Quiz on Chapter 4

**Question 1:**

A "black box" has an interface and an implementation. Explain what is meant by the terms *interface* and *implementation*.

* A black box's interface is its link to the outside world, such as the name and parameters of a subroutine or the dial for adjusting the temperature on a thermostat. The implementation refers to the black box's interior workings. To utilise the black box, you must grasp its interface but have no knowledge of its implementation.

**Question 2:**

A subroutine is said to have a *contract*. What is meant by the contract of a subroutine? When you want to use a subroutine, why is it important to understand its contract? The contract has both "syntactic" and "semantic" aspects. What is the syntactic aspect? What is the semantic aspect?

* A subroutine's contract specifies what must be done to appropriately call the subroutine and what it will do when called. In summary, it contains everything a programmer needs to know about the function in order to properly utilise it. (It does not include the subroutine's "insides" or implementation.)
* The name of the subroutine, the number of parameters, and the type of each parameter are all syntactic components of a subroutine's contract. This is the information required to create a compilable subroutine call statement. The contract's semantic component describes the meaning of the subroutine, that is, the work that the subroutine does. It may also describe what argument values the subroutine can appropriately handle. The semantic component isn't included in the application. It is commonly stated through remarks.

**Question 3:**

Briefly explain how subroutines can be useful in the top-down design of programs.

* Starting with the overall problem to be solved and breaking it down into smaller problems that can be tackled independently is referred to as top-down thinking. You may simply create a subroutine to address each of the minor difficulties when building a programme to solve the problem. Then you may develop and test each function individually.

**Question 4:**

Discuss the concept of *parameters.* What are parameters for? What is the difference between *formal parameters* and *actual parameters*?

* Parameters are used to communicate between a subroutine and the programme section that invokes the procedure. If you think of a subroutine as a black box, then parameters represent the interface to that black box. The subroutine definition has formal parameters. Subroutine call statements include actual arguments. Before the body of the subroutine is run, the values of the real parameters are assigned to the formal parameters when the subroutine is called.

**Question 5:**

Give two different reasons for using named constants (declared with the final modifier).

* A constant has a meaningful name, which makes the program easier to read. It's easier to understand what a name like INTEREST\_RATE is for than it is to figure out how a literal number like 0.07 is being used.
* A second reason for using named constants is that it's easy to modify the value of the constant if that becomes necessary. If a literal value is used throughout the program, the programmer has to track down each occurrence of the value and change it. When a constant is used correctly, it is only necessary to change the value assigned to the constant at one point in the program.
* A third reason is that using the final modifier protects the value of a variable from being changed. This is especially important for member variables that are accessible from outside the class where they are declared.

**Question 6:**

What is an API? Give an example.

* An API is an Applications Programming Interface. It is the interface to a "toolbox" of subroutines that someone has written. It tells you what routines are available, how to call them, and what they do, but it does not tell you how the subroutines are implemented. An example is the standard Java API which describes the interfaces of all the subroutines in all the classes that are available in such packages as java.lang and java.util.

**Question 7:**

What might the following expression mean in a program?

(a,b) -> a\*a + b\*b + 1

* The operator "->" means that this is a "lambda expression," that is, an anonymous function. There must be a functional interface that specifies the type of this lambda expression. It might, for example, be a functional interface that defines a function double f(double x, double y). In any case, the function must take two numeric parameters (since the expression a\*a+b\*b+1 would only be defined when *a* and *b* are numeric), and it must return a numeric value. (Note that this expression can only be used in a context where the compiler can deduce which functional interface applies.)

**Question 8:**

Suppose that *SupplyInt* is a functional interface that defines the method public int get(). Write a lambda expression of type *SupplyInt* that gets a random integer in the range 1 to 6 inclusive. Write another lambda expression of type *SupplyInt* that gets an int by asking the user to enter an integer and then returning the user's response.

A lambda expression of type *SupplyInt* has no parameters and must compute and return a value of type int. For the first expression, the int value can be given as (int)(1+Math.random()\*6), so the lambda expression takes the form

() -> (int)(1+Math.random()\*6)

For the second lambda expression, at least two statements are needed, one to ask the user for an integer and one to read the response. The two statements must be enclosed between braces, and a return is needed to return the integer:

() -> {

System.out.print( "Please enter an integer: " );

return TextIO.getlnInt(); // Read an integer and return it.

}

(This assumes that *TextIO* is available to the program.)

**Question 9:**

Write a subroutine named "stars" that will output a line of stars to standard output. (A star is the character "\*".) The number of stars should be given as a parameter to the subroutine. Use a *for* loop. For example, the command "stars(20)" would output

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

The subroutine could be written as follows:

static void stars(int numberOfStars) {

// output a line containing the specified number of stars

for (int i = 0; i < numberOfStars; i++) {

System.out.print('\*');

}

System.out.println(); // output carriage return after the \*'s

}

**Question 10:**

Write a main() routine that uses the subroutine that you wrote for Question 7 to output 10 lines of stars with 1 star in the first line, 2 stars in the second line, and so on, as shown below.

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*

The main() routine can use a for loop that calls the stars() subroutine ten times, once to produce each line of output. (An occasional beginner's mistake in this problem is to rewrite the body of the subroutine inside the main() routine, instead of just calling it by name.) Here is the main routine—which would, of course, have to be put together with the subroutine in a class in order to be used.

public static void main(String[] args) {

int line; // Line number, and also the number of stars on that line.

for ( line = 1; line <= 10; line++ ) {

stars( line );

}

}

**Question 11:**

Write a function named countChars that has a *String* and a char as parameters. The function should count the number of times the character occurs in the string, and it should return the result as the value of the function.

The returned value will be of type int. The function simply uses a for loop to look at each character in the string. When the character in the string matches the parameter value, it is counted.

static int countChars( String str, char searchChar ) {

// Count the number of times searchChar occurs in

// str and return the result.

int i; // A position in the string, str.

char ch; // A character in the string.

int count; // Number of times searchChar has been found in str.

count = 0;

for ( i = 0; i < str.length(); i++ ) {

ch = str.charAt(i); // Get the i-th character in str.

if ( ch == searchChar )

count++;

}

return count;

}

**Question 12:**

Write a subroutine with three parameters of type *int.* The subroutine should determine which of its parameters is smallest. The value of the smallest parameter should be returned as the value of the subroutine.

I'll call the subroutine smallest and the three parameters x, y, and z. The value returned by the subroutine has to be either x or y or z. The answer will be x if x is less than or equal to both y and z. The correct syntax for checking this is "if (x <= y && x <= z)". Similarly for y. The only other remaining possibility is z, so there is no necessity for making any further test before returning z. (In fact, doing so would be an error in Java, since with no "else" clause in the if statement, the compiler cannot determine that the function definitely returns a value in all possible cases.)

static int smallest(int x, int y, int z) {

if (x <= y && x <= z) {

return x;

}

else if (y <= x && y <= z) {

return y;

}

else

return z;

}

Note: Since a return statement causes the computer to terminate the execution of a subroutine anyway, this could also be written as follows, without the elses:

static int smallest(int x, int y, int z) {

if (x <= y && x <= z) {

return x;

}

if (y <= x && y <= z) {

return y;

}

return z;

}

**Question 13:**

Write a function that finds the average of the first N elements of an array of type double. The array and N are parameters to the subroutine.

Note that the array must be passed as a parameter of type double[], and that the value returned by the function will be a double. For the value of N to make sense, it should be in the range 1 up to the length of the array. My answer throws an *IllegalArgumentException* if N is not in this range:

static double average( double[] numbers, int N ) {

if ( N < 1 ) {

throw new IllegalArgumentException("Can't find an average of " +

N + " numbers.");

}

if ( N > numbers.length ) {

throw new IllegalArgumentsExcpetion( N +

" is more than the length of the array." );

}

double sum = 0; // the sum of the N numbers

for ( int i = 0; i < N; i++ ) {

sum = sum + numbers[i]; // add the i-th number to the sum

}

return sum/N; // Return the average as the value of the function.

}

**Question 14:**

Explain the purpose of the following function, and explain how it works:

static int[] stripZeros( int[] list ) {

int count = 0;

for (int i = 0; i < list.length; i++) {

if ( list[i] != 0 )

count++;

}

int[] newList;

newList = new int[count];

int j = 0;

for (int i = 0; i < list.length; i++) {

if ( list[i] != 0 ) {

newList[j] = list[i];

j++;

}

}

return newList;

}

This function makes a copy of its parameter, except that it leaves out all the elements of list that are equal to zero. It builds a new array that contains all the non-zero elements of list, and it returns that array as the value of the function. (Note that this is an example of using an array type as the return type of a function.)

The function creates a new array to be the return value. But to do that, it must know how long to make the array. The first five lines of the function definition count the number of non-zero elements in list. This is how many spaces we need in the new array, so count is used as the length when the new array is created. The remainder of the function goes through the original list and copies elements into newList. An element is copied only if it is non-zero. We have to keep track of how many spaces in newList have been filled so far. That's what j is for.