Exam 2 Practice Problems

1. Recursion

> A child couldn't sleep, so her mother told a story about a little frog,
> who couldn't sleep, so the frog's mother told a story about a little bear,
> who couldn't sleep, so bear's mother told a story about a little weasel
> ...who fell asleep.
> ...and the little bear fell asleep;
> ...and the little frog fell asleep;
> ...and the child fell asleep.

The String class contains a function named `substring`. For example, if x is a `String`, then `x.substring( 3 )` returns the substring starting in position 3. Positions in a string start counting at zero; consider the following example:

```
String x = new String( "Hello" ) ;
System.out.println( x.substring(1) ) ;
```

will output the string: `ello`

Your task: Write a **recursive** static method that will reverse an input string. An outline of the solution is given below:

```
class main {

    static String reverse( String s )
    {
         // Your implementation here.
    }

    public static void main ( String [] args )
    {
        if ( args.length == 1 ) {
           System.out.println( reverse( args[0] ) ) ;
        }
    }
}
```

Answer:

```
    static String reverse( String s )
    {
       if ( s.length() == 1 ) return s ;
       else {
          return reverse( s.substring(1) ) + s.charAt(0)  ;
       }
    }
```

2. (A much easier recursion problem). Write a **recursive** static Java method to input an odd number `N`, and compute the sum $1 + 3 + 5 + 7 + ... + N$. You may assume that `N` is odd and there is no need to check if it is odd.

   Answer:

```
static int oddsum( int N )
{
   if ( N == 1 ) return 1 ;
   else {
      return N + oddsum( N-2 ) ;
   }
}
```

3. (Program logic) A partial implementation of a list of integers is shown below.

   Your task: Complete the constructor and complete the method `append`

```
class list {
    int allocated_size ;   //  Size of the array when created.
    int n ;                //  Number of elements currently in the array.
    int [] A  ;            //  Array of integers.

    list( int size )       //  Constructor
    {
        //  Your implementation to initialize a list object.
    }
    void append( int x )
    {
        //  Your implementation to add a new integer 'x' to the end
        //  of the list.  Be sure to check if the array is full.
    }
} // end class
```

   Answer:

```
class list {
    int allocated_size ;   //  Size of the array when created.
    int n ;                //  Number of elements currently in the array.
    int [] A  ;            //  Array of integers.

    list( int size )       //  Constructor
    {
        //  Your implementation to initialize a list object.
        allocated_size = size ;
        n = 0 ;
        A = new int[ size ] ;
    }
```

```
    void append( int x )
    {
        //  Your implementation to add a new integer 'x' to the end
        //  of the list.  Be sure to check if the array is full.
        if ( n == allocated_size ) {
            System.err.println( "append():  Array is full error." ) ;
            System.exit(1) ;
         }
         else {
            A[n] = x ; n++ ;
         }
    }
} // end class
```

4. True or false: A non-static method may access both static and non-static data members in a Java object.

   Answer: True.

5. True or false: A static method may access both static and non-static data members in a Java object.

   Answer: False.

6. What does the phrase "overloading a method" mean?

   Answer: A Java method may have several versions with the same name, but different input parameters. Example:

```
// Method to add two integers.
int sum( int a, int b )
{
   return a + b ;
}

// Method to add three integers.
int sum( int a, int b, int c )
{
   return a + b + c ;
}
```

   A method with more than one version (distinguished by its parameter list) is called an *overloaded* method.

7. What is the difference between a static and non-static context?

   Answer: A static context exists during the execution of a static method. Usually, there is no object associated with the call to a static method; non-static data can not be accessed from a static context. A non-static context exists during the execution of a non-static method. There is always an object associated with a call to a non-static method. In a non-static context, non-static data members contained by the object come into scope.

8. What does the keyword "`this`" do in Java?

   Answer: The keyword "`this`" can not be used in a static context. In a non-static context, the keyword "`this`" is a reference to the object on which the currently executing method was called.

9. How does one create a data member in a class and assign it a value that can not be changed later in the program?

   Answer: Using the keyword `final`. Example:

```
final int fred_flintstone = 71 ; // Fred's secret code number
                                 // in the 'Loyal Order of the
                                 // Water Buffalo'.
```

10. Describe how a method can be written to accept parameters.

    Answer: The name of the method is followed by:

    - a left parenthesis, '`(`'
    - a list of zero or more comma-separated input, output, or in/out parameters. Each parameter is of the form: `data_type  formal_parameter_name`
    - a matching right parenthesis, '`)`'

11. What is the difference between actual parameters and formal parameters?

    Answer: Formal parameters are placeholders used in the header of a method definition. In the following example, `A` and `B` are formal parameters.

```
int sum( int A, int B )
{
   return A + B ;
}
```

    Actual parameters are the values that are given to the method when it is called. Actual parameters may be variables or constants. For example;

```
public static void main( String [] args )
{
   int x = 7 ;
   int y = sum( x, 11 ) ;
}
```

    In the example above, the actual parameters are `x` and `11`.

12. Under what circumstances should the following type of variables be used? Discuss.

    - local variables
    - method input parameters
    - object data members

Answer:

Using Variables

(a) If a data value is persistent, (i.e., used later in the program) then the data should usually be stored as a data item belonging to that object (or class).

(b) If a data value represents information that a method needs, and that data will change from one use of the method to the next use, then the data should be an input parameter to the method.

Example: In Lab 4, class `wordlist` has a method named `update`. Each `update` operation will be done with a different word. A good design is to write method `update` so that it accepts a word as an input parameter.

(c) If a data value is temporary, it should be stored using a local variable. Local variables exist only while the method containing them is running. When a method returns, its local variables no longer exist.

Example: Loop indices should be local. E.g.,

```
void simple_method()
{
   int i ;                          // 'i' is a local variable.
   for ( i = 0 ; i < 10 ; i++ ) {
      // Something happens here.
   }
}
```

13. Suppose we have the following string of numerals:

```
String s = new String( "123" ) ;
int N ;
```

Give a simple way to convert the character string `s` to an integer `N`.

Answer: `int N = Integer.parseInt( s ) ;`

14. What is a Java exception ?

Answer: In Java, an exception is an event which occurs during execution of the program. Different types of exceptions are defined corresponding to the different types of events which may occur. Typically, such events need to be processed by an *exception handler*. In Java, an exception handler is a block of code enclosed within a `catch` statement. When an exception occurs the flow of control is immediately transferred to the first statement in the `catch` statement.

An exception is often said to be "thrown". While some programmers may occasionally want to throw their code out of the window (maybe along with their computer), the word "thrown" refers to the immediate transfer of control flow to the corresponding exception handler.

15. How are the Java keywords `try` and `catch` used? What are their purpose?

    Answer: In Java, the `try` keyword is used to enclose a program statement or sequence of statements which may cause an exception. The `catch` keyword is used to define an exception handler associated with a `try` statement. Several `catch` sections of code may be defined for a single `try` section of code. The purpose of multiple `catch` sections is to provide appropriate handling of different exceptions which may arise in the `try` section.

16. What is "type casting" (a.k.a., "type coercion") ? Give an example to convert an integer to a character.

    Answer: In Java, some types are compatible. Type casting refers to converting a value of one type to an equivalent value in another type.

    For example, when reading a file the `read()` method returns an integer. To see it on the screen as a character instead of an integer, it is necessary to type cast the integer to a character. For example:

```
int A  = buff_reader.read(  ) ;  // Note: buff_reader is a
                                 // new BufferedReader
char C = (char) A ;
```

    In general, a type cast is made by putting the type in parenthesis in front of the value to be converted.

    Some types can not be converted using type casting. For example, the following code results in a compile-time error:

```
String s = new String( "hello" ) ;
int x = (int) s ;
```

    When attempting to compile the code above, the following error message is presented:

```
main.java:6: inconvertible types
found   : java.lang.String
required: int
       int x = (int) s ;
                     ^
1 error
```

    When attempting to compile the code above, the following error

17. What class and method can be used to terminate a Java program immediately?

    Answer: `System.exit( 1 ) ;`

18. Suppose we have the following two Java strings and subsequent fragment of a Java program:

```
String A =  new String( ) ;
String B =  new String( ) ;
Scanner scan = new Scanner( System.in ) ;
A = scan.next() ;
B = scan.next() ;

// Your answer goes here.
```

Give a fragment of Java code to check if string A has the same contents as string B. If they are equal, output "equal", otherwise output "not equal".

Answer:

```
if ( A.equals( B ) ) {
     System.out.println( "equal" ) ;
}
else {
     System.out.println( "not equal" ) ;
}
```

19. Refer to the fragment of code in problem 18 Give a fragment of Java code to decide if string A come alphabetically before string B. If so, output "A before B", otherwise output "B before A". In this problem, assume the strings are not equal.

Answer:

```
int t = A.compareTo( B ) ;
if ( t < 0 )  {
     System.out.println( "A before B" ) ;
}
else {
     System.out.println( "B before A" ) ;
}
```

20. Suppose your Java program has two strings `X` and `Y`. What exactly would the following code decide?

```
if ( X == Y )   // Somthing goes here.
```

Answer: It decides if the references `X` and `Y` point to the same place in memory. It does not compare the contents of the two strings.

Bonus Practice Problems using Recursion and Files

21. Write a recursive method that accepts an intger N and returns a string containing the base two representation of N. E.g. if the input N is 43, the output is:

```
1 0 1 0 1 1
```

The header of the method is shown below:

```
static String convert_to_binary_string( int N )
{
   // Your implementation goes here.

}
```

22. Write a static Java method named "`write_it`" in class `main` to write the word "Hello." to a file named "`my_word.txt`". Use the Java classes `FileWriter` and `BufferedWriter`. Remember to close the file. The imports needed are shown below:

```
import java.io.FileWriter      ;
import java.io.BufferedWriter  ;
import java.io.IOException     ;

class main {

   void write_it()
   {
             // Your implementation goes here.
   }
}
```