

# Refactoring Lab: To Java 17 and beyond

Jeanne Boyarsky  
August 10, 2022  
KCDC

[speakerdeck.com/boyarsky](https://speakerdeck.com/boyarsky)



## Jeanne Boyarsky

### Java 17 Cert Book Author



Jeanne Boyarsky is a Java Champion from New York City and has been a Java developer for more than 20 years. She has co-authored Wiley's Oracle Java 8/11/17 certification books. Jeanne also serves as her team's Scrum Master. She volunteers at CodeRanch and mentors the programmers on a high school robotics team in her free time. Jeanne has spoken at numerous conferences including Dev Nexus, KCDC, QCon, and Oracle Code One.

- **Refactoring Lab: To Java 17 and beyond**
- **Refactoring to Java 17 and beyond**

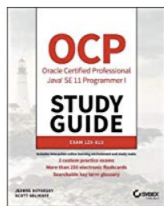
# Pause for a Commercial

## Amazon Best Sellers

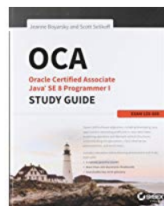
Our most popular products based on sales. Updated hourly.

### Best Sellers in Oracle Certification

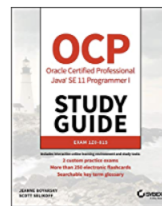
#1



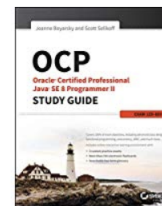
#2



#3



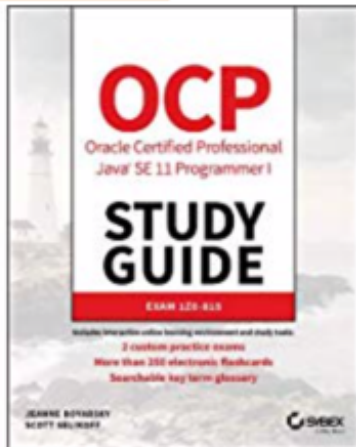
#4



#5



#1 New Release



Java certs: 8/11/17

Book giveaway at 3pm!

@jeanneboyarsky



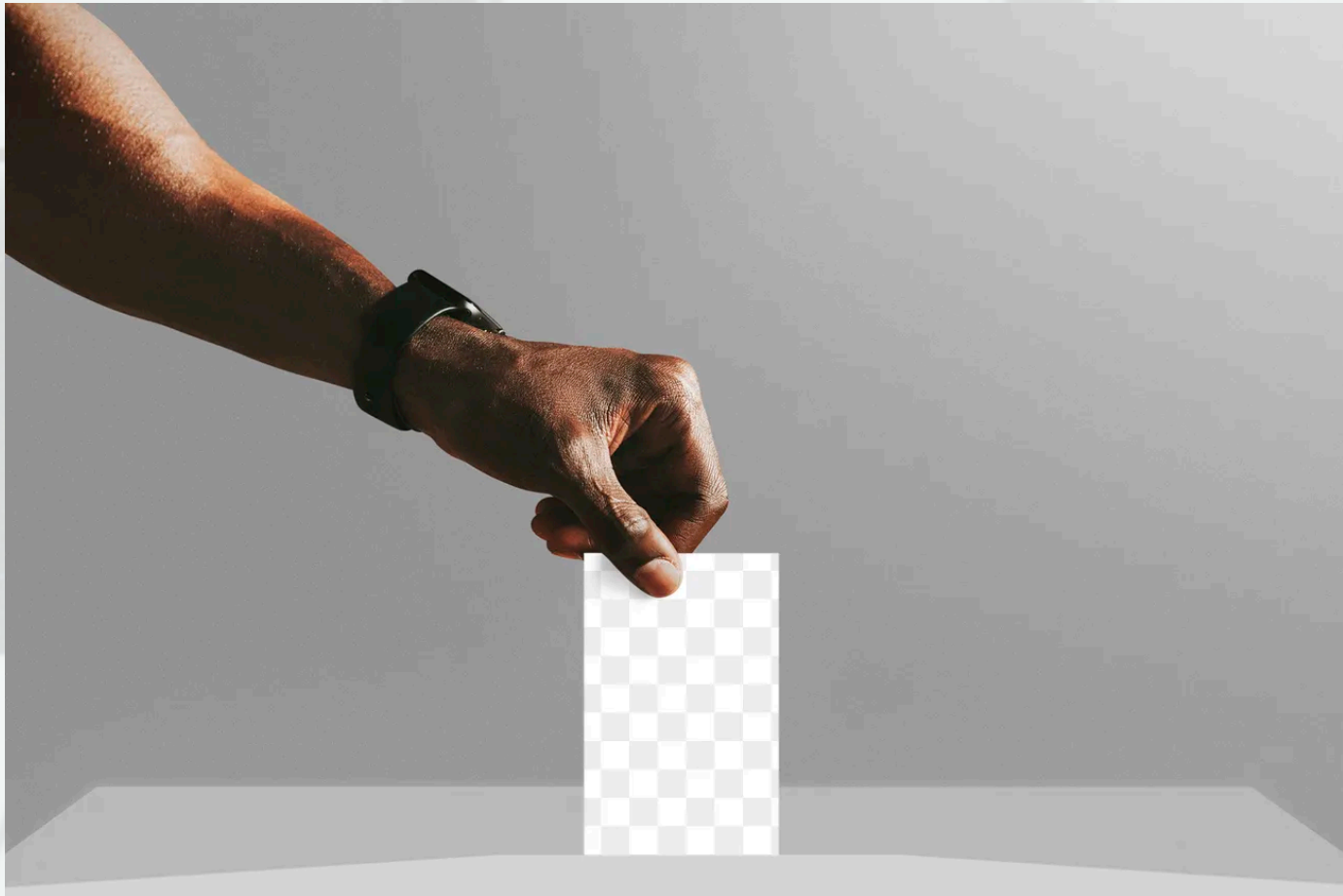
# Materials

<https://github.com/boyarsky/2022-kcdc-java-refactoring>

# Disclaimer

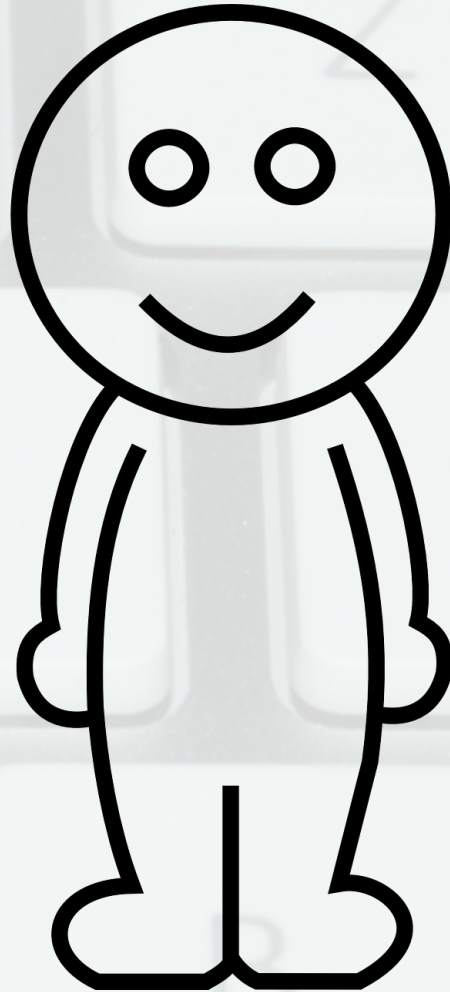
- A bit of the material is from my books.

# Version of Java?

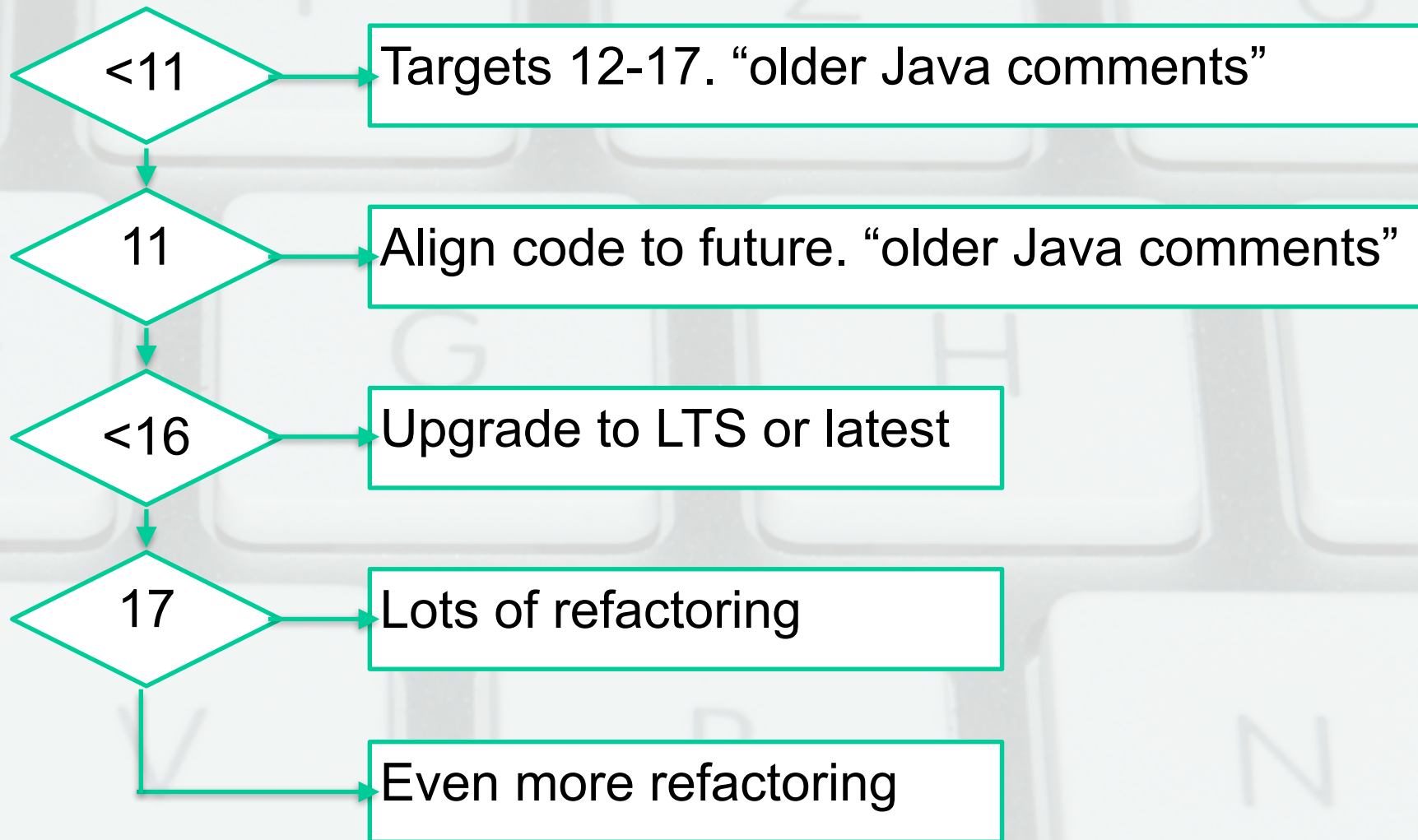




# Introductions



# Version of Java?





# Refactoring

- We are writing legacy code now!
- Refactor for future compatibility

# Agenda

Module	Topics
1	Text blocks and Strings
2	Records and Sealed classes
3	Instanceof and Switch Expressions
4	API changes

# Module Flow

- Review lab from previous module
- Lecture
- Review questions
- Hands on exercises
- 10 minute break

*This means if a colleague needs to call you, the last 15-20 minutes of each hour is best.*



# Required Software for Lab

## Option 1

- Java 17+ - <http://jdk.java.net/17/>
- IDE of your choice

## Option 2

<https://www.jdoodle.com/online-java-compiler/>

# Text blocks and Strings

# Example: REST API Params

```
public String getJson(String search) {  
    String json = "{" +  
        "    \"query\": \"%s\" +  
        "    \"start\": \"1\", +  
        "    \"end\": \"10\" +  
        "    }";  
    return String.format(json, search);  
}
```

This is hard to read



# Take Two

```
public String getJson(String search) {  
    Path path = Path.of(  
        "src/main/resources/query.json");  
    String json = null;  
    try {  
        json = Files.readString(path);  
    } catch (IOException e) {  
        throw new UncheckedIOException(e);  
    }  
    return String.format(json, search);  
}
```

Now the String is far away

# Text Block

15

```
public String getJson(String search) {  
    String json = ""  
        {  
            "query": "%s"  
            "start": "1"  
            "end": "10"  
        }"";  
    return String.format(json, search);  
}
```

Adds line breaks, but still works

# Text Block Syntax

15

```
String textBlock = """  
    kcdc,Kansas City,"session,workshop"  
    meetup,Various,lecture  
    """;
```

start block

incidental  
whitespace

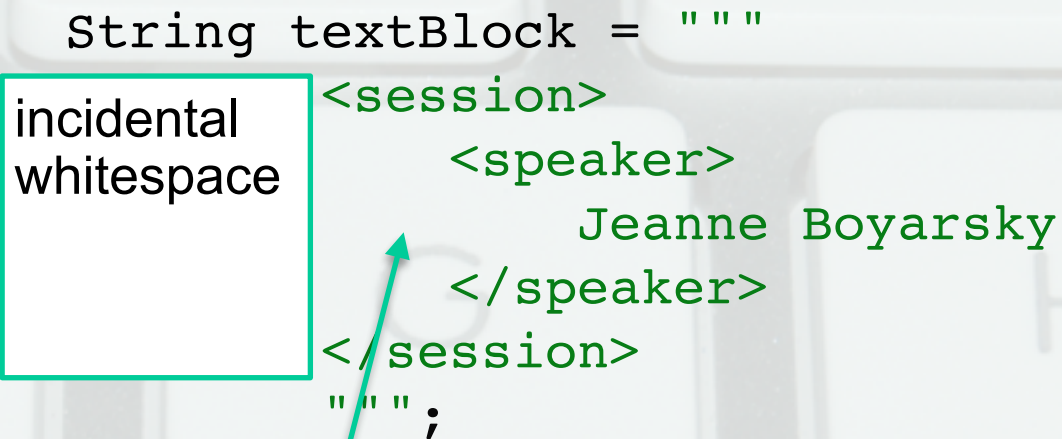
end block



# Essential Whitespace

15

```
String textBlock = ""  
    <session>  
        <speaker>  
            Jeanne Boyarsky  
        </speaker>  
    </session>  
    "";
```



The diagram illustrates the difference between incidental and essential whitespace in XML. A green box labeled "incidental whitespace" points to the spaces between the opening and closing tags of the <session> and <speaker> elements. A green arrow labeled "essential whitespace" points to the spaces between the opening and closing tags of the <session> element.

essential whitespace

# Ending lines

15

```
String textBlock = """
```

```
<session>
```

```
<speaker>
```

```
    Jeanne Boyarsky
```

```
</speaker>
```

```
<title>
```

```
    Becoming one of the first Java 17 \
```

```
    certified programmers \
```

```
    (and learning new features)
```

```
</title>
```

```
</session>
```

```
""";
```

new escape character  
keeps trailing whitespace

\s

tab

continue on next line  
without a line break

# New lines

15

```
String textBlock = ""
```

```
<session>\n
```

```
<speaker>
```

```
  Jeanne\nBoyarsky
```

```
</speaker>
```

```
</session>"";
```

Two new lines  
(explicit and implicit)

One new line (explicit)

no line break at end



# Escaping Three Quotes

15

```
String textBlock = """  
    better \"""  
    but can do \"\\\"\\\"  
    """;
```

# Indent

12

```
String option1 = "  a\n  b\n  c\n";  
String option2 = "a\nb\nc\n".indent(3);  
String option3 = """
```

```
  a  
  b  
  c  
  """.indent(3);
```

```
String option4 = """
```

```
  a  
  b  
  c  
  """;
```

Which do  
you like  
best?

Also normalizes (bye \r)

# Transform

12

```
String option1 = "chiefs".transform(  
    s -> s.toUpperCase());
```

```
String option2 = "  
chiefs  
".transform(s -> s.toUpperCase());
```

Which do  
you like  
best?



# Strip Indent

15

Method	From beginning	From end	Per line
<code>strip()</code>	Leading	Trailing	No
<code>stripIndent()</code>	Incidental	Incidental	Yes
<code>stripLeading()</code>	Leading	n/a	No
<code>stripTrailing()</code>	n/a	Trailing	No

# Opportunities

15

- Externalized data
- Expected values in JUnit
- Formats - CSV, GraphQL, SQL, Text, XML, etc
- Others?

# IDE Support



```
// TODO convert to text block when on Java 17
```

```
String json = "{" +
```

```
    "  \"query\": \"%s\" +
```

```
    "  \"st
```

```
    "  \"en
```

```
    \"}\";
```

```
return String.fo
```

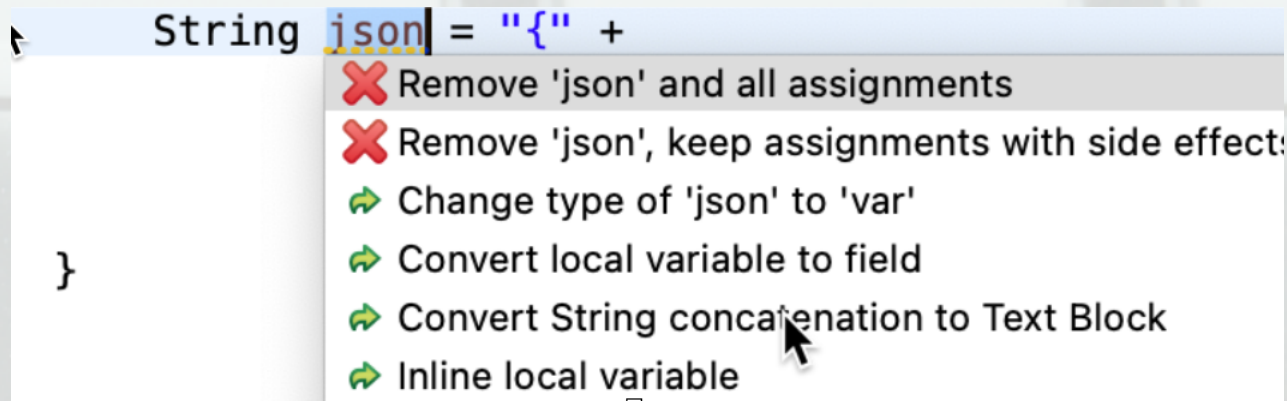
- ✎ Compute constant value of '"{" + " \"query\":'
- ✎ Copy string concatenation text to the clipboard
- ✎ Replace '+' with 'StringBuilder.append()'
- ✎ Replace '+' with 'formatted()'
- ✎ Replace with text block**
- ✎ Split into declaration and assignment

```
String json = """  
    {  "query": "%s"    "start": "1",    "end": "10"}""";
```

Literal refactoring - no \n



# IDE Support



```
String json = ""  
    {\br/>        "query": "%s"\br/>        "start": "1",\br/>        "end": "10"\br/>    }"";  
;
```

Preserve lines but still no \n

# On older Java?

```
public String getJson(String search) {  
    //TODO convert to text block when on Java 17  
    String json = "{" +  
        "    \"query\": \"%s\" +  
        "    \"start\": \"1\", +  
        "    \"end\": \"10\" +  
        "    }";  
    return String.format(json, search);  
}
```

Hard to read but positions for future

# Module 1 - Question 1

Which is true about this text block?

```
String sql = """
    select *
    from mytable
    where weather = 'snow';
    """;
```

- A. Has incidental whitespace
- B. Has essential whitespace
- C. Both A and B
- D. Does not compile



# Module 1 - Question 1

Which is true about this text block?

```
String sql = ""  
    select *  
    from mytable  
    where weather = 'snow';  
    "";
```

A

- A. Has incidental whitespace
- B. Has essential whitespace
- C. Both A and B
- D. Does not compile

# Module 1 - Question 2

Which is true about this text block?

```
String sql = """select *  
                from mytable \  
                where weather = 'snow';  
                """;
```

- A. Has incidental whitespace
- B. Has essential whitespace
- C. Both A and B
- D. Does not compile

## Module 1 - Question 2

Which is true about this text block?

```
String sql = ""select *  
    from mytable \  
    where weather = 'snow';  
"";
```

D

A. Has incidental whitespace

(no line break after opening quotes)

B. Has essential whitespace

C. Both A and B

D. Does not compile



# Module 1 - Question 3

Which is true about this text block?

```
String sql = """
    select *      \s
    from mytable  \s
    where weather = 'snow';
""";
```

- A. Has incidental whitespace
- B. Has essential whitespace
- C. Both A and B
- D. Does not compile

## Module 1 - Question 3

Which is true about this text block?

```
String sql = """
    select *
    from mytable
    where weather = 'snow';
    """;
```

**B**

- A. Has incidental whitespace
- B. Has essential whitespace
- C. Both A and B
- D. Does not compile

# Module 1 - Question 4

Which is true about this text block?

```
String sql = """select * from mytable;""";
```

- A. Has incidental whitespace
- B. Has essential whitespace
- C. Both A and B
- D. Does not compile



## Module 1 - Question 4

Which is true about this text block?

```
String sql = ""select * from mytable;"";
```

- A. Has incidental whitespace
- B. Has essential whitespace
- C. Both A and B
- D. Does not compile

**D**

(one line)

# Module 1 - Question 5

How many lines does this print out?

```
String sql = """  
    select * \n  
    from mytable;  
    """;  
System.out.print(sql);
```

- A. 2
- B. 3
- C. 4
- D. Does not compile

## Module 1 - Question 5

How many lines does this print out?

```
String sql = ""  
    select * \n  
    from mytable;  
    "";  
System.out.print(sql);
```

C

A. 2

B. 3

C. 4

D. Does not compile



# Module 1 - Question 6

How many lines does this print out?

```
String sql = """  
    select * \  
    from mytable;""".stripIndent();  
System.out.print(sql);
```

- A. 1
- B. 2
- C. 3
- D. Does not compile

## Module 1 - Question 6

How many lines does this print out?

```
String sql = ""  
    select * \  
    from mytable;"".stripIndent();  
System.out.print(sql);
```

A

A. 1

B. 2

C. 3

D. Does not compile

# Module 1 - Question 7

How many lines does this print out?

```
String sql = """
    select *      \s
    from mytable;
".stripIndent();
System.out.print(sql);
```

- A. 1
- B. 2
- C. 3
- D. Does not compile



# Module 1 - Question 7

How many lines does this print out?

```
String sql = ""  
    select *    \s  
    from mytable;  
".stripIndent();  
System.out.print(sql);
```

**D**

A. 1

(only one closing quote, not three)

B. 2

C. 3

D. Does not compile

# Module 1 - Question 8

How many whitespace characters are removed by strip()?

```
String sql = """
    select "name"
    from mytable;
    """;
```

- A. 1
- B. 2
- C. 3
- D. Does not compile

## Module 1 - Question 8

How many whitespace characters are removed by strip()?

```
String sql = ""  
    select "name"  
    from mytable;  
    "";
```

C

- A. 1 (two leading whitespace + new line at end)
- B. 2
- C. 3
- D. Does not compile



# Module 1 - Question 9

How many whitespace characters are removed by `stripIndent()`?

```
String sql = """
    select "name"
    from mytable;
    """;
```

- A. 0
- B. 1
- C. 2
- D. 3

# Module 1 - Question 9

How many whitespace characters are removed by `stripIndent()`?

```
String sql = ""  
    select "name"  
    from mytable;  
    "";
```

A

(strip indent only gets rid of incidental whitespace, which text block already takes care of)

A. 0

B. 1

C. 2

D. 3

# Module 1 - Question 10

How many escapes can be removed without changing the behavior?

```
String sql = """
    select \"name\"
    from mytable \
    where value = '\"\"'
    """;
```

- A. 2
- B. 3
- C. 4
- D. Does not compile



# Module 1 - Question 10

How many escapes can be removed without changing the behavior?

```
String sql = """
    select \"name\"
    from mytable
    where value = \"\"';
```

**B**

(two around “name” and the second on value)

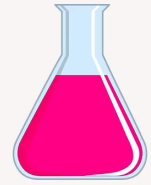
A. 2

B. 3

C. 4

D. Does not compile

# Lab & Break



<https://github.com/boyarsky/2022-kcdc-java-refactoring>



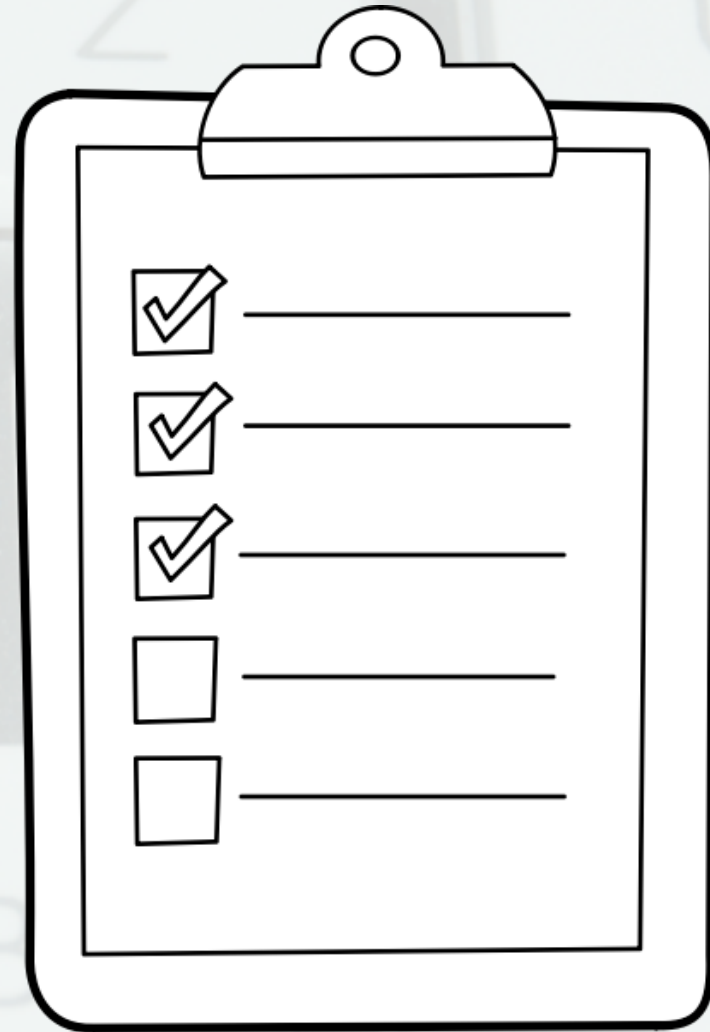
Note: If using JDoodle, comment out code that reads from file

# InstanceOf and Switch Expressions



# Lab Review

- Any questions?



# Casting

```
if (num instanceof Integer) {  
    Integer numAsInt = (Integer) num;  
    System.out.println(numAsInt);  
}  
if (num instanceof Double) {  
    Double numAsDouble = (Double) num;  
    System.out.println(numAsDouble.intValue());  
}
```

# Casting

16

```
if (num instanceof Integer numAsInt) {  
    System.out.println(numAsInt);  
}  
if (num instanceof Double numAsDouble) {  
    System.out.println(numAsDouble.intValue());  
}
```

Pattern  
variable





# Flow Scope

16

```
if (num instanceof Double d1
    && d1.intValue() % 2 == 0) {
    System.out.println(d1.intValue());
}
```

Compiles

```
if (num instanceof Double d2
    || d2.intValue() % 2 == 0) {
    System.out.println(d2.intValue());
}
```

Does not  
compile  
because  
d2 might  
not be  
double

# Does this compile?

16

```
if (num instanceof Double n)  
    System.out.println(n.intValue());
```

```
if (num instanceof Integer n)  
    System.out.println(n);
```

Yes. Only in scope for if statement

# Does this compile?

16

```
if (num instanceof Double n)  
    System.out.println(n.intValue());  
  
System.out.println(n.intValue());
```

No. If statement is over



# Does this compile?

16

```
if (!(num instanceof Double n)) {  
    return;  
}  
System.out.println(n.intValue());
```

Yes. Returns early so rest is like an else

# Does this compile?

16

```
if (!(num instanceof Double n)) {  
    return;  
}  
System.out.println(n.intValue());
```

```
if (num instanceof Double n)  
    System.out.println(n.intValue());
```

No. n is still in scope

# Originally

```
public String getLocation(String store) {  
    String result = "";  
    switch (store) {  
        case "Hallmark":  
            result = "KC";  
            break;  
        case "Crayola":  
            result = "PA";  
            break;  
        default:  
            result = "anywhere";  
    }  
    return result;  
}
```

You remembered the breaks, right?



# Switch Expressions

14

```
public String getLocation(String store) {  
    return switch (store) {  
        case "Hallmark" -> "KC";  
        case "Crayola" -> "PA";  
        default -> "anywhere";  
    };  
}
```

Arrow labels

No break keyword

# More features

14

```
String output = switch(store) {  
    case "Hallmark" -> "KC";  
    case "Legoland" -> {  
        int random = new Random().nextInt();  
        String city = random % 2 == 0  
            ? "KC" : "Carlsbad";  
        yield city;  
    }  
    default -> throw new  
        IllegalArgumentException("Unknown");  
};  
System.out.println(output);
```

Block

yield

throws exception  
so no return value  
needed

# Is this legal?

14

```
private void confusing() {  
    this.yield();  
}
```

```
private void yield() {  
    String store = "Legoland";  
  
    String output = switch(store) {  
        case "Legoland" -> {  
            yield "Carlsbad";  
        }  
        default -> "other";  
    };  
    System.out.println(output);  
}
```

Yes. yield  
is like var



# How about now?

14

```
private void confusing() {  
    yield();  
}  
  
private void yield() {  
    String store = "Legoland";  
  
    String output = switch(store) {  
        case "Legoland" -> {  
            yield "Carlsbad";  
        }  
        default -> "other";  
    };  
    System.out.println(output);  
}
```

No

to invoke a method called yield, qualify the yield with a receiver or type name

# Yield with Switch Stmt

14

```
Position pos = Position.TOP;
```

```
int stmt = switch(pos) {  
    case TOP: yield 1;  
    case BOTTOM: yield 0;  
};
```

Same!

```
int expr = switch(pos) {  
    case TOP -> 1;  
    case BOTTOM -> 0;  
};
```

# Missing value

14

```
enum Position { TOP, BOTTOM };
```

```
Position pos = Position.TOP;
```

```
int stmt = switch(pos) {  
  case TOP: yield 1;  
};
```

```
int expr = switch(pos) {  
  case BOTTOM -> 0;  
};
```

Does not compile  
because assigning  
value

(poly expression)



# Pattern matching for switch

19  
preview

```
public int toInt(Object obj) {  
    return switch (obj) {  
        case Integer i -> i;  
        case Double d -> d.intValue();  
        case String s -> Integer.parseInt(s);  
  
        default -> throw new  
            IllegalArgumentException("unknown type");  
    };  
}
```

Reminder: Syntax can change

# But wait, there's more

19  
preview

```
static void printOddOrEven(Object obj) {  
    switch (obj) {  
  
        case Integer i when i % 2 == 1 ->  
            System.out.println("odd");  
  
        case Integer i when i % 2 == 0 ->  
            System.out.println("even");  
  
        default -> System.out.println("not an int");  
    };  
}
```

Reminder: Feature can still change

# Opportunities

16

- Library code
- Equals methods

```
public boolean equals(Object anObject) {  
    if (this == anObject) {  
        return true;  
    }  
    return (anObject instanceof String aString)  
        && (!COMPACT_STRINGS || this.coder == aString.coder)  
        && StringLatin1.equals(value, aString.value);  
}
```

- Many if/else chains!
- Switch statements with many breaks
- Sets the stage for advanced matching
- Others?



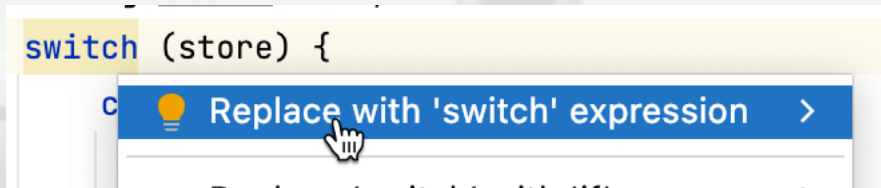
# IDE Support



```
if (num instanceof Integer) {  
    Integer numAsInt = (Integer) num;  
    System.out.print(  
        "Replace 'numAsInt' with pattern variable  >
```

```
if (num instanceof Integer numAsInt) {  
    System.out.println(numAsInt);  
}
```

# IDE Support



```
String result = switch (store) {  
    case "Hallmark" -> "KC";  
    case "Crayola" -> "PA";  
    default -> "anywhere";  
};  
return result;
```

# On older Java?

```
public String getLocation(String store) {  
    //TODO convert to switch expression on Java 17  
    String result = "";  
    switch (store) {  
        case "Hallmark":  
            result = "KC";  
            break;  
        case "Crayola":  
            result = "PA";  
            break;  
        default:  
            result = "anywhere";  
    }  
    return result;  
}
```



# On older Java?

*//TODO convert to pattern var when on Java 17*

```
if (num instanceof Double) {  
    Double numAsDouble = (Double) num;  
    System.out.println(numAsDouble.intValue());  
}
```

Positions for future

# Module 2 - Question 1

What is output?

```
char ch = 'b';  
  
int count = 0;  
switch (ch) {  
    case 'a' -> count++;  
    case 'b' -> count+=2;  
    case 'c' -> count+=3;  
}  
  
System.out.println(count);
```

A. 1  
B. 2

C. 5  
D. Does not compile

## Module 2 - Question 1

What is output?

```
char ch = 'b';  
  
int count = 0;  
switch (ch) {  
    case 'a' -> count++;  
    case 'b' -> count+=2;  
    case 'c' -> count+=3;  
}  
  
System.out.println(count);
```

**B**

A. 1

B. 2

C. 5

D. Does not compile



## Module 2 - Question 2

How many changes are needed to have this code print 2?

```
char ch = 'b';

int value = switch (ch) {
    case 'a' -> 1;
    case 'b' -> yield 2;
    case 'c' -> 3;
}

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

- |      |      |
|------|------|
| A. 1 | C. 3 |
| B. 2 | D. 4 |

## Module 2 - Question 2

How many changes are needed to have this code print 2?

```
char ch = 'b';
```

```
int value = switch (ch) {  
    case 'a' -> 1;  
    case 'b' -> yield 2;  
}
```

(remove yield, add default and add semicolon at end of switch)

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

A. 1

B. 2

C. 3

D. 4

## Module 2 - Question 3

How many lines need changing to make this code compile?

```
char ch = 'b';

int value = switch (ch) {
    case 'a' : yield 1;
    case 'b' : { yield 2; }
    case 'c' -> yield 3;
    default -> 4;
};
System.out.println(value);
```

A. 1

B. 2

C. 3

D. 4



## Module 2 - Question 3

How many lines need changing to make this code compile?

```
char ch = 'b';
```

```
int value = switch (ch)
```

```
case 'a' : yield 1;
```

```
case 'b' : { yield 2; }
```

```
case 'c' : -> yield 3;
```

```
default -> 4;
```

```
};
```

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

**B**

(case c is invalid because can't use yield there. Also case c/default invalid because can't mix and match : and ->

A. 1

B. 2

C. 3

D. 4

# Module 2 - Question 4

What can fill in the blank to have the code print 2?

```
char ch = 'b';  
  
_____ value = switch (ch) {  
    case 'a' -> 1;  
    case 'b' -> 2;  
    case 'c' -> 3.0;  
    default -> 4;  
};  
System.out.println(value);
```

- A. int
- B. Object

- C. Either A or B
- D. None of the above

## Module 2 - Question 4

What can fill in the blank to have the code print 2?

```
char ch = 'b';  
  
_____ value = switch (ch) {  
    case 'a' -> 1;  
    case 'b' -> 2;  
    case 'c' -> 3.0;  
    default -> 4;  
};  
System.out.println(value);
```

**B**

A. int

B. Object

C. Either A or B

D. None of the above



# Module 2 - Question 5

What does the following output?

```
char ch = 'b';  
  
switch (ch) {  
    case 'a' -> System.out.println(1);  
    case 'b' -> System.out.println(2);  
    case 'c' -> { System.out.println(3); }  
};
```

A. 1

B. 2

C. 3

D. Does not compile

## Module 2 - Question 5

What does the following output?

```
char ch = 'b';

switch (ch) {
    case 'a' -> System.out.println(1);
    case 'b' -> System.out.println(2);
    case 'c' -> { System.out.println(3); }
};
```

A. 1

B. 2

C. 3

D. Does not compile

## Module 2 - Question 6

What does the following print?

```
Object robot = "694";
```

```
if (robot instanceof String s) {  
    System.out.print("x");  
}  
if (robot instanceof Integer s) {  
    System.out.print("y");  
}  
System.out.println(robot);
```

A. x694  
B. xy694

C. y694  
D. Does not compile



## Module 2 - Question 6

What does the following print?

```
Object robot = "694";

if (robot instanceof String s) {
    System.out.print("x")
}
if (robot instanceof Integer s) {
    System.out.print("y");
}
System.out.println(robot);
```

A

A. x694

B. xy694

C. y694

D. Does not compile

# Module 2 - Question 7

Which lines have s in scope?

```
Object robot = "694";
```

```
if (robot instanceof String s) {  
    // line 1
```

```
}
```

```
if (robot instanceof int i) {  
    // line 2
```

```
}
```

```
// line 3
```

A. 1

B. 1 and 3

C. 1, 2 and 3

D. Does not compile

## Module 2 - Question 7

Which lines have s in scope?

```
Object robot = "694";
```

```
if (robot instanceof String s) {  
    // line 1  
}
```

```
if (robot instanceof int i) {  
    // line 2  
}
```

```
// line 3
```

**D**

(int needs to be Integer)

A. 1

B. 1 and 3

C. 1, 2 and 3

D. Does not compile



## Module 2 - Question 8

What is true about this class?

```
class Sword {  
    int length;  
  
    public boolean equals(Object o) {  
        if (o instanceof Sword sword) {  
            return length == sword.length;  
        }  
        return false;  
    }  
    // assume hashCode properly implemented  
}
```

A. equals() is correct

B. equals() is incorrect

C. equals() does not compile

## Module 2 - Question 8

What is true about this class?

```
class Sword {  
    int length;  
  
    public boolean equals(Object o) {  
        if (o instanceof Sword sword) {  
            return length == sword.length;  
        }  
        return false;  
    }  
    // assume hashCode properly implemented  
}
```

A. equals() is correct

B. equals() is incorrect

C. equals() does not compile

# Module 2 - Question 9

How many if statements fail to compile?

Number n = 4;

```
if (n instanceof Integer x) {}  
if (n instanceof Integer x  
    && x.intValue() > 1) {}  
if (n instanceof Integer x  
    || x.intValue() > 1) {}  
if (n instanceof Integer x  
    || x.toString().isEmpty()) {}
```

A. 0

B. 1

C. 2

D. 3



## Module 2 - Question 9

How many if statements fail to compile?

```
Number n = 4;
```

```
if (n instanceof Integer x) {}
```

```
if (n instanceof Integer x
```

```
&& x.intValue() > 1) {}
```

```
if (n instanceof Integer x
```

```
|| x.intValue() > 1) {}
```

```
if (n instanceof Integer x
```

```
|| x.toString().isEmpty()) {}
```

(the last two because X could be null)

A. 0

B. 1

C. 2

D. 3

## Module 2 - Question 10

What does `printLength(3)` print?

```
class Sword {  
    int length = 8;  
  
    public void printLength(Object x) {  
        if (x instanceof Integer length) {  
            length = 2;  
        }  
        System.out.println(length);  
    }  
}
```

A. 2  
B. 3

C. 8  
D. Does not compile

## Module 2 - Question 10

What does `printLength(3)` print?

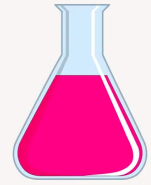
```
class Sword {  
    int length = 8;  
  
    public void printLength(Object x) {  
        if (x instanceof Integer length) {  
            length = 2;  
        }  
        System.out.println(length);  
    }  
}
```

A. 2  
B. 3

C. 8  
D. Does not compile



# Lab & Break



<https://github.com/boyarsky/2022-kcdc-java-refactoring>



# Records and Sealed Classes

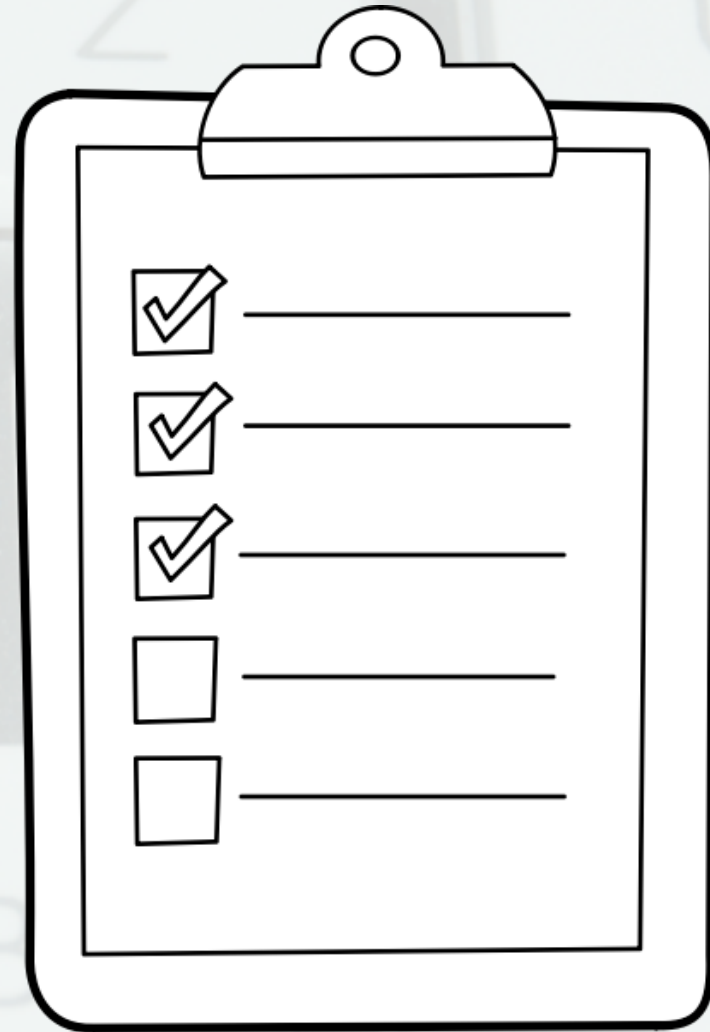
# Book Giveaway





# Lab Review

- Any questions?



# Originally

```
public class Book {  
  
    2 usages  
    private String title;  
  
    2 usages  
    private int numPages;  
  
    public Book(String title, int numPages) {  
        this.title = title;  
        this.numPages = numPages;  
    }  
  
    public String getTitle() {  
        return title;  
    }  
  
    public int getNumPages() {  
        return numPages;  
    }  
}
```

Ran out of room  
on screen!

# Record

16

```
public record Book (String title, int numPages) {  
}
```

New type

Automatically get

- \* final record
- \* private final instance variables
- \* public accessors
- \* constructor taking both fields
- \* equals
- \* hashCode



# Using the Record

16

```
Book book = new Book("Breaking and entering", 289);
```

```
System.out.println(book.title());  
System.out.println(book.toString());
```

← No "get"

Outputs:

Breaking and entering

Book[title=Breaking and entering, numPages=289]

# Add/change methods

16

```
public record Book (String title, int numPages) {  
  
    @Override  
    public String title() {  
        return "'" + title + "'";  
    }  
  
    public boolean isLong() {  
        return numPages > 300;  
    }  
  
}
```

← Change  
behavior

← Custom  
method

# Not really immutable

16

```
public record Book (String title, int numPages,  
    List<String> chapters) {  
}
```

```
Book book = new Book("Breaking and entering", 289,  
    chapters);
```

```
chapters.add("2");  
book.chapters().add("3");  
System.out.println(book.chapters());
```

Prints [1,2,3] because shallow immutability



# Now immutable

16

```
public record Book (String title, int numPages,  
    List<String> chapters) {
```

Compact constructor

```
    public Book {  
        chapters = List.copyOf(chapters);  
    }  
}
```

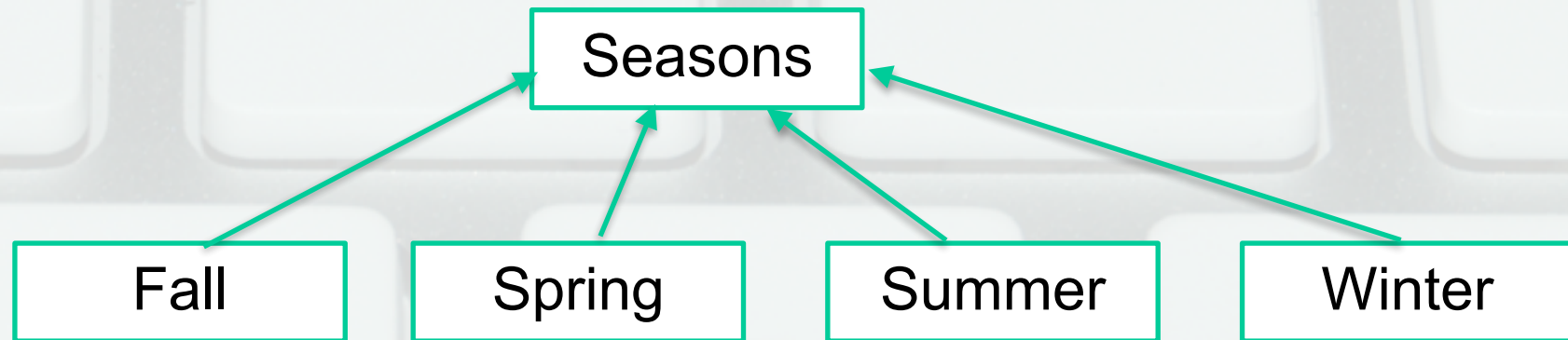
Must match record  
access modifier

# Sealed classes

17

```
public abstract sealed class Seasons  
    permits Fall, Spring, Summer, Winter { }
```

```
final class Fall extends Seasons {}  
final class Spring extends Seasons {}  
final class Summer extends Seasons {}  
final class Winter extends Seasons {}
```



# Subclass modifiers

17

Modifier	Meaning
final	Hierarchy ends here
non-sealed	Others can subclass
sealed	Another layer



# Sealed interface

17

```
public sealed interface TimeOfDay
    permits Morning, Afternoon, Evening {
        boolean early();
    }
public non-sealed class Morning implements TimeOfDay {
    public boolean early() { return true; }
}
public non-sealed class Afternoon implements TimeOfDay {
    public boolean early() { return false; }
}
public record Evening(int hour) implements TimeOfDay {
    public boolean early() { return false; }
}
```

Records are implicitly final

# instanceof

17

```
public class InstanceOf {  
    static sealed class BoolWrapper  
        permits TrueWrapper, FalseWrapper { }  
    static final class TrueWrapper extends BoolWrapper {}  
    static final class FalseWrapper extends BoolWrapper {}  
    public static void main(String[] args) {  
        Map<?, ?> map = new HashMap<>();  
        String string = "";  
        BoolWrapper boolWrapper = new TrueWrapper();  
  
        System.out.println(map instanceof List);           // false  
        System.out.println("" instanceof List);           // error  
        System.out.println(boolWrapper instanceof List);  // error  
    }  
}
```

# instanceof

19  
preview

```
sealed interface Season permits Winter,  
    Spring, Summer, Fall{}
```

```
final class Winter implements Season {}  
final class Spring implements Season {}  
final class Summer implements Season {}  
final class Fall implements Season {}
```

```
public String sealed(Season season) {  
    return switch (season) {  
        case null -> "null";  
        case Winter w -> "Winter";  
        case Spring s -> "Spring";  
        case Summer s -> "Summer";  
        case Fall f -> "Fall";  
    }; };
```

Also “when”



# Module 3 - Question 1

How many lines need to be removed for this code to compile?

```
public record BBQ(String type) {}

public static void main(String[] args) {
    BBQ bbq = new BBQ("chicken");
    System.out.println(bbq.setType("pork"));
    System.out.println(bbq.getType());
    System.out.println(bbq.equals(bbq));
}
```

A. 0

B. 1

C. 2

D. None of the above

## Module 3 - Question 1

How many lines need to be removed for this code to compile?

```
public record BBQ(String type) {}

public static void main(String[] args) {
    BBQ bbq = new BBQ("chicken");
    System.out.println(bbq.setType("pork"));
    System.out.println(bbq.getType());
    System.out.println(bbq.equals(bbq));
}
```

C

(no setters and getter should be type())

A. 0

B. 1

C. 2

D. None of the above

## Module 3 - Question 2

What does this output?

```
public record BBQ(String type) {  
    BBQ {  
        type = type.toUpperCase();  
    }  
}  
  
public static void main(String[] args) {  
    BBQ bbq = new BBQ("chicken");  
    System.out.println(bbq.type());  
}
```

A. chicken  
B. CHICKEN

C. Does not compile  
D. None of the above



## Module 3 - Question 2

What does this output?

```
public record BBQ(String type) {  
    BBQ {  
        type = type.toUpperCase();  
    }  
}
```

C

```
public static void main(String[] args) {  
    BBQ bbq = new BBQ("chicken");  
    System.out.println(bbq.type());  
}
```

(compact constructor must be public because record is)

A. chicken

B. CHICKEN

C. Does not compile

D. None of the above

# Module 3 - Question 3

What does this output?

```
record BBQ(String type) {  
    BBQ {  
        type = type.toUpperCase();  
    }  
}  
  
public static void main(String[] args) {  
    BBQ bbq = new BBQ("chicken");  
    System.out.println(bbq.type());  
}
```

A. chicken  
B. CHICKEN

C. Does not compile  
D. None of the above

## Module 3 - Question 3

What does this output?

```
record BBQ(String type) {  
    BBQ {  
        type = type.toUpperCase();  
    }  
}  
  
public static void main(String[] args) {  
    BBQ bbq = new BBQ("chicken");  
    System.out.println(bbq.type());  
}
```

**B**

A. chicken

B. CHICKEN

C. Does not compile

D. None of the above



# Module 3 - Question 4

How many compiler errors are in the following code?

```
public final record BBQ(String type) {  
    { type = ""; }  
    public BBQ(String type) {  
        type = type.toUpperCase();  
    }  
    public void type() { return ""; }  
    public String toString() { return ""; }  
}
```

- A. 1
- B. 2

- C. 3
- D. 4

## Module 3 - Question 4

How many compiler errors are in the following code?

```
public final record BBQ(String type) {  
    { type = ""; }  
    public BBQ(String type) {  
        type = type.toUpperCase();  
    }  
    public void type() { return ""; }  
    public String toString() { return ""; }  
}
```

(no instance initializer allowed, compact constructor syntax wrong, type() must return String)

A. 1  
B. 2

C. 3  
D. 4

# Module 3 - Question 5

What does this output?

```
public record BBQ(String type)
    implements Comparable<BBQ> {

    public int compareTo(BBQ bbq) {
        return type.compareTo(bbq.type);
    } }

public static void main(String[] args) {
    BBQ beef = new BBQ("beef");
    BBQ pork = new BBQ("pork");
    System.out.println(pork.compareTo(beef));
}
```

A. Negative #  
B. Positive #

C. 0  
D. Does not compile



## Module 3 - Question 5

What does this output?

```
public record BBQ(String type)
    implements Comparable<BBQ> {

    public int compareTo(BBQ bbq) {
        return type.compareTo(bbq.type);
    } }

public static void main(String[] args) {
    BBQ beef = new BBQ("beef");
    BBQ pork = new BBQ("pork");
    System.out.println(pork.compareTo(beef));
}
```

A. Negative #

B. Positive #

C. 0

D. Does not compile

# Module 3 - Question 6

What does this output?

```
static sealed interface Weather permits Wet, Dry{
    boolean needUmbrella(); }
static non-sealed class Wet implements Weather {
    public boolean needUmbrella() { return true; } }
static record Dry(boolean needUmbrella)
    implements Weather {}

public static void main(String[] args) {
    Weather weather = new Dry(false);
    System.out.println(weather.needUmbrella());
}
```

A. true  
B. false

C. Does not compile  
D. None of the above

## Module 3 - Question 6

What does this output?

```
static sealed interface Weather permits Wet, Dry{
    boolean needUmbrella(); }
static non-sealed class Wet implements Weather {
    public boolean needUmbrella() { return true; } }
static record Dry(boolean needUmbrella)
    implements Weather {}

public static void main(String[] args) {
    Weather weather = new Dry(false);
    System.out.println(weather.needUmbrella());
}
```

A. true

B. false

C. Does not compile

D. None of the above



## Module 3 - Question 7

Which of the following are true? (Choose all that apply)

- A. There is only one hyphenated modifier in Java (non-sealed)
- B. A sealed interface may permit another interface.
- C. Sealed records are allowed
- D. Sealed enums are allowed

## Module 3 - Question 7

Which of the following are true? (Choose all that apply)

- A. There is only one hypererated modifier in Java (non-sealed)
- B. A sealed interface may permit another interface.
- C. Sealed records are allowed
- D. Sealed enums are allowed

## Module 3 - Question 8

Given the following, where could Android be?  
(Choose all that apply)

```
package general;  
static sealed class Phone  
    permits iPhone, Android { }
```

- A. In the same file as Phone
- B. In the same package as Phone within a module
- C. In a different package from Phone, but in the same module
- D. In a different module



## Module 3 - Question 8

Given the following, where could Android be?  
(Choose all that apply)

```
package general;  
static sealed class Phone  
    permits iPhone, Android { }
```

A, B

- A. In the same file as Phone
- B. In the same package as Phone within a module
- C. In a different package from Phone, but in the same module
- D. In a different module

## Module 3 - Question 9

Given the following, where could Android be?  
(Choose all that apply)

```
package general;  
static sealed class Phone  
    permits mac.IPhone, google.Android { }
```

- A. In the same file as Phone
- B. In the same package as Phone within a module
- C. In a different package from Phone, but in the same module
- D. In a different module

## Module 3 - Question 9

Given the following, where could Android be?  
(Choose all that apply)

```
package general;  
static sealed class Phone  
    permits mac.IPhone, google.Android { }
```

- A. In the same file as Phone
- B. In the same package as Phone within a module
- C. In a different package from Phone, but in the same module
- D. In a different module



# Module 3 - Question 10

How many compiler errors are in this code?

```
public sealed class Phone {  
    class iPhone extends Phone {  
    }  
    class Android extends Phone {  
    }  
}
```

A. 0

B. 1

C. 2

D. 3

## Module 3 - Question 10

How many compiler errors are in this code?

```
public sealed class Phone {  
  
    class iPhone extends Phone {  
    }  
  
    class iPod extends Phone {  
    }  
}
```

(extending classes need to be sealed/non-sealed final)

A. 0

B. 1

C. 2

D. 3

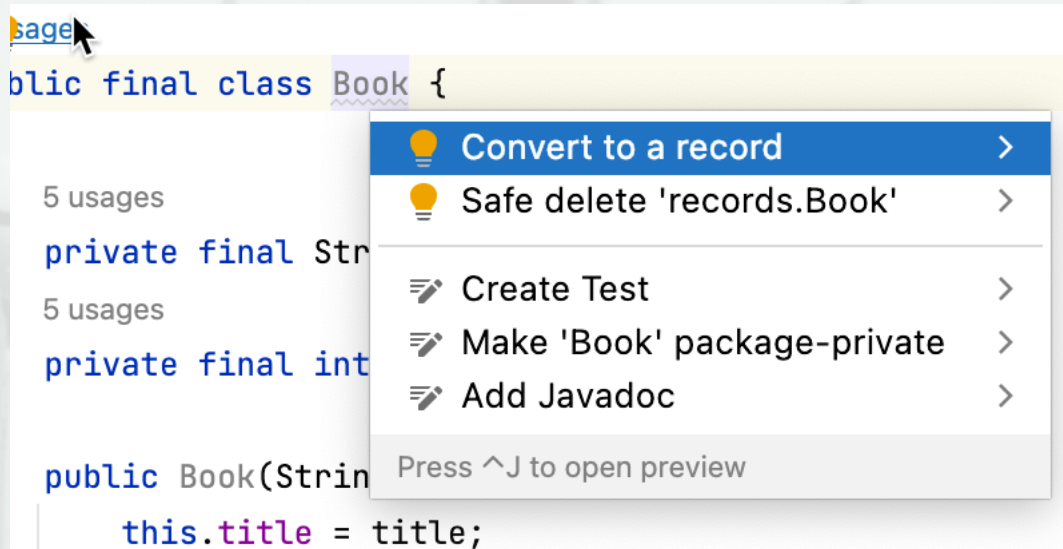
# Opportunities

17

- Immutable POJOs
- Don't have to write equals/hashCode
- Make code coverage tool happy
- Library design
- Others?



# IDE Support



```
public record Book(String title, int numPages) {  
}
```

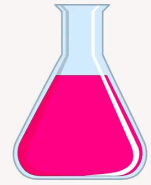
Had to make instance variables final. Also didn't remove my equals() even though generated by IntelliJ

# On older Java?

```
//TODO convert to record when on Java 17
public final class Book {
    private String title;
    private int numPages;

    public Book(String title, int numPages) {
        this.title = title;
        this.numPages = numPages;
    }
    public String title() {
        return title;
    }
    public int numPages() {
        return numPages;
    }
    // hash code, equals
```

# Lab & Break



<https://github.com/boyarsky/2022-kcdc-java-refactoring>



Note: If using JDoodle, combine into one file



# APIs

# toList()

16

```
public List<String> listLonger(  
    Stream<String> stream) {  
  
    return stream.collect(Collectors.toList());  
}  
  
public List<String> listShorter(  
    Stream<String> stream) {  
  
    return stream.toList();  
}
```

# Teeing Collector

12

```
record Separations(String spaceSeparated,  
    String commaSeparated) {}
```

```
var list = List.of("x", "y", "z");  
Separations result = list.stream()  
    .collect(Collectors.teeing(  
        Collectors.joining(" "),  
        Collectors.joining(","),  
        (s, c) -> new Separations(s, c)));
```

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



# Formatting a String

12

```
String firstName = "Jeanne";  
String lastName = "Boyarsky";  
String str = String.format(  
    "Hi %s %s!", firstName, lastName);  
System.out.println(str);  
  
System.out.println("Hi %s %s!".formatted(  
    firstName, lastName));
```

Outputs:  
Hi Jeanne Boyarsky!  
Hi Jeanne Boyarsky!

# Common Conversions

Conversion	What it does
%s	Formattable as String
%d	Decimal integer (no dot)
%c	Char
%f	Float (decimal)
%n	New line

Many more out of scope. Examples:

- %e - scientific notation
- %t - time
- %S - converts to all uppercase

# Conversion Examples

12

Code	Output
<code>"%d%%".formatted(1.2)</code>	exception
<code>"%d%%".formatted(1)</code>	1%
<code>"%s%%".formatted(1)</code>	1%
<code>"%s%%".formatted(1.2)</code>	1.2%
<code>"%f%%".formatted(1.2)</code>	1.200000f



# Formatting a Number

Char	What it does
-	Left justified
+	Always include +/-
space	Leading space if positive

Char	What it does
0	Zero padded
,	Group numbers
(	Negative # in parens

# Flag Examples

12

Code	Output
<code>"%,d".formatted(1234)</code>	1,234
<code>"%+d".formatted(1234)</code>	1234
<code>"% d".formatted(1234)</code>	1234
<code>"%, (d".formatted(-1234)</code>	(1,234)
<code>"%,f".formatted(1.23456789)</code>	1.234568

# Compact Number

12

```
NumberFormat defaultFormat =  
NumberFormat.getCompactNumberInstance();  
NumberFormat shortFormat = NumberFormat  
    .getCompactNumberInstance(  
        Locale.US, NumberFormat.Style.SHORT);  
NumberFormat longFormat = NumberFormat  
    .getCompactNumberInstance(  
        Locale.US, NumberFormat.Style.LONG);  
  
System.out.println(defaultFormat.format(1_000_000));  
System.out.println(shortFormat.format(1_000_000));  
System.out.println(longFormat.format(1_000_000));
```

1M  
1M  
1 million



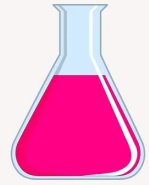
# New Files.mismatch()

12

```
Path kcdc = Path.of("files/kcdc.txt");  
Path kc = Path.of("files/kc.txt");  
  
System.out.println(Files.mismatch(kcdc, kc));  
System.out.println(Files.mismatch(kcdc, kcdc));
```

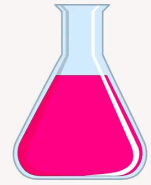
11 (index of first character different)  
-1 (same file contents regardless of whether exists)

# Wed Lab Version



- More APIs
- Hands on practice

# Lab



<https://github.com/boyarsky/2022-kcdc-java-refactoring>

