CS 112 Introduction to Programming

File as Input; Exceptions; while loops; Basic Arrays

Yang (Richard) Yang
Computer Science Department
Yale University
308A Watson, Phone: 432-6400
Email: yry@cs.yale.edu

Admin.

- PS4 status
- Kevin’s Crash Review on Tuesday from 7 to 8 pm at AKW 400
- Puzzle Day from Friday to Monday
Recap

- **switch statement**

- **String processing**
  - Access methods, e.g.,
    - length, substring, charAt, indexOf
  - New string generator methods, e.g.
    - toLowerCase, toUpperCase, replace
  - Output formatting: format string
    - System.out.printf, String.format
  - Boolean methods, e.g.,
    - equals, equalsIgnoreCase, startsWith, endsWith, contains, matches

**String charAt method**

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

  ```java
  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.

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

  Output:
  
  C
  S
char vs. String

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

- String is an object; it contains methods
  ```java
  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
  ```java
  char c = 'h';
  c = c.toUpperCase();   // ERROR
  ```

char vs. String: Comparison

- You can compare char values with relational operators (ordering is alphabetical order):
  - 'a' < 'b' and 'X' == 'X' and 'Q' != 'q'

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

- Relational operators such as != and == compares objects by references (seen later), so it often gives false even when two Strings have the same letters.

```java
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. int**

- 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'
```

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

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

<table>
<thead>
<tr>
<th>Plaintext</th>
<th>Ciphertext</th>
</tr>
</thead>
<tbody>
<tr>
<td>attack said zeus</td>
<td>dwwdfn vdlg chxv</td>
</tr>
<tr>
<td>προσβάλλω</td>
<td>στςφείξζό</td>
</tr>
</tbody>
</table>
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...

Caesar Cipher: Encoding

<table>
<thead>
<tr>
<th>Letter</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>...</th>
<th>...</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orig</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>...</td>
<td>...</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>Shift</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>Cipher (% 26)</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td>25</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

\[
\text{int cipherInt} = (\text{origInt + key}) \mod 26
\]
Caesar Cipher: Decoding

<table>
<thead>
<tr>
<th>Letter</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>...</th>
<th>...</th>
<th>w</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cipher</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>...</td>
<td>...</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>Shift</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td></td>
<td></td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>Text (+26 % 26)</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>0</td>
<td></td>
<td></td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
</tr>
</tbody>
</table>

\[ \text{int origInt} = (\text{cipherInt} - \text{key} + 26) \mod 26 \]

Unifying Design

\[ \text{int cipherInt} = (\text{origInt} + \text{key}) \mod 26 \]
\[ \text{int origInt} = (\text{cipherInt} - \text{key} + 26) \mod 26 \]

- A single encode method with parameter key
  - To encode
    - \text{encode}(\text{key})
  - To decode
    - \text{encode}(26 - \text{key})
Extension: Encrypt a Text File

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

Reading Specific Files in Program

- To read a file, pass a `File` when constructing a `Scanner`.

  - Example:
    ```java
    File file = new File("mydata.txt");
    Scanner input = new Scanner(file);
    ```

  - or (shorter):
    ```java
    Scanner input =
    new Scanner(new File("mydata.txt"));
    ```
Prompting and Checking Files

- Recall: we can ask the user to tell us the file to read.
  ```java
  // prompt for input file name
  Scanner console = new Scanner(System.in);
  System.out.print("Type a file name to use: ");
  String filename = console.nextLine(); // next()
  Scanner input = new Scanner(new File(filename));
  ```

- You can use `File`'s `exists` method to test existence:
  ```java
  File file = new File("data.txt");
  if (!file.exists()) {
    // try a second input file as a backup
    System.out.println("data file not found!");
    file = new File("data2.txt");
  }
  ```

Compiler Error w/ Scanning File

- The program fails to compile with the following error:
  ```java
  ReadFile.java:9: unreported exception
  java.io.FileNotFoundException;
  must be caught or declared to be thrown
   Scanner input = new Scanner(f);
  ```
Exceptions

- **Exception**: representing a runtime error, e.g.,
  - dividing an integer by 0
  - calling `substring` on a `String` and passing too large an index
  - trying to read the wrong type of value from a `Scanner`
  - trying to read a file that does not exist

- Java allows a method to "throw" an exception, when the method detects a run-time error
- **Consequence**: if a method A invokes a method B that may throw an exception, method A must declare how it handles the exception

Handle Checked Exception

- **Declare a throw clause**
- **Use try/catch**
The **throws Clause**

- **throws clause**: Keywords on a method’s header to state that it may generate an exception (and will not handle it).

- **Syntax**:
  
  ```java
  public static <type> <name>(...) throws <type> {
  ```

- **Example**:
  ```java
  public class ReadFile {
      public static void main(String[] args) throws FileNotFoundException {
  ```

- **Like saying, "I hereby announce that this method might throw an exception, and I accept the consequences if this happens."**

The **try/catch Structure**

- **try/catch**: if an exception happens, program execution jumps to the catch statement, skipping the rest in the try block.

```java
import java.io.*; // for File
import java.util.*; // for Scanner

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

- CaesarFile needs to read a file containing an arbitrary number of lines of text. Hence, it will need an indefinite loop.

- **Indefinite loops** are common in programming design, e.g.,
  - Read from user until input is a non-negative number.
  - Repeat until the user has typed "q" to quit.
  - Print random numbers until a prime number is printed.
  - Search for a solution in a set.

- A key to define an indefinite loop is to identify the loop’s test condition.

Scanner Test for Input Status

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasNext()</td>
<td>returns true if there is a next token</td>
</tr>
<tr>
<td>hasNextInt()</td>
<td>returns true if there is a next token and it can be read as an int</td>
</tr>
<tr>
<td>hasNextDouble()</td>
<td>returns true if there is a next token and it can be read as a double</td>
</tr>
<tr>
<td>hasNextLine()</td>
<td>returns true if there is a next line.</td>
</tr>
</tbody>
</table>

- These methods of the Scanner do not consume input; they just give information about what the next input will be.
**CaesarFile using for Loop**

```java
public static void encode(Scanner scan, int key) {
    for (; scan.hasNextLine(); )
        String text = scan.nextLine();
        String result = Caesar.encode(text, key);
        System.out.println(result);
}
}
```

**Java Repetition /Loop Statements**

- Java has three kinds of loop statements:
  - the **for** loop
  - the **while** loop,
  - the **do** loop, and

- They have the same expressive power

- Which loop statements to use will depend on the context to be more intuitive
**The while loop**

- **while loop**: Repeatedly executes its body as long as a logical test is true.

```java
while (<test>) {
    <statement(s>);
}
```

- Example:
  ```java
  int num = 1; // initialization
  while (num <= 10) { // test
      System.out.print(num + " ");
      num = num * 2; // update
  }
  // output: 1 2 4 8
  ```

**The do/while loop**

- **do/while loop**: Similar to while, except move test at the end:

```java
do {
    <statement(s)>;
} while (<test>);
```

- Example:
  ```java
  int num = 1; // initialization
  do {
      System.out.print(num + " ");
      num = num * 2; // update
  } while (num <= 10); // test
  // output: 1 2 4 8
  ```
**CaesarFile using a while Loop**

```java
public static void encode(Scanner scan, int key) {
    while (scan.hasNextLine()) {
        String text = scan.nextLine();
        String result = Caesar.encode(text, key);
        System.out.println(result);
    }
}
```

**Additional Usage of Caesar 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), ...**
Exercise: Breaking Caesar Encryption

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

![Graph showing frequency distribution]

Frequency Analysis Code (Attempt 1)

```java
int nA = 0, nB = 0, nC = 0, nD = 0, nE = 0,
    nF = 0, nG = 0, ... nZ = 0;

while ( scan.hasNextLine() ) {
    String line = scan.nextLine();
    for (int i = 0; i < line.length(); i++) {
        char ch = line.charAt( i );
        if ( ch == 'a' )
            nA ++;
        else if ( ch == 'b' )
            nB ++;
        else if ( ch == 'c' )
            nC ++;
        ...
    }
    // find the max of nA, nB, nC, ...
}
```
Improving Attempt 1

- We need a way to declare many variables in one step.
- We need an effective scheme to refer to the variables.
- The objective of array is to provide these.

Arrays

- **array**: object that stores many values of the same type.
  - **element**: one value in an array.
  - **index**: array element referred to by position number starting from 0.

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>12</td>
<td>49</td>
<td>-2</td>
<td>26</td>
<td>5</td>
<td>17</td>
<td>-6</td>
<td>84</td>
<td>72</td>
<td>3</td>
</tr>
</tbody>
</table>

```
array: [12, 49, -2, 26, 5, 17, -6, 84, 72, 3]
```

- element 0
- element 4
- element 9
**Array Declaration**

\[
\text{<type>[]} \text{<name>} = \text{new} \ \text{<type>[]} \text{<length>};
\]

- Example:
  
  \[
  \text{int}[] \text{ numbers} = \text{new} \ \text{int}[10];
  \]
  
  \[
  \begin{array}{cccccccccc}
  \text{index} & \text{0} & \text{1} & \text{2} & \text{3} & \text{4} & \text{5} & \text{6} & \text{7} & \text{8} & \text{9} \\
  \text{value} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  \end{array}
  \]

- The length in declaration can be any non-negative integer expression, e.g.,
  
  \[
  \text{int} \ x = 2 \times 3 + 1;
  \]
  
  \[
  \text{double}[] \ \text{data} = \text{new} \ \text{double}[x + 2]; \ // \ \text{arr w/ 9 elem}
  \]

**Accessing Array Elements**

\[
\text{<name>[<index>] \ // access} \\
\text{<name>[<index>] = <value>; \ // modify}
\]

\[
\text{int[]} \ \text{numbers} = \text{new} \ \text{int}[10];
\]

| numbers[0] | 85 |
| numbers[1] | 60 |
| numbers[2] | 100 |
| numbers[3] | 72 |
| numbers[4] | 54 |
| numbers[5] | 89 |
| numbers[6] | 90 |
| numbers[7] | 62 |
| numbers[8] | 93 |
| numbers[9] | 81 |
Accessing Elements: Example

```java
int[] numbers = new int[10];
numbers[1] = 3;
numbers[4] = 99;
numbers[6] = 2;

System.out.println(numbers[2-1]);
if (numbers[4] > 10) {
    System.out.println("numbers[4] > 10.");
}
```

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>99</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

BasicArrays.java

Accessing Elements: Example

- What does this code segment do?

```java
int[] numbers = new int[8];
numbers[0] = 1; numbers[1] = 1;
for (int i = 2; i < 8; i++) {
    numbers[i] = numbers[i-1] + numbers[i-2];
}
```

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>13</td>
<td>21</td>
</tr>
</tbody>
</table>

BasicArrays.java
File Input/Output (I/O)

import java.io.*;

- Create a File object to get info about a file on your drive.
  - (This doesn't actually create a new file on the hard disk.)

```java
File f = new File("example.txt");
if (f.exists() && f.length() > 1000) {
    f.delete();
}
```

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>canRead()</td>
<td>returns whether file is able to be read</td>
</tr>
<tr>
<td>delete()</td>
<td>removes file from disk</td>
</tr>
<tr>
<td>exists()</td>
<td>whether this file exists on disk</td>
</tr>
<tr>
<td>getName()</td>
<td>returns file's name</td>
</tr>
<tr>
<td>length()</td>
<td>returns number of bytes in file</td>
</tr>
</tbody>
</table>
File Paths

- **Absolute path**: specifies a drive or a top "/" folder
  - `C:/Documents/smith/prob6/input/data.csv`
  - *Windows can also use backslashes to separate folders.*

- **Relative path**: does not specify any top-level folder
  - `names.dat`
  - `input/kinglear.txt`
  - *Assumed to be relative to the current directory:*
  - `Scanner input = new Scanner(new File("data/readme.txt"));`
  - *If our program is in C:/prob6, File will look for C:/prob6/data/readme.txt*

ReadFile.java