

Building Java Programs

Chapter 4

Lecture 9: Advanced `if/else`; Cumulative sum;
String/char

reading: 4.2, 4.4 - 4.5

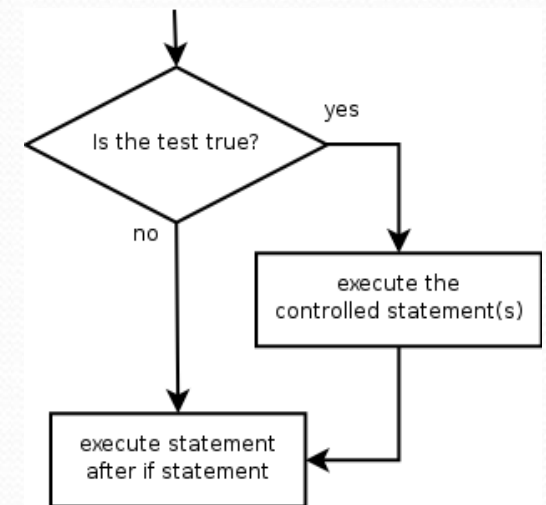
The `if` statement

Executes a block of statements only if a test is true

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

- **Example:**

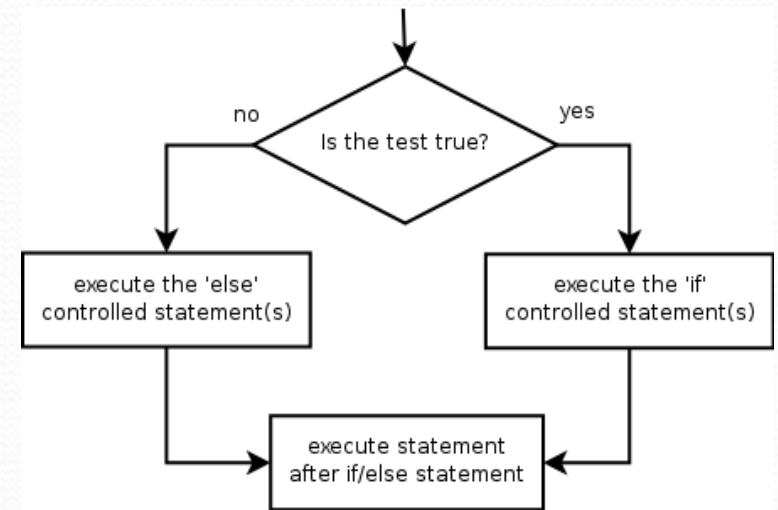
```
double gpa = console.nextDouble();  
if (gpa >= 2.0) {  
    System.out.println("Application accepted.");  
}
```



The if/else statement

Executes one block if a test is true, another if false

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



- **Example:**

```
double gpa = console.nextDouble();  
if (gpa >= 2.0) {  
    System.out.println("Welcome to Mars University!");  
} else {  
    System.out.println("Application denied.");  
}
```

Relational expressions

- `if` statements and `for` loops both use logical tests.

```
for (int i = 1; i <= 10; i++) { ...  
if (i <= 10) { ...
```

- These are `boolean` expressions, seen in Ch. 5.
- Tests use *relational operators*:

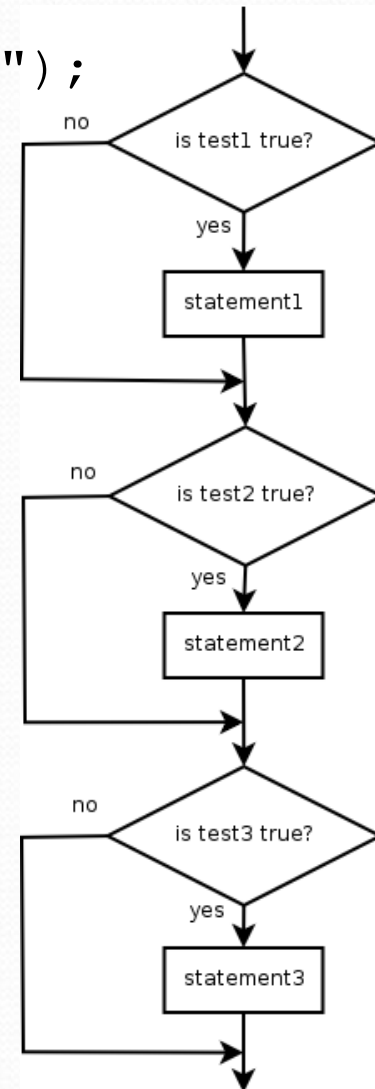
Operator	Meaning	Example	Value
<code>==</code>	equals	<code>1 + 1 == 2</code>	true
<code>!=</code>	does not equal	<code>3.2 != 2.5</code>	true
<code><</code>	less than	<code>10 < 5</code>	false
<code>></code>	greater than	<code>10 > 5</code>	true
<code><=</code>	less than or equal to	<code>126 <= 100</code>	false
<code>>=</code>	greater than or equal to	<code>5.0 >= 5.0</code>	true

Misuse of `if`

- 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!");
}
if (percent < 60) {
    System.out.println("You got an F!");
}
...

```



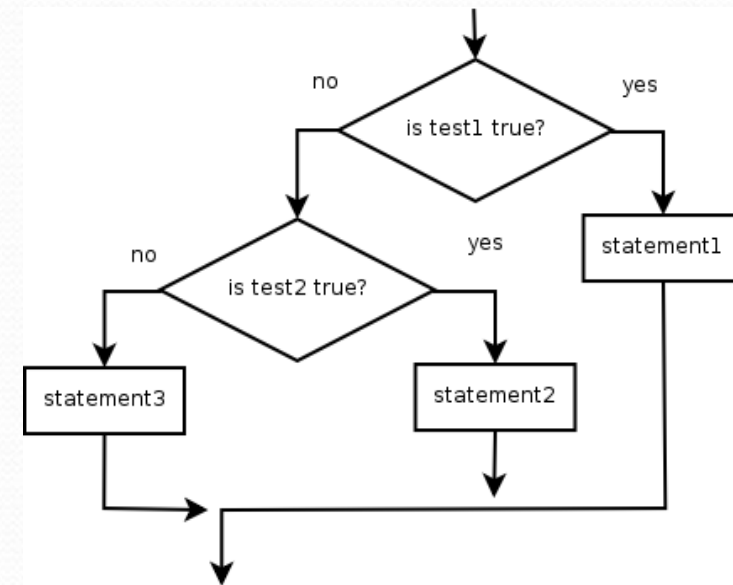
Nested if/else

Chooses between outcomes using many tests

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

- Example:

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



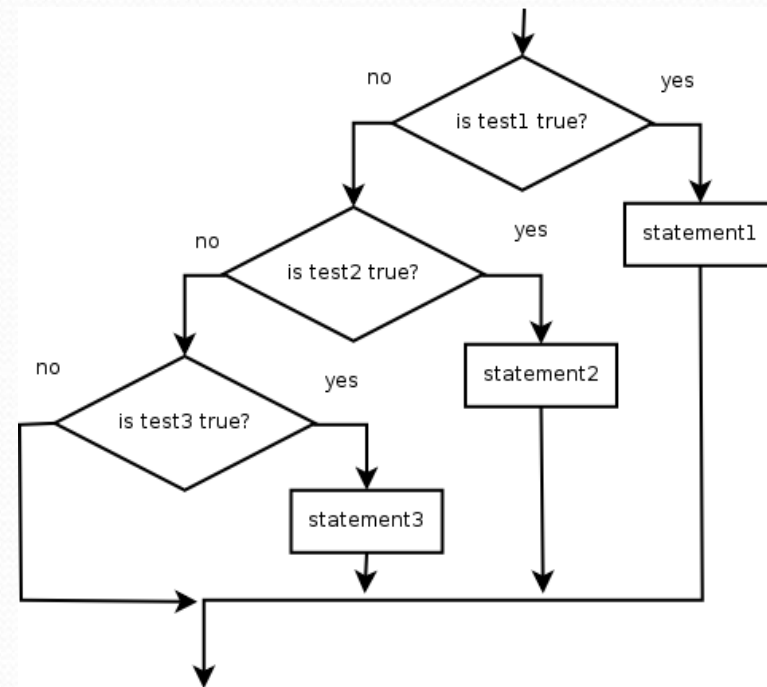
Nested if/else/if

- If it ends with `else`, exactly one path must be taken.
- If it ends with `if`, the code 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("Gold medal!");  
} else if (place == 2) {  
    System.out.println("Silver medal!");  
} else if (place == 3) {  
    System.out.println("Bronze medal.");  
}
```



Nested `if` structures

- exactly 1 path (*mutually exclusive*)

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

- 0 or 1 path (*mutually exclusive*)

```
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**
 - Whether a user is lower, middle, or upper-class based on income.
 - **(2)** nested `if / else if / else`
 - Whether you made the dean's list ($\text{GPA} \geq 3.8$) or honor roll (3.5-3.8).
 - **(3)** nested `if / else if`
 - Whether a number is divisible by 2, 3, and/or 5.
 - **(1)** sequential `if / if / if`
 - Computing a grade of A, B, C, D, or F based on a percentage.
 - **(2)** nested `if / else if / else if / else if / else`

Nested if/else question

Formula for body mass index (BMI):

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

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 output like the following:

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

```
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 BMI = 27.868928571428572
overweight
Person 2 BMI = 23.485824
normal
Difference = 4.3831045714285715
```

Advanced `if/else`

reading: 4.4 - 4.5

Factoring `if/else` code

- **factoring:** Extracting common/redundant code.
 - Can reduce or eliminate redundancy from `if/else` code.
- **Example:**

```
if (a == 1) {  
    System.out.println(a);  
    x = 3;  
    b = b + x;  
} else if (a == 2) {  
    System.out.println(a);  
    x = 6;  
    y = y + 10;  
    b = b + x;  
} else { // a == 3  
    System.out.println(a);  
    x = 9;  
    b = b + x;  
}
```

```
System.out.println(a);  
x = 3 * a;  
if (a == 2) {  
    y = y + 10;  
}  
b = b + x;
```

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

```
// Returns the larger of the two given integers.
```

```
public static int max(int a, int b) {  
    if (a > b) {  
        return a;  
    } else {  
        return b;  
    }  
}
```

- Methods can return different values using `if/else`
 - Whichever path the code enters, it will return that value.
 - Returning a value causes a method to immediately exit.
 - All paths through the code must reach a `return` statement.

All paths must return

```
public static int max(int a, int b) {  
    if (a > b) {  
        return a;  
    }  
    // Error: not all paths return a value  
}
```

- The following also does not compile:

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

- The compiler thinks `if/else/if` code might skip all paths, even though mathematically it must choose one or the other.

Logical operators

- Tests can be combined using *logical operators*:

Operator	Description	Example	Result
&&	and	<code>(2 == 3) && (-1 < 5)</code>	false
	or	<code>(2 == 3) (-1 < 5)</code>	true
!	not	<code>!(2 == 3)</code>	true

- "Truth tables" for each, used with logical values p and q :

p	q	p && q	p q
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

p	!p
true	false
false	true

Evaluating logical expressions

- Relational operators have lower precedence than math; logical operators have lower precedence than relational operators

```
5 * 7 >= 3 + 5 * (7 - 1) && 7 <= 11
```

```
5 * 7 >= 3 + 5 * 6 && 7 <= 11
```

```
35 >= 3 + 30 && 7 <= 11
```

```
35 >= 33 && 7 <= 11
```

```
true && true
```

```
true
```

- Relational operators cannot be "chained" as in algebra

```
2 <= x <= 10
```

```
true <= 10
```

(assume that x is 15)

```
Error!
```

- Instead, combine multiple tests with && or ||

```
2 <= x && x <= 10
```

```
true && false
```

```
false
```

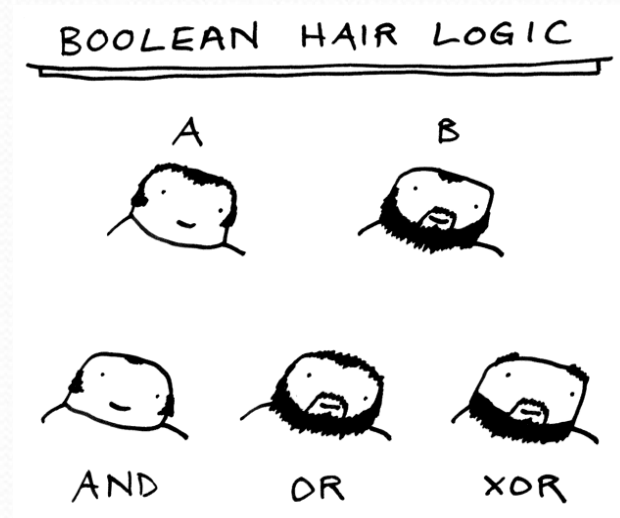
Logical questions

- What is the result of each of the following expressions?

```
int x = 42;  
int y = 17;  
int z = 25;
```

- $y < x \ \&\& \ y \leq z$
- $x \% 2 == y \% 2 \ || \ x \% 2 == z \% 2$
- $x \leq y + z \ \&\& \ x \geq y + z$
- $!(x < y \ \&\& \ x < z)$
- $(x + y) \% 2 == 0 \ || \ !((z - y) \% 2 == 0)$

- **Answers:** true, false, true, true, false



Cumulative algorithms

reading: 4.2

Adding many numbers

- How would you find the sum of all integers from 1-1000?

```
// This may require a lot of typing
```

```
int sum = 1 + 2 + 3 + 4 + ... ;
```

```
System.out.println("The sum is " + sum);
```

- What if we want the sum from 1 - 1,000,000?
Or the sum up to any maximum?
 - How can we generalize the above code?

Cumulative sum loop

```
int sum = 0;
for (int i = 1; i <= 1000; i++) {
    sum = sum + i;
}
System.out.println("The sum is " + sum);
```

- **cumulative sum:** A variable that keeps a sum in progress and is updated repeatedly until summing is finished.
 - The `sum` in the above code is an attempt at a cumulative sum.
 - Cumulative sum variables must be declared *outside* the loops that update them, so that they will still exist after the loop.

Cumulative product

- This cumulative idea can be used with other operators:

```
int product = 1;  
for (int i = 1; i <= 20; i++) {  
    product = product * 2;  
}  
System.out.println("2 ^ 20 = " + product);
```

- How would we make the base and exponent adjustable?

Scanner and cumulative sum

- We can do a cumulative sum of user input:

```
Scanner console = new Scanner(System.in);
int sum = 0;
for (int i = 1; i <= 100; i++) {
    System.out.print("Type a number: ");
    sum = sum + console.nextInt();
}
System.out.println("The sum is " + sum);
```

Cumulative sum question

- Modify the `Receipt` program from Ch. 2.
 - Prompt for how many people, and each person's dinner cost.
 - Use static methods to structure the solution.
- Example log of execution:

How many people ate? 4

Person #1: How much did your dinner cost? 20.00

Person #2: How much did your dinner cost? 15

Person #3: How much did your dinner cost? 30.0

Person #4: How much did your dinner cost? 10.00

Subtotal: \$75.0

Tax: \$6.0

Tip: \$11.25

Total: \$92.25

Cumulative sum answer

```
// This program enhances our Receipt program using a cumulative sum.
import java.util.*;

public class Receipt2 {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        double subtotal = meals(console);
        results(subtotal);
    }

    // Prompts for number of people and returns total meal subtotal.
    public static double meals(Scanner console) {
        System.out.print("How many people ate? ");
        int people = console.nextInt();
        double subtotal = 0.0;           // cumulative sum

        for (int i = 1; i <= people; i++) {
            System.out.print("Person #" + i +
                ": How much did your dinner cost? ");
            double personCost = console.nextDouble();
            subtotal = subtotal + personCost; // add to sum
        }
        return subtotal;
    }
    ...
}
```

Cumulative answer, cont'd.

...

```
// Calculates total owed, assuming 8% tax and 15% tip
```

```
public static void results(double subtotal) {  
    double tax = subtotal * .08;  
    double tip = subtotal * .15;  
    double total = subtotal + tax + tip;  
  
    System.out.println("Subtotal: $" + subtotal);  
    System.out.println("Tax: $" + tax);  
    System.out.println("Tip: $" + tip);  
    System.out.println("Total: $" + total);  
}  
}
```

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.

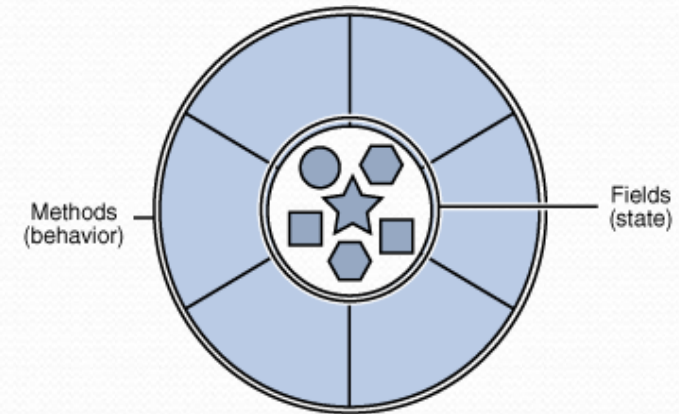
- Solution:

// 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;  
}
```

Objects (usage)

- **object:** An entity that contains data and behavior.
 - *data:* variables inside the object
 - *behavior:* methods inside the object
 - You interact with the methods; the data is hidden in the object.
 - A **class** is a type of objects.



- Constructing (creating) an object:
Type objectName = `new` **Type** (**parameters**) ;
- Calling an object's method:
objectName.methodName (**parameters**) ;

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 = "Ultimate";
```

index	0	1	2	3	4	5	6	7
character	U	l	t	i	m	a	t	e

- First character's index : 0
- Last character's index : 1 less than the string's length
- The individual characters are values of type `char` (seen later)

String methods

Method name	Description
<code>indexOf(str)</code>	index where the start of the given string appears in this string (-1 if not found)
<code>length()</code>	number of characters in this string
<code>substring(index1, index2)</code> or <code>substring(index1)</code>	the characters in this string from <i>index1</i> (inclusive) to <i>index2</i> (<u>exclusive</u>); if <i>index2</i> is omitted, grabs till end of string
<code>toLowerCase()</code>	a new string with all lowercase letters
<code>toUpperCase()</code>	a new string with all uppercase letters

- These methods are called using the dot notation:

```
String starz = "Yeezy & Hova";  
System.out.println(starz.length());    // 12
```

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(1, 7);
System.out.println(s3.toLowerCase());     // "arty s"
```

- Given the following string:

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

- How would you extract the word "Java" ?

Modifying strings

- Methods like `substring` and `toLowerCase` build and return a new string, rather than modifying the current string.

```
String s = "Aceyalone";  
s.toUpperCase();  
System.out.println(s);    // Aceyalone
```

- To modify a variable's value, you must reassign it:

```
String s = "Aceyalone";  
s = s.toUpperCase();  
System.out.println(s);    // ACEYALONE
```

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? Nas
NAS has 3 letters and starts with N
```

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

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

Name border

HELENE
HELEN
HELE
HEL
HE
H
HE
HEL
HELE
HELEN
HELENE
MARTIN
MARTI
MART
MAR
MA
M
MA
MAR
MART
MARTI
MARTIN

- Prompt the user for full name
- Draw out the pattern to the left
- This should be resizable. Size 1 is shown and size 2 would have the first name twice followed by last name twice

Strings question

- Write a program that reads two people's first names and suggests a name for their child

Example Output:

Parent 1 first name? **Danielle**

Parent 2 first name? **John**

Child Gender? **f**

Suggested baby name: JODANI

Parent 1 first name? **Danielle**

Parent 2 first name? **John**

Child Gender? **Male**

Suggested baby name: DANIJO

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("Lance")) {
    System.out.println("Pain is temporary.");
    System.out.println("Quitting lasts forever.");
}
```

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

String test methods

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

```
String name = console.next();
if(name.endsWith("Kweli")) {
    System.out.println("Pay attention, you gotta listen to hear.");
} else if(name.equalsIgnoreCase("NaS")) {
    System.out.println("I never sleep 'cause sleep is the cousin of
        death.");
}
```

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. 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` ?

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'

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, $A \rightarrow D$, $H \rightarrow K$, $X \rightarrow A$, and $Z \rightarrow C$
- 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();
}
}
```