Recursion: Paper Exercise 1

Q1:What do the following two programs print?

Q2: What does xMethod in each program do when called with positive integer as input?
(answer in plain English)

```
public class Lab11Ex1a{
  public static void main(String[] args) {

    xMethod(5);
  }

  public static void xMethod(int n) {
    if (n > 0) {
       System.out.print(n + " ");
       xMethod(n - 1);
    }
  }
}
```

```
public class LabEx1b{
  public static void main(String[] args) {

    xMethod(5);
  }

  public static void xMethod(int n) {
    if (n > 0) {
       xMethod(n - 1);
       System.out.print(n + " ");
    }
  }
}
```

Recursion: Paper Excercise 2

- Q1: What does the following program print?
- Q2: What does xMethod do given positive integer as input (answer in plain English)?

```
public class Lab11Ex2{
  public static void main(String[] args){
     int result = xMethod(4);
     System.out.println("xMethod returned " + result);

  }

  public static int xMethod(int n) {
     if (n == 1)
        return 1;
     else
       return n + xMethod(n - 1);
  }
}
```

- Q3: What would happen if we made xMethod(-3) call in the main, instead of xMethod(5)?

Recursion: Paper Excercise 3

- Q1: What does the following program print?
- Q2: What does xMethod do, given positive integer as input (answer in plain English)?

```
public class Lab11Ex3
{
  public static void main(String[] args) {
      xMethod(7254);
  }

  public static void xMethod(int n) {
      if (n > 0) {
         System.out.print(n % 10);
         xMethod(n / 10);
      }
  }
}
```

Recursion: Paper Exercise 4

Q1: What does the following program print?
Q2: What does xMethod do (answer in plain English)?

```
public class Lab11Ex4 {
  public static void main(String[] args) {
    int[] list = {2,  7,  -11};
    System.out.println(xMethod(list, list.length-1));
  }

  public static int xMethod(int[] list, int high) {
    if (high == 0) {
       return list[0];
    }
    else {
      int tmp=xMethod(list, high - 1);
      if (tmp>list[high])
          return tmp;
      else
          return list[high];
     }
  }
}
```

Recursion: Paper Study

- Open file SumProd.java

It contains contains 2 methods that both compute the sum: 1+2+3 …+n. One computes it in a way that you have seen before (called, iterative way), and the other in recursive way.

Similarly the program contains 2 methods that both compute the product 1*2*3…*n (thus they compute n!) in iterative and factorial way.

Study with TAs and understand well all these 4 solutions.

Recursion: Programming Exercise 1

- Write a recursive method, called m, that computes the following series:

$$m(i) = \frac{1}{3} + \frac{2}{5} + \frac{3}{7} + \frac{4}{9} + \frac{5}{11} + \frac{6}{13} + \ldots + \frac{i}{2i+1}$$

- In the main method, write a test program that displays **m(i)** for **i = 1, 2, . . . , 10**.

Recursion: Programming Exercise 2

- Write a recursive method, called countDigits, that counts the number of digits in a given positive integer n.

- Open the file NumberOfDigitsStudents.java and program your solution in the clearly indicated spaces.

Recursion: Programming Exercise 3

- A string is a palindrome if it reads the same from the left and from the right. For example, word "kayak" is a palindrome, so is a name "Anna", so is a word "a". Word "uncle" is not a palindrome.

- Write a recursive method, called isPalindrome, that returns true if the input string is a palindrome and otherwise returns false. Test your method.

- Notice: a word of length n is a palindrom if 1^{st} and n^{th} letter are the same, AND 2^{nd} and $(n-1)^{st}$ are the same, and so on … until we get to the "middle" of the word.

Idea/Strategy

Checking if a string is a palindrome can be divided into two subproblems:

1. Check if the 1st and the last character in the string are the same
2. Ignore the two end characters and check if the rest of the substring is a palindrome.

Notice that the 2nd subproblem is the same as the original problem but smaller in size.

Useful string methods: length(), charAt() and substring()

9.2.4 String Length, Characters, and Combining Strings

The `String` class provides the methods for obtaining length, retrieving individual characters, and concatenating strings, as shown in Figure 9.3.

java.lang.String	
`+length(): int`	Returns the number of characters in this string.
`+charAt(index: int): char`	Returns the character at the specified index from this string.
`+concat(s1: String): String`	Returns a new string that concatenates this string with string `s1`.

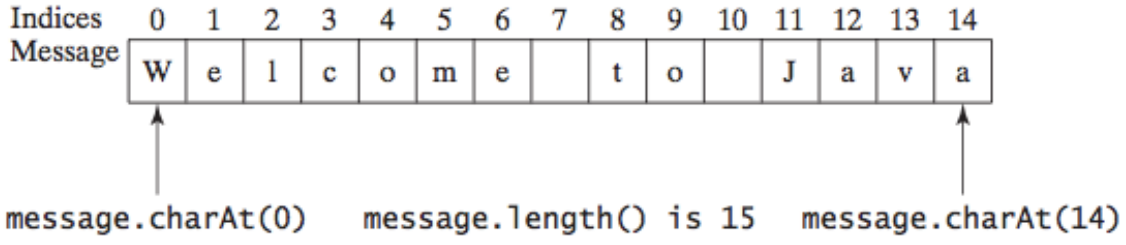
FIGURE 9.3 The `String` class contains the methods for getting string length, individual characters, and combining strings.

You can get the length of a string by invoking its `length()` method. For example, `message.length()` returns the length of the string `message`. `length()`

Caution

`length` is a method in the `String` class but is a property of an array object. So you have to use `s.length()` to get the number of characters in string `s`, and `a.length` to get the number of elements in array `a`. `length()`

The `s.charAt(index)` method can be used to retrieve a specific character in a string `s`, where the index is between `0` and `s.length()-1`. For example, `message.charAt(0)` returns the character `W`, as shown in Figure 9.4. `charAt(index)`



9.2.5 Obtaining Substrings

You can obtain a single character from a string using the `charAt` method, as shown in Figure 9.3. You can also obtain a substring from a string using the `substring` method in the `String` class, as shown in Figure 9.5.

java.lang.String	
`+substring(beginIndex: int): String`	Returns this string's substring that begins with the character at the specified `beginIndex` and extends to the end of the string, as shown in Figure 9.6.
`+substring(beginIndex: int, endIndex: int): String`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`, as shown in Figure 9.6. Note that the character at `endIndex` is not part of the substring.

FIGURE 9.5 The `String` class contains the methods for obtaining substrings.

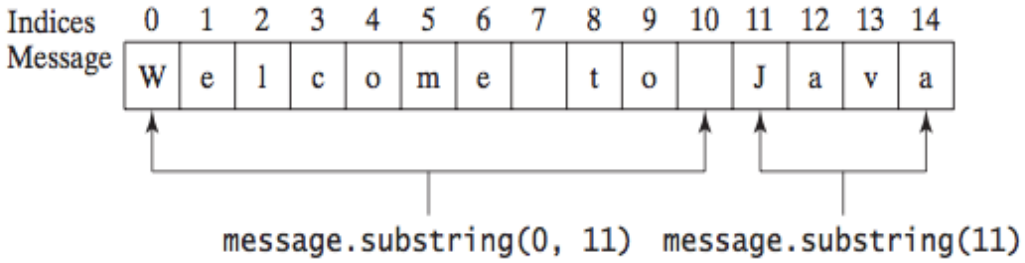


FIGURE 9.6 The `substring` method obtains a substring from a string.

Recursion: Extra Programming Exercise

- Write a recursive method, called checkSorted, that checks if the given array of integers is in sorted order (from smallest to largest)

- TAs will give you hints/overview of the algorithm

- Open the file ArraySortedStudents.java and program your solution in the clearly indicated spaces.

More on Strings: String vs. char[]

- Similarities:
  - both are collections of characters
  - both indexed from **0** up to **length - 1**
  - both are reference variables
    - no **==** comparison!

String vs. char[]

- Differences:
  - Access of single character: **str.charAt(i)** vs **array[i]**
  - Strings cannot be modified internally once they are created
    - No equivalent of **array[i] = 'x'**
  - String variables can be assigned constant strings where using new is optional

```
String str;
str = "abc";
str = new String("def" );
```

  - Most operations on Strings are done with methods

```
array.length        // not a method call; no ( )
str.length( )       // method call; ( ) required
```

Conversions: **String ⇔ char[]**

```
char[] array;
char[] array2;
...
// Create String from array
String str = new String( array );
// Create array from String
array2 = str.toCharArray( );
```

Common Methods of **String**

- Review the various methods available in the String class:

  http://java.sun.com/javase/6/docs/api/java/lang/String.html

  - **charAt(…), indexOf(…), length(…)**
  - **toCharArray(…)**
  - **equals(…), compareTo(…)**
  - **concat(…), substring(…),**
  - **toLowerCase(…), toUpperCase(…)**
  - **…**