

CSCI 136

Data Structures & Advanced Programming

Associations & Dictionaries

Fall 2020

Instructors: (BillJ,BillL)

Outline

- A Ubiquitous Data Structure : Dictionary
- A Useful Class : Association
- Everything's an Object : The Object Class
- A Simple Implementation : Dictionary

Dictionaries

- Look up a value based on a key
 - **Key → Value**
- Examples
 - Account number → Balance
 - **(int → double)**
 - Student name → Grades
 - **(String → String[])**
 - Google:
 - URL → page.html
 - page.html → {a.html, b.html, ...} (links in page)
 - word → {a.html, d.html, ...} (pages with word)
 - **(String → String) or (String → String[])**

Dictionary Goals

- Look-up should be fast
 - We'll return to this many times during course
- Look-up should be unambiguous
 - Each key appears only once
 - But the *value* can be a collection of items
- Operations
 - `contains(key)` → boolean
 - `getValue(key)` → value
 - What if key isn't in dictionary?
 - `add(key,value)` → void? boolean? previous value?
 - `delete(key,value)` → void? boolean? previous value?
- Demo: A Philosopher's Dictionary

Association Class

(from Duane's structure package)

- We want to capture the “key → value” relationship in a general class that we can use everywhere
- Hold a single (key , value) pair
- What types do we use for key and value instance variables?
 - Object!
 - All class types in Java implicitly extend Object class

Class Object

- At the root of all class-based types is the type `Object`
- All class types implicitly *extend* class `Object`
 - `Student`, `Nim`, `String`, ... extend `Object`

```
Object ob = new Student(); // legal!  
Student s = new Object(); // NOT legal!
```
 - `Student`, `Nim`, and `String` are *subclasses* of type `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 (class-based) `Nim` example

Object Equality

- ‘==’ tests whether 2 names refer to same object
 - Each time we use “new” a new object is created
- What do we really want?
 - Depends on object type!
 - String : Same sequence of characters in same order
 - Student : Same name
 - Probably: More likely some unique ID string
 - We could hope that name and age together are good enough....
- Overriding the equals method achieves this
- Let's see an example....

equals()

- We can define equals() for our Student class

```
public boolean equals(Object other) {  
    if ( other instanceof Student ) {  
        Student os = (Student) other;  
        return getName().equals(os.name()) &&  
            getAge() == ot.getAge()      }  
    return false;  
}
```

- Notes

- Must declare other to be of type Object
- Must cast other to type Student
- Use == on primitive types only
- Use instanceof to avoid typecast error
- Pro Tip: add toLowerCase() to avoid upper/lower-case mismatches if case-insensitivity is desired!

Association Class

(from Duane's structure package)

- We want to capture the “key → value” relationship in a general class that we can use everywhere
- Hold a single (key , value) pair
- What types do we use for key and value instance variables?
 - Object!
 - All class types in Java implicitly extend Object class

Association Class

Association Methods

- `public Association (Object key, Object value)`
- `public Object getKey() : return key`
- `public Object getValue() : return value`
- `public Object setValue(Object v)`
 - Returns previous value
- `public boolean equals(Object other)`
 - Returns true *if keys match*; false otherwise

Example: A Philosopher's Dictionary

```
import structure.Association;
class Dictionary {
    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 happen for the worst");

        words[3] = new Association("philosophy",
            "Literally, love of wisdom.");

        words[4] = new Association("premise",
            "A statement used to infer truth of others");
    }
    // implementation continued on next slide..
}
```

Example: A Philosopher's Dictionary

```
// post: returns the definition of word, or "" if not found.

public String lookup(String word) {

    // Note: If words array is not "full", this method would crash
    // If a word wasn't found (Why?)

    for (int i = 0; i < words.length; i++) {

        Association a = words[i];

        // Note: a.getKey() is an Object but word is a String!
        // Java knows to use the equals method for Strings
        if (a.getKey().equals(word)) {
            return (String) a.getValue();
            // note the type-cast above to recover type
        }
    }
    return "";
}
// implementation continued on next slides...
```

Example: Dictionary Implementation

```
// A method to print the defs of words from command line.

public static void main(String args[]) {
    Dictionary dict = new Dictionary();
    System.out.println();

    for (int i = 0; i < args.length; i++) {
        String answer = dict.lookup(args[i]);

        if (!answer.equals(""))
            System.out.println(args[i] + ": " + answer);
        else
            System.out.println("The word '" + args[i] +
                "' was not found.");
    }
    System.out.println();
}
// End of class declaration
```

Association Class Implementation

```
// Association is part of the structure package
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;
    }
}
// Continued on next slide...
```

Association Class Implementation

```
public boolean equals(Object other) {  
    if ( other instanceof Association ) {  
        Association otherAssoc = (Association)other;  
        return getKey().equals(otherAssoc.getKey());  
    }  
    else return false;  
}  
}
```

- **Notes**

- The actual structure package code does NOT do the instanceof check (but it should).
- Instead the method has a “pre-condition” comment that says the other must be a non-null Association!
 - We’ll return to the topic of pre- (and post-) conditions later
- Need to `import structure.Association;`

Summary/Take Aways

- Notes

- Association implements a (key,value) pair type
 - It is part of Duane's `structure` package
- An array of associations allows the building of a simple dictionary
 - We only implemented the `contains` method
 - We'll revisit Dictionaries soon to remedy this
- We were forced to store certain values as variables of type `Object` then *cast them back* to their actual type before being able to invoke methods of their actual type
 - Given `Object o = new Student("Bill J", 18, 'B');`
 - We can't write `o.getName();`
 - We *can* write `o.toString();`
 - Because `Object` class has a `toString` method
 - And happily the correct version (`Student`) of `toString` will be used (if we wrote it!)
- We'll solve this casting "problem" in the next video....

Lecture Ends Here