**QUESTION 1**

1. A search for an item *X* in an array starts at the lower end of the array, and then looks for *X* by comparing array items to *X* in order of increasing subscript. Such a method is called

|  |  |  |
| --- | --- | --- |
|  |  | selection search |
|  |  | lower to upper search |
|  |  | sequential search |
|  |  | binary search |

**QUESTION 2**

1. Constraining a type parameter in a generic class

|  |  |  |
| --- | --- | --- |
|  |  | can only be used when the generic class will be used as a superclass for other classes |
|  |  | causes programs to compile faster |
|  |  | restricts the types that can be used as type arguments |
|  |  | was added to Java in version 1.3 of the language |

**QUESTION 3**

1. In a string that contains a series of words or other items of data separated by spaces or other characters, the programming term for the spaces or other characters is

|  |  |  |
| --- | --- | --- |
|  |  | Delimiter |
|  |  | Separator |
|  |  | Token |
|  |  | Buffer |

**QUESTION 4**

1. In the following statement, which is the interface?  
     
   public class ClassA extends ClassB implements ClassC

|  |  |  |
| --- | --- | --- |
|  |  | ClassA |
|  |  | ClassB |
|  |  | ClassC |
|  |  | Cannot tell |

**QUESTION 5**

1. Look at the following statement.  
     
   import java.util.\*;  
     
   This is an example of

|  |  |  |
| --- | --- | --- |
|  |  | a wildcard import |
|  |  | conditional import |
|  |  | unconditional import |
|  |  | an explicit import |

**QUESTION 6**

1. The scope of a private instance field is

|  |  |  |
| --- | --- | --- |
|  |  | Inside the class, but not inside any method |
|  |  | The instance methods of the same class |
|  |  | The method in which they are defined |
|  |  | Inside the parentheses of a method header |

**QUESTION 7**

1. Under Windows, which of the following statements will open the file InputFile.txt that is in the root directory on the C: drive?

|  |  |  |
| --- | --- | --- |
|  |  | FileReader freader = new FileReader("C:\InputFile.txt"); |
|  |  | FileReader freader = new FileReader("/c/InputFile.txt"); |
|  |  | FileReader freader = new FileReader("C:\InputFile\txt"); |
|  |  | FileReader freader = new FileReader("C:\\InputFile.txt"); |

**QUESTION 8**

1. What will be the value of x[1] after the following code is executed?  
     
   int[] x = { 22, 33, 44 };  
   arrayProcess(x);  
   ...  
   public static void arrayProcess(int[] a)  
   {  
     for(int k = 0; k < 3; k++)  
     {  
     a[k] = a[k] + 5;  
     }  
   }

|  |  |  |
| --- | --- | --- |
|  |  | 49 |
|  |  | 33 |
|  |  | 27 |
|  |  | 38 |

**QUESTION 9**

1. What would be the results of the following code?  
     
   final int SIZE = 25;  
   int[] array1 = new int[SIZE];  
   ... // Code that will put values in array1  
   int value = 0;  
   for (int a = 0; a <= array1.length; a++)  
   {  
     value += array1[a];  
   }

|  |  |  |
| --- | --- | --- |
|  |  | value contains the sum of all the values in array1 |
|  |  | value contains the lowest value in array1 |
|  |  | value contains the highest value in array1 |
|  |  | This would cause the program to crash |

**QUESTION 10**

1. When an array is passed to a method

|  |  |  |
| --- | --- | --- |
|  |  | The method has direct access to the original array |
|  |  | A reference to the array is passed |
|  |  | It is passed just as an object |
|  |  | All of the above |

**QUESTION 11**

1. Which of the following for loops is valid, given the following declaration?  
     
   String[] names = {"abc", "def", "ghi", "jkl"};

|  |  |  |
| --- | --- | --- |
|  |  | for (int i = 0; i < names.length; i++)   System.out.println(names[i].length()); |
|  |  | for (int i = 0; i < names.length(); i++)   System.out.println(names[i].length); |
|  |  | for (int i = 0; i < names.length; i++)   System.out.println(names[i].length); |
|  |  | for (int i = 0; i < names.length(); i++)   System.out.println(names[i].length()); |

**QUESTION 12**

1. Which of the following is the correct boolean expression to test for:  int x being a value between, but not including, 500 and 650, or int y not equal to 1000?

|  |  |  |
| --- | --- | --- |
|  |  | ((x >= 500 && x <= 650) && (y != 1000)) |
|  |  | ((x > 500 && x < 650) || (y != 1000)) |
|  |  | ((x < 500 && x > 650) || !(y == 1000)) |
|  |  | ((x > 500 AND x < 650) OR !(y.equal(1000))) |

**QUESTION 13**

1. A protected member of a class may be directly accessed by

|  |  |  |
| --- | --- | --- |
|  |  | Methods of the same class |
|  |  | Methods of a subclass |
|  |  | Methods in the same package |
|  |  | All of these |

**QUESTION 14**

1. A subclass class can directly access

|  |  |  |
| --- | --- | --- |
|  |  | All members of the superclass class |
|  |  | Only public and private members of the superclass class |
|  |  | Only protected and private members of the superclass class |
|  |  | Only public and protected members of the superclass class |

**QUESTION 15**

1. Every class has a toString method and an equal's method inherited from the Object class.

 True

 False

**QUESTION 16**

1. If a subclass constructor does not explicitly call a superclass constructor,

|  |  |  |
| --- | --- | --- |
|  |  | It must include the code necessary to initialize the superclass fields |
|  |  | The superclass fields will be set to the default values for their data types |
|  |  | Java will automatically call the superclass's default constructor immediately after the code in the subclass's constructor executes |
|  |  | Java will automatically call the superclass's default constructor just before the code in the subclass's constructor executes |

**QUESTION 17**

1. If a superclass does not have a default constructor,

|  |  |  |
| --- | --- | --- |
|  |  | Then a class that inherits from it, must initialize the superclass values |
|  |  | Then a class that inherits from it, must call one of the constructors that the superclass does have |
|  |  | Then a class that inherits from it, does not inherit the data member fields from the superclass |
|  |  | Then a class that inherits from it, must contain the default constructor for the superclass |

**QUESTION 18**

1. If two methods in the same class have the same name but different signatures, the second overrides the first.

 True

 False

**QUESTION 19**

1. An exception's default error message can be retrieved using this method.

|  |  |  |
| --- | --- | --- |
|  |  | getMessage() |
|  |  | getErrorMessage() |
|  |  | getDefaultMessage() |
|  |  | getDefaultErrorMessage() |

**QUESTION 20**

1. If a method does not handle a possible checked exception, what must the method have?

|  |  |  |
| --- | --- | --- |
|  |  | A catch clause in its header |
|  |  | A try/catch clause in its header |
|  |  | A try clause in its header |
|  |  | A throws clause in its header |

**QUESTION 21**

1. In a catch statement, what does the following code do?  
     
   System.out.println(e.getMessage());

|  |  |  |
| --- | --- | --- |
|  |  | It prints the stack trace |
|  |  | It prints the error message for an exception |
|  |  | It prints the code that caused the exception |
|  |  | It overrides the toString method |

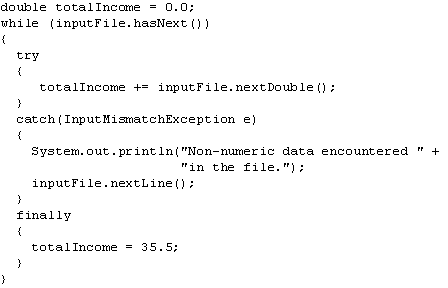
**QUESTION 22**

1. In a try/catch construct, after the catch statement is executed

|  |  |  |
| --- | --- | --- |
|  |  | The program returns to the statement following the statement in which the exception occurred |
|  |  | The program terminates |
|  |  | The program resumes at the statement that immediately follows the try/catch construct |
|  |  | The program resumes at the first statement of the try statement |

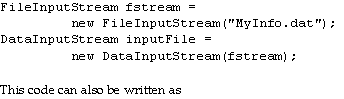
**QUESTION 23**

1. In the following code, assume that inputFile references a Scanner object that has been successfully used to open a file:

  
What will be the value of totalIncome after the following values are read from the file?   
2.5  
8.5  
3.0  
5.5  
abc  
1.0

|  |  |  |
| --- | --- | --- |
|  |  | 19.5 |
|  |  | 0.0 |
|  |  | 35.5 |
|  |  | 75.0 |

**QUESTION 24**

1. Look at the following code:  
   

DataInputStream inputFile =  
new DataInputStream(new FileInputStream("MyInfo.dat"));

**QUESTION 25**

1. The catch clause

|  |  |  |
| --- | --- | --- |
|  |  | follows the try clause |
|  |  | starts with the word catch followed by a parameter list in parentheses containing an *ExceptionType* parameter variable |
|  |  | contains code to gracefully handle the exception type listed in the parameter list |
|  |  | All of these |