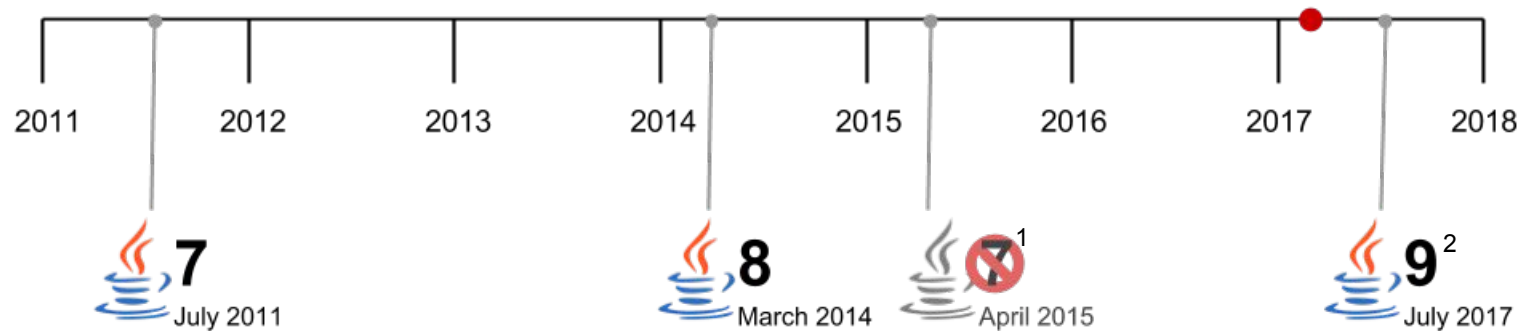




Java release timeline



What's new³ - Language (1/3)

- **Lambdas & Method references**
- **Default methods (interfaces)**
- Repeating annotations
- Type annotations
- Improved type inference
- **Method parameter reflection**

³ <http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html>

What's new³ - APIs (2/3)

- Streaming (`java.util.stream.*`)
- Optional (`java.util.optional`)
- Time (`java.time.*`)

³ <http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html>

What's new³ - Other (3/3)

- Nashorn
- Security improvements
- JavaFX
- Compact profiles
- Improved javac and javadoc tool
- Unicode enhancements
- Concurrency improvements, DB improvements, networking improvements....
- Many more...

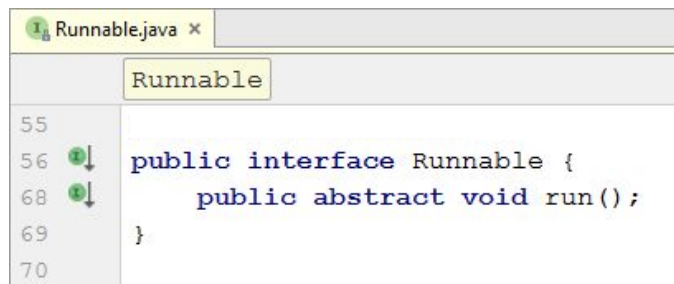
³ <http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html>

Lambdas



Passing behaviour as arguments

Lambdas



The screenshot shows a code editor window titled 'Runnable.java'. A search bar at the top contains the text 'Runnable'. The code defines the 'Runnable' interface with the following structure:

```
55  
56 public interface Runnable {  
68     public abstract void run();  
69 }  
70
```


Lambdas

```
Runnable.java x
Runnable

55
56 public interface Runnable {
68     public abstract void run();
69 }
70
```

```
Lambdas.java x

5
6
7 final Runnable r1 = new Runnable() {
8     @Override
9     public void run() {
10         System.out.println("Executing logic...");
11     }
12 };
13 r1.run();
```

Lambdas

```
Runnable.java x
Runnable
55 @FunctionalInterface
56 public interface Runnable {
68     public abstract void run();
69 }
70
```

```
Lambdas.java x
5
6
7 final Runnable r1 = new Runnable() {
8     @Override
9     public void run() {
10         System.out.println("Executing logic...");
11     }
12 };
13 r1.run();
```

Lambdas



```
Runnable.java x
Runnable
55 @FunctionalInterface
56 public interface Runnable {
68     public abstract void run();
69 }
70
```

```
Lambdas.java x
5
6
7
8
9
10
11
12
13
14

final Runnable r1 = new Runnable() {
    @Override
    public void run() {
        System.out.println("Executing logic...");
    }
};
r1.run();

final Runnable r2 = () -> System.out.println("Executing logic");
r2.run();
```

Java lambdas are syntactic sugar for anonymous classes implementing a functional interface

Lambdas - Functional Interfaces

1 Consumer.java ×

```
41 @FunctionalInterface
42 public interface Consumer<T> {
49     void accept(T t);
67 }
68
```

1 Predicate.java ×

```
41 @FunctionalInterface
42 public interface Predicate<T> {
49     boolean test(T t);
67 }
68
```

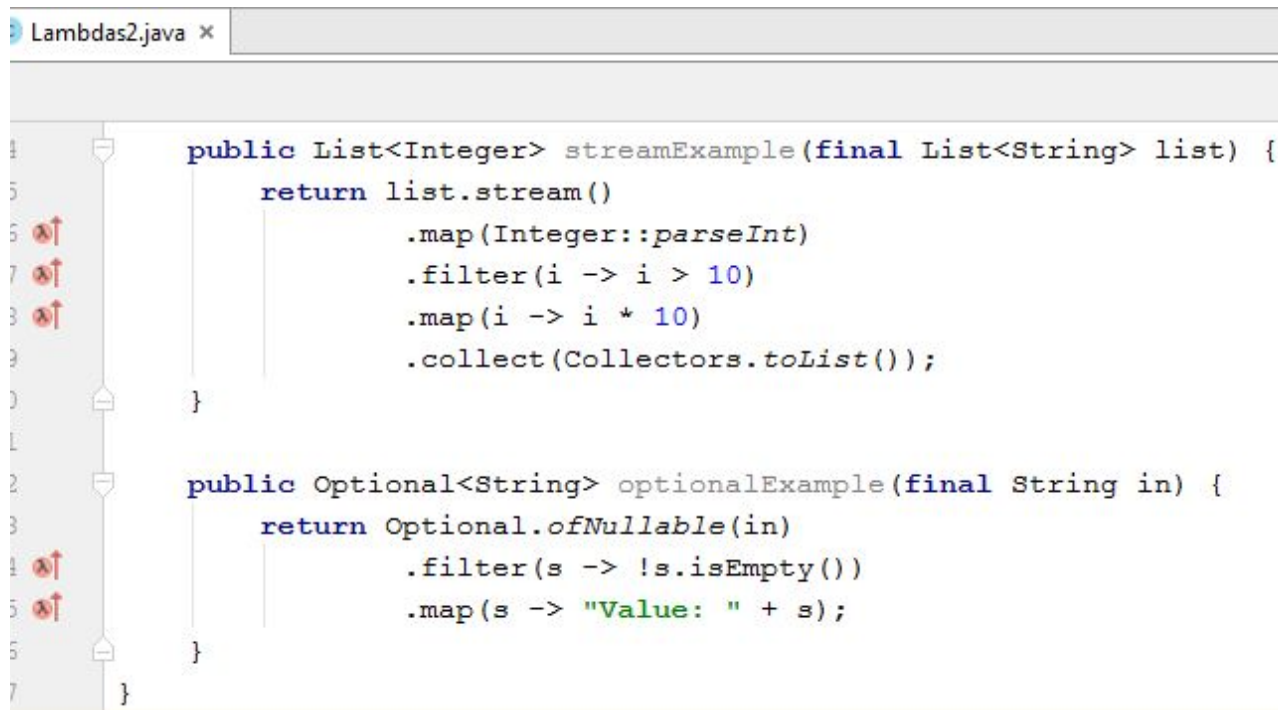
1 Function.java ×

```
41 @FunctionalInterface
42 public interface Function<T, R> {
49     R apply(T t);
67 }
68
```

Lambdas - Examples

```
Lambdas.java x
8      public void runnableExample() {
9          final Thread thread = new Thread(() -> {
10              System.out.println("Running thread now");
11              System.out.println("Executing some logic");
12              System.out.println("Ending thread");
13          });
14          thread.start();
15      }
16
17      public void consumerExample(final List<Integer> numbers) {
18          numbers.forEach((Integer number) -> System.out.println(number));
19          numbers.forEach((number) -> System.out.println(number));
20          numbers.forEach(number -> System.out.println(number));
21          numbers.forEach(System.out::println);
22      }
```

Lambdas - Examples



```
Lambdas2.java x

1 public List<Integer> streamExample(final List<String> list) {
2     return list.stream()
3         .map(Integer::parseInt)
4         .filter(i -> i > 10)
5         .map(i -> i * 10)
6         .collect(Collectors.toList());
7 }

8
9
10 public Optional<String> optionalExample(final String in) {
11     return Optional.ofNullable(in)
12         .filter(s -> !s.isEmpty())
13         .map(s -> "Value: " + s);
14 }
15 }
```

Default methods

Default methods

Interfaces can have method implementations

Default methods

MyInterface v1.0

- abstract a()
- abstract b()



MyClass implements MyInterface

Default methods

MyInterface v1.0

- abstract a()
- abstract b()

MyInterface v2.0

- abstract a()
- abstract b()
- abstract c()



MyClass implements MyInterface

Default methods

MyInterface v1.0

- abstract a()
- abstract b()

MyInterface v2.0a

- abstract a()
- abstract b()
- ~~- abstract c()~~

MyInterface v2.0b

- abstract a()
- abstract b()
- default c() {
 /* Implementation. */
}

Default methods

```
1 package nl.jcore.java8demo.defaultmethods;
2
3 import sun.reflect.generics.reflectiveObjects.NotImplementedException;
4
5 public interface DefaultMethodExample {
6     void demonstrate();
7
8     default void demonstrateDefault() {
9         final String assignmentVar = "like any other";
10        String.format("A method implementation, %s.", assignmentVar);
11    }
12
13    default String demonstrateDefaultB() {
14        throw new NotImplementedException();
15    }
16 }
```

Method parameter reflection

Method parameter reflection

Accessing a method's parameters' names

Method parameter reflection

Reflection!

```
ParameterNamesExample.java x
6  private class Test {
7      public void entryMethod(final String testArg1, final String testArg2) { }
8  }
9
10 private void getMethodInfo() throws NoSuchMethodException {
11     final Method method = Test.class.getDeclaredMethods()[0];
12     final int numParameters = method.getParameterCount();
13     final String arg0Name = method.getParameters()[0].getName();
14     System.out.println(numParameters); // Prints "2"
15     System.out.println(arg0Name); // Prints "arg0"
16 }
```


Method parameter reflection

Compile code (javac) with -parameter flag

→ Command line: `javac *.java -parameters`

→ Allow e.g. Jackson or Spring to get parameter name at runtime

→ Configurable in e.g. Maven or Gradle plugins

```
ParameterNamesController.java x
7  @RestController
8  @RequestMapping("/api/1/demo/parameter-reflection/requestbody")
9  public class ParameterNamesController {
10     @RequestMapping
11     public void demoOld(@RequestParam("name") final String name) { }
12
13     @RequestMapping
14     public void demoNew(@RequestParam final String name) { }
15 }
16
```

Method parameter reflection

```
ParameterNamesDto.java x
8 private class ParameterReflectionDto {
9     private final int id;
10    private final String name;
11
12    @JsonCreator
13    public ParameterReflectionDto(final int id, final String name) {
14        this.id = id;
15        this.name = name;
16    }
17
18    public int getId() {
19        return id;
20    }
21
22    public String getName() {
23        return name;
24    }
25 }
```

```
ParameterNamesDto.java x
27 @RestController
28 @RequestMapping("/api/1/demo/parameter-names/dto")
29 public class ParameterReflectionController {
30     @RequestMapping
31     public void demo(final ParameterReflectionDto dto) {
32     }
33 }
```

Streaming API

Streaming API

“[Streams are] an abstraction for expressing efficient, SQL-like operations on a collection of data.”⁴

⁴ <http://www.oracle.com/technetwork/articles/java/ma14-java-se-8-streams-2177646.html>

From a list of users, get the first 10 names of users with age of 18 or higher

Streaming API



```
StreamingExample.java x
StreamingExample java8Streaming()

57 public List<String> java7StreamingEquivalent() {
58     final List<String> result = new ArrayList<>();
59     for (final User user : getUsers()) {
60         if (user.age >= 18) {
61             result.add(user.getName());
62         }
63         if (result.size() >= 10) {
64             break;
65         }
66     }
67     return result;
68 }
```

Input: list of users

- Filter out users below age 18
- Transform full users to names
 - Limit to 10

Streaming API



StreamingExample.java x

StreamingExample java8Streaming()

```
89 public List<String> java8Streaming() {  
90     return getUsers().stream()  
91         .filter(u -> u.getAge() >= 18)  
92         .map(User::getName)  
93         .limit(10)  
94         .collect(Collectors.toList());  
95 }
```


Streaming API

From a list of users, get the names and last transaction of the 10 users that last performed a transaction and are of age 18 or higher.

Streaming API

Input: list of users

- Filter out users below age 18
- Filter out users that have not performed any transaction
 - Sort by last transaction timestamp
- Transform full users to names with last transaction
 - Limit to 10

Streaming API



```
StreamingExample.java x
StreamingExample java8Streaming()

70 public Map<String, Transaction> java7StreamingMapEquivalent() {
71     final Map<Long, User> sortedUsers = new TreeMap<>();
72     for (final User user : getUsers()) {
73         if (user.age >= 18 && user.hasTransactions()) {
74             sortedUsers.put(getLastTransaction(user).getTimestamp().toEpochMilli(), user);
75         }
76     }
77     final Map<String, Transaction> result = new HashMap<>();
78     final ArrayList<Long> keys = new ArrayList<>(sortedUsers.keySet());
79     for (int i = keys.size() - 1; i >= 0; i--) {
80         final User user = sortedUsers.get(i);
81         result.put(user.getName(), getLastTransaction(user));
82         if (result.size() >= 10) {
83             break;
84         }
85     }
86     return result;
87 }
```

Streaming API

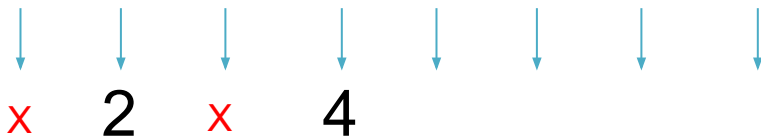
StreamingExample.java x

StreamingExample java8Streaming()

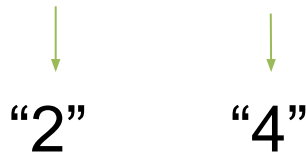
```
97 public Map<String, Transaction> java8StreamingMap() {
98     final Comparator<User> compareLastTransactionTimestamp =
99         Comparator.comparingLong(u -> getLastTransaction(u).getTimestamp().toEpochMilli());
100     return getUsers().stream()
101         .filter(u -> u.getAge() >= 18)
102         .filter(User::hasTransactions)
103         .sorted(compareLastTransactionTimestamp)
104         .limit(10)
105         .collect(Collectors.toMap(User::getName, this::getLastTransaction));
106 }
```

List<Integer> numbers = { 1, 2, 3, 4, 5, 6, 7, 8 };

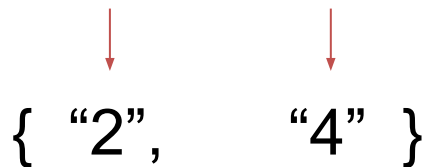
Filtering



Mapping



Collecting



```
numbers.stream()
    .filter(i -> i % 2 == 0)
    .map(String::valueOf)
    .limit(2)

.collect(Collectors.toList());
```

Streaming



Declarative

Highly efficient

Parallelizable

Optional

“A container object which may or may not contain a non-null value.”⁵

⁵ <https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html>

Optional

- Traditionally *null* is used
 - Nullpointer exceptions
 - Holder for optional values
-
- C++: Optional
 - C#: ?T
 - Haskell: *Maybe*
 - Kotlin: T?
 - Scala: Option
 - Swift: Optional

Optional

```
OptionalExample.java x
11 @RestController
12 @RequestMapping("/api/1/demo/optional")
13 public class OptionalExample {
14     @RequestMapping
15     public void searchTransactions(final Optional<String> terms, final Optional<Instant> fromTime) {
16         final Instant reasonableDefaultFromTime = Instant.now().minus(1, ChronoUnit.DAYS);
17         search(terms, fromTime.orElse(reasonableDefaultFromTime));
18     }
19
20     private void search(final Optional<String> terms, final Instant fromTime) {
21     }
22 }
```

Optional

Initialization:

- `Optional.of(valueThatMayNotBeNull);` → throws `NullPointerException` when value is null.
- `Optional.ofNullable(valueThatMaybeNull);`

Optional

```
OptionalExample2.java x
6      public void doSomethingIfPresent(final Optional<String> optString) {
7          if (optString.isPresent()) {
8              doSomethingWithString(optString.get());
9          }
10     }
11
12     private void doSomethingWithString(final String str) {
13     }
14 }
```

Optional



```
OptionalExample3.java x
6      public void doSomethingIfPresent(final Optional<String> optString) {
7          optString.ifPresent(this::doSomethingWithString);
8
9          optString
10             .map(String::trim)
11             .filter(s -> !s.isEmpty())
12             .ifPresent(this::doSomethingWithString);
13     }
14
15     private void doSomethingWithString(final String str) {
16     }
17 }
```

Optional

- Repository
- Streaming
- Controller

Alternatives

- @NonNull
- @Nullable

Time



Time

Java 7 `java.util.Date`:

- Not thread-safe
- Years start at 1900, months start at 1; days start at 0
- Poor concepts

Time

Java 8 java.time.*:

- Immutable values
- Domain Driven Design
- Enhanced support for non-ISO calendar systems

Time - Domain Driven Design

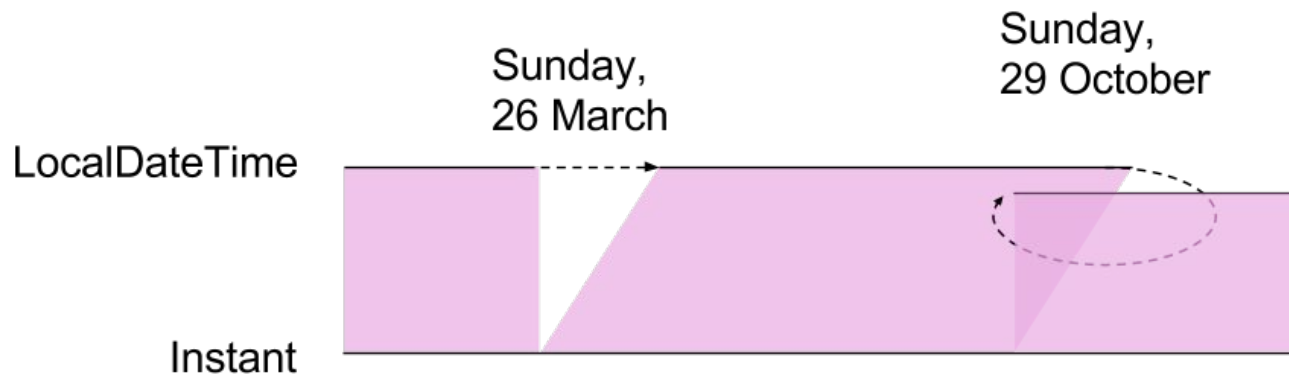
- LocalDate, LocalTime, LocalDateTime
- ZonedDateTime
- Period, Duration
- Instant
- Other classes for non-ISO calendaring systems

Time - Instant

“An instantaneous point on the time-line.”⁶

⁶ <https://docs.oracle.com/javase/8/docs/api/java/time/Instant.html>

Time - Instant



Conclusion

- Language
 - Lambdas
 - Default methods
 - Method parameter reflection
- APIs
 - Streaming
 - Optional
 - Time

Extended reading

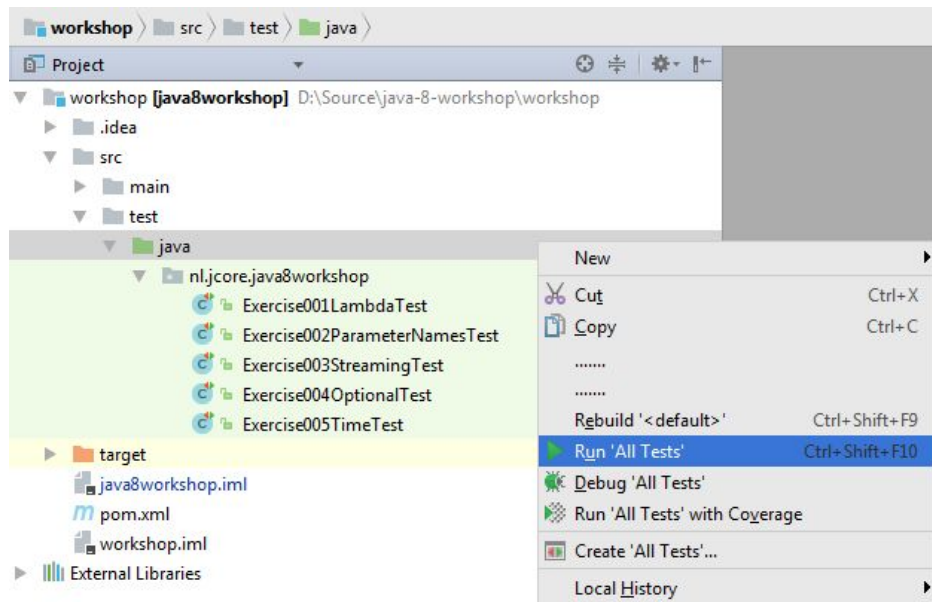
[Java 8 migration guide \(compatibility information, removed features and APIs etc.\)](#)

[Lambdas - A peek under the hood](#)

[Optional Method Parameters](#)

Workshop

git clone <https://github.com/jcoreNL/workshop-java-8>



- Open Maven project
java-8-workshop/workshop
with your IDE
- Run unit tests
- Solve them.

Lambda - Ambiguity

```

22  /**
23   * @return function that converts it's Integer input to a String
24   */
25  @ public static Function<Integer, String> functionToString() {
26      return Integer::toString;
27  }
28  }
29

```

Incompatible types.
 Required: Function <Integer, String>
 Found: Function <Integer, String>

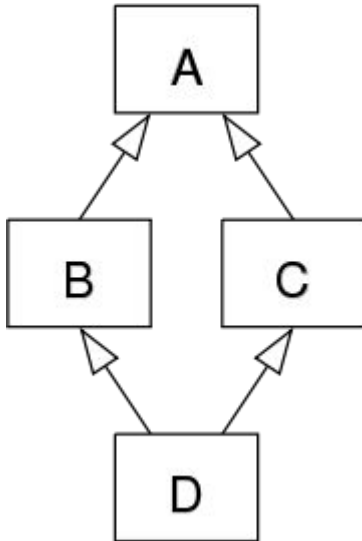
→ Integer.toString(i)
 → (new Integer(i)).toString();

```

22  /**
23   * @return function that converts it's Integer input to a String
24   */
25  @ public static Function<Integer, String> functionToString() {
26      return String::valueOf;
27  }
28  }

```


Default methods - Diamond problem



```

DefaultExample.java x
1 package nl.jcore.java8demo.defaultmethods;
2
3 interface A {
4     void test();
5 }
6
7 interface B extends A {
8     default void test() {
9         System.out.println("B");
10    }
11 }
12
13 interface C extends A {
14     default void test() {
15         System.out.println("C");
16    }
17 }
18
19 class D implements B, C {
20     // D inherits unrelated defaults for test() from types B and C
21 }
22
23 class E implements B, C {
24     public void test() {
25         B.super.test();
26     }
27 }
28
  
```

Type inferencing

Java 6:

```
final Map<String, String> map = new HashMap<String, String>();
```

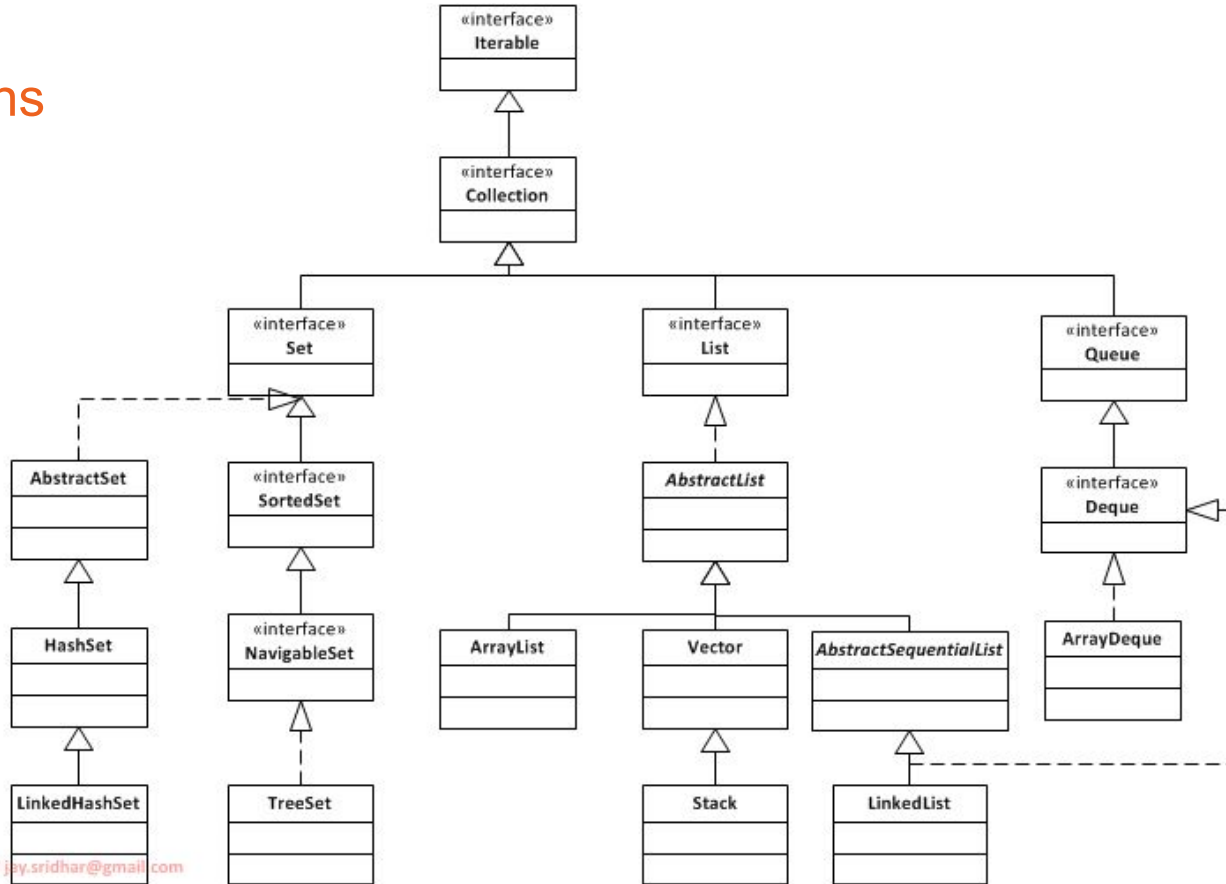
Java 7:

```
final Map<String, String> map = new HashMap<>();  
final ImmutableMap<String, String> map = ImmutableMap<String, String>.of();
```

Java 8:

```
final ImmutableMap<String, String> map = ImmutableMap.of();
```

Collections



Java 9

- Tooling
 - JShell (Read, Eval, Print Loop)
 - Microbenchmarking
- Language
 - Modular JDK / Jigsaw
 - Default garbage collector
- APIs
 - Process API
 - Enhanced deprecation
 - Improved logging API
 - HTTP 2.0

<https://www.jcp.org/en/jsr/detail?id=379>

Homework

Read these articles on Optionals and Streaming in Java 8

- <https://blog.jetbrains.com/idea/2016/07/java-8-top-tips/>
- <https://dzone.com/articles/whats-wrong-java-8-part-iii>
- <https://dzone.com/articles/whats-wrong-java-8-part-iv>

Follow the Optional evolution through Java 8, 9 and 10

- <https://dzone.com/articles/java-8-optional-replace-your-get-calls>
- <http://www.deadcoderising.com/java-9-10-4-new-methods-in-the-optional-api/>