Lecture 17: Objects: I/O, Data

CS 170, Section 000

29 October 2009
Lecture Plan

- Review of String, File **objects** from last lecture
  - **constructor** methods

- Homework 5: Breaking the Caesar cipher
  - Due Wednesday, Nov 4\(^{th}\) (new date)

- Chapter 8 (conclusion): Reading/Writing Files
  - More objects for Text input/output
  - Finding more information: **Java API documentation**

- Managing Data: objects to make your life easier
  - **ArrayList**: resizable general array object
  - **Hashtable**: general, fast, dictionary/lookup object
The String Class: Ch 8.1-8.4

• Constructing a String

```java
String s1 = new String("Welcome to Java");
String s2 = new String("Welcome");
```

• String comparison

```java
if (s1.equals(s2)){
    System.out.println("strings are the same");
} else{
    System.out.println("s1 != s2");
}
```
String object in more depth (Ch. 7.1)

- Contains sequence of characters
  - object data/a.k.a. **data field** a.k.a. property
- Provides useful methods like equals(), length()
  - define **behavior** of an object, can access object data
String object in more depth (Ch. 7.1)

- Contains sequence of characters
  - object data/a.k.a. data field a.k.a. property
- Provides useful methods like equals(), length()
  - define behavior of an object, can access object data

- **Distinction** between class type and class instance:
  - class String (type of object), that defines operations and type of values it can hold:
    - just like int, double, boolean
  - class instance s1 of type String: an actual variable that holds specific value (e.g., “Java”)
    - just like int vs. int Num=10;
Constructors (Ch 7.3)

• **Constructor**: method invoked when an object is created with the **new** operator
  – Has the same name as the class, e.g., constructor for String class is defined in method String(…)

• **Some** String constructors (of many available):
  [http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html](http://java.sun.com/j2se/1.4.2/docs/api/java/lang/String.html)
  – **String()**: initializes a newly created String object so that it represents an empty character sequence.
  – **String(String original)**: Initializes a newly created String object so that it contains a **copy of the argument string**.
StringBuilder Constructors

• Similar to String, but allows to modify object data (character sequence)
  – Can be created empty, from a string, etc...
  – [http://java.sun.com/j2se/1.5.0/docs/api/java/lang/StringBuilder.html](http://java.sun.com/j2se/1.5.0/docs/api/java/lang/StringBuilder.html)

<table>
<thead>
<tr>
<th>java.lang.StringBuilder</th>
<th>Constructs an empty string builder with capacity 16.</th>
</tr>
</thead>
<tbody>
<tr>
<td>+StringBuilder()</td>
<td>Constructs a string builder with the specified capacity.</td>
</tr>
<tr>
<td>+StringBuilder(capacity: int)</td>
<td>Constructs a string builder with the specified string.</td>
</tr>
<tr>
<td>+StringBuilder(s: String)</td>
<td></td>
</tr>
</tbody>
</table>
**toString()**, capacity, length, setLength, and charAt

- **public `String toString()`**: Returns a string representing the data in this sequence. A new String object is allocated and initialized to contain the character sequence currently represented by this object. This **new String object is returned**.

---

<table>
<thead>
<tr>
<th><code>java.lang.StringBuilder</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>+toString(): String</td>
</tr>
<tr>
<td>+capacity(): int</td>
</tr>
<tr>
<td>+charAt(index: int): char</td>
</tr>
<tr>
<td>+length(): int</td>
</tr>
<tr>
<td>+setLength(newLength: int): void</td>
</tr>
<tr>
<td>+substring(startIndex: int): String</td>
</tr>
<tr>
<td>+substring(startIndex: int, endIndex: int): String</td>
</tr>
</tbody>
</table>
| +trimToSize(): void      | Returns a string object from the string builder.  
Returns the capacity of this string builder.  
Returns the character at the specified index.  
Returns the number of characters in this builder.  
Sets a new length in this builder.  
Returns a substring starting at startIndex.  
Returns a substring from startIndex to endIndex-1.  
Reduces the storage size used for the string builder. |
Example revisited: Checking Palindromes Using StringBuilder

- Rewrite the palindromes program that checks whether a string is a palindrome by using StringBuilder class

- CheckPalindromeStringBuilder.java
File I/O

- A `File` object encapsulates the properties of a file or a path but does not contain the methods for reading/writing data from/to a file.

- In order to perform I/O, we need to use I/O classes: `Scanner` and `PrintWriter`. 
Reading data using Scanner class

- Read data from keyboard
  
  `Scanner input = new Scanner(System.in);`

- Read data from a file
  
  `Scanner input = new Scanner(new File(filename));`
# Writing Data Using PrintWriter

<table>
<thead>
<tr>
<th>java.io.PrintWriter</th>
<th>Creates a PrintWriter for the specified file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>+PrintWriter(filename: String)</td>
<td>Creates a PrintWriter for the specified file.</td>
</tr>
<tr>
<td>+print(s: String): void</td>
<td>Writes a string.</td>
</tr>
<tr>
<td>+print(c: char): void</td>
<td>Writes a character.</td>
</tr>
<tr>
<td>+print(cArray: char[]): void</td>
<td>Writes an array of character.</td>
</tr>
<tr>
<td>+print(i: int): void</td>
<td>Writes an int value.</td>
</tr>
<tr>
<td>+print(l: long): void</td>
<td>Writes a long value.</td>
</tr>
<tr>
<td>+print(f: float): void</td>
<td>Writes a float value.</td>
</tr>
<tr>
<td>+print(d: double): void</td>
<td>Writes a double value.</td>
</tr>
<tr>
<td>+print(b: boolean): void</td>
<td>Writes a boolean value.</td>
</tr>
</tbody>
</table>

Also contains the overloaded println methods.

Also contains the overloaded printf methods.

A println method acts like a print method; additionally it prints a line separator. The line separator string is defined by the system. It is \r\n on Windows and \n on Unix.

The printf method was introduced in §3.6, “Formatting Console Output and Strings.”
PrinterWriter Example

- Write student scores to a file named "scores.txt"

- WriteData.java
Problem: Replacing Text

Write a class named `ReplaceText` that replaces a string in a text file with a new string. The filename and strings are passed as command-line arguments as follows:

```
java ReplaceText sourceFile targetFile oldString newString
```

`ReplaceText.java`
Summary

• Objects
  – data, methods, constructors
  – String, File, I/O objects
Homework 5: Breaking the Caesar cipher (due Monday Nov 2 - Wed Nov 4)


- Command line arguments: encrypted text, dictionary file (both file names given from command line).

- Output: original plaintext

- Simplifying assumptions:
  - No punctuation in input
  - All words in plaintext are in dictionary, and are in lower case.
  - Dictionary is small (less than 2000 words).
HW5 (cont’d): Approach

• Overall approach (brute force/dictionary attack)
  – Assume plain text comes from a dictionary
  – Try all possible shifts until each word in a message is found in the dictionary

• Implementation approach:
  – Read encrypted text from a file, split into words, store into array of words (separate string/entry for each word)
  – Read dictionary, store each word (e.g., in array for now), use individual string for each word
  – Write a method to search for a given word from the encrypted text in the dictionary
    • can use linear or binary search, or something else (Thursday lecture)
HW5 (+5 extra credit)

• Break Viginaire cipher (extension of Caesar)
  – Assume fixed key, length >=5 and <=10
  – Assume all words are from dictionary (as before)
  – Requires repeatedly breaking the Caesar cipher, for each key length, for each possible key *(brute force)*³
What is an ArrayList

- A **dynamically re-sizeable array**, so that the total number of elements does not need to be known when created

- Full documentation: [http://java.sun.com/javase/6/docs/api/java/util/ArrayList.html](http://java.sun.com/javase/6/docs/api/java/util/ArrayList.html)
Capacity vs. Size

- Capacity: how many elements the ArrayList can currently hold without growing
- Size: how many elements are currently in the ArrayList.
Constructors

- **ArrayList()**
  - Constructs an empty ArrayList, has a capacity of 10 elements.

- **ArrayList(int n)**
  - Constructs an empty ArrayList with a capacity of n elements.
Important Methods

• public boolean **add**(Object v)
  – Appends the new element v (such as a string) at the end of the list.
  – Returns true if successful in appending.
• public Object **remove**(int i)
  – Removes the ith element (if it exists) and shifts all elements stored after i one position down
  – The element removed is also returned.
  – If the ith element does not exist an exception is thrown.
Important Methods (contd)

- **public Object get(int i)**
  - Returns the ith element in the ArrayList
  - If the ith element does not exist, an exception is generated.

- **public Object set(int i, Object v)**
  - Sets the ith element to the new value v
  - The previous value is returned.
  - If the ith element does not exist an exception is thrown.
Important Methods (contd)

• **public int size()**
  – Number of elements in the ArrayList

• **public boolean isEmpty()**
  – Returns true if the ArrayList is empty (size == 0)

• **public void clear()**
  – Removes all the elements from the ArrayList
Is an Object in the ArrayList?

- `int indexOf(Object elem)`
  - Searches for the first occurrence equal to the given argument
  - Tests for equality using the equals method
  - Returns position in the list, or -1 if not found

- `boolean contains(Object elem)`
  - Returns true if this list contains the specified element
Important: What’s Really Stored

• ArrayLists (as declared here) store object-references to data of type Object
  – So anything added to an ArrayList must be converted to an Object first
    • Some of this is automatic in Java 5.0
  – Important: when retrieving an item with get(), you must convert it to the type you originally stored
  – If you’re storing primitive types (e.g. int, double) you need to use a “wrapper” class (e.g. Integer, Double)
    • See your favorite Java book for more on those
Adding and Retrieving in Java 5.0

• When putting data into an ArrayList:
  – The parameter to add() or set() can be either a primitive type or an object-reference without a problem

• When retrieving:
  – You **must** cast the return value to the appropriate class

```java
int x = (Integer) myIntList.get(i);
String p = (String) myStringList.get(i);
```
Hash Tables
Example

- We design a hash table for a Dictionary storing entries as (SSN, Name), where SSN (social security number) is a nine-digit positive integer.
- Our hash table uses an array of size \( N = 10,000 \) and the hash function \( h(x) = \) last four digits of \( x \).
**Constructor Methods**

**Hashtable()**
Constructs a new, empty hashtable with a default initial capacity (11) and load factor, which is 0.75.

**Hashtable(int initialCapacity)**
Constructs a new, empty hashtable with the specified initial capacity and default load factor, which is 0.75.

```
Hashtable numbers = new Hashtable();
numbers.put("one", new Integer(1));
numbers.put("two", new Integer(2));
numbers.put("three", new Integer(3));
```

```
Integer n = (Integer)numbers.get("two");
if (n != null) {
    System.out.println("two = " + n);
}
```
Object **get**(Object key)
   Returns the value to which the specified key is mapped in this hashtable.

Object **put**(Object key, Object value)
   Maps the specified **key** to the specified **value** in this hashtable.

Object **remove**(Object key)
   Removes the key (and its corresponding value) from this hashtable

void **clear**()
   Clears this hashtable so that it contains no keys.

boolean **contains**(Object value)
   Tests if some key maps into the specified value in this hashtable.

boolean **isEmpty**()
   Tests if this hashtable maps no keys to values.

boolean **containsKey**(Object key)
   Tests if the specified object is a key in this hashtable.