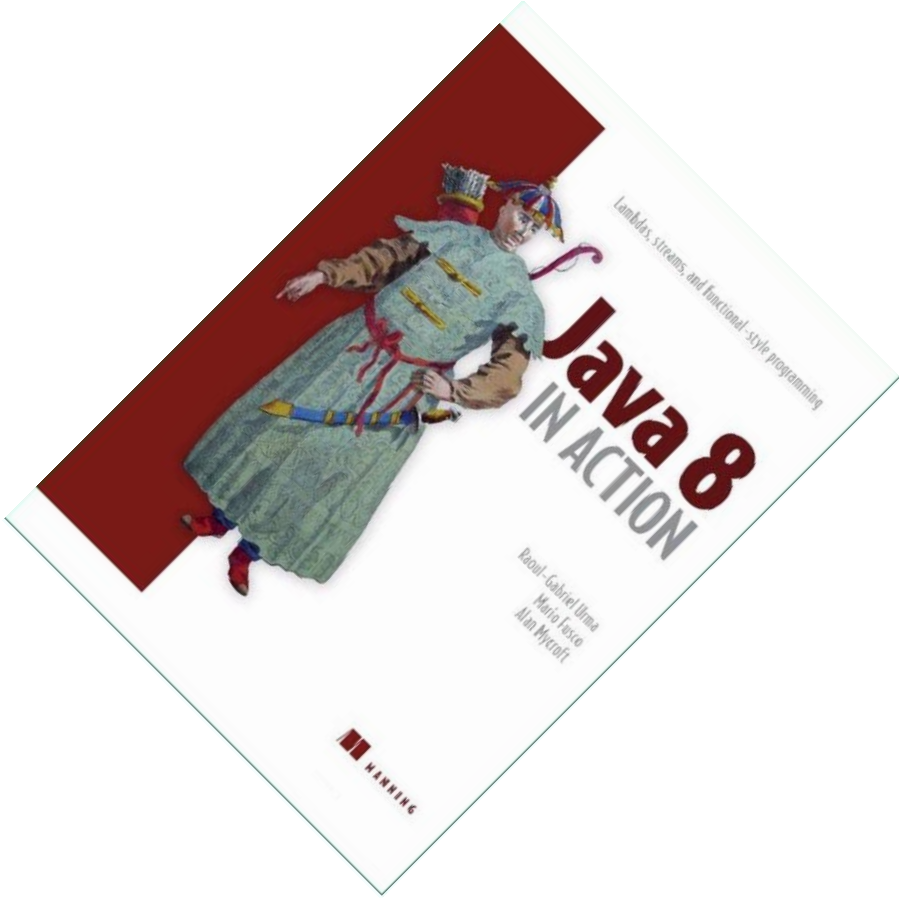
Java 8 quizzes

2014 Olivier Dupuy

with some help from



Java 8: Q 1/8 Compact code thanks to the streams API

```
// Q1: what does it do?
Arrays.stream(new File("c:/").listFiles(File::isHidden)).forEach(
        System.out::println);

// Q2: what does it do?
LongSummaryStatistics stats = Arrays.stream(
        new File("c:/").listFiles(File::isHidden)).collect(
        Collectors.summarizingLong(File::length));
System.out.println(stats);

// Q3: what does it do?
System.out.println(Arrays
        .stream(new File("c:/").listFiles()).filter(File::isHidden)
        .sorted(Comparator.comparingLong(File::length).reversed())
        .limit(5).sorted()
        .peek(t -> System.out.println(t.getName() + ' ' + t.length()))
        .collect(Collectors.summarizingLong(File::length)));
```

Java 8: A 1/8 Compact code thanks to the streams API

```
// Q1: what does it do?
Arrays.stream(new File("c:/").listFiles(File::isHidden)).forEach(
        System.out::println);

// Q2: what does it do?
LongSummaryStatistics stats = Arrays.stream(
        new File("c:/").listFiles(File::isHidden)).collect(
        Collectors.summarizingLong(File::length));
System.out.println(stats);

// Q3: what does it do?
System.out.println(Arrays
        .stream(new File("c:/").listFiles()).filter(File::isHidden)
        .sorted(Comparator.comparingLong(File::length).reversed())
        .limit(5).sorted()
        .peek(t -> System.out.println(t.getName() + ' ' + t.length()))
        .collect(Collectors.summarizingLong(File::length)));
```

- (1) lists the names of all the hidden files of C:\
- (2) displays the statistics on the length (count, min/average/max/total size) of all the hidden files of C:\
- (3) lists the names and sizes of the 5 biggest hidden files of C:\ sorted by name and it displays after the statistics on their length

Java 8: Q 2/8 A lambda is some kind of anonymous class

```
3  public class Java8Quiz2ContentTest {
4      Runnable myRunnable = new Runnable() {
5          public void run() { System.out.println("in runnable 0");    }
6      };
7      @SuppressWarnings("unused")
8      public Java8Quiz2ContentTest() {
9          class MyLocalClass extends Thread {
10             public void run() { System.out.println("in runnable 1"); }
11         };
12         Thread thread2 = new Thread(() -> System.out.println("in runnable 2"));
13     }
14     static class MyStaticClass {
15         Thread thread4 = new Thread(() -> System.out.println("in runnable 3"));
16     }
17     class MyInnerClass implements Runnable {
18         public void run() { System.out.println("in runnable 4"); }
19     }}
```

- Q: Give the names of all the class files in the folder target/classes/org/java8…

Java 8: A 2/8 A lambda is some kind of anonymous class

```
3  public class Java8Quiz2ContentTest {
4      Runnable myRunnable = new Runnable() {
5          public void run() { System.out.println("in runnable 0");    }
6      };
7      @SuppressWarnings("unused")
8      public Java8Quiz2ContentTest() {
9          class MyLocalClass extends Thread {
10             public void run() { System.out.println("in runnable 1"); }
11         };
12         Thread thread2 = new Thread(() -> System.out.println("in runnable 2"));
13     }
14     static class MyStaticClass {
15         Thread thread4 = new Thread(() -> System.out.println("in runnable 3"));
16     }
17     class MyInnerClass implements Runnable {
18         public void run() { System.out.println("in runnable 4"); }
19     }}
```

- Q: Give the names of all the class files in the folder target/classes/org/java8…

Java8Quiz2ContentTest$1.class	// anonymous class of line 4
Java8Quiz2ContentTest$1MyLocalClass.class	// local class of line 9
Java8Quiz2ContentTest$MyInnerClass$1.class	// inner class of line 17
Java8Quiz2ContentTest$MyStaticClass.class	// static class of line 14
Java8Quiz2ContentTest.class	// outer class

Do you see something missing? Why?

Java 8: Q 3/8 Stack trace with a lambda

```
4  import java.util.Objects;
5
6  public class Java8Quiz3StackTraceInLambda {
7
8      public static void main(String[] args) {
9          Apple appleII = new Apple("Special", 2) { // name and weight
10             @Override
11             public int getWeight() {
12                 new RuntimeException().printStackTrace();
13                 return super.getWeight();
14             };
15         };
16         Apple appleIII = new Apple("Apple", 3);
17         Comparator<Apple> byWeight = (a, b) -> a.getWeight() - b.getWeight();
18         Objects.compare(appleII, appleIII, byWeight);
19
20         // what will show the stack trace for this comparison?
```

Java 8: A 3/8 Stack trace with a lambda

```java
import java.util.Objects;

public class Java8Quiz3StackTraceInLambda {

    public static void main(String[] args) {
        Apple appleII = new Apple("Special", 2) { // name and weight
            @Override
            public int getWeight() {
                new RuntimeException().printStackTrace();
                return super.getWeight();
            };
        };
        Apple appleIII = new Apple("Apple", 3);
        Comparator<Apple> byWeight = (a, b) -> a.getWeight() - b.getWeight();
        Objects.compare(appleII, appleIII, byWeight);

        // what will show the stack trace for this comparison?
```

```
        java.lang.RuntimeException
// overridden getWeight() in Java8Quiz3StackTraceInLambda
        at org.java8.Java8Quiz3StackTraceInLambda$1.getWeight(Java8Quiz3StackTraceInLambda.java:12)
// the lambda code itself
        at org.java8.Java8Quiz3StackTraceInLambda.lambda$0(Java8Quiz3StackTraceInLambda.java:17)
// proxy for the functional method compare() of the FI Comparator private static int lambda$0(org.java8.Apple, org.java8.Apple);
        at org.java8.Java8Quiz3StackTraceInLambda$$Lambda$1/8460669.compare(Unknown Source)
// Objects.compare(T a, T b, Comparator<? super T> c)      [New class from Java 7]
        at java.util.Objects.compare(Objects.java:183)
// main method
        at org.java8.Java8Quiz3StackTraceInLambda.main(Java8Quiz3StackTraceInLambda.java:18)
```

Java 8: Q 4/8 Matching the method

- Given **public class** File {
  … **public** File[] listFiles(FileFilter filter) {…}; …}

  and @FunctionalInterface **interface** FileFilter {
  boolean accept(File pathname); }

  and

```
static class FileUtils {
    static boolean isHidden(File file) { return file.isHidden(); }
    boolean isHiddenFile(File file)    { return file.isHidden(); } }
    Predicate<File> hiddenFilePredicatePlusPlus = new Predicate<File>() {
        public boolean test(File t) { return t.isHidden(); };
        public void otherMethod() {} };
    Function<File,Boolean> myFunction = new Function<File,Boolean>() {
        @Override public Boolean apply(File t) { return t.isHidden(); } };
```

- **Which lines compile properly retrieving the hidden files?**

1. **new** File("...").listFiles(File::isHidden);
2. **new** File("...").listFiles(t -> t.isHidden());
3. **new** File("...").listFiles(myFunction);
4. **new** File("...").listFiles(myFunction::apply);
5. **new** File("...").listFiles(FileUtils::isHidden);
6. **new** File("...").listFiles(t -> FileUtils::isHidden);
7. **new** File("...").listFiles(t -> FileUtils::isHidden(t));
8. **new** File("...").listFiles(t -> FileUtils::isHiddenFile);
9. **new** File("...").listFiles(hiddenFilePredicatePlusPlus::test);

Java 8: A 4/8 Matching the method

- Given **public class** File {
  … **public** File[] listFiles(FileFilter filter) {…}; …}

  and @FunctionalInterface **interface** FileFilter {
  boolean accept(File pathname); }

  and

```
static class FileUtils {
    static boolean isHidden(File file) { return file.isHidden(); }
    boolean isHiddenFile(File file)    { return file.isHidden(); } }
    Predicate<File> hiddenFilePredicatePlusPlus = new Predicate<File>() {
        public boolean test(File t) { return t.isHidden(); };
        public void otherMethod() {} };
    Function<File,Boolean> myFunction = new Function<File,Boolean>() {
        @Override public Boolean apply(File t) { return t.isHidden(); } };
```

Which lines compile properly retrieving the hidden files? (FI = FunctionalInterface)

1. **new** File("...").listFiles(File::isHidden); // OK member method File->boolean with same signature as the FI of FileFilter
2. **new** File("...").listFiles(t -> t.isHidden()); // OK lambda File->boolean (same signature...)
3. **new** File("...").listFiles(myFunction); // BAD we need a method taking a File and returning a boolean, not a variable
4. **new** File("...").listFiles(myFunction::apply); // OK member method File->boolean (same...), unboxing is OK
5. **new** File("...").listFiles(FileUtils::isHidden); // OK static method File->boolean
6. **new** File("...").listFiles(t -> FileUtils::isHidden); // BAD if '->' is present then we should have code at right
7. **new** File("...").listFiles(t -> FileUtils::isHidden(t)); // BAD same and method references do not accept parameters
8. **new** File("...").listFiles(t -> FileUtils::isHiddenFile); // BAD if '->' is present then we should have code at right
9. **new** File("...").listFiles(hiddenFilePredicatePlusPlus::test); // OK File->boolean (same signature...)

Java 8: Q 5/8 Lambda in a loop

- Given:

```
5  public class LambadInTheLoop {
6      static Integer myStatic = Integer.valueOf(1);
7      final Integer myMember = Integer.valueOf(2);
8
9      public static void main(String[] args) {
10         LambadInTheLoop instance = new LambadInTheLoop();
11         for (int i = 0; i < 2; i++) {
12             final int j = 2 * i;
13             final int k = 4;
14
15             System.out.println((Supplier<String>) () -> {
16                 return "aaa[" + j + instance.myMember + "]"; });
17
18             System.out.println((Supplier<String>) () -> {
19                 return "bbb[" + System.getenv("yes") + " " + myStatic + k + "]"; });
20             System.out.println();
21         } } }
```

Explain the following output:

```
org.java8.Java8Quiz5LambadInTheLoop$$Lambda$1/12251916@d46ca6
org.java8.Java8Quiz5LambadInTheLoop$$Lambda$2/18340259@105068a

org.java8.Java8Quiz5LambadInTheLoop$$Lambda$1/12251916@132e575
org.java8.Java8Quiz5LambadInTheLoop$$Lambda$2/18340259@105068a
```

Java 8: A 5/8 Lambda in a loop

- Given:

```
5  public class LambadInTheLoop {
6      static Integer myStatic = Integer.valueOf(1);
7      final Integer myMember = Integer.valueOf(2);
8
9      public static void main(String[] args)
10         LambadInTheLoop instance = new Lamb
11         for (int i = 0; i < 2; i++) {
12             final int j = 2 * i;
13             final int k = 4;
14
15             System.out.println((Supplier<String>) () -> {
16                 return "aaa[" + j + instance.myMember + "]"; });
17
18             System.out.println((Supplier<String>) () -> {
19                 return "bbb[" + System.getenv("yes") + " " + myStatic + k + "]"; });
20             System.out.println();
21         } } }
```

```
Explain the following output:
org.java8.Java8Quiz5LambadInTheLoop$$Lambda$1/12251916@d46ca6
org.java8.Java8Quiz5LambadInTheLoop$$Lambda$2/18340259@105068a

org.java8.Java8Quiz5LambadInTheLoop$$Lambda$1/12251916@132e575
org.java8.Java8Quiz5LambadInTheLoop$$Lambda$2/18340259@105068a
```

- 1st lambda: referencing a local variable (final or not) with a changing value or an instance variable (even if always the same, final or primitive) or a member method forces the creation of a new "instance" of Lambda$1/12251916. Google for "java capturing lambda" for more details.
- 2nd lambda: referencing a local "constant" variable or a static method or a static field, the same instance of Lambda$2/18340259 can be reused for all the calls.

Java 8: 6/8 Q Create your own collector

Given this code using the default LongSummaryStatistics collector on the end result of the stream

```
public class Java8Quiz6MyCollector {
    public static void main(String[] args) {
        long[] myValues = new long[] { 1, 3, 5, 7, 8, 9, 10, 11, 12, 13, 14,
                14, 14, 14, 15, 16, 17, 18 };

        LongSummaryStatistics stats = Arrays.stream(myValues).boxed()
                .collect(Collectors.summarizingLong(Long::longValue));
        System.out.println(stats);
        // LongSummaryStatistics{count=18, sum=201, min=1, average=11.166667, max=18}
```

and this collector factory (static method from the Collector interface)

```
    /**
     * Returns a new {@code Collector} described by the given {@code supplier},
     * {@code accumulator}, and {@code combiner} functions.  The resulting
     * {@code Collector} has the {@code Collector.Characteristics.IDENTITY_FINISH}
     * characteristic.
     *
     * @param supplier The supplier function for the new collector
     * @param accumulator The accumulator function for the new collector
     * @param combiner The combiner function for the new collector
     * @param characteristics The collector characteristics for the new
     *                        collector
     * @param <T> The type of input elements for the new collector
     * @param <R> The type of intermediate accumulation result, and final result,
     *           for the new collector
     * @throws NullPointerException if any argument is null
     * @return the new {@code Collector}
     */
    public static<T, R> Collector<T, R, R> of(Supplier<R> supplier,
                                              BiConsumer<R, T> accumulator,
                                              BinaryOperator<R> combiner,
                                              Characteristics... characteristics) {
```

how can you create your own collector with the same results as LongSummaryStatistics but giving the standard deviation as well?

Java 8: 6/8 A Create your own collector (1/3)

```java
55      /** Step 1 : Extend LongSummaryStatistics to get the standard deviation */
56      static class MyStatistics extends LongSummaryStatistics {
57          // we need to keep all the values to calculate the standard deviation at the very end
58          private List<Long> values = new ArrayList<>();
59          private Double standardDeviation = null; // will be calculated at the end (once)
60          @Override // called to accept new values, keep the value for later
61          public void accept(long value) { super.accept(value); values.add(value); }
62          @Override // called to accept new values, keep the value for later
63          public void accept(int value) { super.accept(value); values.add((long)value); }
64          // called to combine 2 MyStatistics in 1 if we process the stream in //
65          public void combine(MyStatistics other) {
66              super.combine(other); // combine average, min, max, total count
67              values.addAll(other.values); // put all the values together
68          }
69          public double getStandardDeviation() { // stream all the values calculating
70              if (standardDeviation == null) {    // the average of (value-average)^2
71                  standardDeviation = Math.sqrt(values.stream()
72                          .mapToDouble(t -> (t - getAverage()) * (t - getAverage())).sum()
73                          / (getCount() - 1)); // see your old math course
74              }
75              return standardDeviation;
76          }
77          @Override  // adds the standard deviation to the inherited toSring()
78          public String toString() {
79              return super.toString()
80                      + ", std dev.=" + Double.toString(getStandardDeviation());
81          }
```

Java 8: 6/8 A Create your own collector (2/3)

```java
41    /**
42     * Step 2 : Creates a new Collector using MyStatistics using the factory in the interface.
43     */
44    static Collector<Long, MyStatistics, MyStatistics> myCollector = Collector.of(
45            MyStatistics::new,// The supplier function for the new collector. Use constructor reference.
46            (r, t) -> r.accept(t),// Accumulator function for new values (t). Call the overridden accept().
47            (l, r) -> {l.combine(r); return l;},//Combiner function for combining 2 results (r) in 1 (l).
48                // Call the overridden accept(). Combining in used if the code is executed in parallel.
49            Collector.Characteristics.IDENTITY_FINISH); //Collector characteristics. No post processing.
50
```

```java
19 public class Java8Quiz6MyCollector {
20     public static void main(String[] args) {
21         long[] myValues = new long[] { 1, 3, 5, 7, 8, 9, 10, 11, 12, 13, 14,
22                 14, 14, 14, 15, 16, 17, 18 };
23
24         LongSummaryStatistics stats = Arrays.stream(myValues).boxed()
25                 .collect(Collectors.summarizingLong(Long::longValue));
26         System.out.println(stats);
27         // LongSummaryStatistics{count=18, sum=201, min=1, average=11.166667, max=18}
28
29         // Step 3 : call your new collector
30         MyStatistics mystats = Arrays.stream(myValues).boxed()
31                 .collect(myCollector);
32         System.out.println(mystats);
33         // MyStatistics{count=18, sum=201, min=1, average=11.166667, max=18}, std dev.=4.829444006484049
```

Java 8: 6/8 A Create your own collector (3/3)

```
// Step 4 : for the same price, call your new collector using parallelization
MyStatistics mystatsParallel = Arrays.stream(myValues).boxed().parallel()
        .collect(myCollector);
System.out.println(mystatsParallel);

// in the real life it would be wise to tell when the parallelization should kick in using a
//                  Spliterator creating multiple streams only if the size is big enough.
// On a quad core system (8 threads), ForkJoin starts with 7 workers to map/reduce the calculation.
```

Debug | Outline

Java8Quiz6MyCollector [Java Application]
org.java8.Java8Quiz6MyCollector at localhost:52727
- Thread Group [system]
  - Thread Group [main]
    - Thread [main] (Suspended (breakpoint at line 66 in Java8Quiz6MyC
    - Daemon Thread [ForkJoinPool.commonPool-worker-1] (Suspended
    - Daemon Thread [ForkJoinPool.commonPool-worker-2] (Suspended
    - Daemon Thread [ForkJoinPool.commonPool-worker-3] (Suspended
    - Daemon Thread [ForkJoinPool.commonPool-worker-4] (Suspended
    - Daemon Thread [ForkJoinPool.commonPool-worker-5] (Suspended
    - Daemon Thread [ForkJoinPool.commonPool-worker-6] (Suspended
    - Daemon Thread [ForkJoinPool.commonPool-worker-7] (Suspended
      - Java8Quiz6MyCollector$MyStatistics.combine(Java8Quiz6MyColl
      - Java8Quiz6MyCollector.lambda$1(Java8Quiz6MyCollector$MySta
      - 4630167.apply(Object, Object) line: not available
      - ReduceOps$3ReducingSink.combine(ReduceOps$3ReducingSink
      - ReduceOps$3ReducingSink.combine(ReduceOps$AccumulatingS
      - ReduceOps$ReduceTask<P_IN,P_OUT,R,S>.onCompletion(Counte
      - ReduceOps$ReduceTask<P_IN,P_OUT,R,S>(CountedCompleter<T
      - ReduceOps$ReduceTask<P_IN,P_OUT,R,S>(AbstractTask<P_IN,P_(
      - ReduceOps$ReduceTask<P_IN,P_OUT,R,S>(CountedCompleter<T
      - ReduceOps$ReduceTask<P_IN,P_OUT,R,S>(ForkJoinTask<V>).doE
      - ForkJoinPool$WorkQueue.runTask(ForkJoinTask<?>) line: 916
      - ForkJoinPool.scan(ForkJoinPool$WorkQueue, int) line: 1689
      - ForkJoinPool.runWorker(ForkJoinPool$WorkQueue) line: 1644
      - ForkJoinWorkerThread.run() line: 157

C:\Program Files (x86)\Java\jdk1.8.0_25\bin\javaw.exe (Feb 15, 2015, 11:32:17

*Java8Quiz6MyCollector.java

```
54
55      /** Step 1 : Extend LongSummaryStatistics to get the standard deviation */
56      static class MyStatistics extends LongSummaryStatistics {
57          // we need to keep all the values to calculate the standard deviation at
58          private List<Long> values = new ArrayList<>();
59          private Double standardDeviation = null; // will be calculated at the en
60          @Override // called to accept new values
61          public void accept(long value) { super.accept(value); values.add(value);
62          @Override // called to accept new values
63          public void accept(int value) { super.accept(value); values.add((long)va
64          // called to combine 2 MyStatistics in 1 if we process the stream in //
65          public void combine(MyStatistics other) {
66              super.combine(other); // combine average, min, max, total count
67              values.addAll(other.values); // put all the value together
68          }
69          public double getStandardDeviation() { // stream all the values calculat
70              if (standardDeviation == null) {    // the average of (value-average
71                  standardDeviation = Math.sqrt(values.stream()
72                          .mapToDouble(t -> (t - getAverage()) * (t - getAverage()
73                          / (getCount() - 1)); // see your old math course
74              }
75              return standardDeviation;
76          }
77          @Override  // adds the standard deviation to the inherited toSring()
78          public String toString() {
79              return super.toString()
80                      + ", std dev.=" + Double.toString(getStandardDeviation());
81          }
82      }
83  }
84
85
```

Java 8: 7/8 Q Changes to the Java interface

Default method

Q1 How can I create a default method?

Q2 Why creating a default method?

Q3 Any restriction for default methods?

Static method

Q4 How can I create a static method?

Q5 Why creating a static method?

Q6 Any restriction for static methods?

Functional method and functional interface

Q7 How can I create a functional method?

Q8 Why creating a functional method?

Q9 Any restriction for functional methods?

Q10 What are the benefits of functional interfaces?

Java 8: 7/8 A Changes to the Java interface

Default method

Q1 How can I create a default method?

Prefix it with default and provide the code e.g. the new **default void List.sort(Comparator<? super E> c) { … }**

Q2 Why creating a default method?

Here, giving access to a new member method to all the sub classes w/o breaking any of them.

Q3 Any restriction for default methods?

Do not override a method of Object or finalize().

Static method

Q4 How can I create a static method?

Prefix it with static and provide the code e.g. the new **public static <T extends Comparable<? super T>>**
Comparator<T> Comparator.reverseOrder() { return Collections.*reverseOrder();* }

Q5 Why creating a static method?

Utility method, factory, builder, composition of instances of the interface, etc.

Q6 Any restriction for static methods?

When you call it, you MUST prefix it w/ the name of the class e.g. … Comparator.reverseOrder() and not an instance name

Functional method and functional interface

Q7 How can I create a functional method?

This has to be the only non default, non static, not present in Object method present. Basically a plain old method.

Q8 Why having a functional method?

With a functional method, tagged or not with @FunctionalInterface, instances of the interface can be created w/ a lambda

Q9 Any restriction for functional methods?

Only one functional method in your FunctionalInterface.

Q10 What are the benefits of functional interfaces?

In addition of being less verbose than anonymous classes, functional interfaces are heavily used in collections and streams and allow functional programming vs imperative data processing (removing if, switch… from the picture)

Java 8: 8/8 Q New @FunctionalInterface (FI) interfaces

- What can I do with the new FIs?

- Q1 Supplier<T>
- Q2 Consumer<T> and BiConsumer<T,U>
- Q3 Predicate<T>
- Q4 Function<T,R> and BiFunction<T,U,R>
- Q5 UnaryOperator<T> and BinaryOperator<T>

- Q6 What should I do if I pass or return primitives?

Java 8: 8/8 A New @FunctionalInterface (FI) interfaces

- What can I do with the new FIs?
  - Use them in the Stream and Collection API to make your code more compact and functional. When using them, avoid modifying the passed in objects. As a FI you can use a lambda to provide the code.
- Q1 Supplier<T>
  - Represents a supplier of results. There is no requirement that a new or distinct result be returned each time the supplier is invoked. T get(). Can be used like a factory.
- Q2 Consumer<T> and BiConsumer<T,U>
  - Represents an operation that accepts a single input argument and returns no result. Unlike most other functional interfaces, Consumer is expected to operate via side-effects. void accept(T t). BiConsumer takes 2 parameters.
- Q3 Predicate<T>
  - Represents a predicate (boolean-valued function) of one argument. This is a FI whose functional method is boolean test(T). Predicates can be composed (e.g. p1.and(p2).test(t).. They can be used to filter. BiPredicate takes 2 parameters.
- Q4 Function<T,R> and BiFunction<T,U,R>
  - public interface Function<T,R> Represents a function that accepts one argument and produces a result. This is a FI whose functional method is R apply(T t). Functions can be composed (e.g. g(f(x)). They are used to transform. BiFunction<T,U,R> takes 2 parameters.
- Q5 UnaryOperator<T> and BinaryOperator<T>
  - This is a specialization of Function for the case where the operand and result are of the same type. BinaryOperator takes 2 parameters.
- Q6 What should I do if I pass or return primitives?
  - These classes offer some specialized versions avoiding the performance penalty of boxing and unboxing primitives

    e.g. DoubleToIntFunction, BooleanSupplier…

Java 8: Have more fun

- Mother's and Father's Day are coming. Put *Java 8 in action, Manning 2014* in your basket.
- As a certified OCP 7, for a limited time pass the beta exam for the upgrade to Java 8 programmer for 50 USD vs. 250 usually.
- Last public Java 7 release in April !
- Be like the Chinese mandarin of war at right. Get ready for the real action !

.