Lab 6 – Small Int

This assignment asks you to write a Java class called SmallInt (similar to a wrapper class). It also asks you to create a testing class called TestSmallInt. You will need to turn in both for your submission.

EDIT: SEE BELOW FOR OPTIONAL EXTRA CREDIT

Design and Implementation

SmallInt contains the following:

- A private instance variable of type int named **value**, which is the value of an integer stored by the class constructor.

- A public static constant named **MAXVALUE**. MAXVALUE should be used as an upper limit for the values SmallInt can represent. You should set MAXVALUE to 1000. However, keep in mind, the grader may change the value of MAXVALUE when testing your program.

- A **constructor** that assigns an integer between 0 and MAXVALUE (inclusive) to value. That's all the constructor has to do, using the int parameter that is passed to it. However, the constructor must check to make sure the parameter value is between 0 and MAXVALUE. If it is not, it should print an informative error message and store the number 0 instead. (hint: using a method to check the value would be a good idea, because you might need to do a similar check elsewhere!)

- An instance getter method **getDec** (returns a String, no parameters) which will return a String representation of the decimal integer. The easiest way to create this String is to concatenate the int representation to an empty String.

- An instance setter method **setDec** (returns void, accepts one int parameter) which assigns an integer between 0 and MAXVALUE to value. However, this setter must check to make sure the parameter value is between 0 and MAXVALUE. If it is not, it should print an informative error message and store the number 0 instead.

- An instance getter method **getBin** (returns a String, takes no parameters) should take the number stored in the instance variable value, generate the corresponding binary string and return that string. This method bears a little more explanation, since it is one of the more substantial parts of the assignment. The generation of the "corresponding binary string" is something that you must accomplish yourself, *without using any convenient methods or classes provided by Java's main library or any external libraries written in Java*. If you happen to be familiar with bit shifting, that approach is OK but you must be sure that you can explain it in your own words (see the directions about comments later).

  There is a nice algorithm for determining the binary representation of an integer. Let's remember that all integers can be expressed as powers of two:

  7 = 2^0 + 2^1 + 2^2 = 1 + 2 + 4 = 7 —> 111
  83 = 2^0 + 2^1 + 2^4 + 2^6 = 1 + 2 + 16 + 64 = 83 —> 1010011

  But how will we know which powers of two to use? You can figure this out manually by seeing how many times you can divide by two (i.e., repeated division). It's simple and a little naive, but it works. The implementation of this algorithm is left to you. It is highly recommended that you work out a few examples on paper before you begin to code. As a hint, remember what pieces of information mod can give you and integer division can give you. Also, note that there are zeroes in particular places. Where? In the places where the powers of 2 are not used. We could re-write the expression of 83 in the following way (the new parts have been highlighted for you):

  2^0 * 1 + 2^1 * 1 + **2^2 * 0** + **2^3 * 0** + 2^4 * 1 + **2^5 * 0** + 2^6 * 1

  Note that the expression above can be simplified into the one presented before. Still, it is important to see the full sequence since it tells us how to create the binary sequence for 83. Read the 0's and 1's from left to right to see how. In your code, you will be expected to have a block of comments explaining how your algorithm works and

why it's correct. You should provide a specific example of a number (try not to choose any of the numbers above or any number that your classmates might have mentioned).

- An instance getter method **getHex** (returns a String, takes no parameters) should take the number stored in the instance variable value, generate the corresponding hexadecimal string and return that string. Just like with getBin, your implementation must consist of only your work.

  Hexadecimal expresses values in terms of different powers of a number, which in this case is 16:

  7 = 16^0 * 7 = 1 * 7 = 7 —> 0x7
  83 = 16^0 * 3 + 16^1 * 5 = 83 —> 0x53

  Here we want to note the values of the constants next to the powers of 16, since those constants give us the correct hex values: 7 and 53, respectively.

  We can use the same algorithm from before (i.e., repeated division with % and /) but choose 16 as the divisor this time. The process should be exactly the same, except now you have to deal with remainder values that will not just be 0 and 1. The values will range from 0-15, but note that this corresponds to the range of hex characters (0-9, A-F, 16 in total).

  If the remainder is less than 10, you can concatenate the remainder as it is to your hexadecimal string. But if the remainder is more than 10, you need to convert the value to the corresponding hexadecimal symbol (A-F). You could create a mapping of values 10-15 to A-F, but there's a handy type cast that you might find even easier:

  ```
  hexString = (char) ('A' + remainder - 10) + hexString;
  //where hexString is the string to be returned by the method
  ```

  As with getBin(), you will need to provide comments with an explanation of how your algorithm works with a specific example.

- Finally, you need to write a second public class (that means you will need two .java files) called TestSmallInt. TestSmallInt will have just one main() method. It will prompt the user (using Scanner) for a number in the range 0 – MAXVALUE (inclusive). You should get MAXVALUE from the SmallInt class. Remember that you can access this variable from the class itself since it is static (e.g., SmallInt.MAXVALUE).

  Then the main() method will simply instantiate a SmallInt object using SmallInt's constructor and print the decimal representation, the binary representation and the hexadecimal representation of the number entered by the user. There is no need to loop until the user decides to quit.

****EXTRA CREDIT****

The following is an OPTIONAL addition to the assignment. You will not lost points if you do not implement this method. At most, you can earn two (2) additional points by completing the following:

- A static method **sameValue** (returns a boolean, accepts two String parameters) should take a binary representation of an int and a hexadecimal representation of an int, convert the bin to hex OR convert the hex to bin, then compare the Strings to see if they are equal. The result of this comparison is returned. Some examples are provided below.

  sameValue ("101", "5") would return true
  sameValue ("101", "6") would return false
  sameValue ("1101", "D") would return true
  sameValue ("1101", "5") would return false
  sameValue ("11010101", "D5") would return true
  sameValue ("1011101", "5D") would return true

Note: In the first two and the last example you can add leading zeros to the binary String if it makes it easier for you.

You must use the String equals( ) method to test string equality (not ==). As an example:

```
binAsHex.equals(hex)
hexAsBin.equals(bin)
```

Grading Rubric

Points	Criteria
3	Correctly creates the SmallInt class, such that SmallInt objects can be instantiated and methods can be called on those objects successfully
3	Implements getBin() correctly, with an adequate explanation of how it works
3	Implements getHex() correctly, with an adequate explanation of how it works
1	Good style
(2)	**Implements sameValue() correctly [NOT REQUIRED]**