```
/**
 *  Gets a command-line argument (int), and prints all the divisors of the given number.
 */
public class Divisors {
    public static void main (String[] args) {

        // Parsing the first command-line argument as an integer
        int num = Integer.parseInt(args[0]);

        // If the number is positive, find divisors from 1 to num/2
        if (num > 0){
            for (int i = 1; i < num/2 + 1; i++){
                // Check if 'i' is a divisor of 'num'
                if (num % i == 0){
                    System.out.println(i);
                }
            }
        }
        // If the number is negative, find divisors from -1 to num/2
        if (num < 0) {
            for (int i = -1; i > num/2 - 1; i--){
                // Check if 'i' is a divisor of 'num'
                if (num % i == 0){
                    System.out.println(i);
                }
            }
        }
        // Print the number itself if it's not equal to zero, as he is the last divisor
        if (num != 0) {
            System.out.println(num);
        }
    }
}
```

```
/**
 * Prints a given string, backward. Then prints the middle character in the string.
 * The program expects to get one command-line argument: A string.
 */
public class Reverse {
    public static void main (String[] args){

        // Retrieve the first command-line argument and store it as a string
        String str = args[0];
        String reversedStr = "";

        // Check if the input string is empty
        if (str.isEmpty()){
            System.out.println("The string is empty");
        }
        else {
            // Reverse the string character by character using a for loop
            for (int i = str.length() - 1; i >= 0; i--){
                reversedStr = reversedStr + str.charAt(i);
            }
            // Print the reversed string
            System.out.println(reversedStr);
            // Print the middle character of the original string
            System.out.println("The middle character is " + str.charAt((str.length() - 1) /
2));
        }
    }
}
```

```java
/**
 *  Generates and prints random integers in the range [0,10),
 *  as long as they form a non-decreasing sequence.
 */
public class InOrder {
    public static void main (String[] args) {

        int range = 10;
        // Generates the first random number
        int num = (int) (Math.random() * range);
        boolean nonDecreasing = true;

        // Loop continues as long as the sequence is non-decreasing
        while (nonDecreasing){
            System.out.println(num);
            // Generate a new random number
            int newNum = (int) (Math.random() * range);
            // Check if the new number is greater than or equal to the current number
            if (newNum >= num){
                // Update the current number if it forms a non-decreasing sequence
                num = newNum;
            } else {
                // Set the flag to false if the sequence is no longer non-decreasing
                nonDecreasing = false;
            }
        }
    }
}
```

```
/**
 *  Gets a command-line argument n (int), and prints an n-by-n damka board.
 */
public class DamkaBoard {
   public static void main(String[] args) {

      // Parse the first command-line argument as an integer to determine board size
      int boardSize = Integer.parseInt(args[0]);
      String line = "";

      // Check if the board size is less than 2 and if so print a message accordingly
      if (boardSize < 2){
         System.out.println("Damka board requires a positive integer bigger than 1");
      }else {
         // Loop through rows (boardSize times) to create the Damka board
         for (int i = 0; i < boardSize; i++){
            // Loop through columns (boardSize times) to create each line of the board
            for (int j = 0; j < boardSize; j++){
               // Check if the row number is even or odd to create alternating pattern of
'* ' and ' *'
               if (i % 2 == 0){
                  line += "* ";
               } else {
                  line += " *";
               }
            }
            // Print the line representing a row of the Damka board
            System.out.println(line);
            // Reset the line for the next row
            line = "";
         }
      }
   }
}
```

```
/**
 *  Gets a command-line argument (int), and chekcs if the given number is perfect.
 */

public class Perfect {
    public static void main (String[] args) {

        // Parse the first command-line argument as an integer
        int num = Integer.parseInt(args[0]);
        int sum = 1;
        // Initialize the string to represent divisors
        String perfectStr = args[0] + " = 1";

        // Check if the number is 0 and if so, print a message accordingly
        if (num == 0){
            System.out.println("0 Doesn't have any divisors");
        }
        else {
            if (num > 0){
                // Find divisors from 2 to num/2 and calculate the sum of divisors
                for (int i = 2; i < num/2 + 1; i++){
                    if (num % i == 0){
                        // Add the divisor to the sum
                        sum += i;
                        perfectStr += String.format(" + %d", i);
                    }
                }
                // Check if the sum of divisors equals the original number and print a
message accordingly
                if (sum == num){
                    String output = String.format("%d is a perfect number since ", num) +
perfectStr;
                    System.out.println(output);
                } else {
                    System.out.println(num + " is not a perfect number");
                }
            }
        }
    }
}
```

```
/**
 *  Computes some statistics about families in which the parents decide
 *  to have children until they have at least one child of each gender.
 *  The program expects to get one command-line argument: an int value
 *  that determines how many families to simulate.
 */
public class OneOfEachStats1 {
   public static void main (String[] args) {

      // Parse the first command-line argument as an integer to determine the number
of iterations (families)
      int T = Integer.parseInt(args[0]);
      // Initialize all the required variables
      int kidsCount, twoKids = 0, threeKids = 0, fourKidsOrMore = 0, totalKids = 0,
commonFamily, max;
      boolean girl, boy;
      double num, averageKids;
      String output = "";

      // Loop 'T' times to simulate 'T' families
      for (int i = 0; i < T; i++){
         kidsCount = 0;
         girl = false;
         boy = false;

         // Simulate the birth of children in a family until at least one boy and one girl
are born
         while(!girl || !boy){
            // Generate a random number between 0.0 (inclusive) and 1.0 (exclusive)
            num = Math.random();
            // If the generated number is bigger than or equal to 0.5 it's a boy
            if (num >= 0.5){
               boy = true;
            }
            // If the generated number is less than 0.5 it's a girl
            if (num < 0.5){
               girl = true;
            }
            // Increment the count of children in the family
            kidsCount ++;
            // Increment the count of total children across all families
            totalKids ++;
         }
         // Update counters based on the number of children in each family
         if (kidsCount == 2) {
            twoKids++;
```

```
            }
            if (kidsCount == 3){
                threeKids ++;
            }
            if (kidsCount >= 4){
                fourKidsOrMore ++;
            }
        }
        // Calculate the average number of children needed to get at least one of each
gender
        averageKids = (double) totalKids / T;
        // Determine the most common number of children in families
        max = Math.max(Math.max(twoKids, threeKids), fourKidsOrMore);
        if (max == twoKids){
            commonFamily = 2;
        } else {
            if (max == threeKids){
                commonFamily = 3;
            }else{
                commonFamily = 4;
            }
        }
        // Construct the output message with statistics about the families and children
        output = "Average: " + averageKids + " children to get at least one of each
gender.\n";
        output += "Number of families with 2 children: " + twoKids + "\n";
        output += "Number of families with 3 children: " + threeKids + "\n";
        output += "Number of families with 4 or more children: " + fourKidsOrMore +
"\n";
        output += "The most common number of children is " + commonFamily + ".";
        // Print the output message displaying the statistics
        System.out.println(output);
    }
}
```