Introduction to Computational Thinking

Lecture #7 Offline:

Strings Processing

Qiao Xiang, Qingyu Song

https://sngroup.org.cn/courses/ct-xmuf25/index.shtml

11/16/2025

This deck of slides are heavily based on cs112 at Yale University and cs101 at UCAS, respectively, by courtesy of Dr. Y. Richard Yang and Dr. Zhiwei Xu.

Outline

- Admin and recap
- Conditionals
- Strings

Discussion

- Functions (methods) that one may need to process text (Strings)?

Roadmap: String Processing

- Access methods, e.g.,
  - `length, substring, charAt, indexOf`
- String generator methods (string->string), e.g.
  - `toLowerCase, toUpperCase, replace, split` (string->strings)
- Conversion from String, e.g.,
  - `Integer.parseInt, Double.parseDouble`
- Input processing, e.g.,
  - `Scanner.next, Scanner.nextLine`
- Output formatting: format string
  - `System.out.printf, String.format`
- Boolean methods, e.g.,
  - `equals, equalsIgnoreCase, startsWith, endsWith, contains, matches`

Strings

- **string**: An object storing a sequence of text characters.
  - Unlike most other objects, a `String` is so common that Java introduces a short cut and you do not need to create String with `new`—to make Strings more similar to native primitive data types

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

```
String name = expression;
```

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

String Internal: Indexes

- A string is a sequence of characters numbered with 0-based *indexes*:

```
String name = "R. Kelly";
```

index	0	1	2	3	4	5	6	7
character	R	.		K	e	l	l	y

  - First character's index : 0
  - Last character's index : 1 less than the string's length

  - The individual characters are values of type `char`

`String` Access Methods

Method name	Description
`length()`	number of characters in this string
`charAt(index)`	character at index
`indexOf(`**`str`**`)`	index where the start of the given string appears in this string (-1 if not found)

- These methods are called using the dot notation:

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

String Method Examples

```
// index      01234567890123456
String s1 = "Xiamen University";
String s2 = "CT 2025, Fall";
System.out.println( s1.length() );          // 17
System.out.println( s1.indexOf("e") );      // 4
```

`String` Generation Methods

Method name	Description

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

`toLowerCase()`	A new string with all lowercase letters
`toUpperCase()`	A new string with all uppercase letters
`replace(`**str1**`, `**str2**`)`	A new string replacing occurrences of *str1* with *str2*

- Java strings are immutable. Hence, each of the above will return a new string. With the current string not modified.

String Method Examples

```
// index      01234567890123456
String s1 = "Xiamen University";
String s2 = "CT 2025, Fall";

System.out.println( s1.length() );          // 17
System.out.println( s1.indexOf("e") );      // 4
System.out.println( s1.substring(7, 17) );// "University"

String s3 = s2.substring(9, 13);
System.out.println( s3.toLowerCase() );    // "fall"
```

Parsing a String

- Instead of substring, indexOf, you can also use scanner to parse a string (in units of tokens only).

```
String s = "Year 2025 is great!";
Scanner scan = new Scanner( s );
String word = scan.next(); // skip Year
int year = scan.nextInt();
```

Exercise

- Given the following string:

```
// index          01234567890123456789
String book = "Building Java Programs";
```

  - How would you extract the word `"Building"` from `book`?
  - More generally, how to extract the first word of a string?

One solution: substring(0, indexOf(" "))

Split a String

- Some common approaches
  - Use indexOf to find the location and then substring to extract a sub string
  - Use scanner to extract tokens

```
String s = "Years 2017 2018 will be great!";
Scanner scan = new Scanner( s );
String word = scan.next(); // skip Year
int year1 = scan.nextInt(); // 2017

word = scan.next();
int year2 = Integer.parseInt( word ); // 2018
```

  - Use split to split a string into multiple strings (more later)

Exercise

- Write a program to convert your “boring” name to a more exciting "gangsta name."
  - last initial
  - *Diddy (stage name of Sean Combs[吹牛老爹, 美国说唱歌手])*
  - first name (all caps)
  - *-izzle (Hip-Pop Ending押韵的韵脚)*

Example Output:

```
Type your name, playa: Marge Simpson
Your gangsta name is "S. Diddy 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();

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

        System.out.println("Your gangsta name is \""
                           + lInitial
                           + ". Diddy "
                           + first + "-izzle\"");
    }
}
```

`String` Boolean (Pattern-Matching) 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
`matches(`**regExp**`)`	whether the string matches a regular expression

```
Scanner console = new Scanner(System.in);
System.out.print("Type your name: ");
String name = console.nextLine();

if (name.startsWith(”Prof. Dr.")) {
   System.out.println(”Are you from Germany?");
} else if (name.endsWith("SquarePants")) {
   System.out.println("And I am Patrick Star!");
}
```

Exercise: Word Rhyme

- Prompt the user for two words and report whether they:
  - *"rhyme"* (end with the same last two letters尾部押韵)
  - *alliterate* (begin with the same letter头部押韵)
  - Example output: (run #1)

```
Type two words: car STAR
They rhyme!
```

(run #2)

```
Type two words: bare bear
They alliterate!
```

(run #3)

```
Type two words: sell shell
They rhyme and alliterate!
```

(run #4)

```
Type two words: extra strawberry
```

Boolean Answer

```
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    System.out.print("Type two words: ");
    String word1 = console.next().toLowerCase();
    String word2 = console.next().toLowerCase();

    boolean isR = rhyme(word1, word2));
    boolean isA = alliterate(word1, word2));

    // output
}

// Returns true if s1 and s2 end with the same two letters.
public static boolean rhyme(String s1, String s2) {
    return s2.length() >= 2
           && s1.endsWith(s2.substring(s2.length() - 2));
}

// Returns true if s1 and s2 start with the same letter.
public static boolean alliterate(String s1, String s2) {
    return s2.length() >= 1
           && s1.startsWith(s2.substring(0, 1));
}
```

Outline

- Strings
  - String access, modify, parse, format
  - String boolean methods
  - String vs char

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

Output:

```
C
S
```

StringExamples.java stringChars

https://en.wikipedia.org/wiki/List_of_Unicode_characters

char vs. int

- Char internal stores the index of a char
- Using `char` in arithmetic expressions will cause char to be converted to an int
- To convert an `int` into the equivalent `char`, type-cast it.

`(char) ('a' + 2)` is `'c'`

```
for (char c = 'a'; c <= 'z'; c = (char)(c+1) ) {
    System.out.print(c);
}
```

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

StringExamples.java charInt method

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
```

char vs. String (char Comparison)

- Consistent with primitive numerical types, you can compare `char` values using relational operators (ordering is alphabetical order):

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

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

ch = word.charAt(0) ;
if ( 'a' <= ch  && ch <= 'z') {
    System.out.println(word
            + "starts w/ a letter");
}
```

StringExamples.java charCompare method

`char` vs. `String` (Comparison)

- <, <=, >, >= are not defined on objects and hence you cannot write str1 <= str2
- != and == are defined on all objects, but they compare *references* (seen later), not values. So == often gives `false` even when two `String`s have the same letters.

```
Scanner console = new Scanner( System.in );

String name = console.next(); // user input Barney

if (name == "Barney") {
    System.out.println("I love you, you love me,");
    System.out.println("We're a happy family!");
}
```

char vs. String (Comparison)

- String comparisons based on content use the method `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!");
}
```

- String defines `compareTo(anotherString)` to compare the lexicographic order of two strings.

StringExamples.java stringCompare method

char vs. String

	String	char
Store	An object w/ a sequence of chars	A primitive value storing a unicode index
Methods	.length, .toUpperCase, …	No methods
==, !=	Compare reference, not value	Compare value
<, <=, >, >=	No defined. Use compareTo(anotherString) method	Lexicographic order
.equals method	Compare content	Not defined

https://symbl.cc/en/unicode-table/

Caesar Cipher

If he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out. If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely D, for A, and so with the others.

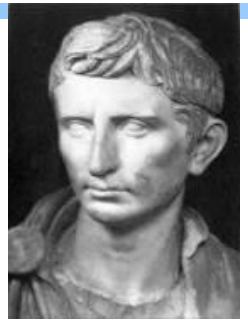
—Suetonius, Life of Julius Caesar 56

(Offline) Caesar Cipher

Plaintext	Ciphertext
attack said zeus	dwwdfn vdlg chxv
προσβάλλω	στςφείξξό

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c

(Offline) Additional Caesar-Style Cipher

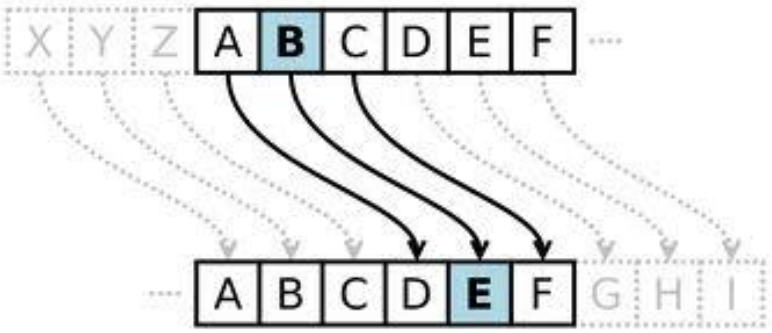


- Augustus
  - Right shift 1 without rotate, with AA for Z
- Mezuzah
  - Shift of one to encrypt the names of God

- Vigenere (a variant) was used by the Confederacy during Civil War

- Mafia boss Provenzano (2006): A (4), …

(Offline)
Why Programming?



Dear Cleopatra,

I love you as big as the Aegean sea, as high as Mount Olympus, as many as the stars of Hyades, Orion, Sirius, and Ursa Major combined... My soul, heart, body and kingdom are yours for the taking. When I think of you, Cleo, my heart beats like that of a gladiator facing death at the claws of a rabid tiger ...

(Offline) Caesar Cipher Encoding (Design 1)

Letter	a	b	c	d	…	…	w	x	y	z
Orig	97	98	99	100	…	…	119	120	121	122
Shift (+3)	100	101	102	103			122	123	124	125
Cipher	100	101	102	103			122	97	98	99

```
int cipher = ch + key;
if (cipher > 'z') cipher -= 26;
ch  = (char)cipher;
```

Q: What if key is 100?

a b c d e f g h i j k l m n o p q r s t u v w x y z

d e f g h i j k l m n o p q r s t u v w x y z a b c

(Offline) Caesar Cipher: Encoding(Design 2)

Letter	a	b	c	d	...	...	w	x	y	z
Orig	0	1	2	3	...	...	22	23	24	25
Shift (+3)	3	4	5	6			25	26	27	28
Cipher (% 26)	3	4	5	6			25	0	1	2

int cipherInt = (origInt + key) % 26

Assumption: a-z mapped to 0 to 25.

How to map a char in a-z to 0 to 25? char – 'a'

a b c d e f g h i j k l m n o p q r s t u v w x y z

d e f g h i j k l m n o p q r s t u v w x y z a b c

(Offline) Caesar Cipher Encoding

Letter	a	b	c	d	…	…	w	x	y	z
Orig	0	1	2	3	…	…	22	23	24	25
Shift (+3)	3	4	5	6			25	26	27	28
Cipher (% 26)	3	4	5	6			25	0	1	2

```
int chRelativeIndex = ch – ‘a’;
int cipherRelInd = (chRelativeIndex + key) % 26;
ch = (char)(cipherRelInd + ‘a’);
```

a b c d e f g h i j k l m n o p q r s t u v w x y z

d e f g h i j k l m n o p q r s t u v w x y z a b c

Offline Exercise/Think:
How to Decode?
(See Backup Slides)

(Offline) Extension: Encrypt a Text File

- Goal: Instead of reading text from terminal, user specifies a file name, which gives the content to be encrypted.

(Offline) CaesarFile using a for Loop

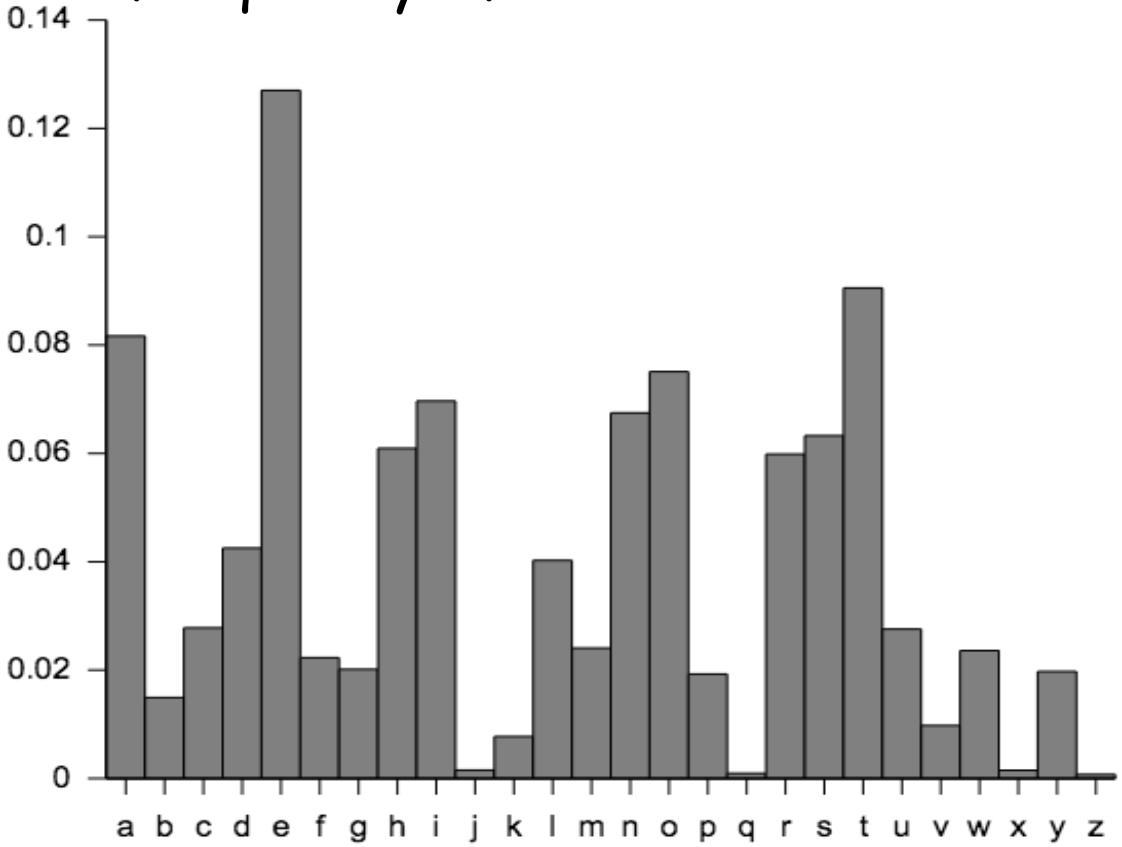
```
public class ReadFile {
    public static void main(String[] args) {
        try {
            Scanner input = new Scanner(new File("data.txt"));
            encode(input, 3);
        } catch (FileNotFoundException e) {
            // print an error message
            System.out.println("File not found exception");
        } // end of catch
    } // end of main
}
```

```
public static void encode(Scanner scan, int key) {

  for (int line = 0; line < NLINES; line ++ ) {
       String text = scan.nextLine();
       String result = Caesar.encode(text, key);
       System.out.println( result );
  }
}
```

(Offline) Exercise: E Caesar Encryption

- Caesar encryption (or permutation ciphers in general) is easy to break because it does not change the frequency of letters.



(Offline) Caesar Cipher: Decoding

Letter	a	b	c	d	…	…	w	x	y	z
Orig	0	1	2	3	…	…	22	23	24	25
Shift (+3)	3	4	5	6			25	26	27	28
Cipher (% 26)	3	4	5	6			25	0	1	2
Back (-3)	0	1	2	3			22	-3	-2	-1
Range (+26% 26)	0	1	2	3			22	23	24	25

(Offline) Unifying Design

$$\text{int cipherInt} = (\text{origInt} + \text{key}) \% 26$$

$$\text{int origInt} = (\text{cipherInt} - \text{key} + 26) \% 26$$

- A single encode method with parameter key
  - To encode
    - encode(key)
  - To decode
    - encode(26 - key)

Caesar.java