

OCA Practice Question Ch. 9 ('Copy')

1. What is the best reason for using StringBuilder instead of String?

- ☐ A StringBuilder adds support for multiple threads.
- ☐ B StringBuilder can use == to compare values.
- ☒ C StringBuilder saves memory by reducing the number of objects created.
- ☐ D StringBuilder supports different languages and encodings.

i Option A is incorrect because StringBuilder does not support multiple threads. In fact, threads aren't even covered on the OCA, which should be your clue that this answer is wrong! You don't need to know this for the exam, but StringBuffer supports multiple threads. Option B is incorrect because == compares references, not values. Option D is incorrect because both String and StringBuilder support languages and encodings. Option C is correct and the primary reason to use StringBuilder. String often creates a new object each time you call certain methods on the object like concat(). StringBuilder optimizes operations like append() because it is mutable.

2. What is not true about a String?

- ☐ A It can be created without coding a call to a constructor.
- ☐ B It can be reused via the string pool.
- ☐ C It is final.
- ☒ D It is mutable.

i A String can be created using a literal rather than calling a constructor directly, making Option A incorrect. A string pool exists for String reuse, making Option B incorrect. A String is final and immutable, making Option C incorrect and Option D correct.

3. Which of the following creates a StringBuilder with a different value than the other options?

- ☐ A new StringBuilder().append("clown")
- ☐ B new StringBuilder("clown")
- ☐ C new StringBuilder("cl").insert(2, "own")
- ☒ D All of them create the same value.

i This question is testing whether you understand how method chaining works. Option A creates an empty StringBuilder and then adds the five characters in clown to it. Option B simply creates the clown when calling the constructor. Finally, Option C creates the same value, just in two parts. Therefore, Option D is correct.

4. `StringBuilder teams = new StringBuilder("333");
teams.append(" 806");
teams.append(" 1601");
System.out.print(teams);`

What is the output of the following?

- ☐ A 333
- ☒ B 333 806 1601
- ☐ C The code compiles but outputs something else.
- ☐ D The code does not compile.

i Since `StringBuilder` is mutable, each call to `append` adds to the value. When calling `print`, `toString()` is automatically called and `333 806 1601` is output. Therefore, Option B is correct.

5. `List frisbees = new _____();`

How many of the types `ArrayList`, `List`, and `Object` can fill in the blank to produce code that compiles?

- ☐ A None
- ☒ B One
- ☐ C Two
- ☐ D Three

i `List` is an interface and not a class. It cannot be instantiated. While `Object` is a concrete class, it does not implement the `List` interface so it cannot be assigned to `frisbees`. Note that if you were to add an explicit cast, it would compile and throw an exception at runtime. Of the three options, only `ArrayList` can fill in the blank, so Option B is correct.

6. `List<String> tools = new ArrayList<>();
tools.add("hammer");
tools.add("nail");
tools.add("hex key");
System.out.println(tools.get(1));`

What is the output of the following?

- ☐ A hammer
- ☐ B hex key
- ☒ C nail
- ☐ D None of the above

i An `ArrayList` does not automatically sort the elements. It simply remembers them in order. Since Java uses zero-based indexes, Option C is correct.

```
7.  StringBuilder sb = new StringBuilder("radical")
    .insert(sb.length(), "robots");
    System.out.println(sb);
```

What is the result of the following code?

- ☐ A radicarobots
- ☐ B radicalrobots
- ☒ C The code does not compile.
- ☐ D The code compiles but throws an exception at runtime.

i Calling the constructor and then insert() is an example of method chaining. However, the sb.length() call is a problem. The sb reference doesn't exist until after the chained calls complete. Just because it happens to be on a separate line doesn't change when the reference is created. Since the code does not compile, Option C is correct.

```
8.  List<String> museums = new ArrayList<>(1);
    museums.add("Natural History");
    museums.add("Science");
    museums.add("Art");
    museums.remove(2);
    System.out.println(museums);
```

What is the output of the following?

- ☒ A [Natural History, Science]
- ☐ B [Natural History, Art, Science]
- ☐ C The code does not compile.
- ☐ D The code compiles but throws an exception at runtime.

i While the ArrayList is declared with an initial capacity of one element, it is free to expand as more elements are added. Each of the three calls to the add() method adds an element to the end of the ArrayList. The remove() method call deletes the element at index 2, which is Art. Therefore, Option A is correct.

```
9.  12:  StringBuilder b= new StringBuilder("12");
    13:  b = b.append("3");
    14:  b.reverse();
    15:  System.out.println(b.toString());
```

What is the output of the following?

- ☐ A 12
- ☐ B 123
- ☒ C 321
- ☐ D The code does not compile.

i On line 12, the value of the StringBuilder is 12. On line 13, it becomes 123. Since StringBuilder is mutable, storing the result in the same reference is redundant. Then on line 14, the value is reversed, giving us 321

and making Option C correct.

10. What is the main benefit of a lambda expression?

- ☐ A It allows you to convert a primitive to a wrapper class.
- ☐ B It allows you to change the bytecode while the application is running.
- ☐ C It allows you to inherit from multiple classes.
- ☒ D It allows you to write code that has the execution deferred.

i Option A is incorrect as it describes autoboxing. Options B and C are not possible in Java. Option D is correct as it describes lambdas. Lambdas use deferred execution and can be run elsewhere in the codebase.

11.

```
5:  StringBuilder line = new StringBuilder("-");
6:  StringBuilder anotherLine = line.append("-");
7:  System.out.print(line == anotherLine);
8:  System.out.print(" ");
9:  System.out.print(line.length());
```

What is the output of the following?

- ☐ A false 1
- ☐ B false 2
- ☐ C true 1
- ☒ D true 2

i A StringBuilder is mutable, so the length is two after line 6 completes. The StringBuilder methods return a reference to the same object so you can chain method calls. Therefore, line and anotherLine refer to the same object. This means that line 7 prints true. Then on line 9, both references point to the same object of length 2, and Option D is correct.

12.

```
public static void secret(_____ mystery) {
    mystery.add("metal");
    String str = mystery.get(0);
    int num = mystery.length();
}
```

The author of this method forgot to include the data type. Which of the following reference types can fill in the blank to complete this method?

- ☐ A ArrayList
- ☐ B ArrayList<String>
- ☐ C StringBuilder
- ☒ D None of the above

i The add() and get() methods are available on ArrayList. However, ArrayList uses size rather than length to get the number of elements. Therefore, Option D is correct. If length was changed to size, Option B would compile if put in the blank. Option A still wouldn't compile in the blank because a cast would be needed to store the value in str.

13. `Predicate<StringBuilder> p = (StringBuilder b) -> {return true;;}`

Which portion of code can be removed so that this line of code continues to compile?

- ☐ A Remove StringBuilder b
- ☐ B Remove →
- ☐ C Remove { and ;}
- ☒ D Remove { return and ;}

i Option A is tricky, but incorrect. While a lambda can have zero parameters, a Predicate cannot. A Predicate is defined as a type mapping to a boolean. Option B is clearly incorrect as → separates the parts of a lambda. Options C and D are similar. Option C is incorrect because return is only allowed when the brackets are present. Option D is correct.

14. `20: List<Character> chars = new ArrayList<>();
21: chars.add('a');
22: chars.add('b');
23: chars.set(1, 'c');
24: chars.remove(0);
25: System.out.print(chars.size() + " " + chars.contains('b'));`

What is the output of the following?

- ☒ A 1 false
- ☐ B 1 true
- ☐ C 2 false
- ☐ D 2 true

i Lines 20–22 create an ArrayList with two elements. Line 23 replaces the second one with a new value. Now chars is [a, c]. Then line 24 removes the first element, making it just [c]. Option A is correct because there is only one element, but it is not the value b.

15. `12: String b = "12";
13: b += "3";
14: b.reverse();
15: System.out.println(b.toString());`

What is the output of the following?

- ☐ A 12
- ☐ B 123
- ☐ C 321
- ☒ D The code does not compile.

i Trick question. There is no reverse method on the String class. There is one on the StringBuilder class. Therefore, the code does not compile, and Option D is correct.

16. `Predicate<String> pred1 = s -> false;`
`Predicate<String> pred2 = (s) -> false;`
`Predicate<String> pred3 = String s -> false;`
`Predicate<String> pred4 = (String s) -> false;`

How many of these lines fail to compile?

- ✓ **A** One
- B** Two
- C** Three
- D** Four

i When creating a lambda with only one parameter, there are a few variants. The `pred1` approach shows the shortest way, where the type is omitted and the parentheses are omitted. The `pred2` approach is similar except it includes the parentheses. Both are legal. The `pred4` approach is the long way with both the parentheses and type specified. The only one that doesn't compile is `pred3`. The parentheses are required if including the type.

17.

```
public class Shoot {
    interface Target {
        boolean needToAim(double angle);
    }
    static void prepare(double angle, Target t) {
        boolean ready = t.needToAim(angle); // k1
        System.out.println(ready);
    }
    public static void main(String[] args) {
        prepare(45, d -> d > 5 || d < -5); // k2
    }
}
```

What does the following do?

- ✓ **A** It prints true.
- B** It prints false.
- C** It doesn't compile due to line k1.
- D** It doesn't compile due to line k2.

i This is a correct example of code that uses a lambda. The interface has a single abstract method. The lambda correctly takes one double parameter and returns a boolean. This matches the interface. The lambda syntax is correct. Since 45 is greater than 5, Option A is correct.

18. `String teams = new String("694");`
`teams.concat(" 1155");`
`teams.concat(" 2265");`
`teams.concat(" 2869");`
`System.out.println(teams);`

What is the output of the following?

- ✓ A 694
- B 694 1155 2265 2869
- C The code compiles but outputs something else.
- D The code does not compile.

i Since String is immutable, each call to `concat()` returns a new object with the new value. However, that return value is ignored and the `teams` variable never changes in value. Therefore it stays as 694, and Option A is correct.

19. Which of these classes are in the `java.util` package?

- I. `ArrayList`
- II. `LocalDate`
- III. `String`

- ✓ A I only
- B II only
- C I and II
- D I, II, and III

i The `ArrayList` class is in the `java.util` package, making I correct. The `LocalDate` class is in the `java.time` package, making II incorrect. The `String` class is in the `java.lang` package, which means you can use it without typing an import, making III incorrect. Therefore, Option A is correct.

20. `StringBuilder sb = new StringBuilder("radical ");`
`sb = _____;`
`System.out.print(sb);`

Which of the answer choices results in a different value being output than the other three choices?

- ☐ A `new StringBuilder("radical ")`
`.append("robots")`
- ☐ B `new StringBuilder("radical ")`
`.delete(1, 100)`
`.append("obots")`
`.insert(1, "adical r")`
- ☒ C `new StringBuilder("radical ")`
`.insert(7, "robots")`
- ☐ D `new StringBuilder("radical ")`
`.insert(sb.length(), "robots")`

i Option A is straightforward and outputs radical robots. Option B does the same in a convoluted manner. First Option B removes all the characters after the first one. It doesn't matter that there aren't actually 100 characters to delete. Then it appends obots to the end, making the builder contain robots. Finally, it inserts the remainder of the string immediately after the first index. Try drawing the flow if this is hard to envision. Option D also creates the same value by inserting robots immediately after the end of the `StringBuilder`. Option C is close, but it has an off-by-one error. It inserts robots after the letter l rather than after the space. This results in the value radicalrobots followed by a space. Option C is different than the others and the correct answer.

21. `String[] array = {"Natural History", "Science"};`
`List<String> museums = Arrays.asList(array);`
`museums.set(0, "Art");`
`System.out.println(museums.contains("Art"));`

What is the output of the following?

- ☒ A true
- ☐ B false
- ☐ C The code does not compile.
- ☐ D The code compiles but throws an exception at runtime.

i Since we are creating the list from an array, it is a fixed size. We are allowed to change elements. At the end of this code, museums is [Art, Science]. Therefore, it contains Art, and Option A is correct.

22. Which is a true statement?

- ☐ A If `s.contains("abc")` is true, then `s.equals("abc")` is also true.
- ☐ B If `s.contains("abc")` is true, then `s.startsWith("abc")` is also true.
- ☐ C If `s.startsWith("abc")` is true, then `s.equals("abc")` is also true.
- ☒ D If `s.startsWith("abc")` is true, then `s.contains("abc")` is also true.

- i Options A and B are not true if the String is "deabc". Option C is not true if the String is "abcde". Option D is true in all cases.

23.

```
20: List<Character> chars = new ArrayList<>();
21: chars.add('a');
22: chars.add('b');
23: chars.set(1, 'c');
24: chars.remove(0);
25: System.out.print(chars.length());
```

What is the output of the following?

- ☐ A 0
☐ B 1
☐ C 2
☒ D None of the above

- i Line 25 does not compile. On an ArrayList, the method to get the number of elements is size. The length() method is used for a String or StringBuilder.

24.

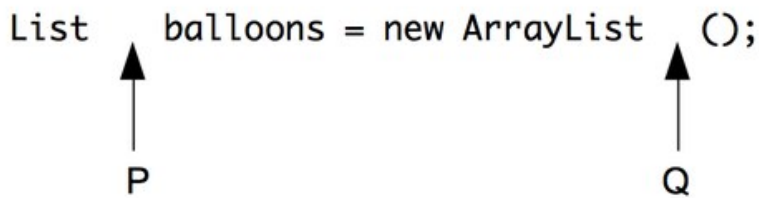
```
public static void secret(_____ mystery) {
    mystery = mystery.replace("1", "8");
    mystery.startsWith("paper");
    String s = mystery.toString();
}
```

The author of this method forgot to include the data type. Which of the following reference types can fill in the blank to complete this method?

- ☒ A ArrayList
☐ B String
☐ C StringBuilder
☐ D None of the above

- i The toString() method call doesn't help in narrowing things down as all Java objects have that method available. The other two methods are more helpful. String is the only type of these three to have a startsWith() method, making Option B correct. String also has the replace() method declared here. If you memorized the whole API, you might know that StringBuilder also has a replace() method, but it requires three parameters instead of two. Please don't memorize the API in that level of detail. We included what you need to know in our study guide. If you do have this outside knowledge, be careful not to read into the questions!

25. `List balloons = new ArrayList ();`



Which statement is true about the following figure while ensuring the code continues to compile?

- ☐ A <> can be inserted at position P without making any other changes.
- ☒ B <> can be inserted at position Q without making any other changes.
- ☐ C <> can be inserted at both positions P and Q.
- ☐ D None of the above.

i The <> is known as the diamond operator. Here, it works as a shortcut to avoid repeating the generic type twice for the same declaration. On the right side of the expression, this is a handy shortcut. Java still needs the type on the left side so there is something to infer. In the figure, position P is the left side and position Q is the right side. Therefore, Option B is correct.

26.

```
import java.util.function.*;
public class Card {
    public static void main(String[] s) {
        Predicate<String> pred = _____ -> true;
    }
}
```

Which of the following can fill in the blank to make the code compile?

- ☐ A (Integer i)
- ☐ B (Object o)
- ☐ C (String s)
- ☒ D None of the above

i The type in the lambda must match the generic declared on the Predicate. In this case, that is String. Therefore, Options A and B are incorrect. While Option C is of the correct type, it uses the variable s, which is already in use from the main() method parameter. Therefore, none of these are correct, and Option D is the answer.

27.

```
5: String line = new String("-");
6: String anotherLine = line.concat("-");
7: System.out.print(line == anotherLine);
8: System.out.print(" ");
9: System.out.print(line.length());
```

What is the output of the following?

- ☒ A false 1
- ☐ B false 2
- ☐ C true 1
- ☐ D true 2

i A String is immutable so a different object is returned on line 6. The object anotherLine points to is of length 2 after line 6 completes. However, the original line reference still points to an object of length 1. Therefore, Option A is correct.

28.

```
Predicate dash = c -> c.startsWith("-");
System.out.println(dash.test("--"));
```

What does the following output?

- ☐ A true
- ☐ B false
- ☒ C The code does not compile.
- ☐ D The code compiles but throws an exception at runtime.

i While it is common for a Predicate to have a generic type, it is not required. However, it is treated like a Predicate of type Object if the generic type is missing. Since startsWith() does not exist on Object, the first line does not compile, and Option C is correct.

29. Of the classes LocalDate, LocalDateTime, LocalTime, and LocalTimeStamp, how many include hours, minutes, and seconds?

- ☐ A One
- ☒ B Two
- ☐ C Three
- ☐ D Four

i LocalDate only includes the date portion and not the time portion. There is no class named LocalTimeStamp. The other two, LocalDateTime and LocalTime, both include the time elements, making Option B correct.

30.

```
1: package rocket;
2: public class Countdown {
3:     public static void main(String[] args) {
4:         String builder = "54321";
5:         builder = builder.substring(4);
6:         System.out.println(builder.charAt(2));
7:     }
8: }
```

What is the output of the following class?

- ☐ A 2
- ☐ B 3
- ☐ C 4
- ☒ D None of the above

i Line 4 creates a String of length 5. Since String is immutable, line 5 creates a new String with the value 1 and assigns it to builder. Remember that indexes in Java begin with 0, so the substring() method is taking the values from the fifth element through the end. Since the first element is the last element, there's only one character in there. Then line 6 tries to retrieve the second indexed element. Since there is only one element, this gives a StringIndexOutOfBoundsException, and Option D is correct.

31. `Predicate<Integer> ip = i -> i != 0;`

Which equivalent code can replace `i -> i != 0` in the following line?

- ☐ A `i -> { i != 0 }`
- ☐ B `i -> { i != 0; }`
- ☐ C `i -> { return i != 0 }`
- ☒ D `i -> { return i != 0; }`

i When you're using brackets, both the return keyword and semicolon are needed for the lambda to compile, making Option D correct.

32.

```
LocalDate xmas = LocalDate.of(2016, 12, 25);
xmas.plusDays(-1);
System.out.println(xmas.getDayOfMonth());
```

What is the output of the following?

- ☐ A 24
- ☒ B 25
- ☐ C 26
- ☐ D None of the above

i Java 8 date and time classes are immutable. The plusDays method returns a LocalDate object presenting Christmas Eve (December 24th). However, this return value is ignored. The xmas variable still represents the original value, so Option B is correct.

33.

```
1: public class Legos {
2:     public static void main(String[] args) {
3:         StringBuilder sb = new StringBuilder();
4:         sb.append("red");
5:         sb.deleteCharAt(0);
6:         sb.delete(1, 2);
7:         System.out.println(sb);
8:     }
9: }
```

What is the output of the following?

- ☒ A e
- ☐ B d
- ☐ C ed
- ☐ D None of the above

i Line 3 creates an empty `StringBuilder`. Line 4 adds three characters to it. Line 5 removes the first character, resulting in `ed`. Line 6 deletes the characters starting at position 1 and ending right before position 2, which removes the character at index 1, which is `d`. The only character left is `e`, so Option A is correct.

34.

```
Predicate clear = c -> c.equals("clear");
System.out.println(clear.test("pink"));
```

What does the following output?

- ☐ A true
- ☒ B false
- ☐ C The code does not compile.
- ☐ D The code compiles but throws an exception at runtime.

i While it is common for a `Predicate` to have a generic type, it is not required. When the generic is omitted, it is treated like a `Predicate` of type `Object`. Since the `equals()` method exists on `Object`, this is fine. Option B is correct because the `Predicate` tests as false.

35. Which starts counting from one rather than zero?

- ☐ A Array indexes
- ☐ B The index used by `charAt` in a `String`
- ☒ C The months in a `LocalDateTime`
- ☐ D The months in a `LocalTime`

i In Java, most things use zero-based indexes, including arrays and a `String`. Months are an exception to this convention starting Java 8. This makes the answer either Option C or D. However, `LocalTime` does not contain date fields, so it has to be Option C.

36. Which statement is not true of Predicate?
- ☐ A A boolean is returned from the method it declares.
 - ☐ B It is an interface.
 - ☒ C The method it declares accepts two parameters.
 - ☐ D The method it declares is named test.

i Predicate is an interface with one method. The method signature is `boolean test(T t)`. Option C is the answer because the method accepts one parameter rather than two.

37. `Period period1 = Period.ofWeeks(1).ofDays(3);`
`Period period2 = Period.ofDays(10);`

Which of these periods represents a larger amount of time?

- ☐ A period1
- ☒ B period2
- ☐ C They represent the same length of time.
- ☐ D None of the above. This code does not compile.

i Be careful here. The `Period` class uses a static helper method to return the period. It does not chain method calls, so `period1` only represents three days. Since three days is less than 10 days, `period2` is larger, and Option B is correct.

38.

```
import java.time.*;
import java.time.format.*;

public class HowLong {
    public static void main(String[] args) {
        LocalDate newYears = LocalDate.of(2017, 1, 1);
        Period period = Period.ofDays(1);
        DateTimeFormatter format = DateTimeFormatter.ofPattern("MM-dd-yyyy");
        System.out.print(format.format(newYears.minus(period)));
    }
}
```

What is the result of the following?

- ☐ A 01-01-2017
- ☒ B 12-31-2016
- ☐ C The code does not compile.
- ☐ D The code compiles but throws an exception at runtime.

i The code starts by correctly creating a date representing January 1, 2017, and a period representing one day. It then explicitly defines the format as month followed by day followed by year. Finally, the code subtracts a day, giving us the formatted version of December 31, 2016.

39. `String happy = " :) - (: ";`
`String really = happy.trim();`
`String question = _____;`
`System.out.println(really.equals(question));`

Which of the following can fill in the blank so the following code prints true?

- ☐ A `happy.substring(0, happy.length() - 1)`
- ☐ B `happy.substring(0, happy.length())`
- ☒ C `happy.substring(1, happy.length() - 1)`
- ☐ D `happy.substring(1, happy.length())`

i The `trim()` method returns a `String` with all leading and trailing white space removed. In this question, that's the seven-character `String`: `" :) - (: "`. Options A and B are incorrect because they do not remove the first blank space in `happy`. Option D is incorrect because it does not remove the last character in `happy`. Therefore, Option C is correct.

40. Which is not a true statement about the `Period` class?

- ☐ A A `Period` is immutable.
- ☐ B A `Period` is typically used for adding or subtracting time from dates.
- ☒ C You can create a `Period` representing 2 minutes.
- ☐ D You can create a `Period` representing 5 years.

i The `Period` class creates immutable objects and is usually used to add/subtract from a `LocalDate` or `LocalDateTime` object. It allows creating date, week, month, or year periods. Since it cannot be used for time, Option C is the answer.

41.

```
1: package rocket;
2: public class Countdown {
3:     public static void main(String[] args) {
4:         StringBuilder builder = new StringBuilder("54321");
5:         builder.substring(2);
6:         System.out.println(builder.charAt(1));
7:     }
8: }
```

What is the output of the following class?

- ☐ A 1
- ☐ B 2
- ☐ C 3
- ☒ D 4

i Line 4 creates a `StringBuilder` of length 5. Pay attention to the `substring()` method `StringBuilder`. It returns a `String` with the value `321`. It does not change the `StringBuilder` itself. Then line 6 is retrieving the second indexed element from that unchanged value, which is 4. Therefore, Option D is correct.

42.

```
List<Integer> pennies = new ArrayList<>();
pennies.add(3);
pennies.add(2);
pennies.add(1);
pennies.remove(2);
System.out.println(pennies);
```

What does the following output?

- ☐ A [3, 1]
- ☒ B [3, 2]
- ☐ C The code does not compile.
- ☐ D The code compiles but throws an exception at runtime.

i This one is tricky. There are two `remove()` methods available on `ArrayList`. One removes an element by index and takes an `int` parameter. The other removes an element by value. Due to the generics, it takes an `Integer` parameter in this example. Since the `int` primitive is a better match, the element with index 2 is removed, which is the value of 1. Therefore, Option B is correct.

43.

```
public static void secret(_____ mystery) {
    char ch = mystery.charAt(3);
    mystery = mystery.insert(1, "more");
    int num = mystery.length();
}
```

The author of this method forgot to include the data type. Which of the following reference types can best fill in the blank to complete this method?

- ☐ A `ArrayList`
- ☐ B `String`
- ☒ C `StringBuilder`
- ☐ D None of the above

i `ArrayList` has a `size()` method rather than a `length()` method, making Option A incorrect. The `charAt()` and `length()` methods are declared on both `String` and `StringBuilder`. However, the `insert()` method is only declared on a `StringBuilder` and not a `String`. Therefore, Option C is correct.

44. What is the smallest unit you can add to a `LocalTime` object?

- ☐ A Second
- ☐ B Millisecond
- ☒ C Nanosecond
- ☐ D Picosecond

i The `minusNanos` and `plusNanos` are the smallest units available, making Option C correct. Option D is incorrect because `LocalTime` is not that granular. Note that while you can add milliseconds by adding many nanoseconds, there isn't a method for it. A millisecond is also larger than a nanosecond. Finally, don't be tricked by the fact that `LocalTime` is immutable. You can still add time; it just gets returned as a different object.

45.

```
import java.time.*;
import java.time.format.*;

public class HowLong {
    public static void main(String[] args) {
        LocalDate newYears = LocalDate.of(2017, 1, 1);
        Period period = Period.ofDays(1);
        DateTimeFormatter format = DateTimeFormatter.ofPattern("mm-dd-yyyy");
        System.out.print(format.format(newYears.minus(period)));
    }
}
```

What is the result of the following?

- A** 01-01-2017
- B** 12-31-2016
- C** The code does not compile.
- ☒ **D** The code compiles but throws an exception at runtime.

i When creating a formatter object, remember that MM represents month while mm represents minute. Since there are not minutes defined on a LocalDate object, the code throws an `UnsupportedTemporalTypeException`. You don't need to know the name of the exception, but you do need to know that an exception is thrown.

46. Which of the following types can you pass as a parameter to the `replace()` method on the `String` class?

- I. `char`
- II. `String`
- III. `StringBuilder`

- A** I
- B** I and II
- C** II and III
- ☒ **D** I, II, and III

i There are two signatures for the `replace()` method. One takes two `char` parameters. The other signature takes a `CharSequence`. Both `String` and `StringBuilder` implement this interface. This makes all three alternatives correct, and Option D is correct.

47.

```
import java.util.*;
import java.util.function.*;

public class PrintNegative {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("-5");
        list.add("0");
        list.add("5");
        print(list, e -> e < 0);
    }
    public static void print(List<String> list, Predicate<Integer> p) {
        for (String num : list)
            if (p.test(num))
                System.out.println(num);
    }
}
```

How many lines does this code output?

- A** One
- B** Two
- ☒ **C** None. The code does not compile.
- D** None. The code throws an exception at runtime.

- i** Pay attention to the data types. The `print()` method is looping through a list of `String` objects. However, the `Predicate` expects an `Integer`. Since these don't match, the `if` statement does not compile.

48.

```
12: List<String> magazines = new ArrayList();
13: magazines.add("Readers Digest");
14: magazines.add("People");
15: magazines.clear();
16: magazines.add("The Economist");
17: magazines.remove(1);
18: System.out.println(magazines.size());
```

What is the output of the following?

- A** 0
- B** 1
- C** The code does not compile.
- ☒ **D** The code compiles but throws an exception at runtime.

- i** Line 12 creates an empty `ArrayList`. While it isn't recommended to use generics on only the left side of the assignment operator, this is allowed. It just gives a warning. Lines 13 and 14 add two elements. Line 15 resets to an empty `ArrayList`. Line 16 adds an element, so now we have an `ArrayList` of size 1. Line 17 attempts to remove the element at index 1. Since Java uses zero-based indexes, there isn't an element there and the code throws an `IndexOutOfBoundsException`.

49. What is the output of the following?

- A** b
- B** black
- ☒ **C** The code does not compile.
- D** The code compiles but throws an exception at runtime.

- i** The declaration of `witch` is incorrect. It tries to store a `char` into a `String` variable reference. This does not compile, making Option C correct. If this was fixed, the answer would be Option B.

50.

```
LocalDate xmas = LocalDate.of(2016, 12, 25);
xmas.setYear(2017);
System.out.println(xmas.getYear());
```

What is the result of the following?

- A** 2016
- B** 2017
- ☒ **C** The code does not compile.
- D** The code compiles but throws an exception at runtime.

- i** The Java 8 date and time classes are immutable. This means they do not contain setter methods and the code does not compile.