Java 8

and other cool stuff at JavaOne 2013

What's new?
Lambdas!
Lambdas!
Lambdas!
Lambdas!
Lambdas!
Lambdas!
Streams!

Cool Stuff

- Java 8
- JavaFx
  - Desktop
  - On Raspberry Pi
  - On iOS
- Java 9 and beyond
- jmh
- Hazelcast

Java 8

- First revolutionary version since 1.5
  - Date/Time API
  - Lambdas!
  - Futures
  - Streams
  - Compact Profiles
  - Type annotations
  - Nashorn - JavaScript on the server
- Release date: March 18, 2014

Java 8 Date/Time API

- JSR-310
- “new Date()” used only to get instants
- Inspired by Joda Time
- Backport available in Maven Central for JDK 1.7
    - 'org.threeten:threetenbp:0.8.1'

Java 8 Date/Time API

- LocalDate - year-month-day
  - `date=LocalDate.now();`
  - `date=LocalDate.of(2014, MARCH, 18);`
  - `boolean isLeap=date.isLeapYear();`

Java 8 Date/Time API

- Dates are immutable

```
date=date.plusMonths(2).minusDays(1);

date=date.withDayOfMonth(24);
// set to the 24th of the same month and year
```

Java 8 Date/Time API

- January is 1, December is 12 (yeah!)
  - `date=date.with(lastDayOfMonth());`
  - `date=date.with(next(TUESDAY));`
  - `date=date.with(next3rdFriday());`
    `// roll your own`

Java 8 Date/Time API

- LocalTime
  - hours-minute-second-nanosecond
  - `time=LocalTime.now()`
  - `time.toString(); // 13:30`
  - `time=time.truncatedTo(SECONDS);`
- LocalDateTime
  - combination of LocalDate and LocalTime
    - `dt.toString(); // 2013-09-12T13:30`

Java 8 Date/Time API

- TimeZones
  - Syria changed DST with 3 days notice
  - Western Australia had 3 year DST experiment
  - Brazil changes DST every year
  - Egypt had two DST periods in 2010

Java 8 Date/Time API

- TimeZones
  - Timezone database
  - JSR-310 separates responsibility
    - ZoneOffset – from +14:00 to -12:00
    - ZoneRules – rules for switching zone offsets
    - ZoneId – fixed identifier for a location/ government

Java 8 Date/Time API

- TimeZones
  - ZonedDateTime - closest equivalent to GregorianCalendar
  - ZonedDateTime.rules
    - nextTransition
      - when is the next DST transition?

Java 8 Date/Time API

- Instants
  - timestamps, such as for logging
  - time zone not relevant
  - who uses Date() for anything else?
  - Nanoseconds is default

Java 8 Date/Time API

- Amounts of time
  - not tied to the timeline - 6 years, 24 minutes, etc.
  - Duration class
    - seconds, minutes, hours (1 day is 24 hours)
  - Period class
    - date based, years, months, days
    - period of a day can be 23 or 25 hours, depending on DST

Java 8 Date/Time API

- Custom clocks
  - Can stop time, run slowly, run faster
    - Good for tests
  - `LocalDate.now(Clock.system())`
  - `LocalDate.now(Clock.system(zone))`

Java 8 Date/Time API

- Existing classes not deprecated
  - JSR-310 classes don't reference old classes

Java 8 Date/Time API

- Formatting and Parsing
  - DateTimeFormatter - now THREAD-SAFE!
    - many common formatters provided
    - Can use FormatStyle.FULL (don't need to know zzzz, VV, etc)
- http://threeten.github.io
  - Backported to JDK 1.7 on Maven Central
  - JDK 8 developer preview
- CON6091_Converting to the New Date and Time API in_JDK 8.pdf

Java 8 Futures

- Example use case: pull data from multiple systems
  - Parallelize
  - Non-blocking
  - Fork-Join good for this (hard to use!)
  - CompletableFuture
    - computation as a chain/flow of events/tasks
      - do this *then* that (chaining)
      - do this *and* that (joining)
      - do this on failure (recovering)

Java 8 Futures

- CompletableFuture
  - thenAccept* - run function when complete
  - thenApply* - convert result using a function when complete
  - * - additional behavior happens in the current thread (blocking)
  - *Aysnc - additional behavior happens somewhere

Java 8 Futures

```
public Result index(Request request, Response response)
{
  CompletableFuture<List<Blog>> blogs= latestBlogs(request);
  CompletableFuture <UserData> user  = currentUserData(request);
  CompletableFuture <Html> html =
blogs.thenCombineAsync(user,HomePage::renderHomePage);
  //static method - note that it doesn't know about response

  // combine the two into the response
  return async(html, response);
}
```

Java 8 Futures

- Asynchronous Join
  - thenCombine* - convert results of two futures into new thing when both done
  - thenAcceptBoth* - run function when both futures done
  - *Async

Java 8 Futures

- Recovering failures

```
public CompletableFuture<T>{
  exceptionally(
    Function<Throwable, ? extends T>
    recovery){
      ...
    }
}
```

- Note that we don't need Executors
- Provide good states when failures happen
- CON6591_Reactive Programming Patterns with Java 8 Futures.pptx

Java 8 Lambdas

- Largest change to Java programming model ever
  - Method references
  - Allows you to treat code as data
  - Behavior can be stored in variables and passed to methods
  - External iteration - client knows about collection, usually sequentially
  - Internal iteration - collection runs logic, picks best way with parallelism, lazy, etc.

Java 8 Lambdas

- Functional interfaces instead of function types
  - Preserve the core
  - Existing libraries are forward compatible to lambdas
- Better APIs
  - More permeable client-library boundary
  - Client can provide functionality
  - Client determines *what*
  - Library determines *how*
  - More opportunities for optimization

Java 8 Lambdas

- Example - sorting using comparators
- Old way:

```
new Comparator<Person>
{
  public int compare (person x, person y)
  {
    return x.getLastName().compareTo(y.getLastName());

  }
};
```

- New way:

```
Comparators.comparing(Person::getLastName);
```

Java 8 Lambdas

- Interface evolution
  - Collections.sort()
    - Static method, not good
  - forEach() - new default method on Collection
- Default methods
  - Virtual interface method
  - Default is the opposite of abstract
  - Subclasses can override with better implementations
    - List has better impl than Collection

Java 8 Lambdas

- Default methods for interfaces
  - Child classes inherit methods from interface
  - Multiple inheritance of behavior (but not of state)
  - Always had multiple inheritance of types (interfaces)
  - Java interfaces are stateless

Java 8 Lambdas

- Default Methods
  - Class wins
    - If a class can inherit from a superclass and superinterface,
      - Prefer the superclass method
  - Subtypes win
    - If class has two interfaces, one more specific,
      - Prefer the more specific
        - i.e., List over Collection
    - Inheritance tree doesn't matter

Java 8 Lambdas

- Optional methods
  - Example - remove on iterator - just throws an exception, no behavior
- Combinators - reversed() calls compare with opposite order
  - Sort example:

```
people.sort(comparing(p->p.getLastName()))

people.sort(comparing(Person::getLastName))

people.sort(comparing(Person::getLastName)).reversed()

people.sort(comparing(Person::getLastName))
            .thenComparing(Person::getFirstName)
```

Java 8 Lambdas

- Bulk operations on collections
  - Instead of ifs, use .filter
    - .stream().filter().forEach()

```
List<Shape> allShapes = getShapes();
List<Shape> blue = allShapes.stream()
                            .filter(s-> s.getColor()==BLUE)
                            .collect(Collectors.toList());
```

  - Also use .map, .sum
  - .map(Shape::getContainingBox)
  - .mapToInt(Shape::getWeight).sum();

Java 8 Lambdas

- Expressive and composable
  - Each stage does one thing
  - Client code reads like problem statement

Java 8 Streams

- Represents a stream of values
- Not a data structure, doesn't store
- Source can be collection, generating function, I/O, …
- Can be infinite
- Operations that produce stream are lazy
- Efficient - does a single pass on the data
- Eliminates temp collections for intermediate results
- Makes it easier to parallelize code

Java 8 Streams

- Useful methods:
  - findFirst
  - findAny
  - count
  - max
  - min
  - forEach
  - collect

Java 8 Streams

- Useful methods creating substreams:
  - filter
  - distinct
  - sorted
  - limit
  - mapToInt/Long/Double
  - reduce

Java 8 Streams

- Many ways to make stream
  - collection.stream() or array.stream()
  - Factories (IntStream.range(1..1000))
  - I/O-backed (Files.walk())
  - Roll your own with Spliterator
    - Parallel analog to iterator, recursive decomposition

Java 8 Streams

- Stream source manages
  - Access to stream elements
  - Decompositions for parallel operations
  - Stream characteristics (sized, ordered, distinct, sorted)
- Stream sources
  - Excellent - ArrayList, Array
  - Poor - LinkedList, BufferedReader

Java 8 Streams

- Old way - what does this do?

```
public static void main(String[] args) {
    List<String> symbols = Tickers.symbols;
    findStockImperative(symbols);
  }

public static void findStockImperative(List<String> symbols) {
  List<StockInfo> stockPrices = new ArrayList<>();

  for(String ticker : symbols) {
    stockPrices.add(StockUtil.getPrice(ticker));
  }

  List<StockInfo> stocksLessThan500 = new ArrayList<>();
  for(StockInfo stockInfo : stockPrices) {
    if(StockUtil.isPriceLessThan(500).test(stockInfo))
      stocksLessThan500.add(stockInfo);
  }

  StockInfo highPriced = new StockInfo("", 0.0);
  for(StockInfo stockInfo : stocksLessThan500) {
    highPriced = StockUtil.pickHigh(highPriced, stockInfo);
  }

  System.out.println(highPriced);
}
```

Java 8 Streams

- New way

```
public static void main(String[] args) {
    List<String> symbols = Tickers.symbols;
    findStockDeclarative(symbols.stream());
  }

public static void findStockDeclarative(Stream<String> tickers) {
  StockInfo highPriced = tickers
                        .map(StockUtil::getPrice)
                        .filter(StockUtil.isPriceLessThan(500))
                        .reduce(new StockInfo("", 0.0), StockUtil::pickHigh);
  System.out.println(highPriced);
}
```

Java 8 Streams

- New way with parallelization

```
public static void main(String[] args) {
    List<String> symbols = Tickers.symbols;
    findStockDeclarative(symbols.parallelStream());
  }

public static void findStockDeclarative(Stream<String> tickers) {
  StockInfo highPriced = tickers
                          .map(StockUtil::getPrice)
                          .filter(StockUtil.isPriceLessThan(500))
                          .reduce(new StockInfo("", 0.0), StockUtil::pickHigh);
  System.out.println(highPriced);
}
```

- Note that no change was required to the method which does the work!

Java 8 Streams

- Other examples

```
Stream.of("to", "be", "or", "not", "to", "be")
      .forEach(s -> System.out.print(s + " "));
```

```
to be or not to be
```

```
int[] arr = new int[] { 0, 1, 2, 3, 4, 5, 6, 7 };
Arrays.stream(arr, 2, 6)
      .forEach(i -> System.out.print(i + " "));
```

```
2345
```

```
Random rand = new Random();
Stream.generate(() -> rand.nextInt(100))
      .limit(10)
      .forEach(i -> System.out.print(i + " "));
```

```
70 17 65 11 22 94 23 18 16 27
```

Java 8 Optional Objects

- What happens when this runs?

```
public static Fruit find(String name, List<Fruit> fruits) {
   for(Fruit fruit : fruits) {
      if(fruit.getName().equals(name)) {
         return fruit;
      }
   }
   return null;
}
.
.
.
.


List<Fruit> fruits = asList(new Fruit("apple"),
                            new Fruit("grape"),
                            new Fruit("orange"));

Fruit found = find("lemon", fruits);
.
.
.
String name = found.getName(); //!
```

Java 8 Optional Objects

- Returning null if something isn't found is evil
- What does "null" mean?
  - Not found?
  - Error?
  - Something else?
- Testing for null
  - Works, but nothing forces user of your API to do so.

Java 8 Optional Objects

- This is safer and more explicit:

```
public static Optional<Fruit> find(String name, List<Fruit> fruits) {
   for(Fruit fruit : fruits) {
      if(fruit.getName().equals(name)) {
         return Optional.of(fruit);
      }
   }
   return Optional.empty();
}

.
.
.
.

List<Fruit> fruits = asList(new Fruit("apple"),
                            new Fruit("grape"),
                            new Fruit("orange"));

Optional<Fruit> found = find("lemon", fruits);
if(found.isPresent()) {
   Fruit fruit = found.get();
   String name = fruit.getName();
}
```

Java 8 Optional Objects

- Other useful examples

```
Optional<Fruit> found = find("lemon", fruits);
String          name  = found.orElse(new Fruit("Kiwi")).getName();

Optional<Fruit> found = find("lemon", fruits);
found.ifPresent(f -> { System.out.println(f.getName()); });

Optional<Fruit> found = find("lemon", fruits);
Fruit fruit = found.orElseGet(() -> new Fruit("Lemon"));
```

Other Cool Stuff

- Java 9
- jmh
- Hazelcast
- Java on iOS

Java 9 and Beyond

- Sumatra
- Reification
- JNI 2.0
- Memory efficient data structures
- Modular platform (Jigsaw+)

Java 9 and Beyond

- Sumatra
  - Enable Java applications to take advantage of graphics processing units (GPUs) and accelerated processing units (APUs)
  - Initial focus on the Hotspot JVM, enabling code generation, runtime support and garbage collection on GPUs
  - How to best expose GPU support to application and/or library developers
    - Leverage the new Java 8 Lambda language and library features

Java 9 and Beyond

- Reification
  - Generic type parameters are not reified
  - Available at compile time
  - Not available at runtime
  - Generics are implemented using erasure
    - Type parameters are simply removed at runtime

jmh

- Java micro-benchmarking
- http://openjdk.java.net/projects/code-tools/jmh/
  - Eliminates JIT effects
  - Provides absolute "which way is better" proof

jmh

- Demo - StringBuilder vs. concatenation

  Which is faster, this?

```
String dibble = "dibble"+"dabble"+"lklklk"+"dononond";
```

Or this?

```
String sb = new StringBuilder();
sb.append("dibble");
sb.append("dabble");
sb.append("lklklk");
sb.append("dononond")
String dibble = db.toString();
```

jmh

- Demo - StringBuilder vs. concatenation in a loop

  Which is faster, this?

```
String result = "";
for(int i=0; i<1000;i++){
  result = result +i;
}
```

  Or this?

```
String sb = new StringBuilder();
for(int i=0; i<1000;i++){
  sb.append(i+"");
}
String result = sb.toString();
```

What about those fancy new streams?

```
StringBuilder sb = new StringBuilder();
textStuff.stream().forEach(text -> sb.append(text));
String out = sb.toString();
```

Hazelcast

- Distributed data grid
- Very lightweight, quick, easy to use
  - Add a single jar in your project
- Community edition is free
- Enterprise edition has low-heap features, security
- Integrates with Spring and Hibernate

Hazelcast

- Features:
  - Distributed java.util.{Queue, Set, List, Map}
  - Distributed java.util.concurrency.locks.Lock
  - Distributed java.util.concurrent.ExecutorService
  - Distributed MultiMap for one-to-many mapping
  - Distributed Topic for publish/subscribe messaging
  - Distributed Indexing and Query support
  - Transaction support and J2EE container integration via JCA
  - Socket level encryption for secure clusters
  - Write-Through and Write-Behind persistence for maps
  - Java Client for accessing the cluster remotely
  - Dynamic HTTP session clustering
  - Support for cluster info and membership events
  - Dynamic discovery
  - Dynamic scaling
  - Dynamic partitioning with backups
  - Dynamic fail-over
  - Web-based cluster monitoring tool

Hazelcast

- When is Hazelcast useful?
  - Share data/state among many servers
  - Cache data (distributed cache)
  - Cluster applications
  - Provide secure communication among servers
  - Partition in-memory data
  - Distribute workload onto many servers
  - Parallel processing
  - Fail-safe data management

Hazelcast

- What does the code look like?

```
public static void main(String[] args) {
  Config cfg = new Config();
  HazelcastInstance instance = Hazelcast.newHazelcastInstance(cfg);

  Map<Integer, String> mapCustomers = instance.getMap("customers");

  mapCustomers.put(1, "Joe");
  mapCustomers.put(2, "Ali");
  mapCustomers.put(3, "Avi");

  Queue<String> queueCustomers = instance.getQueue("customers");
  queueCustomers.offer("Tom");
}
```

Hazelcast

- Demo

Java on iOS

- It IS possible to write iOS apps in Java!
  - Share code between iOS, Android, desktop
  - JIT not allowed - must work around
- 3 ways (currently) of doing this:
  - RoboVM
  - J2ObjC
  - Nuvos

Java on iOS

- RoboVM
  - Compiles to native code
  - Interfaces with iOS classes
  - Can write entire app in Java, or use iOS wrapper
  - Version 0.0.6
  - "HelloWorld.java" compiles/links 1700 classes
  - http://www.robovm.org/

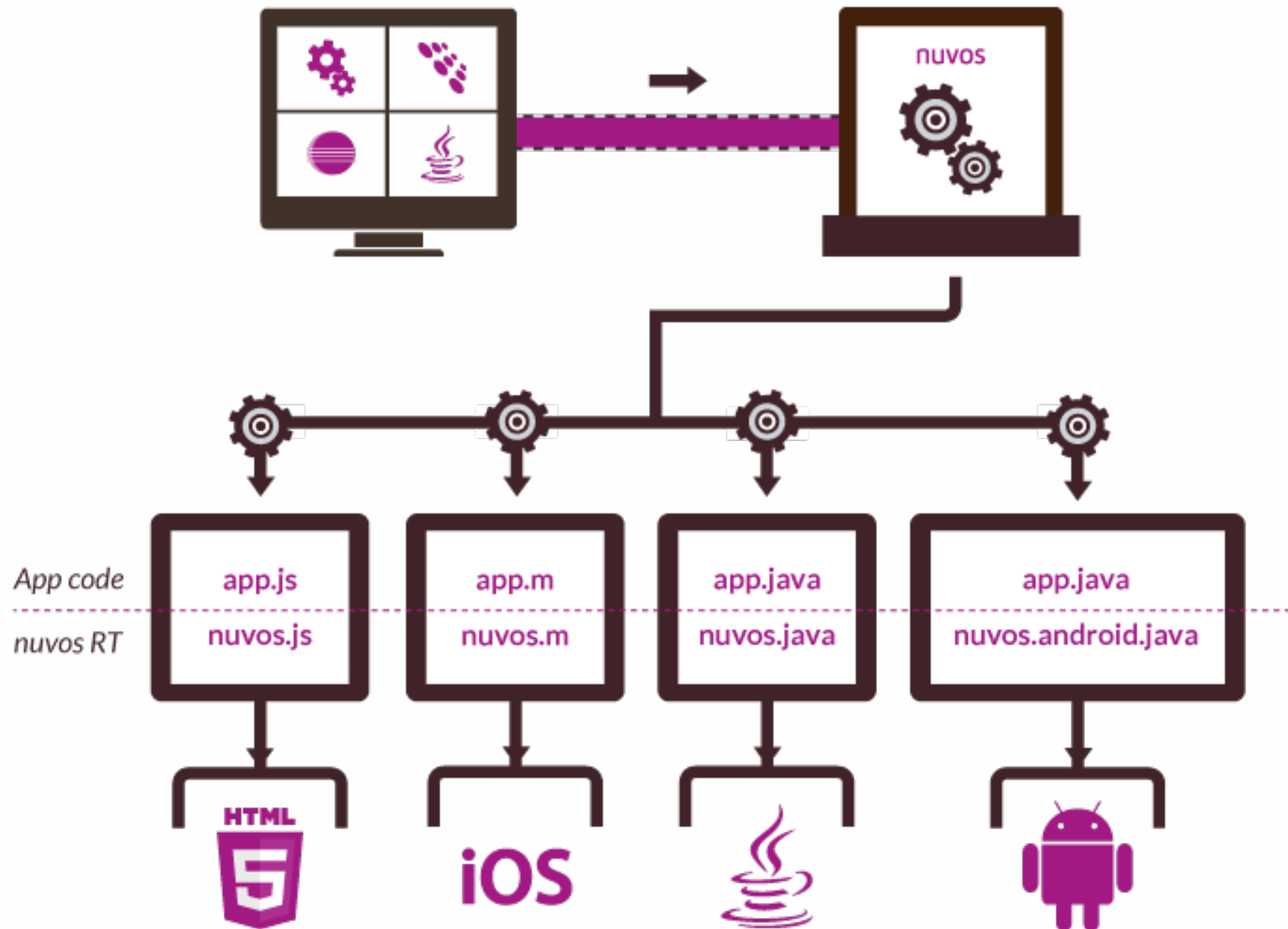
Java on iOS

- J2ObjC
  - Google project
  - Generate Objective-C from Java
  - Only non-UI code
  - http://code.google.com/p/j2objc/

Java on iOS

- Nuvos
    - Like GWT, but...
    - Custom SDK which generates
        - iOS
        - HTML5/JS
        - Java
        - Android Java
        - Wrappers for above to integrate with native code

Java on iOS



Java on iOS

- Nuvos
  - http://www.nuvos.com/sdk.html