1. **How can a recursive method cause a stack overflow?**

Stack overflow occurs when the limit of the size of the stack is exceeded. This happens when a method makes a call to itself, and the recursive calls continue on in a long chain. As a result, the stack will have lots of suspended computations – if there are too many suspended computations, the stack will grow beyond its limit and cause an error (this is stack overflow). A very common cause of stack overflow is infinite recursion (recursion where there is no stopping case).

1. **As a surprise for your young cousin's birthday, you have placed your small gift into a box, placed the box into a larger box, and continued placing smaller boxes into larger boxes until you ran out of boxes.  Write a recursive algorithm that your cousin can follow to open your gift.**Given parameter n number of boxes in giftBox method:

if(n == 0){

System.out.println(“gift”);

else{

System.out.println(“box”);

giftBox(n-1);

}

This method will print box each time there is a new box, and will eventually print gift once your young cousin finally opens your gift.

1. **Write a recursive method to compute 2n for a positive integer n.**  public static int powerOf2(int n){

int finalNum = 0;

if(n==0){

finalNum = 1;

}

else{

finalNum = powerOf2(n-1) \* 2;

}

return finalNum;

}

1. **The factorial of n, or n!, is defined as the series of products 1 × 2 × ... × n.  
   Write a recursive method and an iterative method that each compute n!.  
   Verify that both methods return the same result for any given positive value of n.  
   Then, execute both methods for larger and larger values of n.  
   What is the largest value of n that will successfully produce a result for each method?**Recursive method:

public static int recursiveFactorial(int n){

int finalNum = 0;

if(n==0){

finalNum = 1;

}

else{

finalNum = recursiveFactorial(n-1) \* n;

}

return finalNum;

}

Iterative method:

public static int iterativeFactorial(int n){

int finalNum = 1;

for(int i = 1; i <= n; i++){

finalNum = finalNum \* i;

}

return finalNum;

}

Finally, the largest value of n that will successfully produce a result for these methods is n=16. After 16, the correct answer is no longer generated.

1. **What is the output of the following program?  
   public class Exercise5  
   {  
       public static void main(String[] args)  
       {  
           System.out.println("Sum = " + guessResult(4));  
       } // end main  
        
       public static int guessResult(int n)  
       {  
           boolean result;  
           if (n == 1) {  
               result = 1;  
           } else {  
               result = n + guessResult(n - 1);  
           }  
           return result;  
       } // end guessOutput  
   } // end Exercise5**

As the code above is, it will not compile because guessResult is supposed to return an int and result is return, yet declared as a Boolean. However, assuming that Boolean was a typo and result was supposed to be declared as an int, the following would be output:

Sum = 10

1. **What is the output of the following program?  
   public class Exercise6  
   {  
       public static void main(String[] args)  
       {  
           guessOutput(7654321);  
       } // end main  
        
       public static void guessOutput(int n)  
       {  
           if (n > 0) {  
               System.out.print((n % 10) + " ");  
               guessOutput(n / 10);  
           }  
       } // end guessOutput  
   } // end Exercise6**

The above program would output the following:

1 2 3 4 5 6 7

1. **Identify and fix the errors in the following program:  
     
   public class Exercise7  
   {  
       public static void main(String[] args)  
       {  
           guessOutput(7654321);  
       } // end main  
        
       public static void guessOutput(int n)  
       {  
           if (n != 0)  
           {  
               System.out.println(n);  
               guessOutput(n / 10);  
           } // end if  
       } // end guessOutput  
   } // end Exercise7**

The program above does not appear to have any errors. It will output the following:

7654321

765432

76543

7654

765

76

7

However, in the if statement in guessOutput, it would make more sense for the condition to be >0 rather than !0 so that the method does not allow use of negative numbers. So, the code would look like the following:

public class Exercise7  
{  
    public static void main(String[] args)  
    {  
        guessOutput(7654321);  
    } // end main  
     
    public static void guessOutput(int n)  
    {  
        if (n > 0)  
        {  
            System.out.println(n);  
            guessOutput(n / 10);  
        } // end if  
    } // end guessOutput  
} // end Exercise7

1. **Identify and fix the errors in the following program:  
     
   public class Exercise8  
   {  
       public static void main(String[] args)  
       {  
           Exercise8 ex = new Exercise8();  
           System.out.println(ex);  
       } // end main  
        
       public Exercise8()  
       {  
           Exercise8 ex = new Exercise8();  
       } // end default Exercise8 constructor  
   } // end Exercise8**

This program will cause a stack overflow error as it stands. This is because when you create a new Exercise8 object, it calls the Exercise8 constructor. However, the Exercise8 constructor also creates a new Exercise8 object which then calls the constructor once again, and so on. Thus, there is a case of infinite recursion. One way to fix this would be to just have an empty (default) constructor as shown below:

public class Exercise8  
{  
    public static void main(String[] args)  
    {  
        Exercise8 ex = new Exercise8();  
        System.out.println(ex);  
    } // end main  
     
    public Exercise8()  
    {

} // end default Exercise8 constructor  
} // end Exercise8

1. **Consider the method displayRowOfCharacters that displays any given character the specified number of times on one line. For example, the call  
       displayRowOfCharacters('\*', 5);  
   produces the line  
       \*\*\*\*\*  
   Implement this method in Java by using recursion.**

public static void displayRowOfCharacters(char x, int y){

if (y >= 1) {

System.out.print(x);

displayRowOfCharacters(x, y - 1);

}

}

}

1. **Write a recursive method that asks the user for integer input that is between 1 and 100, inclusive. If the input is out of range, the method should recursively ask the user to enter a new input value.** public static int inRange() throws InputMismatchException {

Scanner input = new Scanner(System.in);

System.out.print("Please enter a number between 1 and 100 (inclusive): ");

int userNum = input.nextInt();

if(userNum < 1 || userNum > 100){

System.out.println("Please enter valid int");

userNum = inRange();

}

return userNum;

}

1. **Write a recursive method that displays the entries in a given array in backward order. That is, display the last entry first and the first entry last.  
   Hint: Consider the last entry of the array first.**  public static void backwardsArray(int[] x, int arraySize){

if(arraySize >= 0){

System.out.print(x[arraySize] + " ");

backwardsArray(x, arraySize-1);

}

}

When the backwardsArray method is called in main, arraySize should be set to the length of the array - 1.

1. **Repeat Exercise 11, but have your solution instead consider the first entry of the array first.**

public static void backwardsArray(int[] x, int n){

if(n < x.length){

backwardsArray(x, n+1);

System.out.print(x[n] + " ");

}

}

When the backwardsArray method is called in main, n should be set to 0.

1. **Repeat Exercises 11 and 12, but write a string backward instead of an array.** public static void backwardsString(String x, int stringLength){

if(stringLength >= 0){

System.out.print(x.charAt(stringLength));

backwardsString(x, stringLength-1);

}

}

When the backwardsString method is called in main, stringLength should be set to the length of the string – 1.

1. **Write a recursive method that displays a given message n times.** public static void displayMessage(String input, int n){

if(n==0){

return;

}

else{

System.out.println(input);

displayMessage(input,n-1);

}

}

In this method, n is the number of times the message should be displayed.

1. **Define a recursive method f that, given a nonnegative integer n, returns  
       n % 2 when n ranges from 0 to 9  
       f(n / 10) + (n % 10) % 2 when n is greater than or equal to 10** public static int f(int n){

if(n >= 0 && n <= 9){

return n % 2;

}

else{

return f(n / 10) + (n % 10) % 2;

}

}

Can use error throwing in main method to make sure that integer n is nonnegative.

1. **Jack and Jill take turns using a one-gallon pail to empty a tank of water.  
       On his turn, Jack always removes one gallon of water.  
       Jill, however, removes one or two gallons so that either an even number of gallons are left in the tank or the tank becomes empty.  
       Using recursion, how many turns in total must Jack and Jill make to empty the tank, if it initially contains n gallons?  
       Write a recursive method that counts and returns the total number of turns given the number of gallons n.**

public static int emptyWell(int gallons, int count, boolean jackTurn){  
 if(jackTurn){  
 gallons--;  
 count++;  
 jackTurn = false;  
 }  
 else{  
 if(gallons % 2 == 0){  
 gallons = gallons - 2;  
 }  
 else{  
 gallons--;  
 }  
 count++;  
 jackTurn = true;  
 }  
  
 if(gallons <= 0){  
 return count;  
 }  
 else{  
 return emptyWell(gallons, count, jackTurn);  
 }  
}