Answer

1. (E)
   The program will compile. The compiler can figure out that the local variable price will be initialized, since the value of the condition in the if statement is true. The two instance variables and the two static variables are all initialized to the respective default value of their type.

2. (B)
   Only (B) is a valid method declaration. Methods must specify a return type or must be declared void. This makes (D) and (E) invalid. Methods must specify a list of zero or more comma-separated parameters enclosed by parentheses, ( ). The keyword void cannot be used to specify an empty parameter list. This makes (A) and (C) invalid.

3. (A) and (D)
   The first and the third pairs of methods will compile. The second pair of methods will not compile, since their method signatures do not differ. The compiler has no way of differentiating between the two methods. Note that the return type and the names of the parameters are not a part of the method signatures. Both methods in the first pair are named fly and, therefore, overload this method name. The methods in the last pair do not overload the method name glide, since only one method has that name. The method named Glide is distinct from the method named glide, as identifiers are case-sensitive in Java.

4. (B) and (E)
   A constructor can be declared private, but this means that this constructor can only be used within the class. Constructors need not initialize all the fields when a class is instanstiated. A field will be assigned a default value if not explicitly initialized. A constructor is non-static and, as such, it can directly access both the static and non-static members of the class.

5. (B)
   Evaluation of the actual parameter i++ yields 0, and increments i to 1 in the process. The value 0 is copied into the formal parameter i of the method addTwo() during method invocation. However, the formal parameter is local to the method, and changing its value does not affect the value in the actual parameter. The value of the variable i in the main() method remains 1.

6. (B) and (E)
   The size of the array cannot be specified, as in (B) and (E). The size of the array is given implicitly by the initialization code. The size of the array is never specified in the declaration of an array reference. The size of an array is always associated with the array instance (on the right-hand side), not the array reference (on the left-hand side).

7. (D)
   The variables `a` and `b` are local variables that contain primitive values. When these variables are passed as arguments to another method, the method receives copies of the primitive values in the variables. The actual variables are unaffected by operations performed on the copies of the primitive values within the called method. The variable `bArr` contains a reference value that denotes an array object containing primitive values. When the variable is passed as a parameter to another method, the method receives a copy of the reference value. Using this reference value, the method can manipulate the object that the reference value denotes. This allows the elements in the array object referenced by `bArr` to be accessed and modified in the method `inc2()`.

8. (D)
   The length of the array passed to the `main()` method is equal to the number of program arguments specified in the command line. Unlike some other programming languages, the element at index 0 does not

contain the name of the program. The first argument given is retrieved using `args[0]`, and the last argument given is retrieved using `args[args.length-1]`.

9. (C) and (D)
A class or interface name can be referred to by using either its fully qualified name or its simple name. Using the fully qualified name will always work, but in order to use the simple name it has to be imported. By importing `net.basemaster.*` all the type names from the package `net.basemaster` will be imported and can now be referred to using simple names. Importing `net.*` will not import the subpackage `basemaster`.

10. (B)
Remember that a constructor does not have a return type; if a return type is provided, it is treated as a method in that class. In this case, since `Color` had `void` return type, it became a method named `Color()` in the `Color` class, with the default `Color` constructor provided by the compiler. By default, data values are initialized to zero, hence the output.

11. (A)
The compiler looks for the method `Color()` when it reaches this statement: `Color(10, 10, 10);`. The right way to call another constructor is to use the this keyword as follows: `this(10, 10, 10);`.

12. (C)
The `toString()` implementation has the expression `"The color is: " + red + blue + green`. Since the first entry is string, the + operation becomes the string concatenation operator with resulting string `"The color is: 10"`. Following that, again there is a concatenation operator + and so on until finally it prints `"The color is: 101010"`.

13. (A)
No access modifier is specified for the `toString()` method. Object's `toString()` method has a public access modifier; you cannot reduce the visibility of the method. Hence, it will result in a compiler error.