Building Java Programs

Chapter 4

Lecture 4-1: `if` and `if/else` Statements

reading: 4.2

self-check: #4-5, 7, 10, 11

exercises: #7

videos: Ch. 4 #2-4

Loops with `if/else`

- `if/else` statements can be used with loops or methods:

```
int evenSum = 0;
int oddSum = 0;

for (int i = 1; i <= 10; i++) {
    if (i % 2 == 0) {
        evenSum = evenSum + i;
    } else {
        oddSum = oddSum + i;
    }
}

System.out.println("Even sum: " + evenSum);
System.out.println("Odd sum: " + oddSum);
```

Nested `if/else`

reading: 4.2, 4.5

self-check: #6, 8, 9, 24-27

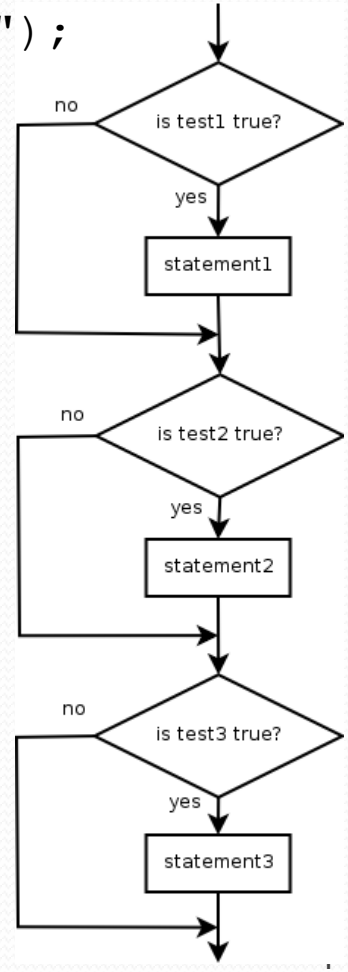
exercises: #10-14

videos: Ch. 4 #4

Sequential `if` bug

- What's wrong with the following code?

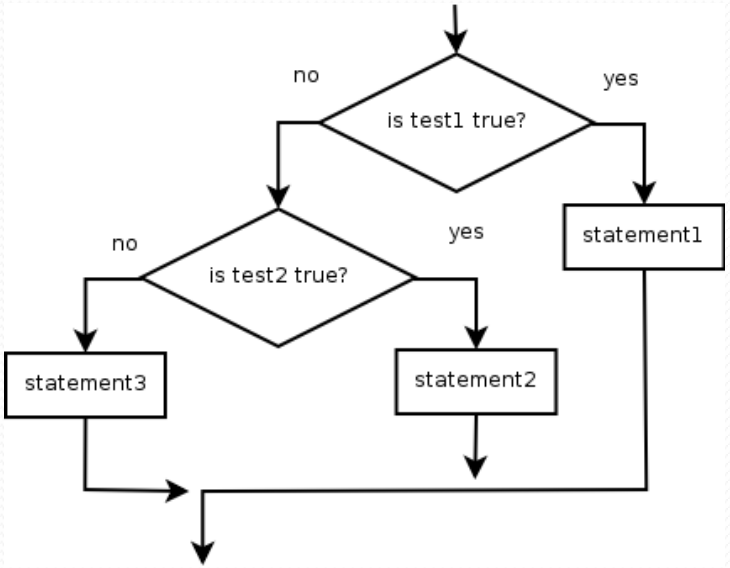
```
Scanner console = new Scanner(System.in);
System.out.print("What percentage did you earn? ");
int percent = console.nextInt();
if (percent >= 90) {
    System.out.println("You got an A!");
}
if (percent >= 80) {
    System.out.println("You got a B!");
}
if (percent >= 70) {
    System.out.println("You got a C!");
}
if (percent >= 60) {
    System.out.println("You got a D!");
}
else {
    System.out.println("You got an F!");
}
...
```



Nested `if/else`

Chooses between outcomes using many tests

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else {
    statement(s);
}
```



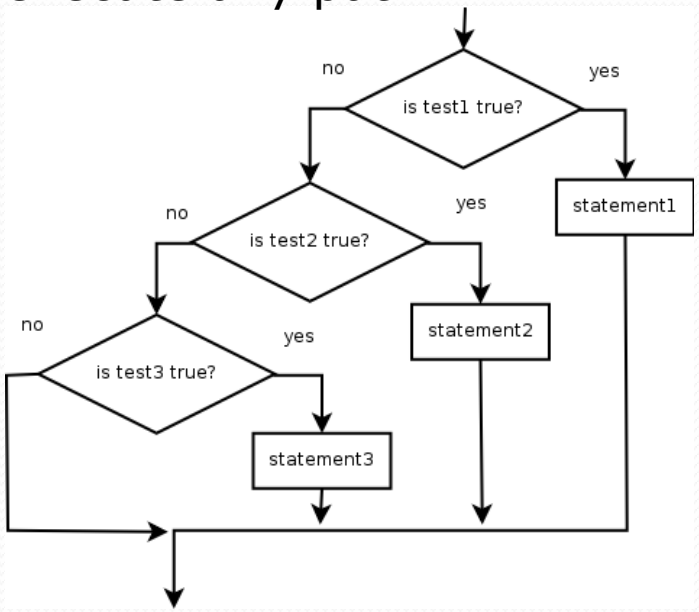
- Example:

```
if (number > 0) {
    System.out.println("Positive");
} else if (number < 0) {
    System.out.println("Negative");
} else {
    System.out.println("Zero");
}
```

Nested `if/else/if`

- If it ends with `else`, one code path must be taken.
- If it ends with `if`, the program might not execute any path.

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else if (test) {
    statement(s);
}
```



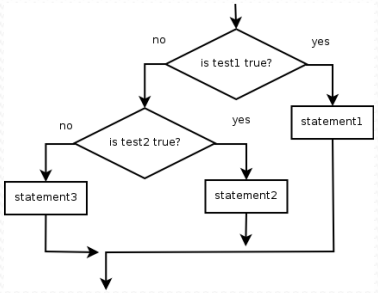
- Example:

```
if (place == 1) {
    System.out.println("You win the gold medal!");
} else if (place == 2) {
    System.out.println("You win a silver medal!");
} else if (place == 3) {
    System.out.println("You earned a bronze medal.");
}
```

Structures

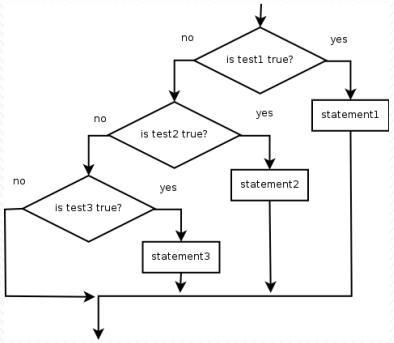
- Exactly 1 path: (mutually exclusive)

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else {
    statement(s);
}
```



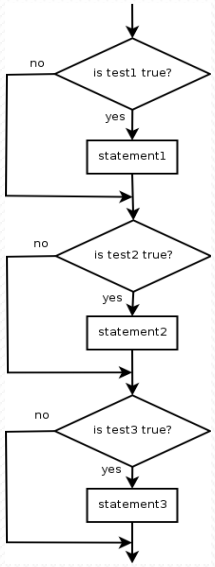
- 0 or 1 path:

```
if (test) {
    statement(s);
} else if (test) {
    statement(s);
} else if (test) {
    statement(s);
}
```



- 0, 1, or many paths: (independent tests, not exclusive)

```
if (test) {
    statement(s);
}
if (test) {
    statement(s);
}
if (test) {
    statement(s);
}
```



Which nested `if/else`?

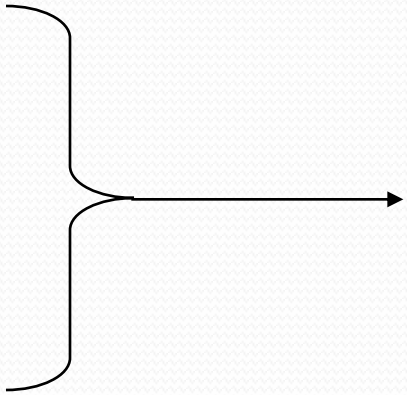
- **(1) if/if/if (2) nested if/else (3) nested if/else/if**
  - Reading the user's GPA and printing whether the student is on the dean's list (3.8 to 4.0) or honor roll (3.5 to 3.8).
    - **(3)** nested `if / else if`
  - Printing whether a number is even or odd.
    - **(N/A)** simple `if / else`
  - Printing whether a user is lower-class, middle-class, or upper-class based on their income.
    - **(2)** nested `if / else if / else`
  - Reading a number from the user and printing whether it is divisible by 2, 3, and/or 5.
    - **(1)** sequential `if / if / if`
  - Printing a grade of A, B, C, D, or F based on a percentage.
    - **(2)** nested `if / else if / else if / else if / else`

Factoring `if/else` code

- **factoring**: extracting common/redundant code
  - Factoring `if/else` code can reduce the size of `if/else` statements or eliminate the need for `if/else` altogether.

- Example:

```
if (a == 1) {
    x = 3;
} else if (a == 2) {
    x = 6;
    y++;
} else {  // a == 3
    x = 9;
}
```



```
x = 3 * a;
if (a == 2) {
    y++;
}
```

Code in need of factoring

```
if (money < 500) {
    System.out.println("You have, $" + money + " left.");
    System.out.print("Caution!  Bet carefully.");
    System.out.print("How much do you want to bet? ");
    bet = console.nextInt();
} else if (money < 1000) {
    System.out.println("You have, $" + money + " left.");
    System.out.print("Consider betting moderately.");
    System.out.print("How much do you want to bet? ");
    bet = console.nextInt();
} else {
    System.out.println("You have, $" + money + " left.");
    System.out.print("You may bet liberally.");
    System.out.print("How much do you want to bet? ");
    bet = console.nextInt();
}
```

Code after factoring

```
System.out.println("You have, $" + money + " left.");

if (money < 500) {
    System.out.print("Caution!  Bet carefully.");
} else if (money < 1000) {
    System.out.print("Consider betting moderately.");
} else {
    System.out.print("You may bet liberally.");
}

System.out.print("How much do you want to bet? ");
bet = console.nextInt();
```

- If the start of each branch is the same, move it *before* the `if/else`.
- If the end of each branch is the same, move it *after* the `if/else`.
- If similar but code exists in each branch, look for patterns.

The "dangling `if`" problem

- What can be improved about the following code?

```
if (x < 0) {
    System.out.println("x is negative");
} else if (x >= 0) {
    System.out.println("x is non-negative");
}
```

- The second `if` test is unnecessary and can be removed:

```
if (x < 0) {
    System.out.println("x is negative");
} else {
    System.out.println("x is non-negative");
}
```

  - This is also relevant in methods that use `if` with `return`...

if/else with return

- Methods can return different values using `if/else`:

```
// Returns the largest of the three given integers.
public static int max3(int a, int b, int c) {
    if (a >= b && a >= c) {
        return a;
    } else if (b >= c && b >= a) {
        return b;
    } else {
        return c;
    }
}
```

  - Whichever path the code enters, it will return the appropriate value.
  - Returning a value causes a method to immediately exit.
  - All code paths must reach a `return` statement.
    - All paths must also return a value of the same type.

All paths must return

```
public static int max3(int a, int b, int c) {
    if (a >= b && a >= c) {
        return a;
    } else if (b >= c && b >= a) {
        return b;
    }
    // Error: not all paths return a value
}
```

- The following also does not compile:

```
public static int max3(int a, int b, int c) {
    if (a >= b && a >= c) {
        return a;
    } else if (b >= c && b >= a) {
        return b;
    } else if (c >= a && c >= b) {
        return c;
    }
}
```

  - The compiler thinks `if/else/if` code might skip all paths.

if/else question

A person's body mass index (BMI) is defined to be:

$$BMI = \frac{weight}{height^2} \times 703$$

BMI	Weight class
below 18.5	underweight
18.5 - 24.9	normal
25.0 - 29.9	overweight
30.0 and up	obese

- Write a program that produces the following output:

```
This program reads data for two people and computes
their body mass index (BMI) and weight status.

Enter next person's information:
height (in inches)? 70.0
weight (in pounds)? 194.25

Enter next person's information:
height (in inches)? 62.5
weight (in pounds)? 130.5

Person #1 body mass index = 27.87
overweight
Person #2 body mass index = 23.49
normal
Difference = 4.38
```

if/else answer

```
// This program computes two people's body mass index (BMI)
// and compares them.  The code uses parameters and returns.

import java.util.*;  // so that I can use Scanner

public class BMI {
    public static void main(String[] args) {
        introduction();
        Scanner console = new Scanner(System.in);

        double bmi1 = person(console);
        double bmi2 = person(console);

        // report overall results
        report(1, bmi1);
        report(2, bmi2);
        System.out.printf("Difference = %.2f\n", Math.abs(bmi1 - bmi2));

    }

    // prints a welcome message explaining the program
    public static void introduction() {
        System.out.println("This program reads in data for two people");
        System.out.println("and computes their body mass index (BMI)");
        System.out.println("and weight status.");
        System.out.println();
    }
...
```

Scanner BMI solution, cont.

```
// reads information for one person, computes their BMI, and returns it
public static double person(Scanner console) {
    System.out.println("Enter next person's information:");
    System.out.print("height (in inches)? ");
    double height = console.nextDouble();

    System.out.print("weight (in pounds)? ");
    double weight = console.nextDouble();
    System.out.println();

    double bodyMass = bmi(height, weight);
    return bodyMass;
}

// Computes/returns a person's BMIbased on their height and weight.
public static double bmi(double height, double weight) {
    return (weight * 703 / height / height);
}

// Outputs information about a person's BMI and weight status.
public static void report(int number, double bmi) {
    System.out.printf("Person #%d body mass index = %.2f\n", number, bmi);
    if (bmi < 18.5) {
        System.out.println("underweight");
    } else if (bmi < 25) {
        System.out.println("normal");
    } else if (bmi < 30) {
        System.out.println("overweight");
    } else {
        System.out.println("obese");
    }
}

}
```

`if/else`, `return` question

- Write a method `countFactors` that returns the number of factors of an integer.
  - `countFactors(24)` returns `8` because 1, 2, 3, 4, 6, 8, 12, and 24 are factors of 24.

- Write a program that prompts the user for a maximum integer and prints all prime numbers up to that max.

```
Maximum number? 52
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
15 primes (28.84%)
```

if/else, return answer 1

```
// Prompts for a maximum number and prints each prime up to that maximum.
import java.util.*;

public class Primes {
    public static void main(String[] args) {
        // read max from user
        Scanner console = new Scanner(System.in);
        System.out.print("Maximum number? ");
        int max = console.nextInt();
        printPrimes(max);
    }

    // Prints all prime numbers up to the given maximum.
    public static void printPrimes(int max) {
        int primes = 0;
        for (int i = 2; i <= max; i++) {
            if (countFactors(i) == 2) {       // i is prime
                System.out.print(i + " ");
                primes++;
            }
        }
        System.out.println();

        double percent = 100.0 * primes / max;
        System.out.printf("%d primes (%.2f%%)\n", primes, percent);
    }
```

if/else, return answer 2

```
    ...

    // Returns how many factors the given number has.
    public static int countFactors(int number) {
        int count = 0;
        for (int i = 1; i <= number; i++) {
            if (number % i == 0) {
                count++;  // i is a factor of number
            }
        }
        return count;
    }
}
```

Building Java Programs

Chapter 4
Lecture 4-2: Strings

reading: 3.3, 4.3 - 4.4

self-check: Ch. 4 #12, 15

exercises: Ch. 4 #15, 16

videos: Ch. 3 #3

Objects and classes

- **object:** An entity that contains:
  - *data* (variables), and
  - *behavior* (methods).

- **class**: A program, or a type of objects.

- Examples:
  - The class `String` represents objects that store text.
  - The class `DrawingPanel` represents graphical window objects.
  - The class `Scanner` represents objects that read information from the keyboard, files, and other sources.

Strings

- **string**: An object storing a sequence of text characters.
  - Unlike most other objects, a `String` is not created with `new`.

```
String name = "text";
String name = expression;
```

  - Examples:

```
String name = "Marla Singer";

int x = 3;
int y = 5;
String point = "(" + x + ", " + y + ")";
```

Indexes

- Characters of a string are numbered with 0-based *indexes*:

```
String name = "P. Diddy";
```

index	0	1	2	3	4	5	6	7
char	P	.		D	i	d	d	y

  - The first character's index is always 0
  - The last character's index is 1 less than the string's length

  - The individual characters are values of type `char` (seen later)

String methods

Method name	Description
`indexOf(`**`str`**`)`	index where the start of the given string appears in this string (-1 if it is not there)
`length()`	number of characters in this string

`substring(`**`index1`**`, `**`index2`**`)`
or
`substring(`**`index1`**`)`
	the characters in this string from *index1* (inclusive) to *index2* (exclusive);
	if *index2* omitted, grabs till end of string

`toLowerCase()`	a new string with all lowercase letters
`toUpperCase()`	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String gangsta = "Dr. Dre";
System.out.println(gangsta.length());   // 7
```

String method examples

```
//      index 012345678901
String s1 = "Stuart Reges";
String s2 = "Marty Stepp";
System.out.println(s1.length());          // 12
System.out.println(s1.indexOf("e"));      // 8
System.out.println(s1.substring(7, 10))   // "Reg"

String s3 = s2.substring(2, 8);
System.out.println(s3.toLowerCase());     // "rty st"
```

- Given the following string:

```
//          index 0123456789012345678901
String book = "Building Java Programs";
```

  - How would you extract the word "Java" ?
  - How would you extract the first word from any string?

Modifying strings

- Methods like `substring`, `toLowerCase`, etc. create/return a new string, rather than modifying the current string.

```
String s = "lil bow wow";
s.toUpperCase();
System.out.println(s);   // lil bow wow
```

- To modify a variable, you must reassign it:

```
String s = "lil bow wow";
s = s.toUpperCase();
System.out.println(s);   // LIL BOW WOW
```

Strings as parameters

```
public class StringParameters {
    public static void main(String[] args) {
        sayHello("Marty");

        String teacher = "Helene";
        sayHello(teacher);
    }

    public static void sayHello(String name) {
        System.out.println("Welcome, " + name);
    }
}
```

Output:

```
Welcome, Marty
Welcome, Helene
```

Strings as user input

- Scanner's next method reads a word of input as a String.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
name = name.toUpperCase();
System.out.println(name + " has " + name.length() +
    " letters and starts with " + name.substring(0, 1));
```

Output:

```
What is your name? Madonna
MADONNA has 7 letters and starts with M
```

- The nextLine method reads a line of input as a String.

```
System.out.print("What is your address? ");
String address = console.nextLine();
```

Comparing strings

- Relational operators such as < and == fail on objects.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name == "Barney") {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

  - This code will compile, but it will not print the song.

  - == compares objects by *references* (seen later), so it often gives `false` even when two `Strings` have the same letters.

The equals method

- Objects are compared using a method named equals.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney")) {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

  - Technically this is a method that returns a value of type boolean, the type used in logical tests.

String test methods

Method	Description
`equals(`**`str`**`)`	whether two strings contain the same characters
`equalsIgnoreCase(`**`str`**`)`	whether two strings contain the same characters, ignoring upper vs. lower case
`startsWith(`**`str`**`)`	whether one contains other's characters at start
`endsWith(`**`str`**`)`	whether one contains other's characters at end
`contains(`**`str`**`)`	whether the given string is found within this one

```
String name = console.next();
if (name.startsWith("Dr.")) {
    System.out.println("Are you single?");
} else if (name.equalsIgnoreCase("LUMBERG")) {
    System.out.println("I need your TPS reports.");
}
```

Strings question

- Write a program that reads a person's name and converts it into a "gangsta name."

Output (run 1):

```
Type your name, playa: Peter Griffin
(M)ale or (F)emale? m
Your gangsta name is "P. GRIFFIN Daddy Peter-izzle"
```

Output (run 2):

```
Type your name, playa: Marge Simpson
(M)ale or (F)emale? F
Your gangsta name is "M. SIMPSON Goddess Marge-izzle"
```

Strings answer

```
// This program prints your "gangsta" name.
import java.util.*;

public class GangstaName {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Type your name, playa: ");
        String name = console.nextLine();

        System.out.print("(M)ale or (F)emale: ");
        String gender = console.next();

        // split name into first/last name and initials
        String first = name.substring(0, name.indexOf(" "));
        String last = name.substring(name.indexOf(" ") + 1);
        last = last.toUpperCase();
        String fInitial = first.substring(0, 1);

        String title;
        if (gender.equalsIgnoreCase("m")) {
            title = "Daddy";
        } else {
            title = "Goddess";
        }

        System.out.println("Your gangsta name is \"" + fInitial + ". "
                + last + " " + title + " " + first + "-izzle\"");
    }
}
```

Type char

- `char` : A primitive type representing single characters.
  - Each character inside a `String` is stored as a `char` value.
  - Literal `char` values are surrounded with apostrophe (single-quote) marks, such as `'a'` or `'4'` or `'\n'` or `'\''`
  - It is legal to have variables, parameters, returns of type `char`

```
char letter = 'S';
System.out.println(letter);                    // S
```

- `char` values can be concatenated with strings.

```
char initial = 'P';
System.out.println(initial + " Diddy");  // P Diddy
```

The charAt method

- The chars in a String can be accessed using the charAt method.

```
String food = "cookie";
char firstLetter = food.charAt(0);   // 'c'

System.out.println(firstLetter + " is for " + food);
System.out.println("That's good enough for me!");
```

- You can use a for loop to print or examine each character.

```
String major = "CSE";
for (int i = 0; i < major.length(); i++) {
    char c = major.charAt(i);
    System.out.println(c);
}
```

Output:

```
C
S
E
```

char vs. int

- All char values are assigned numbers internally by the computer, called *ASCII* values.
  - Examples:
    ```
    'A' is 65,    'B' is 66,   ' ' is 32
    'a' is 97,    'b' is 98,   '*' is 42
    ```
  - Mixing char and int causes automatic conversion to int.
    ```
    'a' + 10  is 107,          'A' + 'A'  is 130
    ```
  - To convert an int into the equivalent char, type-cast it.
    ```
    (char) ('a' + 2) is 'c'
    ```

char vs. String

- "h" is a String
  'h' is a char (the two behave differently)

- String is an object; it contains methods

```
String s = "h";
s = s.toUpperCase();          // 'H'
int len = s.length();         //  1
char first = s.charAt(0);     // 'H'
```

- char is primitive; you can't call methods on it

```
char c = 'h';
c = c.toUpperCase();   // ERROR: "cannot be dereferenced"
```

  - What is s + 1 ? What is c + 1 ?
  - What is s + s ? What is c + c ?

Comparing char values

- You can compare `char` values with relational operators:

```
'a' < 'b'   and   'X' == 'X'   and   'Q' != 'q'
```

  - An example that prints the alphabet:

```
for (char c = 'a'; c <= 'z'; c++) {
    System.out.print(c);
}
```

- You can test the value of a string's character:

```
String word = console.next();
if (word.charAt(word.length() - 1) == 's') {
    System.out.println(word + " is plural.");
}
```

String/char question

- A *Caesar cipher* is a simple encryption where a message is encoded by shifting each letter by a given amount.
  - e.g. with a shift of 3, A → D, H → K, X → A, and Z → C

- Write a program that reads a message from the user and performs a Caesar cipher on its letters:

```
Your secret message: Brad thinks Angelina is cute
Your secret key: 3
The encoded message: eudg wklqnv dqjholqd lv fxwh
```

Strings answer 1

```
// This program reads a message and a secret key from the user and
// encrypts the message using a Caesar cipher, shifting each letter.

import java.util.*;

public class SecretMessage {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("Your secret message: ");
        String message = console.nextLine();
        message = message.toLowerCase();

        System.out.print("Your secret key: ");
        int key = console.nextInt();

        encode(message, key);
    }

    ...
```

Strings answer 2

```
// This method encodes the given text string using a Caesar
// cipher, shifting each letter by the given number of places.
public static void encode(String text, int shift) {
    System.out.print("The encoded message: ");
    for (int i = 0; i < text.length(); i++) {
        char letter = text.charAt(i);

        // shift only letters (leave other characters alone)
        if (letter >= 'a' && letter <= 'z') {
            letter = (char) (letter + shift);

            // may need to wrap around
            if (letter > 'z') {
                letter = (char) (letter - 26);
            } else if (letter < 'a') {
                letter = (char) (letter + 26);
            }
        }
        System.out.print(letter);
    }
    System.out.println();
}
}
```