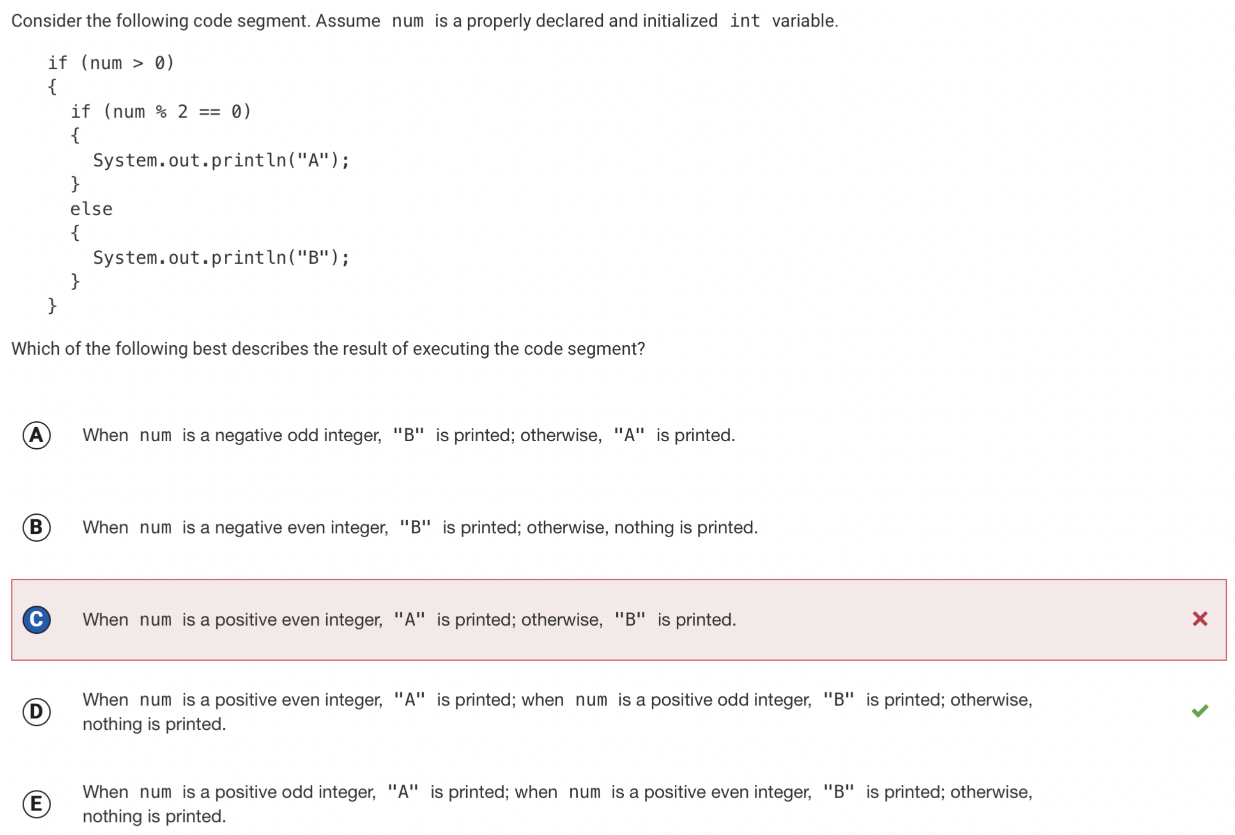
Question 2



Consider the following code segment. Assume `num` is a properly declared and initialized `int` variable.

```
if (num > 0)
{
  if (num % 2 == 0)
  {
    System.out.println("A");
  }
  else
  {
    System.out.println("B");
  }
}
```

Which of the following best describes the result of executing the code segment?

(A) When `num` is a negative odd integer, `"B"` is printed; otherwise, `"A"` is printed.

(B) When `num` is a negative even integer, `"B"` is printed; otherwise, nothing is printed.

[✓ (C)] When `num` is a positive even integer, `"A"` is printed; otherwise, `"B"` is printed. ✗

(D) When `num` is a positive even integer, `"A"` is printed; when `num` is a positive odd integer, `"B"` is printed; otherwise, nothing is printed. ✓

(E) When `num` is a positive odd integer, `"A"` is printed; when `num` is a positive even integer, `"B"` is printed; otherwise, nothing is printed.

This was an unnecessary error by me. The code segment first only works if the number is greater than 0, or if it's positive. I didn't read carefully and notice that and didn't read the final two answer choices to notice this.

Question 5

Consider the following class definition.

```
public class Bird
{
  private String species;
  private String color;
  private boolean canFly;
  public Bird(String str, String col, boolean cf)
  {
    species = str;
    color = col;
    canFly = cf;
  }
}
```

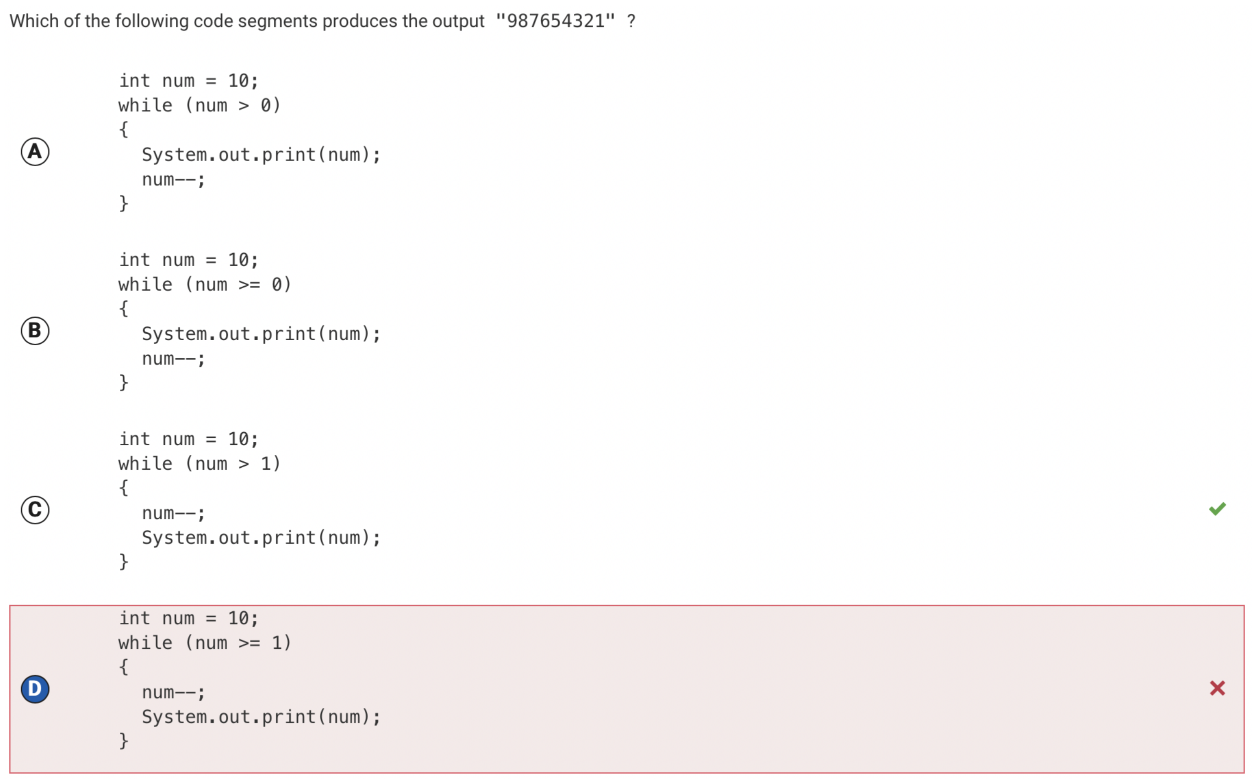
Which of the following constructors, if added to the `Bird` class, will cause a compilation error?

Ⓐ ✗

```
public Bird()
{
  species = "unknown";
  color = "unknown";
  canFly = false;
}
```

Explanation: I was not sure about this because it depended on the parameters being set. Turns out that any constructor, in this case, would work as long as the parameters listed are not the exact same as that of when the variable Bird is first declared because the matching signatures would result in a compiling error. Thus, the correct answer is E, as the line public Bird(String col, String str, boolean cf) has the matching signature and thus would cause the compiling error.

Question 11



Which of the following code segments produces the output "987654321" ?

(A)

```
int num = 10;
while (num > 0)
{
  System.out.print(num);
  num--;
}
```

(B)

```
int num = 10;
while (num >= 0)
{
  System.out.print(num);
  num--;
}
```

(C) ✓

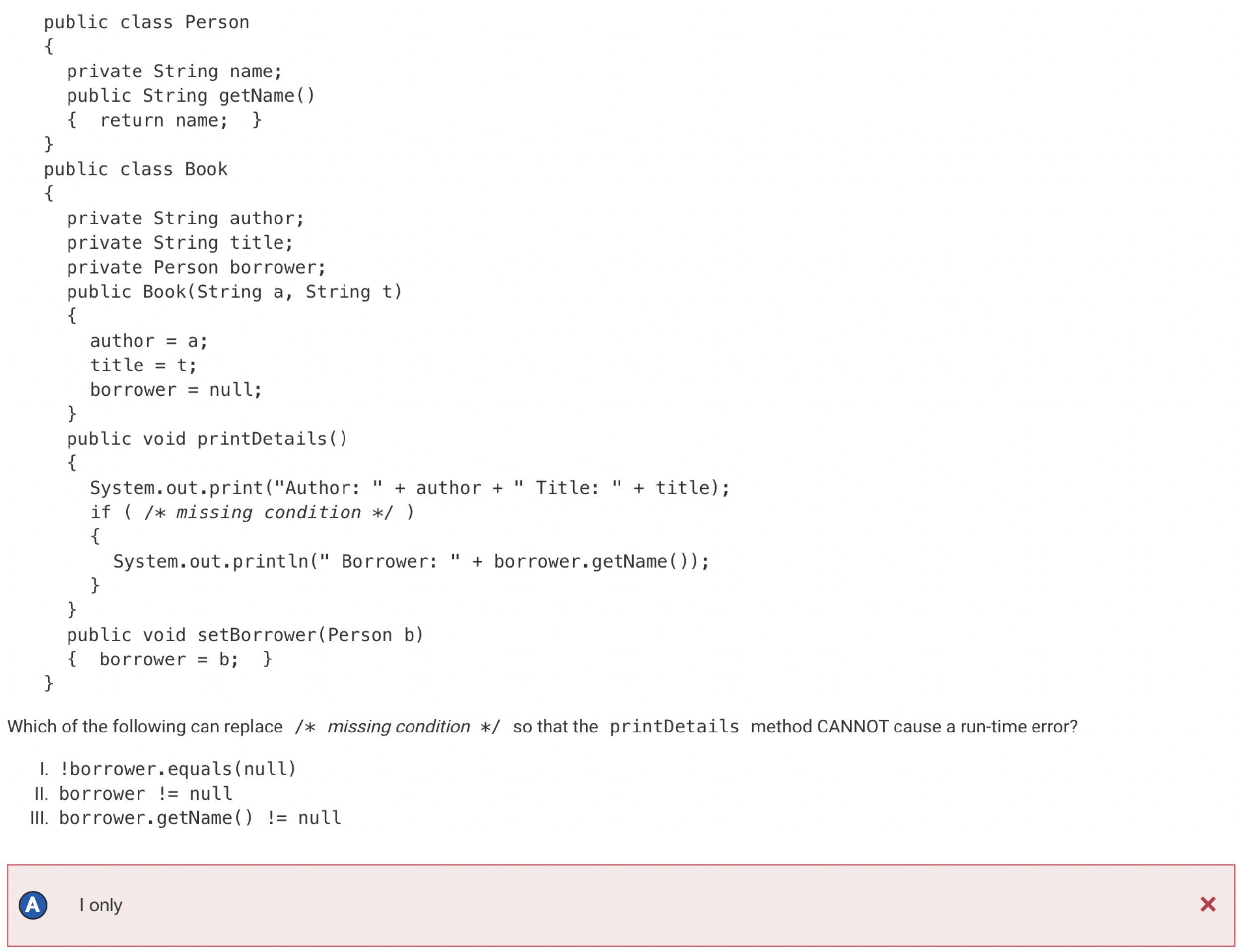
```
int num = 10;
while (num > 1)
{
  num--;
  System.out.print(num);
}
```

[✓ D] ✗

```
int num = 10;
while (num >= 1)
{
  num--;
  System.out.print(num);
}
```

This was a miscalculation by me. Option D’s while statement first subtracts a number and then prints it. Thus, when it gets to 1, it will subtract 1 and print 0, which is wrong as 0 should not be printed. Thus, the while statement needs to terminate when num is greater than 1 so that it would not print 0. In this case, option C would be correct.

Question 12



```
public class Person
{
  private String name;
  public String getName()
  {  return name;  }
}
public class Book
{
  private String author;
  private String title;
  private Person borrower;
  public Book(String a, String t)
  {
    author = a;
    title = t;
    borrower = null;
  }
  public void printDetails()
  {
    System.out.print("Author: " + author + " Title: " + title);
    if ( /* missing condition */ )
    {
      System.out.println(" Borrower: " + borrower.getName());
    }
  }
  public void setBorrower(Person b)
  {  borrower = b;  }
}
```

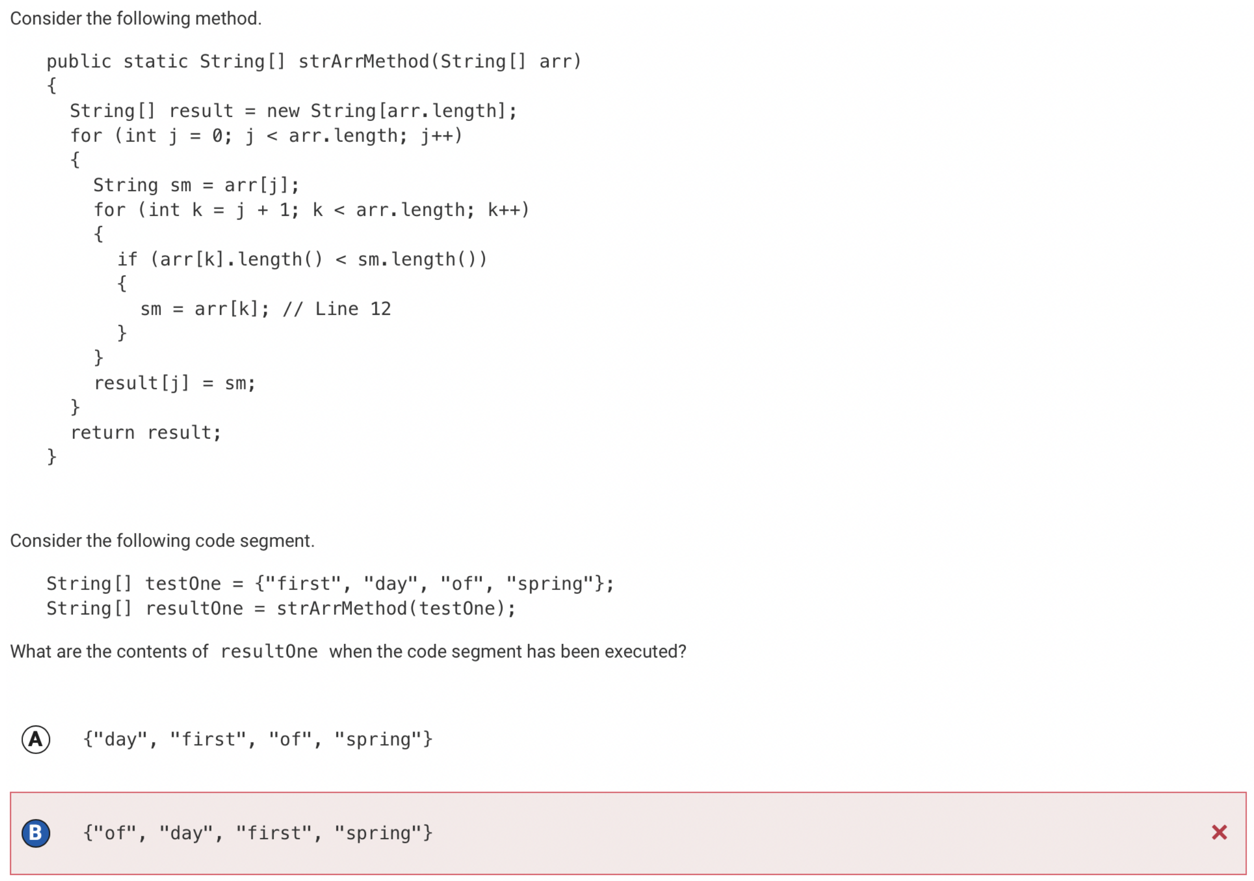
Which of the following can replace `/* missing condition */` so that the `printDetails` method CANNOT cause a run-time error?

I. `!borrower.equals(null)`
II. `borrower != null`
III. `borrower.getName() != null`

A I only ✗

Option I is wrong because if the no Person object has been assigned to borrower, the method call borrower.equals(null) throws a NullPointerException. Condition II is correct. This condition ensures that borrower contains a reference to an object when it is used in the println method call that follows. Option II would only be correct because this condition ensures that borrower contains a reference to an object when it is used in the println method call that follows.

Question 21



Consider the following method.

```
public static String[] strArrMethod(String[] arr)
{
  String[] result = new String[arr.length];
  for (int j = 0; j < arr.length; j++)
  {
    String sm = arr[j];
    for (int k = j + 1; k < arr.length; k++)
    {
      if (arr[k].length() < sm.length())
      {
        sm = arr[k]; // Line 12
      }
    }
    result[j] = sm;
  }
  return result;
}
```

Consider the following code segment.

```
String[] testOne = {"first", "day", "of", "spring"};
String[] resultOne = strArrMethod(testOne);
```

What are the contents of `resultOne` when the code segment has been executed?

(A) `{"day", "first", "of", "spring"}`

[✓ (B)] `{"of", "day", "first", "spring"}` ✗

Option B is wrong because it would represent the contents of resultOne if the method assigned values to elements of resultOne in increasing order of string length. This is wrong as the method assigns the shortest string that occurs in any element of arr between arr[n] and arr[arr.length - 1], inclusive, to result[n]. The shortest string found between arr[0] and arr[3] is "of", so result[0] is assigned the value "of". The shortest string found between arr[1] and arr[3] is also "of", so result[1] is also assigned the value "of". The same is true for the part of the array that begins at index 2 and ends at index 3, so result[2] is also assigned the value "of". In the last iteration of the outer for loop, there are no values to consider after arr[3], so result[3] is assigned the value "spring".

Question 22

Consider the following method.

```
public static String[] strArrMethod(String[] arr)
{
  String[] result = new String[arr.length];
  for (int j = 0; j < arr.length; j++)
  {
    String sm = arr[j];
    for (int k = j + 1; k < arr.length; k++)
    {
      if (arr[k].length() < sm.length())
      {
        sm = arr[k]; // Line 12
      }
    }
    result[j] = sm;
  }
  return result;
}
```

Consider the following code segment.

```
String[] testTwo = {"last", "day", "of", "the", "school", "year"};
String[] resultTwo = strArrMethod(testTwo);
```

How many times is the line labeled `//` Line 12 in the `strArrMethod` executed as a result of executing the code segment?

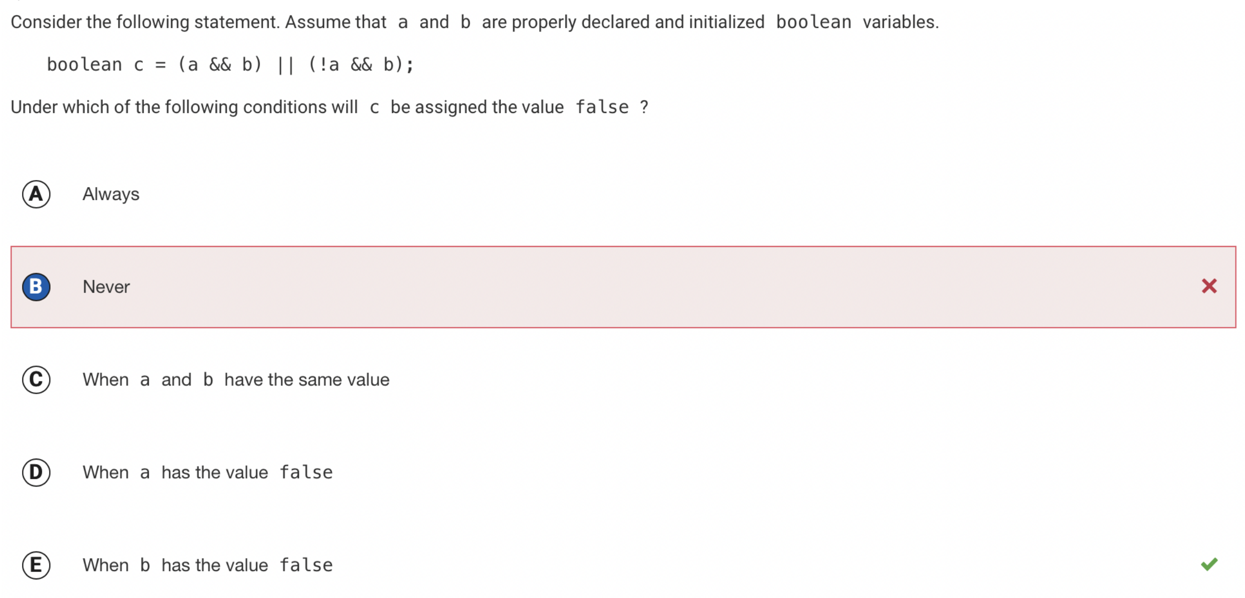
(A) 4 times ✓

(B) 5 times

[✓ (C) 6 times] ✗

Option C is wrong because line 12 was not executed for every element of arr; line 12 is executed each time the variable sm is updated because a new smallest value is found. When j has the value 0, sm is updated for "day" and "of". When j has the value 1, sm is updated for "of". When j has the value 4, sm is updated for "year". When j has any of the values 2, 3, or 5, sm is not updated. Line 12 is executed four times. Thus answer A is correct.

Question 27



Consider the following statement. Assume that a and b are properly declared and initialized `boolean` variables.

```
boolean c = (a && b) || (!a && b);
```

Under which of the following conditions will c be assigned the value `false` ?

(A) Always

[✓ (B)] Never ✗

(C) When a and b have the same value

(D) When a has the value `false`

(E) When b has the value `false` ✓

Option B is wrong because variable c will be assigned the value false when b has the value false, regardless of the value of a. When b has the value false, both of the expressions (a && b) and (!a && b) evaluate to false, regardless of the value of a. The entire expression evaluates to false || false, or false. When b has the value true, one of the expressions (a && b) or (!a && b) evaluates to true. The entire expression, in this case, is either true || false or false || true, or true. A truth table can be used to summarize these results.

Question 30

Consider the following class definitions.

```
public class Rectangle
{
  private int height;
  private int width;
  public Rectangle()
  {
    height = 1;
    width = 1;
  }
  public Rectangle(int x)
  {
    height = x;
    width = x;
  }
  public Rectangle(int h, int w)
  {
    height = h;
    width = w;
  }
  // There may be methods that are not shown.
}
public class Square extends Rectangle
{
  public Square(int x)
  {
    /* missing code */
  }
}
```

Which of the following code segments can replace `/* missing code */` so that the `Square` class constructor initializes the `Rectangle` class instance variables `height` and `width` to `x` ?

Option E is wrong because the code segment would result in a compiler error. The instance variables height and width are defined as private in the superclass and cannot be accessed directly from the subclass. The super class is needed as a call to the one-argument superclass constructor with the single parameter x will set both the height and the width instance variables to x.