# Java: Did You Know That?

Jeanne Boyarsky
Thursday February 20, 2020
DevNexus

speakerdeck.com/boyarsky

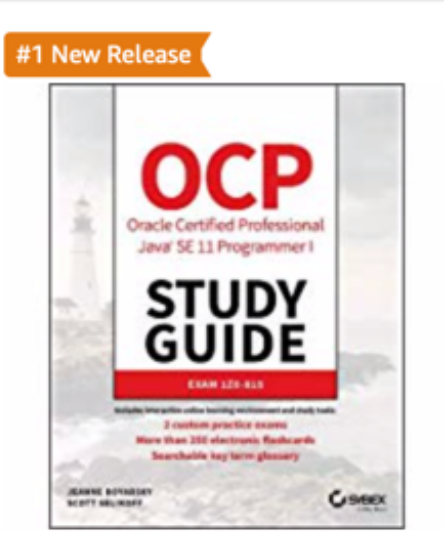
@jeanneboyarsky

# About Me


- Java Champion
- Author
- Developer at NYC bank for 17+ years
- FIRST Robotics Mentor

@jeanneboyarsky

# Pause for a Commercial


Java 11 certs

- 1Z0-815 - Out now
- 1Z0-816 - April ETA

@jeanneboyarsky

# With Contributions By

- Janeice DelVecchio
- Elena Felder
- Scott Selikoff

# Removing in a Loop

# Removing in a Loop

```
List<String> list = new ArrayList<>();
list.add("red");
list.add("alliance");
list.add(“blue");
list.add("alliance");

for(String current : list) {
    list.remove(current);
}

System.out.println(list);
```

Output:
ConcurrentModificationException

# Removing in a Loop

```
List<String> list = new ArrayList<>();
list.add("red");
list.add("alliance");

for(String current : list) {
    list.remove(current);
}

System.out.println(list);
```

Output:
[alliance]

# Behavior Explanation

1. List starts as [red, alliance]
2. First iteration through loop, current = red
3. In loop, remove red
4. Now list is [alliance]
5. For loop checks size.
6. Size = 1; already saw one element
7. Done!

# Behavior Explanation

1. vs List as [red, alliance, blue, alliance]
2. Size = 3 after removal and saw one
3. Not Done!
4. Exception

# Removing in a Loop

```
List<String> list
    = new CopyOnWriteArrayList<>();
list.add("red");
list.add("alliance");
list.add("blue");
list.add("alliance");

for(String current : list) {
    list.remove(current);
}
```

Output:
[]

# Or ditch the loop

```
list.clear();

(or)

list.removeIf(x -> true);
```

Output:
[]

# Creating a Set

# Creating a Set

```
String[] words = { "all", "the", “words",
  "in", "the", "world" };

Set<String> set = new HashSet<>(
  Arrays.asList(words));

System.out.println(set);
```

Output like:
[all, the, world, in, words]

# Creating a Set

```
String[] words = { "all", "the", “words",
   "in", "the", "world" };

Set<String> set = Set.of(words);


System.out.println(set);
```

Output:
IllegalArgumentException: duplicate element: the

# Behavior Explanation

1. Set.of() takes varargs.
2. Doc says can't have duplicates

# Backed Collection

# Collection

```
Map<String, String> map = new HashMap<>();
map.put("Braves", "Atlanta");
map.put("Mets", "NYC");

Set<String> keys
    = new HashSet<>(map.keySet());
keys.remove("Mets");

System.out.println(map);
```

```
{Mets=NYC, Braves=Atlanta}
```

# Collection

```
Map<String, String> map = new HashMap<>();
map.put("Braves", "Atlanta");
map.put("Mets", "NYC");

Set<String> keys = map.keySet();
keys.remove("Mets");

System.out.println(map);
```

```
{Braves=Atlanta}
```

# Behavior Explanation

1. keySet(), values(), and entrySet() are backed collections
2. When change any, it affects original map

# Overloading

# Overloading

```
public void check(Number number) {
    System.out.print("Number "); }
public void check(Integer integer) {
    System.out.print("Integer "); }
public void delegator(Number number) {
    check(number); }

Integer num = Integer.valueOf(42);
Overloading target = new Overloading();
target.check(num);
target.delegator(num);
```

Output:
Integer Number

# Overloading

```
public void check(Number number) {
    System.out.print("Number "); }
public void check(Integer integer) {
    System.out.print("Integer "); }
public void delegator(Number number) {
    check(number); }

Number num = Integer.valueOf(42);
Overloading target = new Overloading();
target.check(num);
target.delegator(num);
```

Output:
Number Number

# Behavior Explanation

1. Method chosen at compile time, not runtime.
2. (vs polymorphism on objects)

==

# Equality

```
Integer int1 = Integer.valueOf(8);
Integer int2 = Integer.valueOf(8);
System.out.println((int1 == int2)
    + " " + int1.equals(int2));
```

```
true true
```

# Equality

```
Integer int1 = Integer.valueOf(8_000);
Integer int2 = Integer.valueOf(8_000);
System.out.println((int1 == int2)
    + " " + int1.equals(int2));
```

false true

# Behavior Explanation

1. Wrapper classes cache small values
2. Lesson: always use equals() for objects

# var and inference

# var and inference

```
List<Integer> x1 = List.of();
List<Integer> x2 = List.of(1, 2, 3);
Stream.of(x1, x2)
        .flatMap(x -> x.stream())
        .map(x -> x + 1)
        .forEach(System.out::print);
```

Output:
234

# var and inference

```
List<Integer> x1 = List.of();
var x2 = List.of(1, 2, 3);
Stream.of(x1, x2)
        .flatMap(x -> x.stream())
        .map(x -> x + 1)
        .forEach(System.out::print);
```

Output:
234

# var and inference

```
var x1 = List.of();
var x2 = List.of(1, 2, 3);
Stream.of(x1, x2)
        .flatMap(x -> x.stream())
        .map(x -> x + 1)
        .forEach(System.out::print);
```

Compiler error:
Operator ‘+’ cannot be applied to capture<?>, int

# Behavior Explanation

1. var x2 = List.of(1,2,3)
2. x2 is a List<Integer>
3. var x1 = List.of()
4. x1 is a not a List<Integer>
5. Stream.of(x1, x2) is a Stream<List<? extends Object>>
6. flatMap makes Stream<? extends Object>
7. Object doesn't go with +

# Sorting Characters

@jeanneboyarsky

# Sorting Characters

```
Stream<String> ohMy = Stream.of(
   "lions", "tigers", "bears");

Comparator<Character> c =
   Comparator.naturalOrder();

System.out.println(ohMy.collect(
   Collectors.groupingBy(String::length,
   Collectors.mapping(s ->
   s.charAt(0), Collectors.minBy(c)))));
```

Output:
{5=Optional[b], 6=Optional[t]}

# Sorting Characters

```
Stream<String> ohMy = Stream.of(
   "lions", "tigers", "bears");

System.out.println(ohMy.collect(
   Collectors.groupingBy(String::length,
   Collectors.mapping(s ->
   s.charAt(0), Collectors.minBy(
   Comparator.naturalOrder()))))));
```

Compiler error
No suitable method found for….

# The actual message

Error:(18, 51) java: no suitable method found for groupingBy(String::length,java.util.stream.Collector<java.lang.Object,capture#1 of ?,java.util.Optional<T>>)
    method java.util.stream.Collectors.<T,K>groupingBy(java.util.function.Function<? super T,? extends K>) is not applicable
      (cannot infer type-variable(s) T,K
        (actual and formal argument lists differ in length))
    method java.util.stream.Collectors.<T,K,A,D>groupingBy(java.util.function.Function<? super T,? extends K>,java.util.stream.Collector<? super T,A,D>) is not applicable
      (inference variable U has incompatible upper bounds java.lang.Object,java.lang.Comparable<? super T>,T,T)…

# The other message

```
Error:(19, 52) java: cannot find symbol
  symbol:   method charAt(int)
  location: variable s of type java.lang.Object
```

# Behavior Explanation

1. char != Character
   Inferred type check fails
2. Propagates error to other call
3. But, this suggests a
   workaround…

# Sorting Characters

```
Stream<String> ohMy = Stream.of(
   "lions", "tigers", "bears");

System.out.println(ohMy.collect(
   Collectors.groupingBy(String::length,
   Collectors.mapping((String s) ->
   s.charAt(0), Collectors.minBy(
   Comparator.naturalOrder()))))));
```

Output:
{5=Optional[b], 6=Optional[t]}

# URL Equality

# URL Equality

```
URL url1 = new URL("https://google.com");
URL url2 = new URL("https://google.com");
System.out.println(url1.equals(url2));
```

Output:
true

# URL Equality

```
URL url1 = new URL("cloudURL");
URL url2 = new URL(“cloudURL");
System.out.println(url1.equals(url2));
```

Output:
false if DNS resolution changes during program

# Behavior Explanation

1. URL's equals method calls the URLStreamHandler's equals method
2. Which uses DNS resolution
3. Cloud URLs change often

Instead, use URI
https://news.ycombinator.com/item?id=21765788

# Week of the Year

# Week of Year

```
LocalDate xmasEve = LocalDate.of(
    2019, Month.DECEMBER, 24);
WeekFields weekFields = WeekFields.of(
    Locale.getDefault());

int weekNumber = xmasEve.get(
    weekFields.weekOfWeekBasedYear());
int week = xmasEve.get(
    weekFields.weekOfYear());
System.out.println(weekNumber
     + " " + week);
```

Output:
52 52

# Week of Year

```
LocalDate newYearsEve = LocalDate.of(
    2019, Month.DECEMBER, 31);
WeekFields weekFields = WeekFields.of(
    Locale.getDefault());

int weekNumber = newYearsEve.get(
    weekFields.weekOfWeekBasedYear());
int week = newYearsEve.get(
    weekFields.weekOfYear());
System.out.println(weekNumber
     + " " + week);
```

Output:
1 53

# Behavior Explanation

1. Most years are 52 weeks + 1 day
2. 2020 started on a Wednesday.
3. The week based week starts the Sunday before
4. Whereas weekOfYear() returns 1-54

# Week of Year

```
LocalDate xmasEve = LocalDate.of(
    2019, Month.DECEMBER, 24);
DateTimeFormatter format =
    DateTimeFormatter.ofPattern(
    "yyyy-MM-dd");
DateTimeFormatter format2 =
    DateTimeFormatter.ofPattern(
    "YYYY-MM-dd");
System.out.println(xmasEve.format(format)
    + " " + xmasEve.format(format2));
```

Output:
2019-12-24 2019-12-24

# Week of Year

```
LocalDate newYearsEve = LocalDate.of(
    2019, Month.DECEMBER, 31);
DateTimeFormatter format =
    DateTimeFormatter.ofPattern(
    "yyyy-MM-dd");
DateTimeFormatter format2 =
     DateTimeFormatter.ofPattern(
     "YYYY-MM-dd");
System.out.println(
    newYearsEve.format(format) + " "
    + newYearsEve.format(format2));
```

Output:
2019-12-31 2020-12-31

# Behavior Explanation

1. y is year
2. Y is week of year
3. Be careful with week of year!

# Closing Resources

# JDBC

```
PreparedStatement stmt =
    conn.prepareStatement(
    "update mytable set updated = now()");
try (stmt) {
    stmt.executeUpdate();
}
```

"Good"

# JDBC

```
PreparedStatement stmt =
    conn.prepareStatement(
    "update mytable set updated = now()" +
    " where key = ?");

stmt.setString(1, "abc");
try (stmt) {
    stmt.executeUpdate();
}
```

Resource leak!

# Behavior Explanation

1. What happens if an exception is thrown while calling the PreparedStatement setter?
2. The try-with-resources doesn't run
3. Resource leak!

# JDBC

```
PreparedStatement stmt =
    conn.prepareStatement(
    "update mytable set updated = now()" +
    " where key = ?");

try (stmt) {
    stmt.setString(1, "abc");
    stmt.executeUpdate();
}
```

“good”

# IO

```
try (BufferedReader reader =
    Files.newBufferedReader(path)) {

    String line = null;
    while
      ((line = reader.readLine())!= null) {
      // process line
     }
}
```

No leak

# IO

```
Files.lines(path).count();
```

Resource leak!

# Behavior Explanation

1. Resource not closed by terminal operation
2. Affects
    1. find()
    2. lines()
    3. list()
    4. newDirectoryStream()
    5. walk()

# IO

```
try (Stream<String> stream
     = Files.lines(path)) {

  stream.count();
}
```

Good

# PSA: Free Tools

- PMD
- FindBugs
- CheckStyle
- SonarQube - examples:
  - No return in finally
  - Close resources
  - All sorts of bugs

# Other Favorites?