ANSWER KEY

1. E	8. A	15. D
2. D	9. D	16. C
3. E	10. A	17. B
4. E	11. A	18. E
5. B	12. D	19. C
6. C	13. D	20. A
7. C	14. D	21. E

ANSWERS EXPLAINED

1. (E) A programmer should never make unilateral decisions about a program specification. When in doubt, check with the person who wrote the specification.
2. (D) In I and II a three-digit number is the object being manipulated. For III, however, the object is a six-character string, which suggests a class other than a `ThreeDigitNumber`.
3. (E) Top-down programming consists of listing the methods for the main object and then using stepwise refinement to break each method into a list of subtasks. Eliminate choices A, C, and D: Top-down programming refers to the design and planning stage and does not involve any actual writing of code. Choice B is closer to the mark, but "top-down" implies a list of operations, not an essay describing the methods.
4. (E) All three considerations are valid when choosing an algorithm. III is especially important if your code will be part of a larger project created by several programmers. Yet even if you are the sole writer of a piece of software, be aware that your code may one day need to be modified by others.
5. (B) A process that causes excessive data movement is inefficient. Inserting an element into its correct (sorted) position involves moving elements to create a slot for this element. In the worst case, the new element must be inserted into the first slot, which involves moving every element up one slot. Similarly, deleting an element involves moving elements down a slot to close the "gap." In the worst case, where the first element is deleted, all elements in the array will need to be moved. Summing the five smallest elements in the list means summing the first five elements. This requires no testing of elements and no excessive data movement, so it is efficient. Finding the maximum value in a sorted list is very fast—just select the element at the appropriate end of the list.
6. (C) "Robustness" implies the ability to handle all data input by the user and to give correct answers even for extreme values of data. A program that is not robust may well run on another computer without modification, and a robust program may need modification before it can run on another computer.
7. (C) Eliminate choice D because 0 is an invalid weight, and you may infer from the method description that invalid data have already been screened out. Eliminate

choice E because it tests two values in the range 10–25. (This is not wrong, but choice C is better.) Eliminate choice A since it tests only the endpoint values. Eliminate B because it tests *no* endpoint values.

8. (A) The statement is syntactically correct, but as written it will not find the mean of the integers. The bug is therefore an intent or logic error. To execute as intended, the statement needs parentheses:

```
double average = (n1 + n2 + n3 + n4) / (double) 4;
```

9. (D) The error that occurs is a run-time error caused by an attempt to divide by zero (`ArithmeticException`). Don't be fooled by choice C. Simply reading an expression `8/0` from the input file won't cause the error. Note that if the operands were of type `double`, the correct answer would be E. In this case, dividing by zero does not cause an exception; it gives an answer of `Infinity`. Only on inspecting the output would it be clear that something was wrong.

10. (A) A precondition does not concern itself with the action of the method, the local variables, the algorithm, or the postcondition. Nor does it initialize the parameters. It simply asserts what must be true directly before execution of the method.

11. (A) The best case causes the fewest computer operations, and the worst case leads to the maximum number of operations. In the given algorithm, the initial test `if (a1 > a2)` and the assignment to `max` will occur irrespective of which value is the largest. The second test, `if (max < a3)`, will also always occur. The final statement, `max = a3`, will occur only if the largest value is in `a3`; thus, this represents the worst case. So the best case must have the biggest value in `a1` or `a2`.

12. (D) The precondition is an assertion about the variables in the loop just before the loop is executed. Variables `N`, `k`, and `sum` have all been initialized to the values shown in choice D. Choice C is wrong because `k` may equal `N`. Choice A is wrong because `k` may be less than `N`. Choice E is wrong because `mean` is not defined until the loop has been exited. Choice B is wrong because it omits the assertions about `N` and `k`.

13. (D) Eliminate choices A, B, and E since `i` is initialized to `3` in the `for` loop. Choice C is wrong because the value of `i` after final exit from the loop is `n+1`.

14. (D) `a` is being added to `total` `b` times, which means that at the end of execution `total = a*b`.

15. (D) It makes sense for an `Item` to be responsible for its name, unit price, quantity, and total price. It is *not* reasonable for it to be responsible for other `Items`. Since an `ItemList`, however, will contain information for all the `Items` purchased, it is reasonable to have it also compute the total `amountDue`. It makes just as much sense to give an `Invoice` the responsibility for displaying information for the items purchased, as well as providing a final total, `amountDue`.

16. (C) The *is-a* relationship defines inheritance, while the *has-a* relationship defines association. These types of relationship are mutually exclusive. For example, a graduate student *is-a* student. It doesn't make sense to say a student *has-a* graduate student!

17. (B) Even though it's convenient for a `Tire` object to inherit `Circle` methods, an inheritance relationship between a `Tire` and a `Circle` is incorrect: It is false to say

that a `Tire` *is-a* `Circle`. A `Tire` is a car part, while a `Circle` is a geometric shape. Notice that there is an *association* relationship between a `Tire` and a `Circle`: A `Tire` *has-a* `Circle` as its boundary.

18. (E) Independent classes do not have relationships with other classes and can therefore be more easily coded and tested.

19. (C) The word "program" is never included when it's used in this context. The word "integer" describes the type of coordinates x and y and has no further use in the specification. While words like "direction," "boundary," and "simulation" may later be removed from consideration as classes, it is not unreasonable to keep them as candidates while you ponder the design.

20. (A) A `GridPoint` object knows only its x and y coordinates. It has no information about whether a `BumperCar` is at that point. Notice that operations in all of the other choices depend on the x and y coordinates of a `GridPoint` object. An `isEmpty` method should be the responsibility of the `Grid` class that keeps track of the status of each position in the grid.

21. (E) A `BumperCar` is responsible for itself—keeping track of its own position, selecting an initial direction, making a move, and reversing direction. It is not, however, responsible for maintaining and updating the grid. That should be done by the `Grid` class.