CSCI 136
Data Structures & Advanced Programming

Lecture 5

Spring 2020

Bill Jannen & Dan Barowy

Administrative Details

- Read and prepare for Lab 2
  - Bring a design document!
  - We'll collect them
- We had our weekly TA meeting last night
  - Unanimously voted the most challenging lab **at the time**
  - Challenges?
    - First time we design multiple classes that work together ("Has-A" relationship)
    - It's impossible to debug if you don't incrementally build and test (don't wait until all classes are written to compile and run!)
    - "The lab handout was long, and I didn't read it all the way through before starting… I missed things at the end that would have been helpful to know"
  - We'll start by talking about our designs with a partner
    - Be prepared to draw your design on a paper (I mean literally draw boxes and arrows…)

Last Time

- Practice Quiz! Topics:
  - Types and user input
  - Static context and creating objects
- Vectors: Extensible arrays
- Java Generics
  - Give the compiler enough information to help us write type-safe code
  - Give us the infrastructure to build classes that are reusable and adaptable

Today

- Inheritance and the `Object` class
  - Object provides default `toString`() and `equals`() methods that we can override
- Associations & dictionary interface
- Vectors of Associations
- Code Samples
  - Dictionary (Associations, Vectors)
  - WordFreq (Vectors, Associations, histograms)

Aside about “static” Variables

- Static variables are shared by all instances of class
- What would this print?

```
public class A {
     static protected int x = 0;
     public A() {
         x++;
         System.out.println(x);
     }
     public static void main(String args[]) {
         A a1 = new A();
         A a2 = new A();
     }
}
```

- Since static variables are shared by all instances of A, it prints 1 then 2. (Without static, it would print 1 then 1.

Aside about "static" Methods

- Static methods are shared by all instances of class
  - Can only access static variables and other static methods

```
public class A {
     public A() { … }
     public static int tryMe() { … }
     public int doSomething() { … }
     public static void main(String args[]) {
            A a1 = new A();
            int n = a1.doSomething();
            A.doSomthing(); //WILL NOT COMPILE
            A.tryMe();
            a1.tryMe();     // LEGAL, BUT MISLEADING!
            doSomething();  // WILL NOT COMPILE
            tryMe();        // Ok
     }
}
```

Memory Management in Java

- Where do “old” objects go?

```
String s = new String(“Dan”);
…
s = new String(“Bill”);
```

- What happens to poor Dan?
- Java has a *garbage collector*
  - Runs periodically to “clean up” memory that had been allocated but is no longer in use
  - Automatically runs in background
- Not true for many other languages!

Class Object

- At the root of all class-based types is the type `Object`
- All class types implicitly *extend* class `Object`
  - Ex.: `CoinStrip`, `Vector`, … `extend Object`

    ```
    Object ob = new CoinStrip(); // legal!
    ```

  - This is the “is a” relationship: a `CoinStrip` *is an* `Object`
  - But we can’t make assign a more general class to a more specific type–it’s unsafe

    ```
    CoinStrip c = new Object(); // NOT legal!
    ```

(All squares are rectangles, but not all rectangles are squares)

Class Object

- Class `Object` defines some methods that all classes should support, including:

```
public String toString()
public boolean equals(Object other)
```

- But we usually *override* (redefine) these methods
  - As we did with `toString()` in our CoinStrip class
  - What about `equals()`?

Object Equality

- Suppose we have the following code:

```
String s1 = new String(“abc”);
String s2 = new String(“abc”);
if (s1 == s2) { System.out.println(“SAME”); }
else { System.out.println(“Not SAME”); }
```

- What is printed?
- How about:

```
String s3 = s2;
if (s2 == s3) { System.out.println(“SAME”); }
else { System.out.println(“Not SAME”); }
```

- ‘==‘ tests whether 2 names refer to same object
  - Each time we use “`new`” a unique object is created

Equality

- What do we really want it to mean for Strings to be equal?
  - They represent the same sequence of characters!
- How would we check this?

```
return (s1.length() == s2.length() &&
        s1.charAt(0) == s2.charAt(0) &&
        s1.charAt(1) == s2.charAt(1) &&
        …
        s1.charAt(s1.length()-1) == s2.charAt(s2.length-1));
```

- This works, but is cumbersome if we must do it every time we compare two Strings…
  - `equals()` to the rescue!

equals()

- For non-primitive types, we can use:
  `if (obj1.equals(obj2)) { … }`
- We should define `equals`() for each class we write
- What makes Two "Eph" objects equal?

```
public boolean equals(Object other) {
  if (other instanceof Eph) {
      Eph otherEph = (Eph) other;
      return this.studentID() ==
             otherEph.studentID();
  } else {
      return false;
  }
}
```

- Note: Must cast `other` to type `Eph`

Next Up: Associations

- In prose, how would you describe a dictionary?
  - A book that maps words to their definitions
- Abstractly, how would you describe the “Interface” of a dictionary?
  - Given a word, you can “get” it’s definition by searching through the list of all word/definition pairs until you find the entry that matches target word
    - Word is the key, definition is the value
- A dictionary data structure stores key-value pairs
    - How would you represent a key-value pair?

Association

- The `Association` class is a generic container class that holds a key and a value.

```
Association<String, String> dictionaryEntry =
    new Association<String, String>(“CS136”,
                                    “An exciting class where
                                    every day is more fun than
                                    the last”);
```

- Accessors: `getKey(), getValue()`
- Setters: `setValue(E value)`
- equals(): two associations are equal if they have keys that are equal, e.g., `a.getKey().equals(b.getKey())`

Association Examples

- Word → Definition
- Account number → Balance
- Student ID → Course Schedule
- 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

```
// Association is part of the structure package
class Association<K, V> {
   protected K key;
   protected V value;

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

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

Example Usage: Word Counts

- Goal: Determine word frequencies in files
- Idea: Keep a Vector of (word, freq) pairs
  - When a word is read…
  - If it’s not in the Vector, add it with freq =1
  - If it is in the Vector, increment its frequency
- How do we store a (word, freq) pair?
  - An *Association*<*String, Integer*>

WordFreq.java

- Uses a Vector<Association<String, Integer>>
  - Each entry is an Association<String, Integer>
  - Each Association is a (String, Integer) pair

- Notes:
  - Include structure5.*;
  - Can create a Vector with an initial capacity and still grow on demand

Notes About Vectors

- Primitive Types and Vectors

```
Vector<Integer> v = new Vector<Integer>();
v.add(5);
```

  - This (technically) shouldn't work! Can't use primitive data types with vectors…they aren't Objects!
  - Java is now smart about some data types, and converts them automatically for us -- called autoboxing

- We used to have to "box" and "unbox" primitive data types:

```
Integer num = new Integer(5);
v.add(num);
…
Integer result = v.get(0);
int res = result.intValue();
```

- Similar wrapper classes (Double, Boolean, Character) exist for all primitives

Vector Summary So Far

- Vectors: "extensible arrays" that automatically manage adding elements, removing elements, etc.
  1. Must store Objects of the same type
  2. Use wrapper classes (with capital letters) for primitive data types (use "Integers" not "ints")
  3. Must define equals() method for Objects being stored for contains(), indexOf(), etc. to work correctly

Application: Dictionary Class

- Again, what is a Dictionary?
  - A *map* from words to definitions…
    - Given a word, lookup and return definition
  - Example: java Dictionary some_word
    - Prints definition of some_word
- What do we need to write a Dictionary class?
  - A Vector of Associations of (String, String)

Dictionary.java

```
protected Vector<Association<String, String>> defs;
public Dictionary() {
   defs = new Vector<Association<String, String>>();
}

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

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

Dictionary.java

```
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");
}
```