CS 259
Computer Programming Fundamentals
Code Cracker Game
Enhancement*

Instructor:
Joel Castellanos

e-mail: joel@unm.edu

Web: http://cs.unm.edu/~joel/

Office:

krl gl drlrpcj cpr ougei tm arjgrvr tfct wfgef tfry wgqf tm ar tpur.

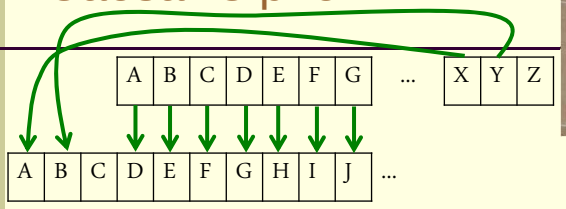
10/26/2016

Code Cracker Game

- Code Cracker is a two player game.
- The first player enters an sentence in clear (unencrypted) text.
- The first player then enters a single keyword.
- The program uses the keyword to create a *substitution cipher*, encrypts the original sentence, and displays the encrypted sentence for the second player.
- The second player must crack the cipher in less than some fixed number of guesses or suffer a barrage of rude (≤PG-13) insults from the program.
- If the original message contained grammatical or spelling errors, then the second player is rubber and the first, glue.

2

Caesar Cipher



The *Caesar Cipher* is named after Julius Caesar, who used it with a wrap around shift of 3, to protect messages of military significance.

> *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

3

Keyword Substitution Cipher

- A *keyword substitution cipher* is a method of encryption by which units of plaintext are replaced with ciphertext according to fixed, bijection defined by the keyword.
- The bijection (one-to-one onto map) is created by first writing out the keyword, removing repeated letters in it, then writing all the remaining letters in alphabetic order.
- Example: keyword= "Byzantine":
  Alphabet without "**BYZANTIE**": **CDFGHJKLMOPQRSUVWX**

a	b	c	d	e	f	g	g	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
B	Y	Z	A	N	T	I	E	C	D	F	G	H	J	K	L	M	O	P	Q	R	S	U	V	W	X

4

Keyword Substitution Cipher Example

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
B	Y	Z	A	N	T	I	E	C	D	F	G	H	J	K	L	M	O	P	Q	R	S	U	V	W	X

The map used in this project is:

- Case-insensitive (convert plain text to all lowercase and cipher text to all uppercase).
- Only maps letters (numbers, spaces and punctuation are not ciphered).

```
advance legion to gaul by new moon.
```

Enciphers to:

```
BASBJZN GNICKJ QK IBRG YW JNU HKKJ.
```

5

What is a Word?

- Player 1 must input a sentence consisting of 7 or more words.
- A word is defined as 1 or more characters that:
  - Contains at least 1 alphabetic character.
  - May contain one or more punctuation marks.
  - Does not contain a space character.
- Examples:

  The woman said, "I want to ask - " when the earthquake began to shake the room.

  Has a word count of 15. That is, even though the dash and ending quotation mark are separated by spaces, they do not count as words.

  Hyphenated words (i.e., sweet-smelling) count as one word.

6

Algorithm for Counting Words

- Let *count* be the number of words already counted (thus, at the start of the string, *count*=0).
- Let *insideword* be a boolean set initially to false.
- Loop through each character, *c*, of the string.
  - If *c* is an alphabetic character, AND *insideword* is false, then increment *count*.
  - If *c* is an alphabetic character, set *insideword* to true.
  - If *c* is a space, set *insideword* to false.

7

Syntax of Plaintext Input Message

Ask player 1 to enter a message with up to 150 characters and at least 7 words.

Print an error message and exit if the sentence:

- Has more than 150 characters.
- Has less than 7 words.
- Contains a non-alphabetic character other than space or the following punctuation marks:

```
! , . ? - : ; ( ) [ ] / " '
```

- Does not end with '.', '!' or '?'.

Display a message showing the input sentence where each lowercase letter of the input is replaced with the same letter in uppercase.

8

Syntax Tests of Input Message

Program must recognize each sentence is valid (does not show an error message) and display each in all capitals.

C-1) "The woods are lovely, dark and deep."

C-2) "The only other sound's the sweep -- of easy wind and downy flake."

9

Syntax Tests of Input Message

Each case must provide meaningful error message.

Empty string error: ""

No ending mark error: "Whose woods these are I think I know"

Too many character error: "His house is in the village though; he will not see me stopping here to watch his woods fill up with snow. My horse must think it queer. To stop without a farmhouse near."

Note: it is totally okay that the above "sentence" is actually 3 sentences. The error is that the message has 170 characters (and the limit is 150).

Too few words error: "My horse must think it queer."

Bad Character error: "He gives his harness bells a shake+to ask if there is some mistake."

10

Syntax Check of keyword

Ask player 1 to enter a keyword with up to 10 characters.

Print an error message and exit if the keyword:

- Has more than 10 characters.
- Has less than 3 characters.
- Contains any non-alphabetic characters.

After player 1 has entered correct message and key word, display a bunch of blank lines and ask player 2 for a guess in the form: $a=b$.

Print an error message and ask again if the guess:

- Is not 3 characters.
- The second character is not '='.
- The first and last characters are not both alphabetic.

11

Syntax Tests for Keyword

Each error case must provide a meaningful error message.

Too few letters error: "" (empty, not null) String.

Too few letters error: "No"

Illegal character error: "Fro st"

Too many letter error: "ABCDEFGHIJK"

12

Syntax of Player 2 Guesses

When player 2 makes a syntax error in a guess, report the error, do not count the guess and allow the player to keep guessing (DO NOT EXIT THE GAME).

Each error case must provide a meaningful error message.

Missing equal error: "AB"
Guess includes spaces error: "A = B"
Bad character error: "A=1"

Guesses are case insensitive: "A=e", "a=e", "a=E", and "A=E" are all identical guesses.

After player 2 enters a guess, exit the game ***ONLY IF***:

1) The player enters a blank line (just presses enter).
2) The entire message has been decrypted.

13

Definitions

- The cypher is a *bijection* (a one-to-one onto map).
- The *cardinality* (size) of the bijection is 26 (the letters of the English alphabet).
- The letters to be encrypted form a *well-ordered set* (every non-empty subset of the set has a ***least*** element in this ordering).
- An *ordinal* denotes a position in a sequence ("first", "second", "third", etc.) of a well-ordered set.
- The ordinal of a plaintext letter in the cypher is: (A=0, B=1, C=2, ... Z=25).

14

Algorithm for Adding Key to Map

1) Start with `String map = ""` (an empty **`String`**).
2) Loop through each character in the key. Let c_i be the i^{th} character in the input key.
3) Before appending c_i to the end of the map, make sure the character is not already in the map:
   a) You can use **`map.indexof(c)`** to return the index of **`c`** in **`map`**.
   b) If **`c`** is already in the map, then **`map.indexof(c)`** will return its index which must be an integer ≥0.
   c) If **`c`** is not in map, **`map.indexof(c)`** returns -1.

15

Finding the Ordinal of a Letter

```
1)  public static void main(String[] args)
2)  {
3)    char letter = 'a';
4)    while (letter <= 'z')
5)    {
6)      int ordinal = letter - 'a';
7)      System.out.println(letter+ ": " + ordinal);
8)
9)      letter++; //Adding 1 to a char
10)   }
11) }
```

```
a: 0    g: 6    m: 12   s: 18   y: 24
b: 1    h: 7    n: 13   t: 19   z: 25
c: 2    i: 8    o: 14   u: 20
d: 3    j: 9    p: 15   v: 21
e: 4    k: 10   q: 16   w: 22
f: 5    l: 11   r: 17   x: 23
```

16

Simple Java Implementation of Cipher Map

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	a	t	u	v	w	x	y	z
Z	E	B	R	A	S	C	D	F	G	H	I	J	K	L	M	N	O	P	Q	T	U	V	W	X	Y

```
String map =
   "ZEBRASCDFGHIJKLMNOPQTUVWXY";

//Then the cipher of "abcz" is:
String cipher =
   map.charAt(0) + map.charAt(1) +
   map.charAt(2) + map.charAt(25);
```

Ordinal of Plaintext Letter

ZEBY

17

Algorithm for Encrypting a String

1) Let **`plainText`** be a trimmed unencrypted String.
2) Let **`cypherText=""`**
3) Loop through each character of **`plainText`** where c is the current character.
   a) If c is a letter, then encrypt it:
      i. Let **`ordinal`** be the ordinal of c.
      ii. Set **`c`** to the character in **`map`** at **`ordinal`**.
   b) If **`c`** is not a letter, leave it unchanged.
   c) Append c to **`cypherText`**.

18

Project Requirements: What verses How

- Usually, project requirements will tell you ***what*** your program needs to do.
- Usually, ***how*** the program does it should be left to the programmer.
- The algorithms presented in these slides are suggestions - not requirements.
- Most assignments in this course will to stick to these rules.
- Professional programmers should carefully review project requirements specifically looking for any that dictate "how" something should be done.
- Usually, the programmer should try to politely novitiate the removal of any "*how*" requirements.

19

Algorithm for Maintaining Cypher State

`String plainText`: Trimmed, lowercase plaintext message.

`String cypherText`:

1) Initialize to "".
2) Then loop through each character of **`plaintext`**:
   a) Use **`map`** to encrypt that character.
   b) Concatenate the encrypted character to the end of **`cyphertext.`**
   c) Used to check correctness of final conjecture.

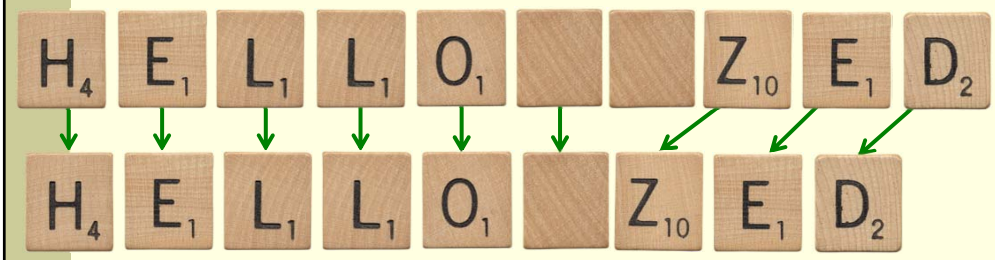
`String workingText:`

1) Initialize to **`cypherText`**.
2) After each correct guess, E=p, loop through **`workingText`** and for each character c_i=E replace with p.

20

Removing Double/Triple Spaces

Before attempting to write Java code, develop an algorithm in English, pencil-&-paper or manipulatives.




```
String original = "HELLO  ZED";
String trimmed = "";
```

loop through each character, c_i, of **`original`**.

If c_i is a space and i = 0, next i

If c_i is a space and c_{i-1} is a space, next i

otherwise append c_i to the end of **`trimmed`**.

21

Example Test Case 1

Plaintext: **`Nostalgia isn't what it used to be.`**

Key: **`Zebras`**

Guesses: **`f=i, k=s, q=t,`** *blank line*

Conjecture: **`x`**

History:

```
KLPQZICFZ FPK'Q VDZQ FQ TPAR QL EA.
KLPQZICiZ iPK'Q VDZQ iQ TPAR QL EA.
K=S
KLPtZICiZ iPK't VDZt it TPAR tL EA.
```

Guess Count: 3

22

Example Test Case 2

Plaintext: **Nostalgia isn't what it used to BE.**

Key: **POVITAMUN**

Guesses: **n=, tbb, N=s, n=I, W=W,** *blank line*

Conjecture: **nostalgia isn't what IT used to be.**

History:

```
FGLQPDMNP NLF'Q WUPQ NQ RLTI QG OT.
N=S
FGLQPDMiP iLF'Q WUPQ iQ RLTI QG OT.
FGLQPDMiP iLF'Q wUPQ iQ RLTI QG OT.
```

Guess Count: 3

23

Example Test Case 3

Plaintext: **All I ask is a chance to prove money can't make me happy.**

Key: **pizi**

Guesses: **j=l, p=a, f=i, n=o, n=p, e=h, x=y, r=s, h=k, S=I, s=t, m=o, z=c, l=n, b=e, k=m, q=u, q=q, u=x, u=b,** *blank line*

History: (*see next slide*)

24

Test Case 3 History

```
PJJ F PRH FR P ZEPLZB SM NQMUB KMLBX ZPL'S KPHB KB EPNNX.
Pll F PRH FR P ZEPLZB SM NQMUB KMLBX ZPL'S KPHB KB EPNNX.
all F aRH FR a ZEaLZB SM NQMUB KMLBX ZaL'S KaHB KB EaNNX.
all i aRH iR a ZEaLZB SM NQMUB KMLBX ZaL'S KaHB KB EaNNX.
N=O
all i aRH iR a ZEaLZB SM pQMUB KMLBX ZaL'S KaHB KB EappX.
all i aRH iR a ZhaLZB SM pQMUB KMLBX ZaL'S KaHB KB happX.
all i aRH iR a ZhaLZB SM pQMUB KMLBy ZaL'S KaHB KB happy.
all i asH is a ZhaLZB SM pQMUB KMLBy ZaL'S KaHB KB happy.
all i ask is a ZhaLZB SM pQMUB KMLBy ZaL'S KakB KB happy.
S=I
all i ask is a ZhaLZB tM pQMUB KMLBy ZaL't KakB KB happy.
all i ask is a ZhaLZB to pQoUB KoLBy ZaL't KakB KB happy.
all i ask is a chaLcB to pQoUB KoLBy caL't KakB KB happy.
all i ask is a chancB to pQoUB KonBy can't KakB KB happy.
all i ask is a chance to pQoUe Koney can't Kake Ke happy.
all i ask is a chance to pQoUe money can't make me happy.
Q=U
Q=Q
U=X
U=B
```

25

Example Test Case 4

Plaintext: **A two-year-old is kind of like having a blender, but you don't have a top for it.**

Key: **Seinfeld**

Guesses: **s=i, s=a, t=t, w=w, m=o, y=y, l=t, l=f,** *blank line*

History:

```
S TWM-YFSQ-MHN BR GBKN ML HBGF ASVBKD S EHFKNFQ, EUT YMU NMK'T ASVF S TMO LMQ BT.
S=I
a TWM-YFaQ-MHN BR GBKN ML HBGF AaVBKD a EHFKNFQ, EUT YMU NMK'T AaVF a TMO LMQ BT.
a tWM-YFaQ-MHN BR GBKN ML HBGF AaVBKD a EHFKNFQ, EUt YMU NMK't AaVF a tMO LMQ Bt.
a twM-YFaQ-MHN BR GBKN ML HBGF AaVBKD a EHFKNFQ, EUt YMU NMK't AaVF a tMO LMQ Bt.
a two-YFaQ-oHN BR GBKN oL HBGF AaVBKD a EHFKNFQ, EUt YoU NoK't AaVF a toO LoQ Bt.
a two-yFaQ-oHN BR GBKN oL HBGF AaVBKD a EHFKNFQ, EUt yoU NoK't AaVF a toO LoQ Bt.
L=T
a two-yFaQ-oHN BR GBKN of HBGF AaVBKD a EHFKNFQ, EUt yoU NoK't AaVF a toO foQ Bt.
```

26

Example Test Case 5

Plaintext: **A two-year-old is kind of like having a blender, but you don't have a top for it.**

Key: **ississippi**

Guesses: **m=t, m=f, m=i, m==o, m=O, I=a,** *blank line*

History:

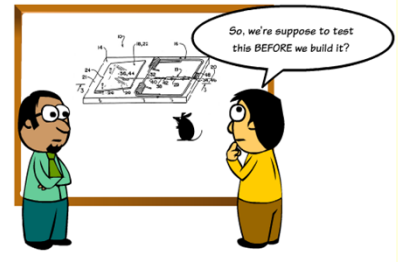
```
I TWM-YBIQ-MJA FR HFLA MC JFHB EIVFLD I SJBLABQ, SUT YMU AML'T EIVB I TMN CMQ FT.
M=T
M=F
M=I
I TWo-YBIQ-oJA FR HFLA oC JFHB EIVFLD I SJBLABQ, SUT YoU AoL'T EIVB I ToN CoQ FT.
a TWo-YBaQ-oJA FR HFLA oC JFHB EaVFLD a SJBLABQ, SUT YoU AoL'T EaVB a ToN CoQ FT.
```

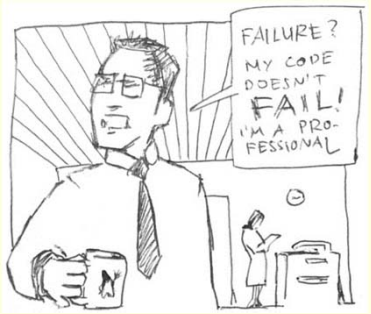
27

Unit Tests

In computer programming, ***unit testing*** is a method by which individual units of source code are tested to determine if they are fit for use.

The goal of unit testing is to isolate each part of the program and show that the individual parts are correct.





28

Unit Tests: Benefits

Find problems early: Unit tests find problems early in the development cycle.

Facilitates change: Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly.

Simplifies integration: Unit testing reduces uncertainty in the units themselves. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier.

Documentation: Unit testing provides a sort of living documentation of the system. By contrast, narrative documentation is more susceptible to drift in the program and becomes outdated.

29

Unit Text Example: `removeExtraSpaces`

```
//Use Java VM option:  -enableassertions
assert(removeExtraSpaces(
    "This is a place too fair.").equals(
    "This is a place too fair."));

assert(removeExtraSpaces(
    "  This is a   place too fair.  ").equals(
    "This is a place too fair."));

assert(removeExtraSpaces("This").equals("This"));
assert(removeExtraSpaces(" This ").equals("This"));
assert(removeExtraSpaces("").equals(""));
assert(removeExtraSpaces("      ").equals(""));
```

30

Grading Rubric (1 of 5)

When the user enters valid input, your program must replace the original plaintext message with a String:

[A 1 point] That is ***fully uppercase***.

[B 2 points] Has each set of adjacent multiple space characters replaced with a single space character and any leading and trailing space characters removed.

[C 2 points] All user input must be case-insensitive.

[D 3 points] Passes message syntax checks.

[E 3 points] Passes keyword syntax checks.

[F 2 points] Passes guess syntax checks.

31

Grading Rubric (2 of 5)

[G 3 points] If Player 2 enters invalid input, then show an error message, and return to prompting the player for a guess.

[H 15 points] Game must display the correctly encrypted, trimmed plaintext message for each of 5 unknown test cases (3 points each).

[I 4 points] If the guess was correct, then, display a complement followed by the message with each not-yet-guessed letter displayed in uppercase and all occurrences of each correctly guessed letter (from the current guess and all past guesses) as lowercase plaintext.

[J 5 points] If the guess is incorrect, display a rude insult (≤PG-13), and the total number of incorrect guesses. Then redisplay the message as above.

32

Grading Rubric (3 of 5)

[K 2 points] Exit the program if player 2 enters a blank guess.

[L 4 points] Your program must display a score and rating when all letters in the message have been decrypted. Then your program must exit (**`System.exit(0)`**).

Score	Rank
0 incorrect guesses	Epic Cryptologist
1 - 3 incorrect guesses	Master Cryptologist
4 to 6 incorrect guesses	Decoder
7 to 10 incorrect guesses	Aspiring Decoder
11 to 15 incorrect guesses	Noob Decoder
> 15 incorrect guesses	Keyboard Masher

33

Grading Rubric (4 of 5)

[M 4 points] Your program must define the class fields:

```
private static final int MAX_MSG_LENGTH = 150;
private static final int MIN_KEY_LENGTH = 3;
private static final int MAX_KEY_LENGTH = 10;
private static final int MIN_WORD_COUNT = 7;
```

You must use these fields within your code such that changing the value of one or more of these fields causes the behavior of your program to change appropriately.

You may assume that when this feature of your code is tested, only reasonable values for these fields will be used. For example:

- All values will always be positive,
- **`MIN_KEY_LENGTH`** will be less than **`MAX_KEY_LENGTH`**,

34

Grading Rubric (5 of 5)

Extra Credit: You are encouraged to negotiate ideas for extra credit with the instructor. All extra credit proposals must be extensions that do not cause your code to fail any of the test cases.

Penalties:

- **[-5 points]:** Code does not adhere to the hallowed CS-259 coding standard: This includes indenting, class, method, and in-line comments, removing all warnings, etc. Note: all 5 points are lost if any ***one*** of the standards is severely broken.

35

Code Cracker Game with GUI (1 of 5)

- Create a JavaFX application with 3 scenes.
- I suggest doing this by having your

```
public void start(Stage stage)
```

  call 3 private methods each that creates all the components for one scene, adds those components to a container (such as a Vbox) and sets the scene to a class field.
- For example, you might have the class fields:
  - `private Scene sceneChooseGameMode;`
  - `private Scene sceneEnterMsgAndKey;`
  - `private Scene sceneGuessLetters;`

36

Code Cracker Game with GUI (2 of 5)

`sceneChooseGameMode` Scene

- This scene is very simple: two buttons and an initially hidden dropdown menu.
- The two buttons allow the player to select a single or two player game.
- If the player clicks the two player game button, switch to the **`sceneEnterMsgAndKey`** scene.
- If the player clicks single player, then:
  1) Read the tab delimited text file of with topic, message and author.
  2) Populate the hidden dropdown menu with the topics and make the dropdown visible.

37

Code Cracker Game with GUI (3 of 5)

If Single player game:

- When the user selects a topic form the topic menu on the **`sceneChooseGameMode`** scene:
  - Without using a key, generate a random substitution cypher map.
  - Branch to the **`sceneGuessLetters`** scene. This is totally the same scene as used in two player mode since after encrypting the message, the single and two player games are the same.

38

Code Cracker Game with GUI (4 of 5)

`sceneEnterMsgAndKey` Scene

- This scene is displayed after the user selects 2 player mode form the **`sceneChooseGameMode`** scene.
- Display edit fields for player 1 to enter message and key.
- Display error messages until the input message and key are ok.
- After input is ok, enable an "encrypt" button.
- When the player clicks "encrypt", branch to the **`sceneGuessLetters`** scene.

39

Code Cracker Game with GUI (5 of 5)

`sceneGuessLetters` Scene

- This same scene is used by both the single and two player game modes after the message has been encrypted.
- This scene must display the encrypted message, provide an interface for guessing which encrypted letter decrypts to which clear text letters, and a way of showing the player's progress in decrypting the message.
- I suggest first getting this to work using an edit field to enter a guess as "A=b".
- Finally, improve the interface by adding dropdown menus, auto graying or removing of already used letters, or whatever you like.

40