Today

- Vectors and Associations

- How *generics* in Java can help us avoid objects

- Using Vectors and Associations to create a Dictionary

- Towards the end: example program to calculate frequencies of words in a document
Dictionary

- One of (if not the) most important data structures that exist
- Google, one could argue, essentially just makes Dictionaries
  - OK fine they do way more than that.
  - BUT their ability to scale, particularly in the context of dictionary operations, is what puts them ahead
- We will learn several ways of implementing a dictionary in this course
- First: what is a dictionary? And how can we implement it using Java?
Dictionary data structure

- Store data associated with a set of *keys*
- Goal: for a given key, want to be able to look up the associated data (which we call a *value*)
- For example: let’s say we have a list of words. We want to be able to look up the definition of any word.
  - keys are the words
  - definitions are the values
- For Google: given a keyword, find all websites that contain that keyword
- Given a course name, find the list of all students that are taking that course
Dictionary Goals

• Should be fast
  • We’ll be improving dictionary performance throughout the course

• method contains(key) returns a boolean

• method getValue(key) should get the value associated with a key

• Want to be able to update dictionary: add(key, value) and delete(key, value)

• Each key should appear once. (Why?)
  • Unambiguous lookup! If a query a key, I should know exactly what value I’m getting
Implementing a dictionary

- Any ideas? What do we need to do conceptually?
- Need to store our keys.
  - How?
  - Perhaps in an array
  - Downside of an array?
    - Fixed size: would need to know how many keys are in our dictionary ahead of time
- For each key, need to store an associated value
- How can we store the relationship between a key and its value?
- Let’s make a class for that. We can call it an Association
Association

- Stores a pair of objects (for us, it will be a key and a value)
- What data do we want to store? What is the type of this data?
- What operations do we want?
- Let’s look at a simple Association implementation
public class Association {
    protected Object theKey;
    protected Object theValue;

    public Association (Object key, Object value) {
        theKey = key;
        theValue = value;
    }

    public Object getKey() {return theKey;}
    public Object getValue() {return theValue;}
    public Object setValue(Object value) {
        V old = theValue;
        theValue = value;
        return old;
    }
}
Association: Downsides

- What’s annoying about the current kind of Association?
- Let’s try to use it to store a word and its definition. How do we get the definition?
- The problem: everything we’re storing is an `Object`. We’re not storing its type.
- This is not very Java-y! And, in fact, may lead to issues
- Example: if we mess up the type in Java usually, it’s a compile-time error
- If we get the wrong type here, it’s a run-time error
- Let’s look at a simple program that stores items in an association.
public class UseAssociation {
    public static void main(String[] args) {
        Student a = new Student(19, "Sam", 'A');
        String gradingMessage = "Great job Sam!";
        Association pair = new Association(a, gradingMessage);
        //System.out.println(pair.getKey().getName()); // compile time error
        System.out.println(((Student)pair.getKey()).getName()); // works
        System.out.println(((Student)pair.getValue()).getName()); // run time error
    }
}
What do we really want out of an Association?

- We’d like to be able to store two objects of a *particular type* in our Association

- We always know what type an object is when we store it

- We always told Java “I want an array of *ints*” (or something)

- Can we do the same for Associations? “I want an Association between a String and a String”

- Then methods like `getValue(key)` will return an object of the particular type we want!
Generics
Generics

• A way to create a general class that allows us to \textit{fill in} the type

• We tell Java what kind of Association (etc.) we want

• Can use multiple kinds of Association for various use cases

• But, the underlying code logic remains the same!!
A Generic Association

```java
public class Association<K, V> {
    protected K theKey;
    protected V theValue;
    //pre: key != null
    public Association(K key, V value) {
        theKey = key;
        theValue = value;
    }
    public K getKey() {return theKey;}
    public V getValue() {return theValue;}
    public V setValue(V value) {
        V old = theValue;
        theValue = value;
        return old;
    }
}
```
Every time we use the word Association, we use angle brackets to denote the type of the key and the type of the value.

*every* time you write Association, you should write the type in angle brackets.
A Note on Generics

- Can’t use primitive types with generics in Java

- Instead, need to use the object equivalent of each primitive type: Integer, Character, Boolean, etc.

- An Association that associates an integer with another integer would be Association<Integer, Integer>

- To be clear: can’t do Association<int, int>. But they do exactly the same thing!

- (Java handles casting between int and Integer for you.)
So: we can store a key-value pair using an association

How do we store all key-value pairs?

Could use an array, but those are not great

Right away: can’t resize!

Can be annoying to use

It would be really nice if there was an array-like class that was resizeable and had some useful methods
Vectors
Note on Vectors

• We’ll focus on structure5, the code that comes with the textbook

• You need to place it on the machines you code with this semester. Instructions linked from the lab assignment page; let us know in lab if you have issues

• Java has a built-in, very similar, Vector class (from java.util.Vector). Don’t use this in this class! Use the structure5 version instead.
Vectors

- An OOP version of arrays
- Don’t need to know the size up front
- Come with other useful methods:
  - Check if an item exists in the Vector
  - “Insert” an item in the middle of the Vector
- Implemented with a Java class that we can all read
Vectors

- API can be found in javadocs (linked from lectures page, and here: http://www.cs.williams.edu/~bailey/JavaStructures/doc/structure5/structure5/Vector.html
- Highlights:
  - `get(int)` and `set(int, E)` are equivalent to `[]`
  - `size()` instead of `.length`
  - Extra stuff like `add(int, E)` to add an element at a location (shifting remaining elements down), `contains(E)` to check if the Vector contains a given element
  - and `toString()` (finally!)
We’ll talk about how to implement a Vector next class. Let’s focus on using them for now.

Vectors use generics! *Always* specify the type of items in your Vector every time you write Vector.

So a vector of ints would be of type Vector<Integer>.

Basic idea: can access specific elements using get(int) and set(int,E).
Quick Vector Example

```java
import structure5.*;
public class UseVector{
    public static void main(String[] args) {
        Vector<Integer> newVector = new Vector<Integer>();
        newVector.add(1);
        newVector.add(2);
        newVector.set(1, 4);
        System.out.println(newVector);
    }
}
```
Creating a `Dictionary<K,V>`

- How can we store a dictionary? Specifically, with keys of type `K` and values of type `V`. So a `Dictionary<K,V>`

- Each key-value pair is stored in an `Association<K,V>`

- All of the pairs are stored in a vector. What is the type of item stored in the vector?
  - Each item in the vector is of type `Association<K,V>`
  - So we’re looking for a `Vector<Association<K,V>>`

- Let’s look quickly at how to implement a (very simple, with many missing methods) `Dictionary<K,V>`. We’ll come back to this on Monday.
Count Word Frequencies
Let’s Solve a Problem Together!

- User inputs a sequence of words
- We want to keep track of how many times each word appears
- Let’s plan this out
- What data structure do we want to use?
  - What does our data structure need to store?
  - What operations do we need to support?
- We want to keep track of, for a given word, how many times it appears
- Sounds like each pair is a Association<String, Integer>
- Store all pairs in a Vector< Association<String, Integer> >
Keeping track of word frequency counts

• What happens when a new word comes in?
  • Depends on if it’s stored already or not
  • If it’s stored, increment the relevant count
  • Otherwise, add a new association with count 1

• How can we print things out when we’re done?
  • Loop through the Vector, printing each item
Let’s look at the code!