

## Chapter 2 - Bag Implementations that use Arrays

### True/False (10)

1. Any methods that a core method might call are part of the core group.

Answer: true

2. You should implement all methods of an ADT implementation before testing to make testing easier.

Answer: false

3. You should never suppress compiler warnings.

Answer: false

4. When defining the bag class, you should implement the *isArrayFull* and *remove* operations first.

Answer: false

5. When defining the bag class write methods that have simple implementation first to get them out of the way.

Answer: false

6. A class should not return a reference to an array that is a private data field.

Answer: true

7. A final class is more secure than one that is not final.

Answer: true

8. Write stubs early in the implementation of an ADT so you can begin testing early.

Answer: true

9. When testing a method, you only need to check for arguments that lie within the legal range of their corresponding parameter.

Answer: false

10. When comparing objects in a bag, we assume that the class to which the objects belong defines its own version of *equals*.

Answer: true

### Short Answer (7)

1. It is a good practice to identify a group of core methods to implement and test before continuing with the rest of the class definition. What type of methods should you begin with?

Answer: You should begin with methods that add to the collection or methods that have complex implementations.

2. Why is it better to implement the add operation in a collection before implementing the remove operation?

Answer: You cannot test remove functionality until you have written and tested add functionality.

3. What is the difference between the *numberOfEntries* data field in the ArrayBag implementation and the DEFAULT\_CAPACITY field?

Answer: If the client does not specify the size of the bag, the *numberOfEntries* field will be set to the DEFAULT\_CAPACITY value in the constructor. Otherwise, the *numberOfEntries* field will be set to the size specified by the client.

4. Why is it a safer practice for the *toArray* method to return a copy of the array instead of a reference to the array?

Answer: A client would then have direct access to the private data and could either inadvertently or maliciously alter the data without using the public methods of the class.

5. What is fail-safe programming?

Answer: It is the practice of checking for anticipated errors in your program.

6. Why is it a good security practice to declare the ArrayBag to be a final class?

Answer: A programmer cannot use inheritance to change its behavior.

7. Why doesn't the *contains* method return the index of a located entry?

Answer: Returning the index implies an explicit position in the array. A client should be shielded from this level of detail.

Multiple Choice (30) WARNING: CORRECT ANSWERS ARE IN THE SAME POSITION AND TAGGED WITH \*\*. YOU SHOULD RANDOMIZE THE LOCATION OF THE CORRECT ANSWERS IN YOUR EXAM.

1. Which of the following methods is a good candidate for a core method:
  - a. `add()` \*\*

- b. `clear()`
  - c. `contains()`
  - d. `remove()`
2. Which of the following methods is a good candidate for a core method:
- a. `add()`
  - b. `toArray()`
  - c. `isArrayFull()`
  - d. all of the above \*\*
3. Which one of the following Java statements allocates an array in the bag constructor causing a compiler warning for an unchecked operation? Assume *capacity* is an integer.
- a. `bag = (T[ ]) new Object[capacity];` \*\*
  - b. `bag = new T[capacity];`
  - c. `bag = new Object[capacity];`
  - d. `bag = new (T[ ]) Object[capacity];`
4. Which instruction suppresses an unchecked-cast warning from the compiler?
- a. `@SuppressWarnings("unchecked")` \*\*
  - b. `@SuppressWarnings()`
  - c. `@SuppressWarnings()`
  - d. `@Warning("suppress unchecked")`
5. What are the consequences of returning a reference to the bag array in the `toArray` method?
- a. the return variable is an alias for the private instance array variable
  - b. the client will have direct access to the private instance array variable
  - c. the client could change the contents of the private instance array variable without using the public access methods
  - d. all of the above \*\*
6. Which one of the following is considered a safe and secure programming practice?
- a. validating input data and arguments to a method \*\*
  - b. identifying a group of core methods to implement first
  - c. using generic data types
  - d. none of the above
7. Which one of the following is considered a safe and secure programming practice?
- a. making no assumptions about the actions of clients and users \*\*
  - b. using `@SuppressWarnings("unchecked")`
  - c. adding the comments and headers of the public methods to the class by copying them from the interface

- d. all of the above
8. When implementing the bag ADT, which scenario could result in a security problem?
- a. a constructor throw an exception or error before completing its initialization \*\*
  - b. the programmer validates input data to a method
  - c. generics are used to restrict data types of the entries in the collection
  - d. a group of core methods is not defined
9. When implementing the bag ADT, which scenario could result in a security problem?
- a. a client attempts to create a bag whose capacity exceeds a given limit \*\*
  - b. a `SecurityException` is thrown in the constructor
  - c. an `IllegalStateException` is thrown in the constructor
  - d. the *delete* method is implemented before the *add* method
10. An incomplete definition of a method is called a \_\_\_\_.
- a. stub \*\*
  - b. core method
  - c. fail-safe method
  - d. security problem
11. A stub is created for what purpose?
- a. to avoid syntax errors \*\*
  - b. to avoid duplicate code
  - c. to protect the integrity of the ADT
  - d. to practice fail-safe programming
12. A stub should
- a. report that it was invoked by displaying a message
  - b. include a return statement that returns a dummy value
  - c. be written early for testing programs
  - d. all of the above \*\*
13. A test driver for the bad add method should check for which one of the following
- a. an over capacity condition \*\*
  - b. printing elements of the bag
  - c. adding elements of the correct type
  - d. an empty bag condition
14. The method *remove* that has no parameter in the bag implementation
- a. removes the last entry in the array \*\*
  - b. removes the first entry in the array
  - c. removes a random entry in the array

- d. none of the above
15. If a bag is empty before the *remove* method executes, it
- a. returns null \*\*
  - b. throw an exception
  - c. returns an error message
  - d. all of the above
16. Decrementing the private class variable *numberOfEntries* in a bag
- a. causes the last entry to be ignored \*\*
  - b. causes an *IllegalState* exception to be thrown
  - c. destroys the integrity of the bag container
  - d. is a security problem
17. In the *remove* method, setting the last entry of the array to null
- a. flags the removed object for garbage collection
  - b. prevents malicious code from accessing it
  - c. is a good security practice
  - d. all of the above \*\*
18. The *remove* method replaces the removed entry with null because
- a. the entry could potentially be scheduled for garbage collection \*\*
  - b. the client expects a null return value
  - c. it is a fail-safe programming practice
  - d. otherwise it could be considered a duplicate value
19. When calling the *remove* method with an argument if there are multiple entries
- a. exactly which occurrence removed is unspecified
  - b. only one occurrence is removed
  - c. the first occurrence is removed
  - d. all of the above \*\*
20. The most efficient approach to dealing with a gap left in an array after removing an entry from a bag is to
- a. replace the entry being removed with the last entry in the array and replace the last entry with null \*\*
  - b. replace the entry being removed with the first entry in the array and replace the first entry with null
  - c. shift subsequent entries and replace the duplicate reference to the last entry with null
  - d. replace the entry being removed with null

21. If an array bag contains the entries "lions", "elephants", "otters", "bears", "tigers", "lemurs" and a call to the *remove* method with the entry "bears" is made, what does the array look like after remove?
- a. ""lions", "elephants", "otters", "lemurs", "tigers", null \*\*
  - b. "lions", "elephants", "otters", null, "tigers", "lemurs"
  - c. "lions", "elephants", "otters", "tigers", "lemurs", null
  - d. "lions", "elephants", "otters", "tigers", "lemurs"
22. A return value of -1 from the *getIndex* method indicates
- a. the entry was not present in the bag \*\*
  - b. the entry was found at the end of the array
  - c. the bag was empty
  - d. the bag was full
23. A fixed size array
- a. has a limited capacity
  - b. can waste memory
  - c. prevents expansion when the bag becomes full
  - d. all of the above \*\*
24. In order to resize an array to accommodate a larger bag, you must
- a. define an alias that references the original array
  - b. create a new array that is larger than the original array and make the alias reference it
  - c. copy the contents of the original array reference by the alias to the new array and discard the original array
  - d. all of the above \*\*
25. When the need to expand the size of a bag occurs, the common practice is to
- a. double the size of the array \*\*
  - b. increase the size of the array by one to accommodate the new entry
  - c. use the Fibonacci sequence to determine the incremental size of the new array
  - d. prompt the user for how much larger the bag will need to be
26. The Java Class Library method to resize an array is called
- a. *Arrays.copyOf* \*\*
  - b. *Arrays.resize*
  - c. *Arrays.copy*
  - d. *Arrays.double*
27. When resizing an array to increase the bag size, if the copy exceeds the maximum memory specified for the computer, the *checkCapacity* method should
- a. throw an *IllegalStateException* \*\*

- b. throw a `MaxSizeExceededException`
- c. throw a `MaxMemoryExceededException`
- d. return false

28. In a `ResizableArrayBag` class, why does the *add* method always return true?

- a. to conform to the bag interface \*\*
- b. because the array will always double in size
- c. returning void is not a fail-safe programming practice
- d. all of the above

29. Which of the following is an advantage of using an array to implement the ADT bag?

- a. adding an entry to a bag is fast \*\*
- b. removing a particular entry requires time to locate the entry
- c. increasing the size of the array requires time to copy its entries
- d. the client has control over the size of the bag

30. Which of the following is a disadvantage of using an array to implement the ADT bag?

- a. increasing the size of the array requires time to copy its entries \*\*
- b. adding an entry to a bag is fast
- c. removing an unspecified entry from the array is fast
- d. all of the above