1. Which of the following is dereferencing object o?
   a. Object o = new Object();
   b. o.setValue(8);
   c. field = o.fieldName;
   d. Object o = null;
   e. a and b
   f. b and c
   g. c and d
   h. All of the above
   i. None of the above

2. Which of the following is NOT an example of object dereferencing?
   b. o.getField();
   c. o = null;
   d. Object p = o;
   e. a and b
   f. b and c
   g. c and d
   h. All of the above
   i. None of the above

3. If you are going to dereference an object, it is best practice to:
   a. Check if the object is null prior to the dereference
   b. Ensure that the object is not null after dereferencing it
   c. Create a new instance of the object prior to dereferencing it
   d. Set all object attributes prior to dereferencing to retrieve any of them.
   e. a and b
   f. c and d
   g. c and b
   h. All of the above
   i. None of the above

4. The value null indicates:
   a. an empty object
   b. no object
   c. a new object
   d. an object with no attributes
   e. a and b
   f. a and c
   g. c and d
   h. All of the above
   i. None of the above

5. A NullPointerException is thrown when:
   a. a dereferenced object is null and should not be
   b. A piece of code is run with compilation errors
c. a **null** object is created and then dereferenced immediately after
d. More than one object is trying to access the same piece of information at the
   same time
e. a and c
f. b and d
g. a and d
h. All of the above
i. None of the above

6. The best way to avoid a `NullPointerException` is by:
   a. Using getter and setter methods to access object attributes/fields (i.e.
      `o.getField()`)  
   b. Make sure every method you implement either throws or catches
      `NullPointerException`
   c. Debug your program to make sure no objects are found to be **null** during
      execution
   d. Checking the object you are trying to access to make sure it is not **null**
      before completing any operations with the value (i.e. `if o != null`
      `{ ... }`

7. Say you have the following code:

```java
public Element getParagraphElement(int pos) {
    Element e;
    for (e = getDefaultRootElement(); ! e.isLeaf(); ) {
        int index = e.getElementIndex(pos);
        e = e.getElement(index);
    }
    if(e != null)
        return e.getParentElement();
    return e;
}
```

A static analysis tool gives the following warning on the code as written:

**A value is checked here to see whether it is null, but this value can't be null**
**because it was previously dereferenced and if it were null a null pointer**
**exception would have occurred at the earlier dereference. Essentially, this code**
**and the previous dereference disagree as to whether this value is allowed to be**
**null. Either the check is redundant or the previous dereference is erroneous.**
Upon executing your code, you do not get a NullPointerException. To test the accuracy of the notification, and prevent problems from manifesting later, you might:

a. Move the first dereferencing of e inside the null check at line 8
b. remove the if statement at line 8
c. add an if statement in to check if the value returned by getDefaultRootElement() is null before proceeding; this includes moving line 10 to preserve functionality
d. Move line 10 above the null check at line 8
e. a and b
f. b and c
g. c and d
h. All of the above
i. None of the above

8. Say you are calling a Java API method, getObject(), that returns object o, given o is not null. Inside the method that is calling getObject(), there is the following line of code:

```java
1    if (getObject() != null) {
2        return getObject() ... }
```

Running a static analyzer on the code produces the following notification:

This method contains a redundant check of a known non-null value against the constant null.

Which of the following is the simplest way to fix the problem, or prevent it from manifesting, without modifying the code for getObject():

a. remove the if statement at line 1
b. make sure the method you are writing throws a NullPointerException
c. create a new method that performs the same functionality as getObject() but does not check that
d. Nothing - the code is fine as it is.