CSCI 136
Data Structures & Advanced Programming

Jeannie Albrecht

Lecture 5

Feb 19, 2014

Administrative Details

- Lab 2 is today
- We'll go over design at the beginning of lab
- You may work with a partner

Last Time

- Continued reviewing Java and arrays
  - Pokerhand example
  - I'm skipping the rest of PokerHand to save time…
    - Code is posted and I posted extra notes/slides

Today's Outline

- Quickly review Strings in Java (end of Java refresher!)
- Learn about Assertions and pre/post conditions
- Discuss Associations and Vectors
- We need to go quickly…we'll slow down on Friday!

Quick Review: Strings in Java

- Useful methods (also check javadocs)
  - `indexOf(string);`
    `indexOf(string, startIndex);`
  - `substring(start, end); //[start,end)`
  - `charAt(int index);`
  - `equals(other);`
  - `toLowerCase();`
  - `toUpperCase();`
  - `compareTo(string);`
  - `length();`
  - `startsWith(string);`

Using Strings

- Suppose we want to parse an XML listing of our music library
  - XML = eXtended Markup Language
  - XML is used for many things
  - CD info:

```
<CD>
    <TITLE>Big Willie style</TITLE>
    <ARTIST>Will Smith</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <YEAR>1997</YEAR>
</CD>
```

- How can we find and print just the titles?
  - See CDTitles.java
  - Redirecting System.in in Unix: java CDTitles < cds.xml

Moving on…

Pre and Post Conditions

- Recall `charAt(int index)` in Java String class
- What are the pre-conditions for charAt?
  - 0 <= index < length()
- What are the post-conditions?
  - Method returns char at position index in string
- We put pre and post conditions in comments above most methods

```
/* pre: 0 ≤ index < length
 * post: returns char at position index
 */
public char charAt(int index) { … }
```

Pre and Post Conditions

- Pre and post conditions "form a contract"
- Post-condition is guaranteed if method is called when pre-condition is true
- Examples:
  - `s.charAt(s.length() - 1)`: index < length, so valid
  - `s.charAt(s.length() + 1)`: index > length, not valid
- These conditions document requirements that the program should satisfy

Other Examples

- Other places pre and post conditions are useful (see CardPrePost.java):

```
public class Card {
  //pre: TWO <= rank <= ACE
  //pre: CLUBS <= suit <= SPADES
  public Card(int rank, int suit) { … }
```

- Also:

```
//pre: other is a Card
//post: returns true if other has same rank and suit
public boolean equals (Object other) { … }
```

Assert Class

- Pre and post condition comments are useful as a programmer, but it would be really helpful to know as soon as a pre-condition is violated (and return an error)
- The Assert class (in structure5 package) allows us to programmatically check for pre and post conditions

Assert Class

```
public class Assert {
  public static void pre(boolean test, String message);
  public static void post(boolean test, String message);
  public static void condition(boolean test, String message);
  public static void fail(String message);
}
```

Card.java

- Let's look at Card.java again (CardAssert.java)
- This time, we'll use assertions to check for pre-conditions
  - Have to import structure5.* in bailey.jar
- Use `instanceof` to check `Object other` in `equals()` method
  - This allows Java to print **useful** error messages when something is wrong

General Rules about Assert

1. State pre/post conditions in comments
2. Check conditions in code using "Assert"
3. Fail in unexpected cases (such as the default block of a switch statement)

- Any questions?
- You should use Assertions in Lab 2

Moving on…Dictionary Class

- Now we're going to discuss our first general data structure!
- What is a Dictionary?
  - Really just a *map* from word to definition…
  - These mappings are called Associations
  - Given word, lookup and return definition
  - java Dictionary <word>
    - Prints definition

Other Associations

- Word → Definition
- Account number → Balance
- Student name → Grades
- Google:
  - URL → page.html
  - page.html → {a.html, b.html, …} (links in page)
  - Word → {a.html, d.html, …} (pages with Word)
- In general:
  - Key → Value

Association Class

- We want to capture the "key → value" relationship in a general class that we can use everywhere
- What type do we use for key and value instance variables?
  - Object!
  - We can treat any thing as an Object since all classes inherently extend Object class in Java…

Association Class

```
import structure5.*;
class Association {
   protected Object key;
   protected Object value;

   //pre: key != null
   public Association (Object K, Object V) {
        Assert.pre (K!=null, "Null key");
        key = K;
        value = V;
   }

   public Object getKey() {return key;}
   public Object getValue() {return value;}
   public Object setValue(Object V) {
        Object old = value;
        value = V;
        return old;
   }
}
```

Dictionary Class

- Now that we have an Association class, let's implement Dictionary.java
- A Dictionary object is really just a collection of Associations
- What should we use to store our Associations?
  - An array!

Dictionary.java (version 1)

```
protected Association words[] = new Association[5];
public Dictionary() {
   words[0] = new Association("perception", "Awareness of an
              object of thought");
   words[1] = new Association("person", "An individual capable of
              moral agency");
   words[2] = new Association("pessimism", "Belief that things
              generally happen for the worst");
   words[3] = new Association("philosophy", "Literally,
              love of wisdom.");
   words[4] = new Association("premise", "A statement whose
              truth is used to infer that of others");
}

// post: returns the definition of word, or "" if not found.
public String lookup(String word) {
     for (int i = 0; i < words.length; i++) {
         Association a = words[i];
         if (a.getKey().equals(word)) {
             // note cast to recover type from Object
             return (String)a.getValue();
          }
     }
     return "";
}
```

Problems with Arrays

- Dictionary is a fixed size
  - How do we support addWord?
- Possible solutions:
  - Big array and keep a counter of current number of words
    - Error prone. What if we run out of space in array?
  - Big array-like data structure that can dynamically grow and manage itself

Vectors

- Vectors are collections of Objects
- Methods include:
  - `add(Object o), remove(Object o)`
  - `contains(Object o)`
  - `indexOf(Object o)`
  - `get(int index), set(int index, Object o)`
  - `remove(int index)`
  - `add(int index, Object o)`
  - `size(), isEmpty()`

Dictionary.java (version 2)

```
protected Vector defs;
public Dictionary() {
   defs = new Vector();
}

public void addWord(String word, String def) {
   defs.add(new Association(word, def));
}

// post: returns the definition of word, or "" if not found.
public String lookup(String word) {
   for (int i = 0; i < defs.size(); i++) {
       Association a = (Association)defs.get(i);
       if (a.getKey().equals(word)) {
           return (String)a.getValue();
       }
   }
   return "";
}
```

Dictionary.java (version 2)

```
public static void main(String args[]) {
   Dictionary dict = new Dictionary();
   dict.addWord("perception", "Awareness of an object of
        thought");
   dict.addWord("person", "An individual capable of moral
        agency");
   dict.addWord("pessimism", "Belief that things generally
        happen for the worst");
   dict.addWord("philosophy", "Literally, love of
        wisdom.");
   dict.addWord("premise", "A statement whose truth is used to
        infer that of others");
}
```