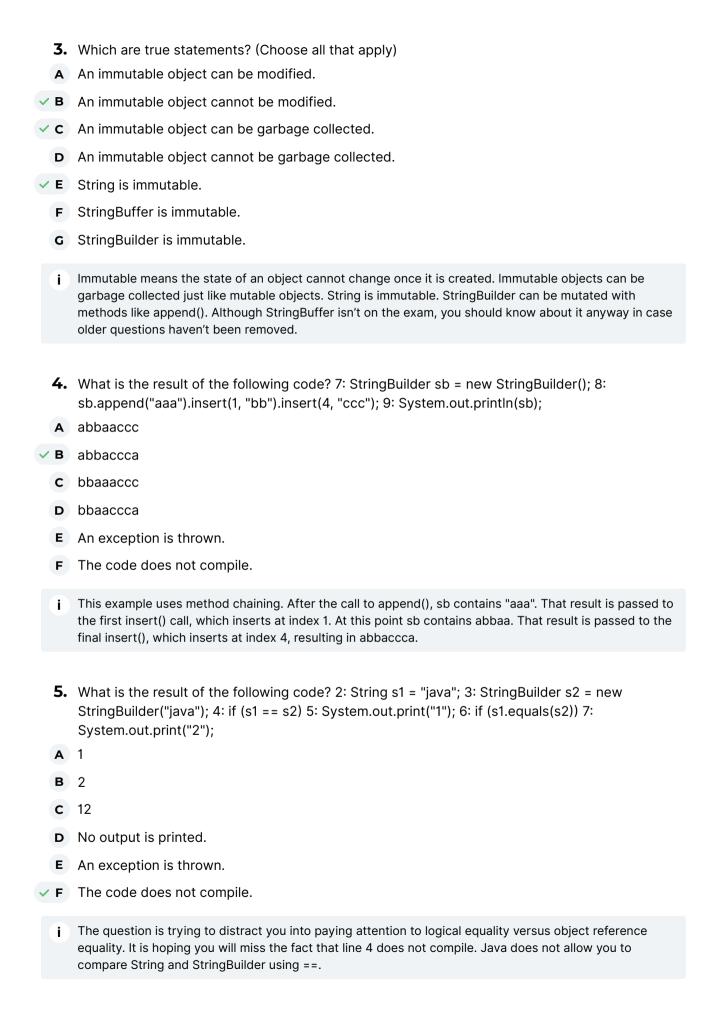
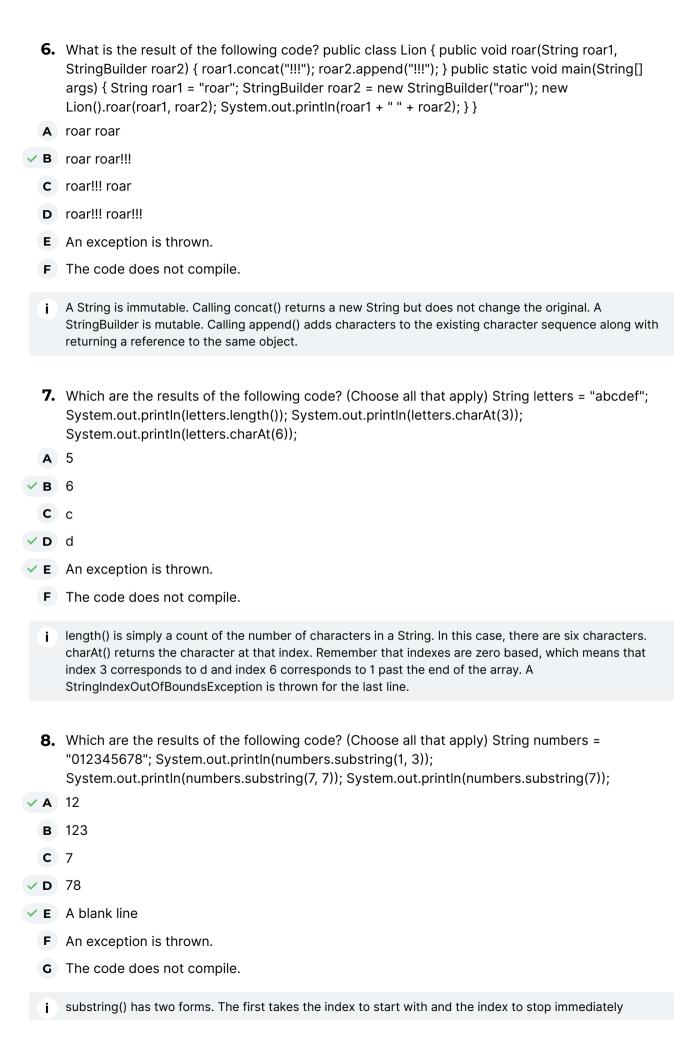


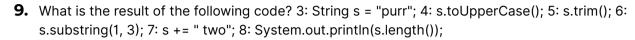
## Java OCA Chapter 3 Review ('Copy')

| 1.         | What is output by the following code? 1: public class Fish { 2: public static void main(String[] args) { 3: int numFish = 4; 4: String fishType = "tuna"; 5: String anotherFish = numFish + 1; 6: System.out.println(anotherFish + " " + fishType); 7: System.out.println(numFish + " " + 1); 8: } }  |
|------------|---|
| A          | 4 1   |
| В          | 41  |
| C          | 5   |
| D          | 5 tuna  |
| E          | 5tuna Stuna   |
| F          | 51tuna  |
| ⁄ G        | The code does not compile.  |
| i          | Line 5 does not compile. This question is checking to see if you are paying attention to the types. numFish is an int and 1 is an int. Therefore, we use numeric addition and get 5. The problem is that we can't store an int in a String variable. Supposing line 5 said String anotherFish = numFish + 1 + "";. In that case, the answer would be options A and D. The variable defined on line 5 would be the string "5", and both output statements would use concatenation.   |
| 2.         | Which of the following are output by this code? (Choose all that apply) 3: String s = "Hello"; 4: String t = new String(s); 5: if ("Hello".equals(s)) System.out.println("one"); 6: if (t == s) System.out.println("two"); 7: if (t.equals(s)) System.out.println("three"); 8: if ("Hello" == s) System.out.println("four"); 9: if ("Hello" == t) System.out.println("five");   |
| <b>/</b> A | one   |
| В          | two   |
| / C        | three   |
| / D        | four  |
| E          | five  |
| F          | The code does not compile.  |
| i          | The code compiles fine. Line 3 points to the String in the string pool. Line 4 calls the String constructor explicitly and is therefore a different object than s. Lines 5 and 7 check for object equality, which is true, and so print one and three. Line 6 uses object reference equality, which is not true since we have different objects. Line 7 also compares references but is true since both references point to the object from the string pool. Finally, line 8 compares one object from the string pool with one that was explicitly constructed and returns false. |

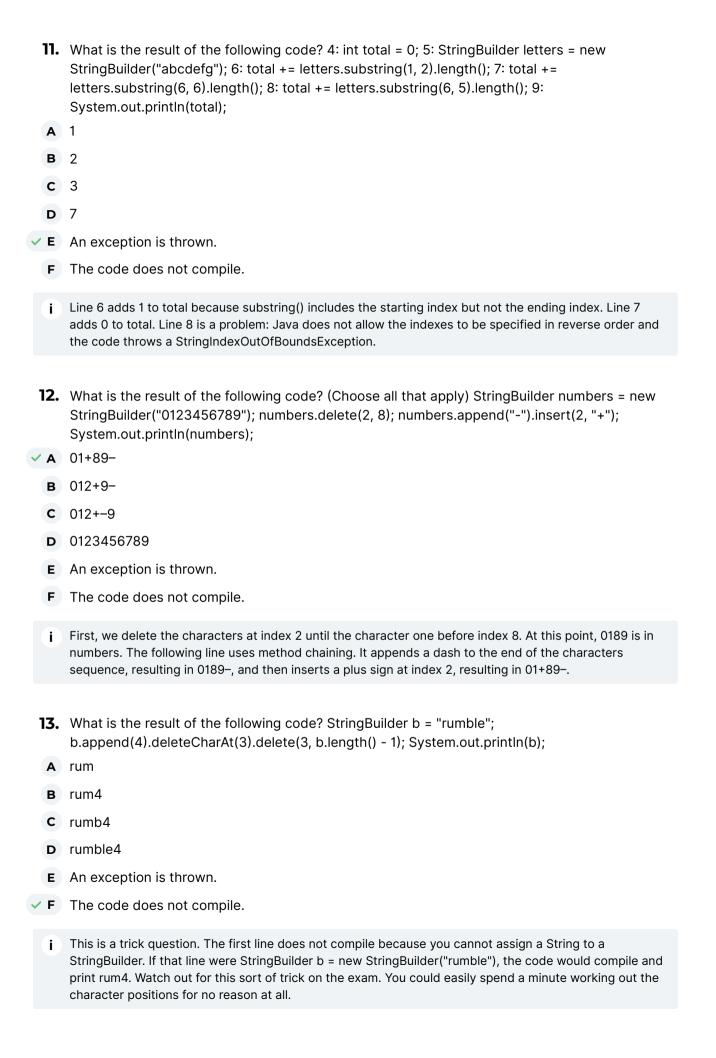




before. The second takes just the index to start with and goes to the end of the String. Remember that indexes are zero based. The first call starts at index 1 and ends with index 2 since it needs to stop before index 3. The second call starts at index 7 and ends in the same place, resulting in an empty String. This prints out a blank line. The final call starts at index 7 and goes to the end of the String.

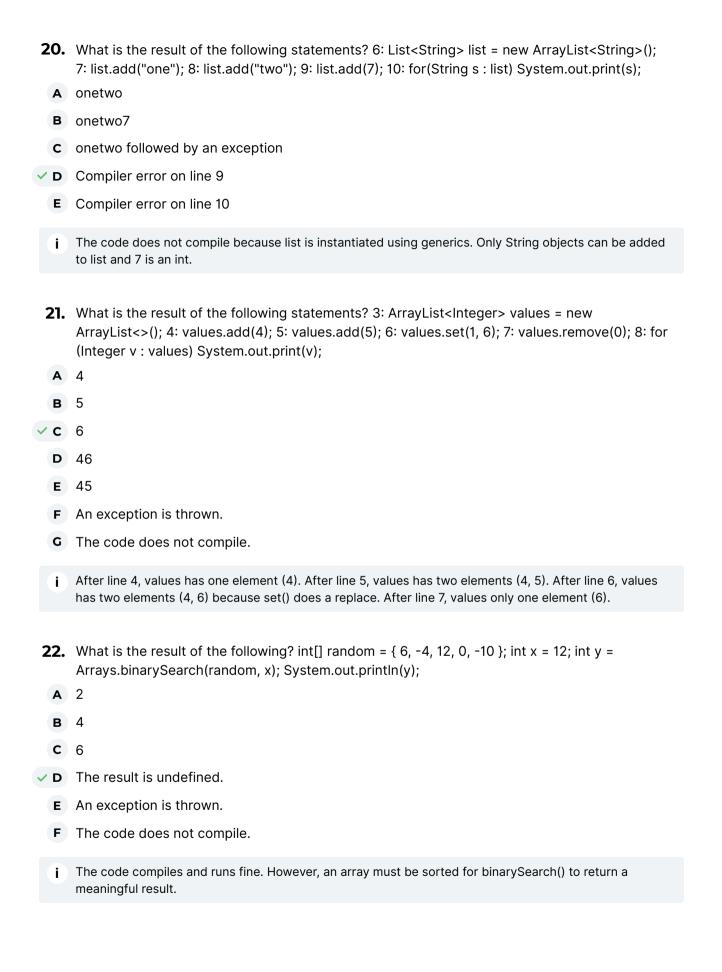


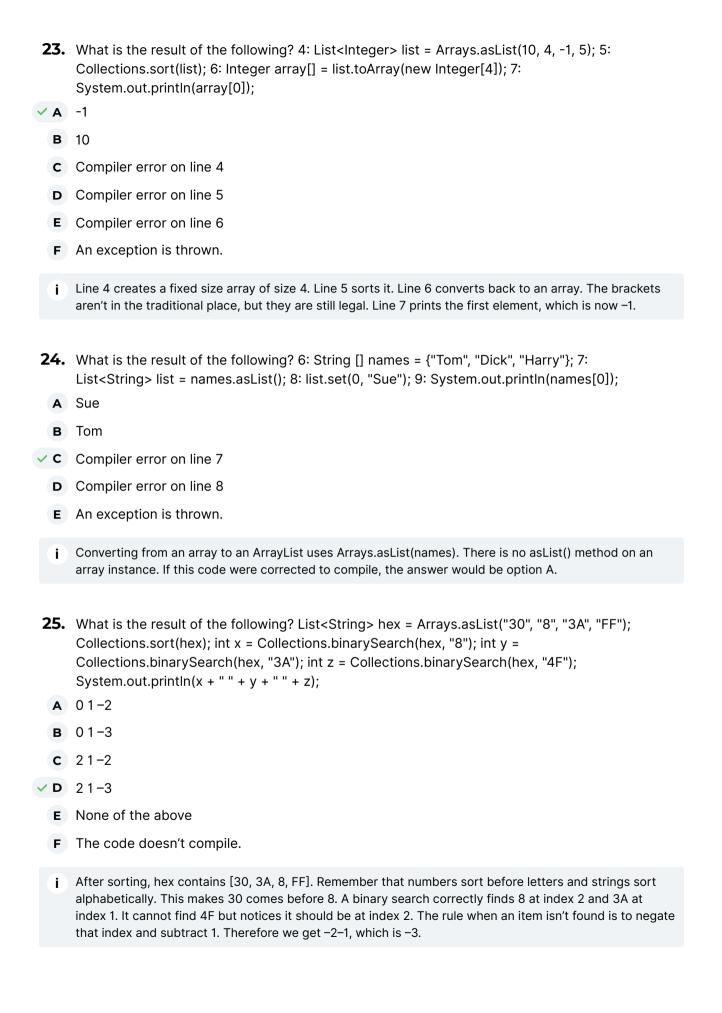
- **A** 2
- **B** 4
- **✓ C** 8
  - **D** 10
  - **E** An exception is thrown.
  - **F** The code does not compile.
  - i This question is trying to see if you know that String objects are immutable. Line 4 returns "PURR" but the result is ignored and not stored in s. Line 5 returns "purr" since there is no whitespace present but the result is again ignored. Line 6 returns "ur" because it starts with index 1 and ends before index 3 using zero-based indexes. The result is ignored again. Finally, on line 6 something happens. We concatenate four new characters to s and now have a String of length 8.
- **10.** What is the result of the following code? 13: String a = ""; 14: a += 2; 15: a += 'c'; 16: a += false; 17: if ( a == "2cfalse") System.out.println("=="); 18: if ( a.equals("2cfalse")) System.out.println("equals");
- A Compile error on line 14
- **B** Compile error on line 15
- **C** Compile error on line 16
- **D** Compile error on another line
- E ==
- ✓ F equals
  - **G** An exception is thrown.
  - i a += 2 expands to a = a + 2. A String concatenated with any other type gives a String. Lines 14, 15, and 16 all append to a, giving a result of "2cfalse". The if statement on line 18 returns false because the values of the two String objects are the same using object equality. The if statement on line 17 returns false because the two String objects are not the same in memory. One comes directly from the string pool and the other comes from building using String operations.

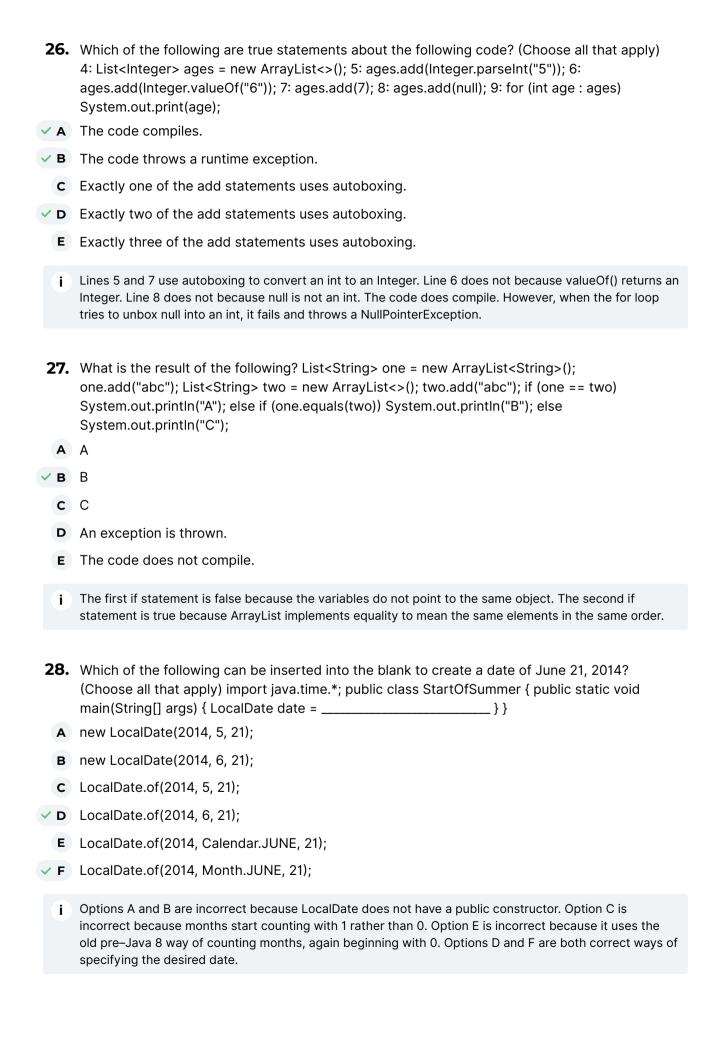


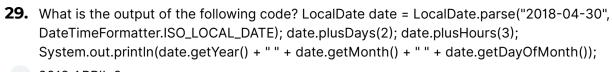
- **14.** Which of the following can replace line 4 to print "avaJ"? (Choose all that apply) 3: StringBuilder puzzle = new StringBuilder("Java"); 4: // INSERT CODE HERE 5: System.out.println(puzzle);
- ✓ A puzzle.reverse();
  - **B** puzzle.append("vaJ\$").substring(0, 4);
- puzzle.append("vaJ\$").delete(0, 3).deleteCharAt(puzzle.length() 1);
  - puzzle.append("vaJ\$").delete(0, 3).deleteCharAt(puzzle.length());
  - E None of the above
  - i The reverse() method is the easiest way of reversing the characters in a StringBuilder; therefore, option A is correct. Option B is a nice distraction—it does in fact return "avaJ". However, substring() returns a String, which is not stored anywhere. Option C uses method chaining. First it creates the value "JavavaJ\$". Then it removes the first three characters, resulting in "avaJ\$". Finally, it removes the last character, resulting in "avaJ". Option D throws an exception because you cannot delete the character after the last index. Remember that deleteCharAt() uses indexes that are zero based and length()counts starting with 1.
  - 15. Which of these array declarations is not legal? (Choose all that apply)
  - A int[][] scores = new int[5][];
  - B Object[][][] cubbies = new Object[3][0][5];
- v c String beans[] = new beans[6];
  - p java.util.Date[] dates[] = new java.util.Date[2][];
- L int[][] types = new int[];
- ✓ F int[][] java = new int[][];
  - i Option C uses the variable name as if it were a type, which is clearly illegal. Options E and F don't specify any size. Although it is legal to leave out the size for later dimensions of a multidimensional array, the first one is required. Option A declares a legal 2D array. Option B declares a legal 3D array. Option D declares a legal 2D array. Remember that it is normal to see on the exam types you might not have learned. You aren't expected to know anything about them.
- **16.** Which of these compile when replacing line 8? 7: char[]c = new char[2]; 8: // INSERT CODE HERE
- A int length = c.capacity;
- B int length = c.capacity();
- c int length = c.length;
  - **D** int length = c.length();
  - **E** int length = c.size;
  - f int length = c.size();
  - **G** None of the above
  - i Arrays define a property called length. It is not a method, so parentheses are not allowed.



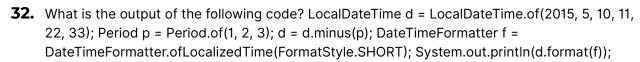








- **A** 2018 APRIL 2
- **B** 2018 APRIL 30
- **c** 2018 MAY 2
- ✓ D The code does not compile.
  - **E** A runtime exception is thrown.
  - i A LocalDate does not have a time element. Therefore, it has no method to add hours and the code does not compile.
- **30.** What is the output of the following code? LocalDate date = LocalDate.of(2018, Month.APRIL, 40); System.out.println(date.getYear() + " " + date.getMonth() + " " + date.getDayOfMonth());
  - **A** 2018 APRIL 4
  - **B** 2018 APRIL 30
  - c 2018 MAY 10
  - D Another date
  - **E** The code does not compile.
- ✓ F A runtime exception is thrown.
  - j Java throws an exception if invalid date values are passed. There is no 40th day in April—or any other month for that matter.
  - **31.** What is the output of the following code? LocalDate date = LocalDate.of(2018, Month.APRIL, 30); date.plusDays(2); date.plusYears(3); System.out.println(date.getYear() + " " + date.getMonth() + " " + date.getDayOfMonth());
  - **A** 2018 APRIL 2
- **✓ B** 2018 APRIL 30
  - **c** 2018 MAY 2
  - **D** 2021 APRIL 2
  - **E** 2021 APRIL 30
  - **F** 2021 MAY 2
  - **G** A runtime exception is thrown.
  - **i** The date starts out as April 30, 2018. Since dates are immutable and the plus methods have their return values ignored, the result is unchanged. Therefore, option B is correct.



- **A** 3/7/14 11:22 AM
- **B** 5/10/15 11:22 AM
- **c** 3/7/14
- **D** 5/10/15
- ✓ **E** 11:22 AM
  - F The code does not compile.
  - **G** A runtime exception is thrown.
  - **i** Even though d has both date and time, the formatter only outputs time.
- **33.** What is the output of the following code? LocalDateTime d = LocalDateTime.of(2015, 5, 10, 11, 22, 33); Period p = Period.ofDays(1).ofYears(2); d = d.minus(p); DateTimeFormatter f = DateTimeFormatter.ofLocalizedDateTime(FormatStyle.SHORT); System.out.println(f.format(d));
  - **A** 5/9/13 11:22 AM
- ✓ **B** 5/10/13 11:22 AM
  - **c** 5/9/14
  - **D** 5/10/14
  - **E** The code does not compile.
  - **F** A runtime exception is thrown.
  - **i** Period does not allow chaining. Only the last Period method called counts, so only the two years are subtracted.