambda only calls a method, you can use a method reference

```
void createUI() {
  Button button = new Button("Guess What I'm Thinking");
  button.setOnAction(this::guess);  // use method reference
  ...
}

void guess(ActionEvent event) {
  Button button = (Button) event.getSource();
  button.setText("You love Lambda!");
}
```

Method References

- Method references let us reuse a method as a lambda expression

```
FileFilter x = (File f) -> f.canRead();
```

```
FileFilter x = File::canRead;
```

Constructor References

- Same concept as a method reference
  - For the constructor

```
Factory<List<String>> f = () -> return new ArrayList<String>();
```

Equivalent to

```
Factory<List<String>> f = ArrayList<String>::new;
```

Four Kinds of Method References

1) Reference to a static method:

```
ContainingClass::staticMethodName
```

2) Reference to an instance method of a particular object:

```
ContainingObject::instanceMethodName
```

3) Reference to an instance method of an arbitrary object of a particular type:

```
ContainingType::methodName
```

4) Reference to a constructor:

```
ClassName::new
```


Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

Default Methods

- Provide a mechanism to add new methods to existing interfaces
  - Without breaking backwards compatability
  - Gives Java multiple inheritance of behaviour, as well as types
    - but not state!

```
public interface Set<T> extends Collection<T> {
    ...   // The existing Set methods

  default Spliterator<E> spliterator() {
    return Spliterators.spliterator(this, Spliterator.DISTINCT);
  }
}
```

Static Methods In Interfaces

- Previously it was not possible to include static methods in an interface
- Static methods, by definition, are not abstract
  - `@FunctionalInterface` can have zero or more static methods

```
static <T> Predicate<T> isEqual(Object target) {
        return (null == target)
                ? Objects::isNull
                : object -> target.equals(object);
}
```

Streams API

Aggregate Operations

- Most business logic is about aggregate operations
  - “Most profitable product by region”
  - “Group transactions by currency”
- As we have seen, up to now, Java mostly uses external iteration
  - Inherently serial
  - Frustratingly imperative
- Java SE 8’s answer: **`Streams`**
  - With help from Lambdas

Stream Overview

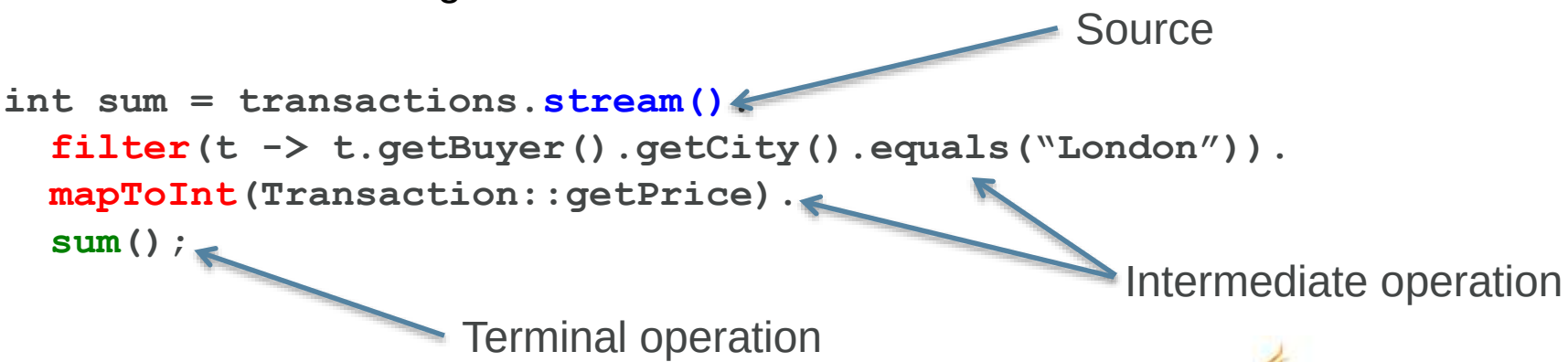
High Level

- Abstraction for specifying aggregate computations
  - Not a data structure
  - Can be infinite
- Simplifies the description of aggregate computations
  - Exposes opportunities for optimization
  - Fusing, laziness and parallelism

Stream Overview

Pipeline

- A stream pipeline consists of three types of things
  - A source
  - Zero or more intermediate operations
  - A terminal operation
    - Producing a result or a side-effect



Stream Overview

Execution

- The **`filter`** and **`map`** methods don't really do any work
  - They set up a pipeline of operations and return a new **`Stream`**
- All work happens when we get to the **`sum()`** operation
  - **`filter()/map()/sum()`** fused into one pass on the data
    - For both sequential and parallel pipelines

```
int sum = transactions.stream().
  filter(t -> t.getBuyer().getCity().equals("London")).   // Lazy
  mapToInt(Transaction::getPrice).                        // Lazy
  sum();                                          // Execute the pipeline
```

Stream Sources

Many Ways To Create

- From collections and arrays
  - `Collection.stream()`
  - `Collection.parallelStream()`
  - `Arrays.stream(T array)` or `Stream.of()`
- Static factories
  - `IntStream.range()`
  - `Files.walk()`
- Roll your own
  - `java.util.Spliterator()`

Stream Sources

Manage Three Aspects

- Access to stream elements
- Decomposition (for parallel operations)
  - Fork-join framework
- Stream characteristics (See Spliterator javadoc)
  - **`ORDERED`**
  - **`DISTINCT`**
  - **`SORTED`**
  - **`SIZED`**
  - **`SUBSIZED`**
  - **`NONNULL`**
  - **`IMMUTABLE`**
  - **`CONCURRENT`**

Stream Intermediate Operations

- Uses lazy evaluation where possible
- Can affect stream characteristics
  - **`map()`** preserves **`SIZED`** but not **`DISTINCT`** or **`SORTED`**
- Some operations fuse/convert to parallel better than others
  - Stateless operations (**`map`**, **`filter`**) fuse/convert perfectly
  - Stateful operations (**`sorted`**, **`distint`**, **`limit`**) fuse/convert to varying degrees

Stream Terminal Operations

- Invoking a terminal operation executes the pipeline
  - All operations can execute sequentially or in parallel
- Terminal operations can take advantage of pipeline characteristics
  - **`toArray()`** can avoid copying for **`SIZED`** pipelines by allocating in advance

Optional<T>

Reducing NullPointerException Occurences

- Indicates that reference may, or may not have a value
  - Makes developer responsible for checking
  - A bit like a stream that can only have zero or one elements

```
Optional<GPSData> maybeGPS = Optional.of(gpsData);
maybeGPS = Optional.ofNullable(gpsData);

maybeGPS.ifPresent(GPSData::printPosition);

GPSData gps = maybeGPS.orElse(new GPSData());

maybeGPS.filter(g -> g.lastRead() < 2).ifPresent(GPSData.display());
```

`java.util.function` Package

- **`Predicate<T>`**
  - Determine if the input of type T matches some criteria
- **`Consumer<T>`**
  - Accept a single input argumentof type T, and return no result
- **`Function<T, R>`**
  - Apply a function to the input type T, generating a result of type R
- Plus several more

Stream Example 1

Convert words in list to upper case

```
List<String> output = wordList.
  stream().
  map(String::toUpperCase).
  collect(Collectors.toList());
```

Stream Example 2

Find words in list with even length

```
List<String> output = wordList.
  stream().
  filter(w -> (w.length() & 1 == 0).
  collect(Collectors.toList());
```

Stream Example 3

Count lines in a file

- BufferedReader has new method
  - `Stream<String> lines()`

```
long count = bufferedReader.
  lines().
  count();
```

Stream Example 4

Find the length of the longest line in a file

```
int longest = reader.
  lines().
  mapToInt(String::length).
  max().
  getAsInt();
```

Stream Example 5

Collect all words in a file into a list

```
List<String> output = reader.
  lines().
  flatMap(line -> Stream.of(line.split(REGEXP))).
  filter(word -> word.length() > 0).
  collect(Collectors.toList());
```

Stream Example 6

List of words lowercased, in aphabetical order

```
List<String> output = reader.
  lines().
  flatMap(line -> Stream.of(line.split(REGEXP))).
  filter(word -> word.length() > 0).
  map(String::toLowerCase).
  sorted().
  collect(toList());
```

Annotations On Java Types

- Annotations can currently only be used on type declarations
  - Classes, methods, variable definitions
- Extension for places where types are used
  - e.g. parameters
- Permits error detection by pluggable type checkers
  - e.g. null pointer errors, race conditions, etc

```
public void process(@notnull List data) {…}
```

Concurrency Updates

- Scalable update variables
  - `DoubleAccumulator`, `DoubleAdder`, etc
  - Multiple variables avoid update contention
  - Good for frequent updates, infrequent reads
- `ConcurrentHashMap` updates
  - Improved scanning support, key computation
- `ForkJoinPool` improvements
  - Completion based design for IO bound applications
  - Thread that is blocked hands work to thread that is running

Parallel Array Sorting

- Additional utility methods in **`java.util.Arrays`**
  - **`parallelSort`** (multiple signatures for different primitives)
- Anticipated minimum improvement of 30% over sequential sort
  - For dual core system with appropriate sized data set
- Built on top of the fork-join framework
  - Uses Doug Lea's **`ParallelArray`** implementation
  - Requires working space the same size as the array being sorted

Date And Time APIs

- A new date, time, and calendar API for the Java SE platform
- Supports standard time concepts
  - Partial, duration, period, intervals
  - date, time, instant, and time-zone
- Provides a limited set of calendar systems and be extensible to others
- Uses relevant standards, including ISO-8601, CLDR, and BCP47
- Based on an explicit time-scale with a connection to UTC

Base64 Encoding and Decoding

- Currently developers are forced to use non-public APIs
  - **`sun.misc.BASE64Encoder`**
  - **`sun.misc.BASE64Decoder`**
- Java SE 8 now has a standard way
  - **`java.util.Base64.Encoder`**
  - **`java.util.Base64.Decoder`**
  - **`encode`**, **`encodeToString`**, **`decode`**, **`wrap`** methods

Nashorn JavaScript Engine

- Lightweight, high-performance JavaScript engine
  - Integrated into JRE
- Use existing `javax.script` API
- ECMAScript-262 Edition 5.1 language specification compliance
- New command-line tool, `jjs` to run JavaScript
- Internationalised error messages and documentation

Removal Of The Permanent Generation

Permanently

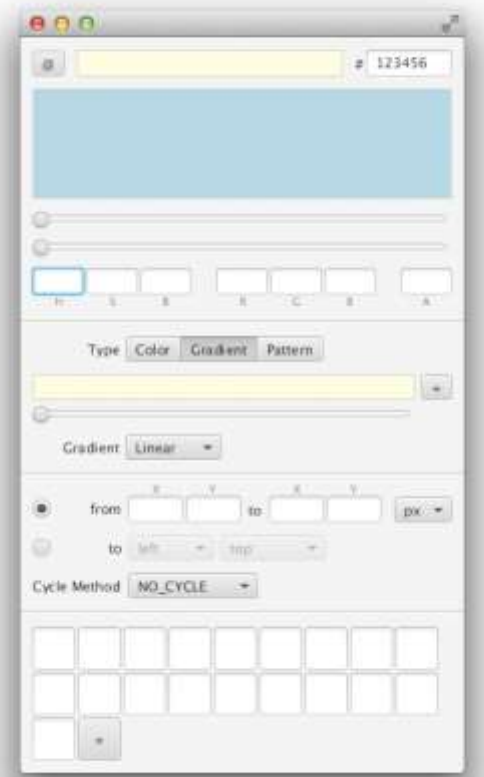
- No more need to tune the size of it
- Current objects moved to Java heap or native memory
  - Interned strings
  - Class metadata
  - Class static variables
- Part of the HotSpot, JRockit convergence

JavaFX 8
New Theme

Caspian

vs

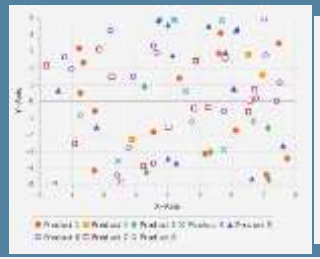
Modena

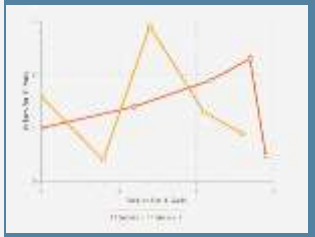


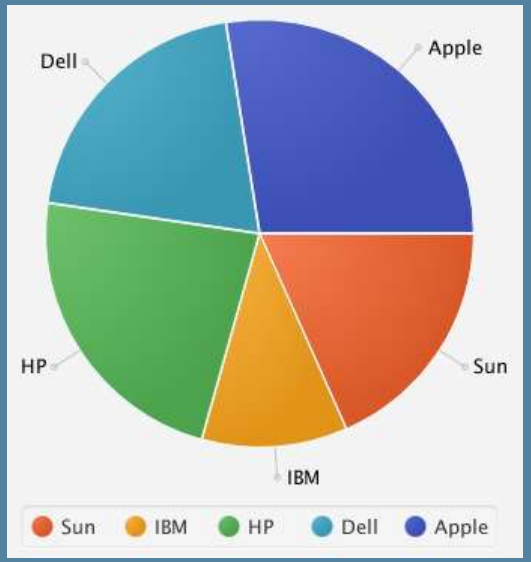
Modena

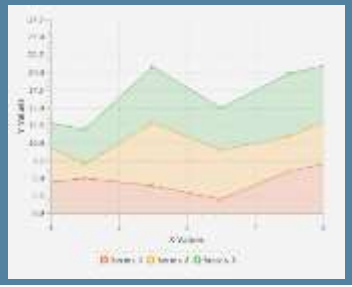
4
9
Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

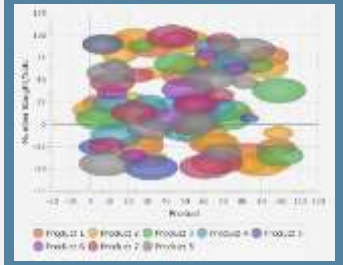
Modena











JavaFX 8

Improvements To Full Screen Mode

- Customisable
  - Configurable key combinations to exit full screen mode
  - Ability to prevent user exiting full screen mode

```
// Set the key combination that the user can use to exit
stage.setFullScreenExitKeyCombination(KeyCombination.NO_MATCH);

// Setup the click handler so we can escape from full screen
Rectangle r = new Rectangle(0, 0, 250, 250);
r.setOnMouseClicked(e -> stage.setFullScreen(false));

// Set full screen again
stage.setFullScreen(true);
```

JavaFX 8

Printing Support

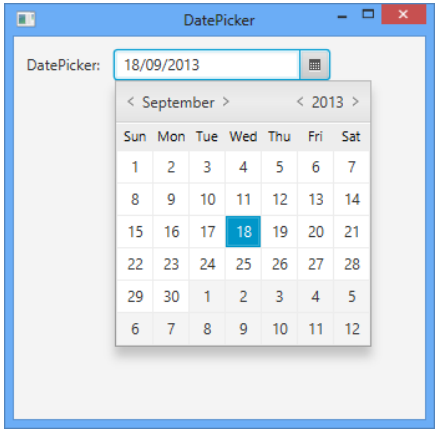
- New API for printing
  - Currently only supported on desktop
- Any node can be printed

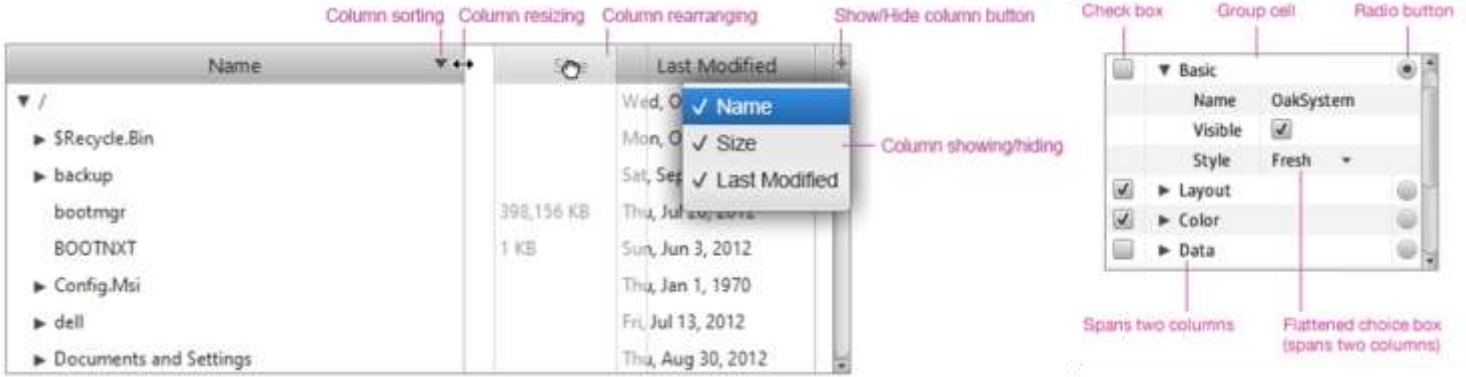
```
PrinterJob job = PrinterJob.createPrinterJob(printer);
job.getJobSettings().setPageLayout(pageLayout);
job.getJobSettings().setPrintQuality(PrintQuality.HIGH);
job.getJobSettings().setPaperSource(PaperSource.MANUAL);
job.getJobSettings().setCollation(Collation.COLLATED);

if (job.printPage(someRichText))
  job.endJob();
```

JavaFX 8

New Controls

- **DatePicker**
- **TreeTableView**



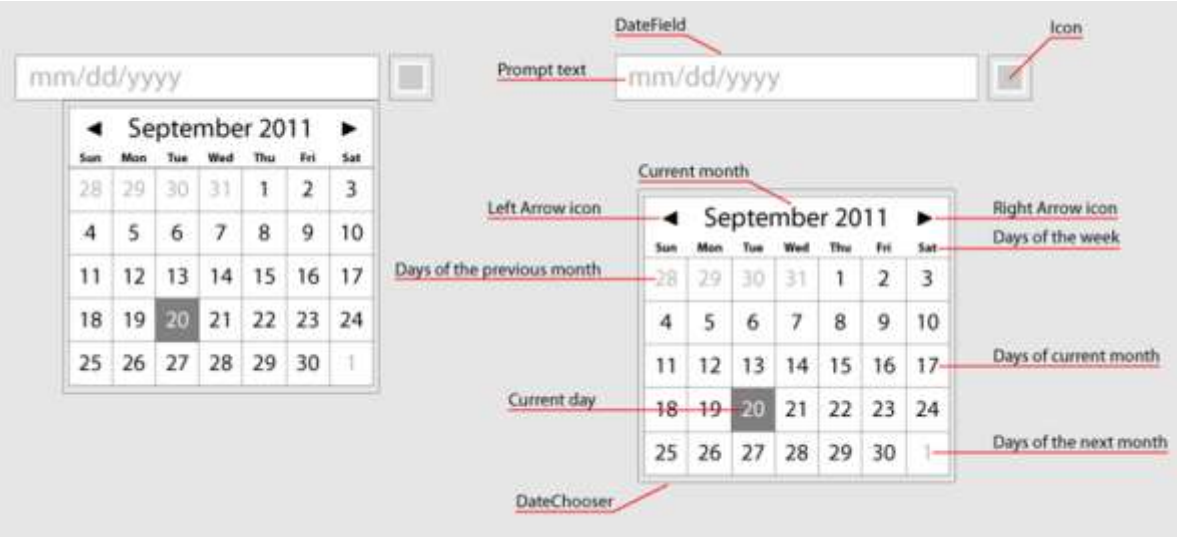


DatePicker

The JavaFX DatePicker is designed with the new Calendaring APIs in Java 8 using **`java.time`**

Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

DatePicker

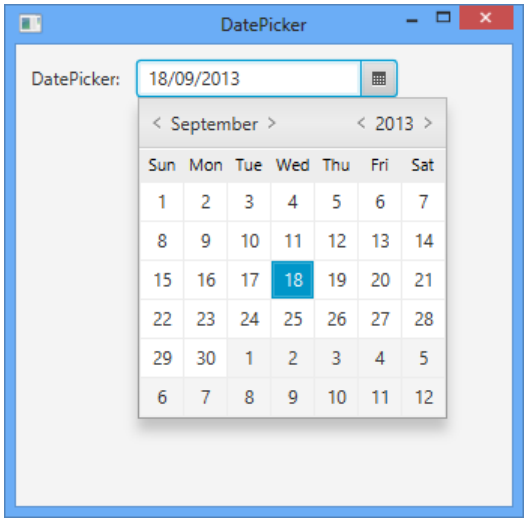


55

DatePicker

```
import java.time.LocalDate; // From new Date/Time API
...
    LocalDate ld;

    DatePicker dp = new DatePicker();
      dp.setOnAction(e -> {
        ld = dp.getValue();
        System.out.println("Date selected " + ld);
      });
```

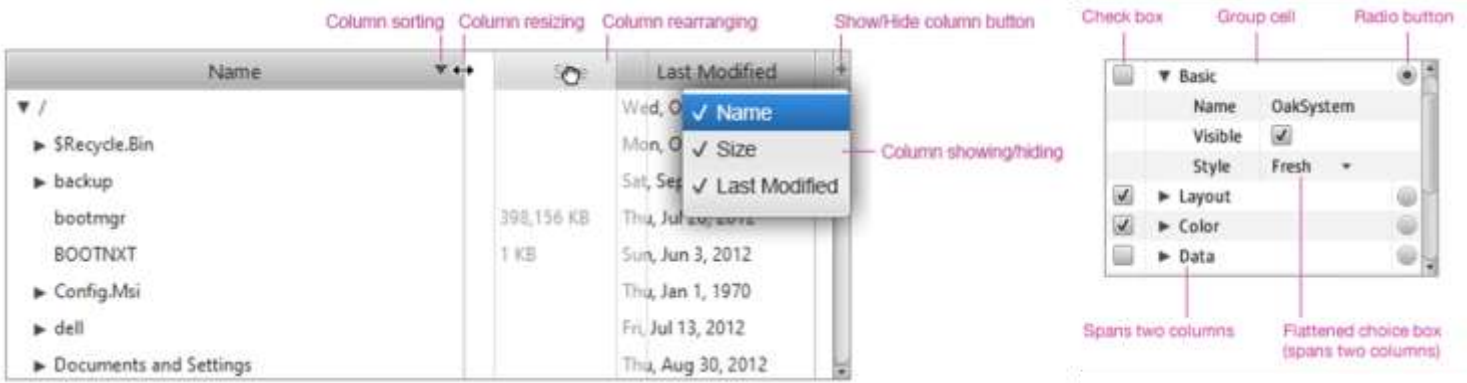
Date Picker



Copyright © 2012, Oracle and/or its affiliates. All rights reserved.

TreeTableView

UI control with combined TreeView and TableView controls functionality



58

JavaFX 8

Touch Support

- Gestures
  - Swipe
  - Scroll
  - Rotate
  - Zoom
- Touch events and touch points

Touch Gestures

Class GestureEvent

java.lang.Object
    java.util.EventObject
        javafx.event.Event
            javafx.scene.input.InputEvent
                javafx.scene.input.GestureEvent

All Implemented Interfaces:

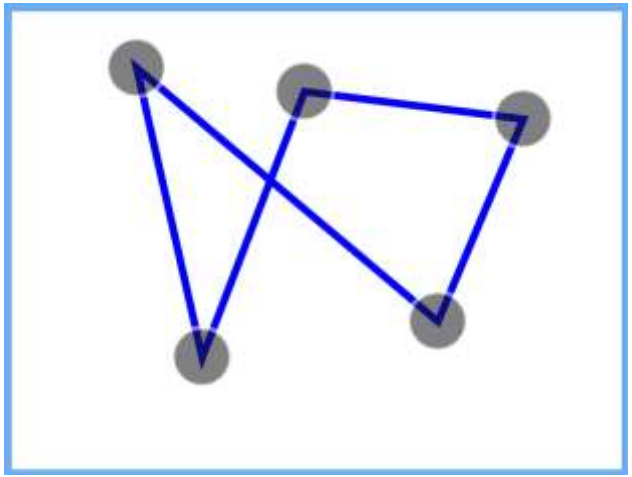
java.io.Serializable, java.lang.Cloneable

Direct Known Subclasses:

RotateEvent, ScrollEvent, SwipeEvent, ZoomEvent

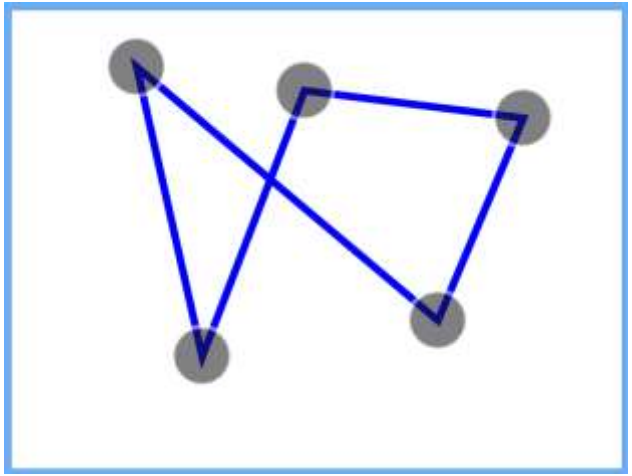
Handling Touch (`TouchEvent/TouchPoint`)

- A **`TouchEvent`** contains information about a touch, including:
  - Event type: Pressed, released, moved, or stationary
  - Touch points: The **`TouchPoint`** instances that represent each of the points that were touched
- Each **`TouchEvent`** has a unique ID to identify the events and touch points in a multi-touch action



Responding to Touch Events

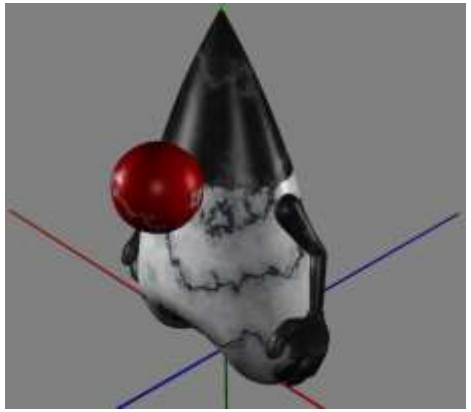
```
    root.setOnTouchPressed(e -> updatePoly(e));
    root.setOnTouchReleased(e -> updatePoly(e));
    root.setOnTouchMoved(e -> updatePoly(e));
}

private void updatePoly(TouchEvent te) {
    poly.getPoints().clear();
    for (TouchPoint tp : te.getTouchPoints()) {
        poly.getPoints()
                .addAll(tp.getX(), tp.getY());
    }
}
```

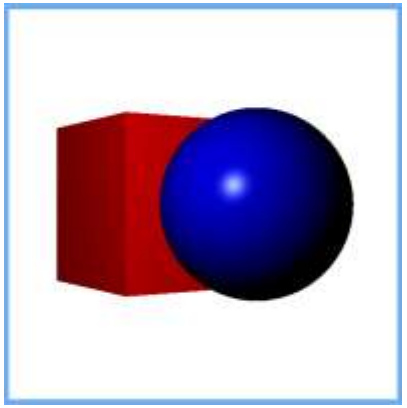


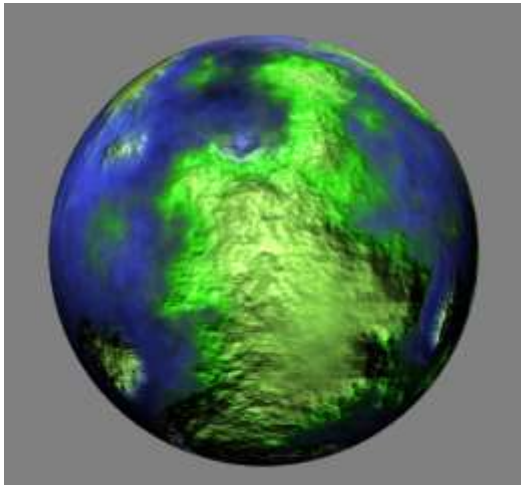
JavaFX 8

3D Support

- Predefined shapes
  - Box
  - Cylinder
  - Sphere
- User-defined shapes
  - `TriangleMesh`, `MeshView`
- `PhongMaterial`
- Lighting
- Cameras

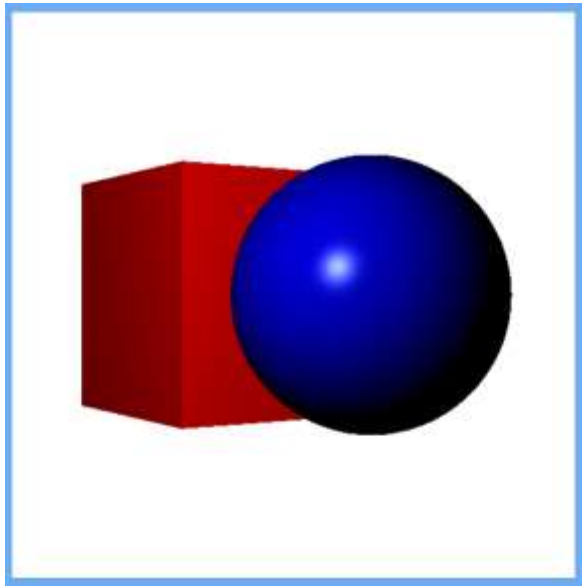




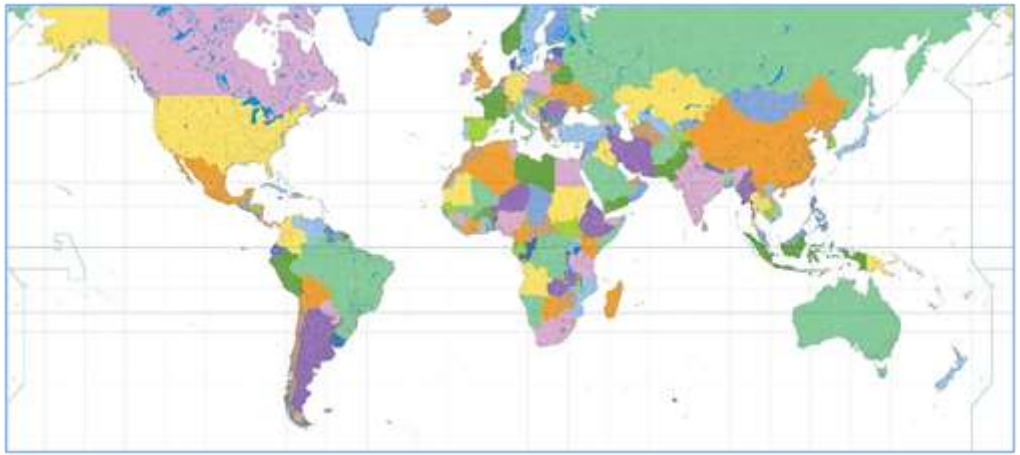


Creating Primitive Shapes and Materials

```
PhongMaterial mat =
    new PhongMaterial();
mat.setDiffuseColor(Color.BLUE);
mat.setSpecularColor(Color.LIGHTBLUE);

final Sphere blue = new Sphere(200);
blue.setMaterial(mat);
```



Placing a Texture on a Sphere

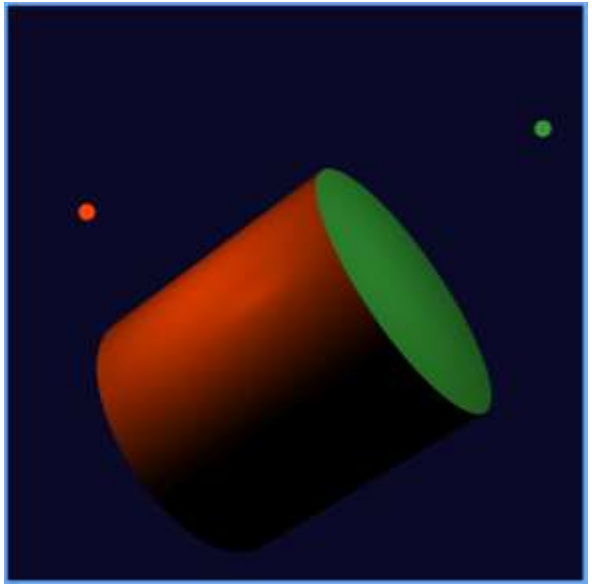


Placing a Texture on a Sphere

```
Material earthMaterial =
   new PhongMaterial(Color.TRANSPARENT,
                     diffuseMap, // Image
                     null, null, null);

final Sphere earth = new Sphere(400);
earth.setMaterial(earthMaterial);
```

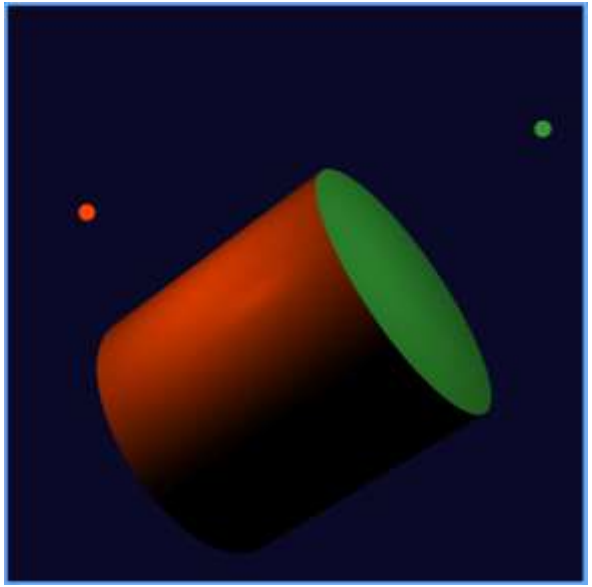
3D Lights

- Lights are nodes in the scene graph
  - **`PointLight`**
  - **`AmbientLight`**
- Default light provided if no active lights



Lights, Camera, Action!

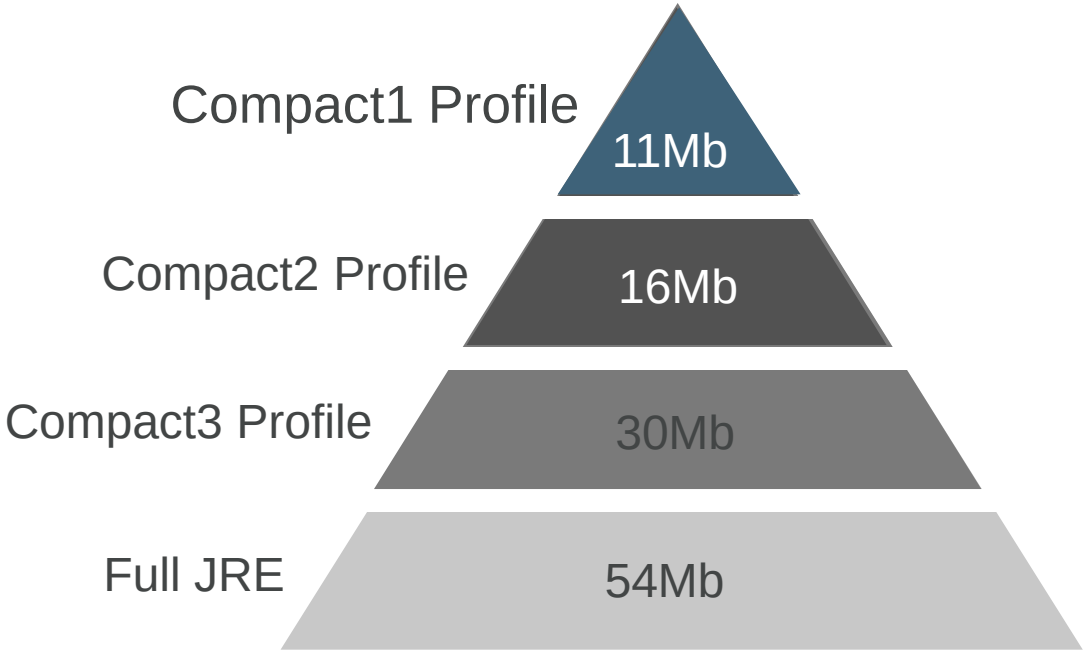
```
PointLight lightA =
  new PointLight(light1Color);
lightA.setTranslateX(700);
lightA.setTranslateY(200);
lightA.setTranslateZ(-200);

PerspectiveCamera camera =
  new PerspectiveCamera();
camera.setTranslateZ(-10);
```



Example 3D multi-touch app: ZenGuitar3D

Compact Profiles

Approximate static footprint goals

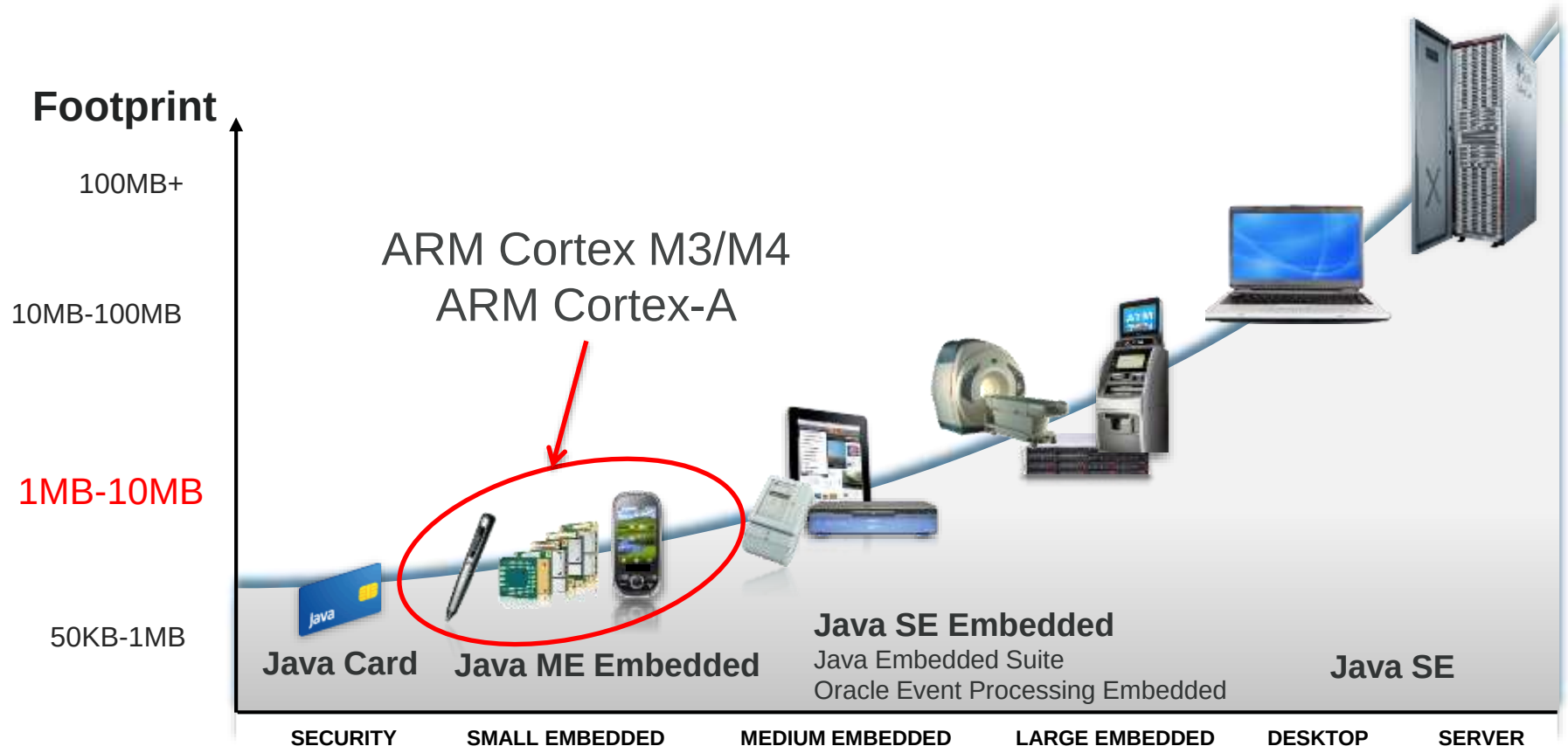


Java SE Embedded

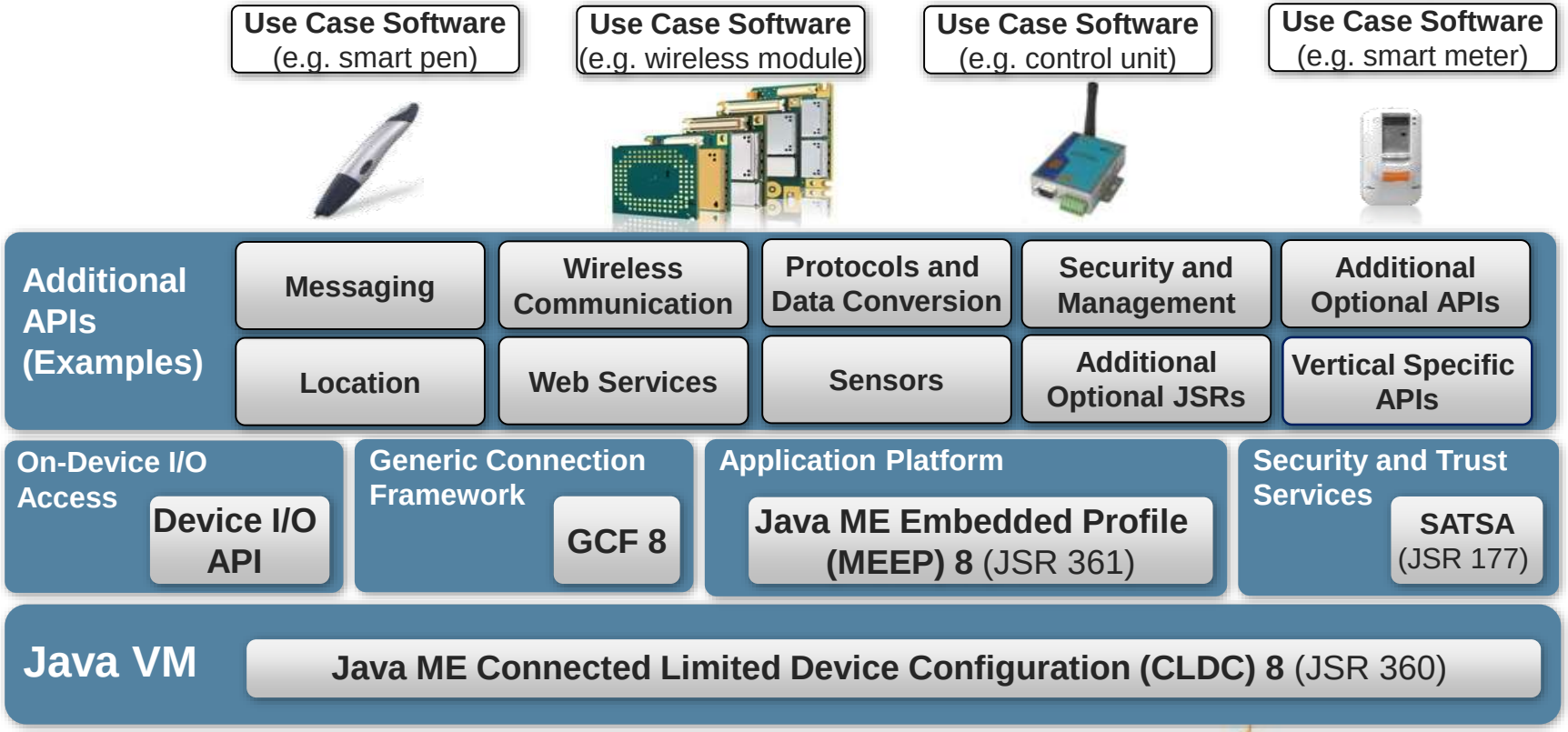
- Optimised binary for embedded use from devices to gateways
- Latest Java innovations in the smallest footprint
- Use of compact profiles
  - Approximate JRE size 20.4Mb (ARM v7 VFS, Hard Float)
    - Compact profile 1: 10.4Mb
    - JavaFX: 10Mb
- Production ready binary for popular platforms that run Linux
  - ARM v6/7 with Hard floating point
  - Raspberry Pi is one of the reference platforms

Java ME 8: Building The Internet of Things

Java ME 8 Focus



Java ME 8 Platform Overview



Java ME 8 Key Features

- Aligned with Java SE 8
  - Language, libraries, VM
- Designed for embedded
  - Fully headless operation
  - Remote software provisioning and management
- Highly portable and scalable
  - Minimum RAM footprint of 1Mb
- Consistent across devices

Java ME 8 Key Features

- Advanced application platform
  - Multi-application model
- Modularised software services
  - Faster and more flexible software development and deployment
- Multi-client domains (“partitioning”)
  - Different clients can have different security domains
- Access to peripheral devices (Device I/O API)
- Compatible with JCP/JSR standard APIs

Connected Limited Device Configuration 8

CLDC 8 Key Features

- Language Alignment with SE 8
- VM Alignment with SE 8
- Library Alignment with SE 8
- Compact Configuration for very small devices
- GCF 8 to provides flexible networking
- Developer leverage of tools, APIs and knowledge

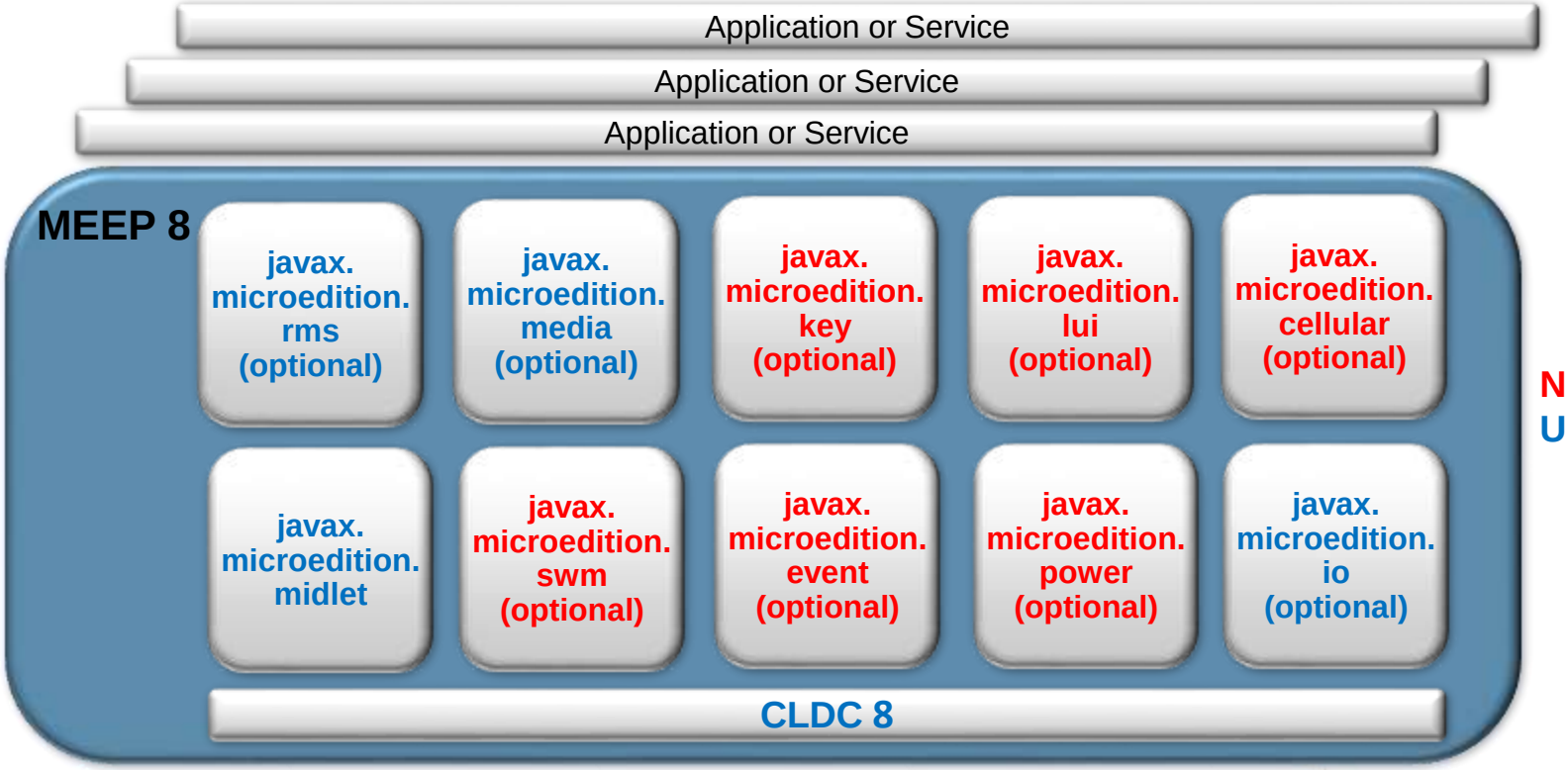
Generic Connection Framework 8

Substantially Increased Range of Connection Types

- SecureServerConnection
- SecureDatagramConnection
- ModemConnection
- UDPMulticastConnection
- CommConnection
- HttpConnection
- HttpsConnection
- SecureConnection
- ServerSocketConnection
- SocketConnection
- UDPDatagramConnection

Java ME Embedded Profile (MEEP) 8

Architecture



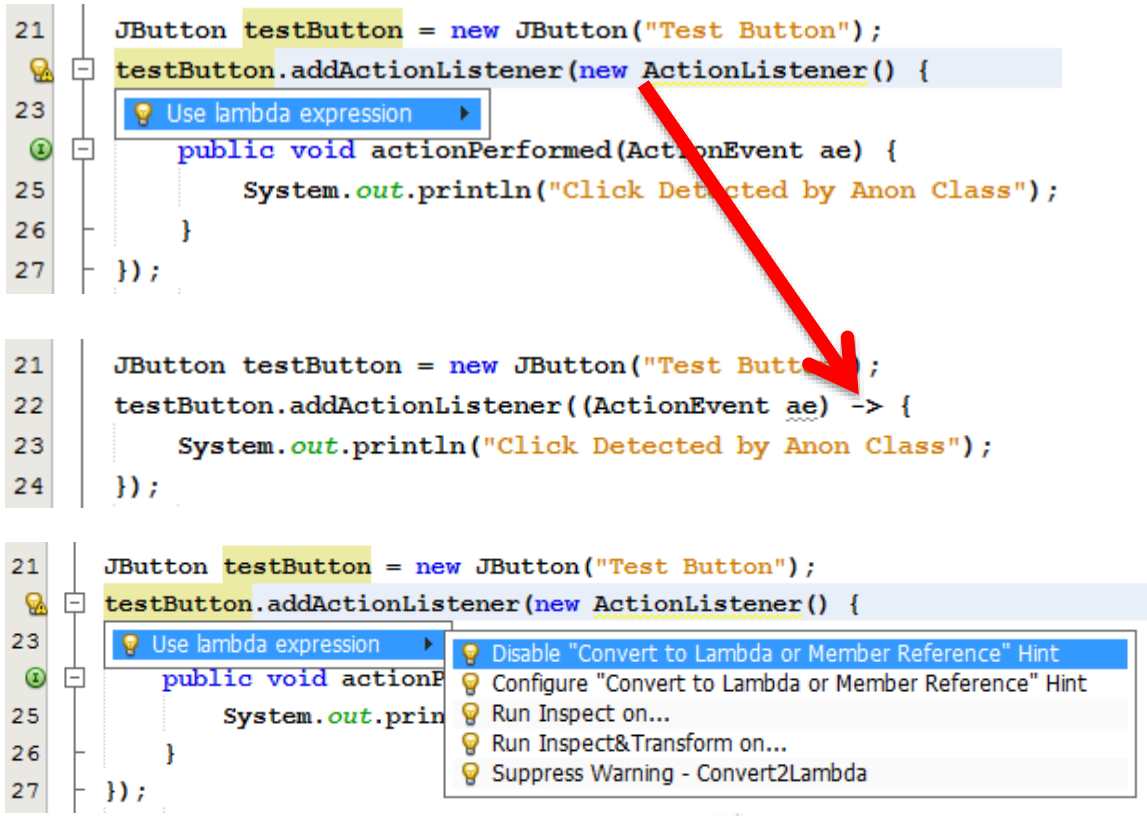
NEW
UPDATED

NetBeans 8:
The IDE For Java 8

Tools for Java SE 8

Lambda Expressions

- Quickly convert anonymous inner classes to lambdas



Tools for Java SE 8

Lambda Expressions

- Static analysis
  to ensure
  safe transformation,
  automatically
  add cast
  for correct type

```
27    try {
28        conv = AccessController.doPrivileged(
                  new PrivilegedExceptionAction<B2CConverter>() {
30                    @Override
                      public B2CConverter run() throws Exception {
32                        return new B2CConverter(enc);
33                    }
34                });
35    } catch (PrivilegedActionException ex) {
36        Exception e = ex.getException();
37        if (e instanceof IOException) {
38            throw (IOException) e;
39        }
40    }
```

```
27  try {
28      conv = AccessController.doPrivileged(
29              (PrivilegedExceptionAction<B2CConverter>) () -> new B2CConverter(enc));
30  } catch (PrivilegedActionException ex) {
31      Exception e = ex.getException();
32      if (e instanceof IOException) {
33          throw (IOException) e;
34      }
35  }
```

Tools for Java SE 8

Internal Iterators via Java 8 Streams

- Editor support for functional operations over collections

```
public int getTotalWeight(List<Block> blocks) {
    return blocks.stream()
            .map(b -> b.getWeight())
            .reduce(0, (a, b) -> a + b);
}
```

1

```
public void changeColor(List<Block> blocks) {
    blocks.stream()
            .filter(b -> b.getColor() == Color.RED)
            .forEach(b -> b.setColor(Color.BLUE));
}
```

2

```
public boolean areAnyBlocksBlue(List<Block> blocks){
    return blocks.stream().anyMatch(b -> b.getColor() == Color.BLUE);
}
```

3

Tools for Java SE 8

Internal Iterators via Java 8 Streams

- Smoothly convert to internal iterators via hints and tips

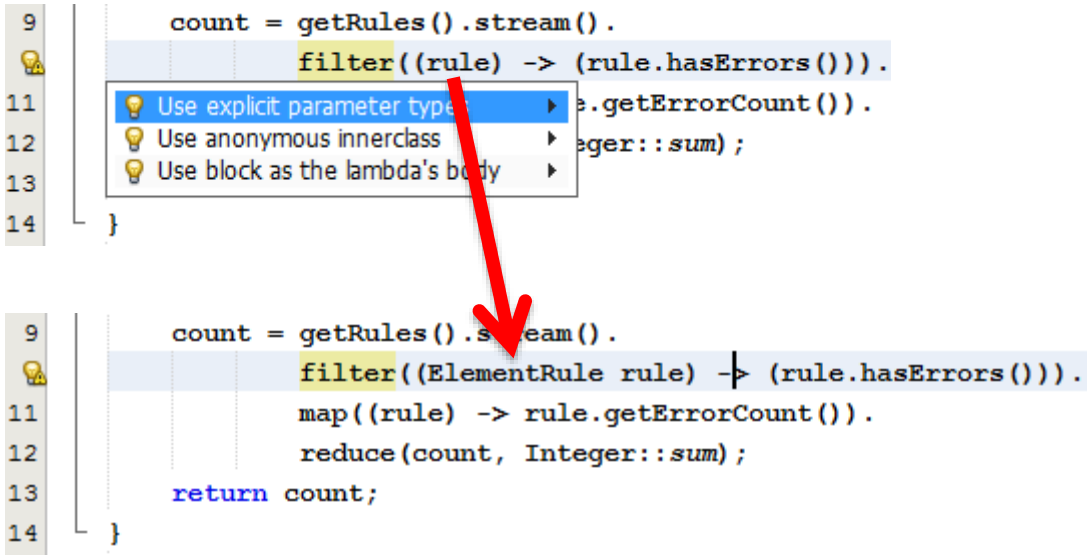
```
7  public int getNumberOfErrors(String grammarName){
8      int count = 0;
9      for (ElementRule rule : getRules()) {
10         if(rule.hasErrors()){
11             count += rule.getErrorCount();
12         }
13     }
14     return count;
15 }
```

```
7  public int getNumberOfErrors(String grammarName) {
8      int count = 0;
9      count = getRules().stream().
10             filter((rule) -> (rule.hasErrors())).
11             map((rule) -> rule.getErrorCount()).
12             reduce(count, Integer::sum);
13     return count;
14 }
```

Tools for Java SE 8

Explicit Parameter Types For Lambda Expressions

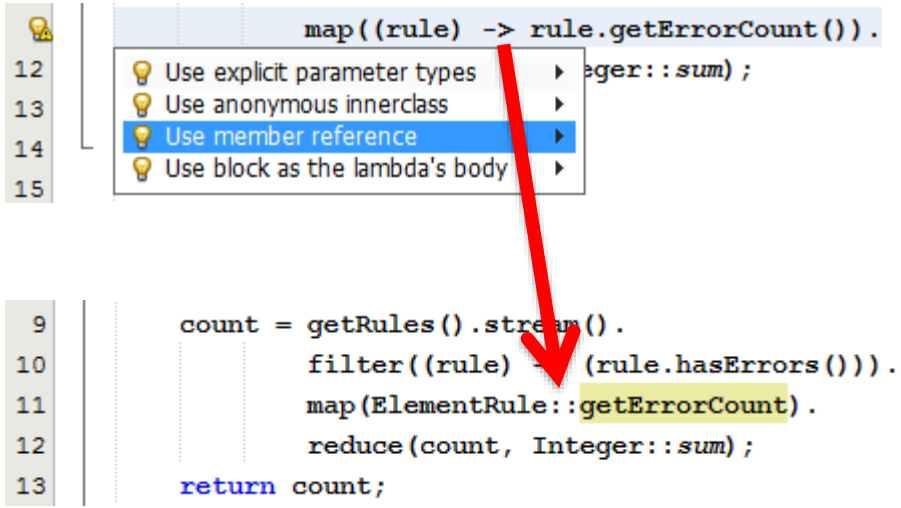
- Quickly add explicit parameter types for readability



Tools for Java SE 8

Method References

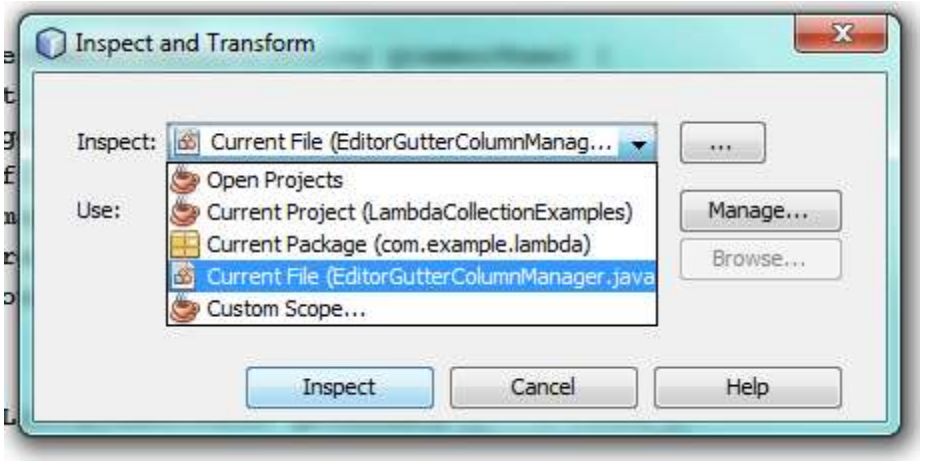
- Easily convert from lambdas to method references

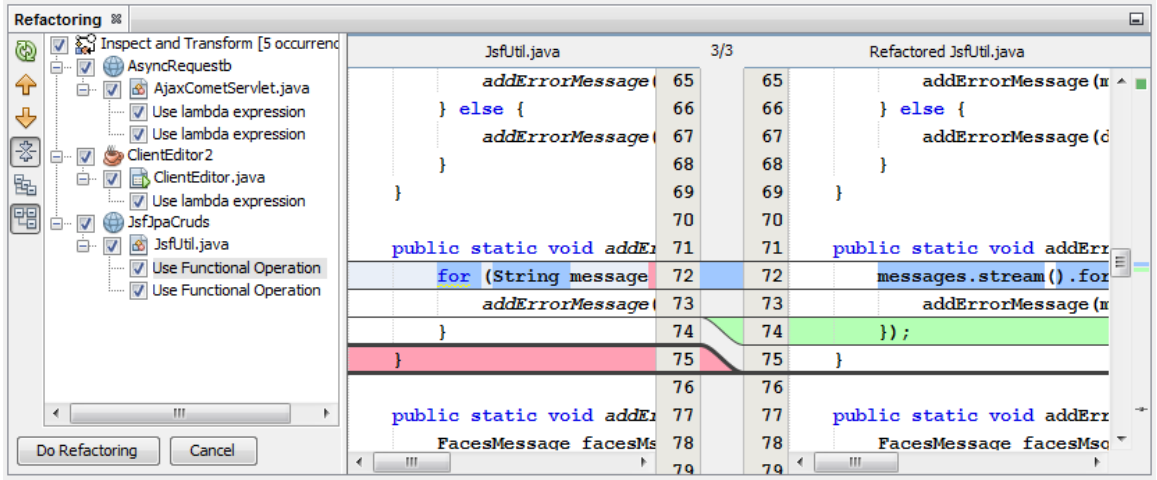


Tools for Java SE 8

Refactoring in Batch Mode

- Specify a scope for upgrading to Java 8
  - All/current projects
  - Specific package
  - Specific class
- Run converters
- Visually preview proposals for refactoring

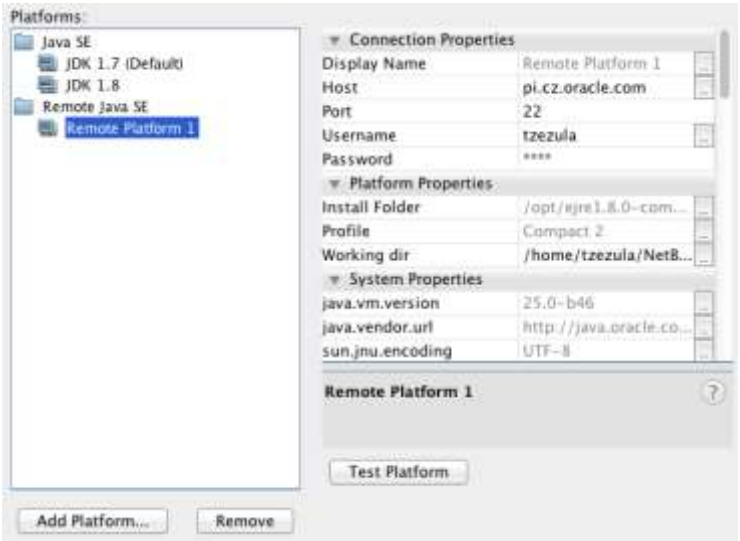




Tools for Java SE Embedded 8

Seamless Integration

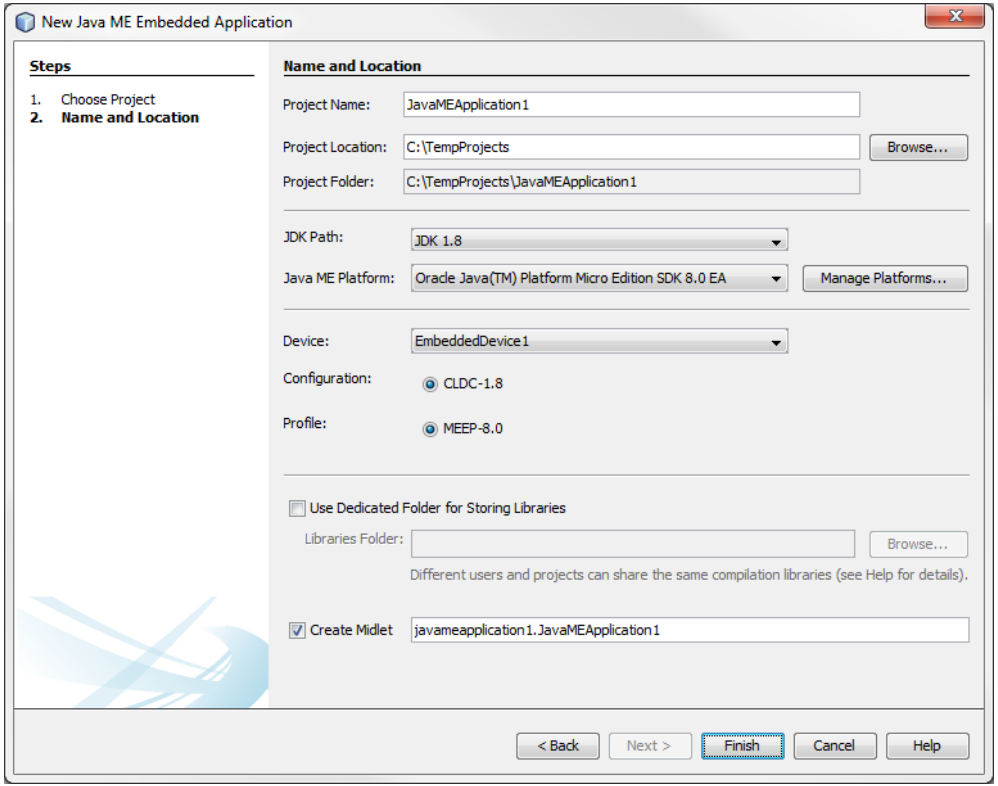
- Full Development Cycle support against a remote platform
  - Intuitive development
  - One-click deploy
  - Remote debugging
  - Comprehensive remote profiling
- Complete end to end integration for Raspberry Pi and other embedded devices,
  - e.g.Web Services.



Tools for Java ME Embedded 8

Seamless Integration

- Java ME 8 CLDC Platform Emulator
- Intuitive tools and editors for JDK 8 on Java ME
- Simple customization of optional packages



Where Next?

Java 8

Learn More & Resources

- **Download:** java.oracle.com
- Documentation: docs.oracle.com/javase
- Training: education.oracle.com/java
- Java 8 Central: www.oracle.com/java8
- Java Magazine: www.oracle.com/javamagazine


@java @javaembedded @netbeans


facebook.com/netbeans


Nighthacking.com


Youtube.com/java


blogs.oracle.com/ java

Conclusions

- Java SE 8 adds powerful new features to the core
  - Lambda expressions, streams and functions
  - New controls, stylesheet and 3D support in JavaFX
- Java ME 8 focused on embedded development
  - Device I/O APIs
- NetBeans 8, ready for Java 8 development
  - Lambda support, embedded support
- Java continues to evolve

Java 9 And Beyond

Java Never Stops

- Modularisation of the Java platform
  - Project Jigsaw
- Foreign function interface
  - Like JNI
- Enhanced volatiles


ORACLE
Java
CREATE
THE FUTURE


Java™