Week 12 - Wednesday

COMP 2000

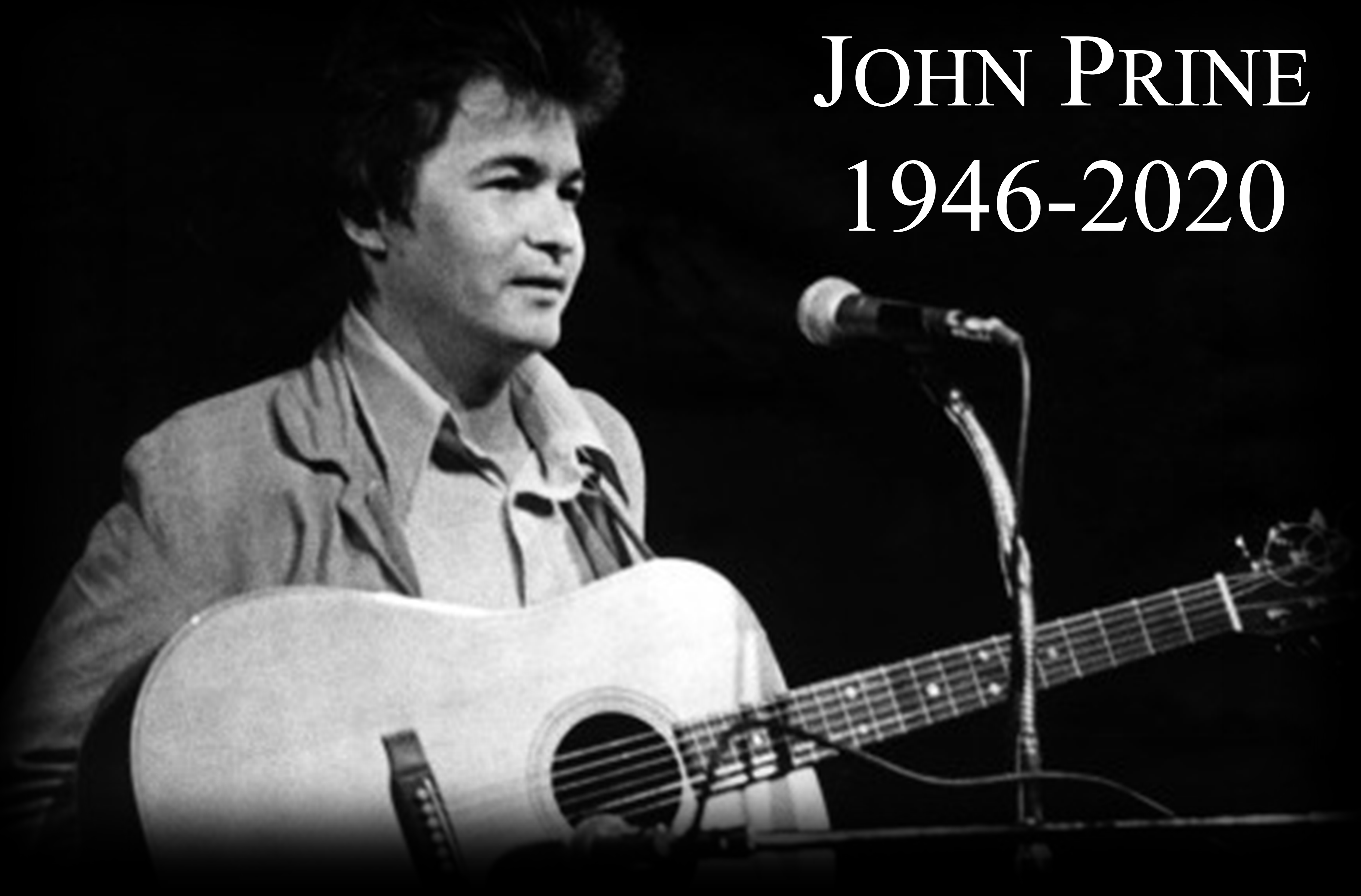
Last time

- What did we talk about last time?
- Exam 2 post mortem
- Java Collections Framework
  - **`List`**
  - **`ArrayList`**
  - **`LinkedList`**

Questions?

Project 4

JOHN PRINE
1946-2020



`List` Practice

List<E> methods

- The **List<E>** interface is one of the biggest you'll ever see
- Here are a few important methods in it

Returns	Method	Description
boolean	**add(E element)**	Adds **element** to the end of the list
void	**add(int index, E element)**	Adds **element** before **index**
boolean	**addAll(Collection<? extends E> collection)**	Adds everything from **collection** to this list
void	**clear()**	Removes everything from this list
boolean	**contains(Object object)**	Returns **true** if this list contains **object**
E	**get(int index)**	Return the element at **index**
int	**indexOf(Object object)**	Returns the first index where something that equals **object** can be found
boolean	**isEmpty()**	Returns **true** if the list is empty
boolean	**remove(int index)**	Remove the element at **index**
E	**set(int index, E element)**	Set the item at location **index** to **element**
int	**size()**	Returns the size of the list

List practice 1 (Fizz Buzz)

- Create an **`ArrayList`** of **`String`** values to hold
- Prompt the user for a positive integer
- From 1 up to the number they enter, add the **`String`** equivalent of that number to the list
- Exceptions:
  - If the number is divisible by 3, add Fizz to the list instead
  - If the number is divisible by 5, add Buzz to the list instead
  - If the number is divisible by both, add Fizz Buzz to the list instead
- Output the list
- Example for 16:
  - **`1, 2, Fizz, 4, Buzz, Fizz, 7, 8, Fizz, Buzz, 11, Fizz, 13, 14, Fizz Buzz, 16`**

List practice 2 (a real job interview question)

- There are ***n*** prisoners standing in a circle, about be executed
- The executions are carried out starting with the ***k***th person, and removing every successive ***k***th person going clockwise until no one is left
- Prompt the user for ***n*** and ***k***
- Determine where a prisoner should stand in order to be the last survivor
- For example, if ***n*** = 5 and ***k*** = 2, the order of executions would be [1, 3, 0, 4, 2] (assuming 0-based numbering)
- **Hint:** Use a list and repeatedly remove indexes

Maps

Maps

- Maps are a kind of data structure that holds a (key, value) pair
- For example, a map might use social security numbers as keys and have **`Person`** objects as the value
- In a map, the keys must be unique, but the values could be repeated
- Both Java and C++ use the name map for the symbol table classes in their standard libraries
- Python calls it a dictionary (and supports it in the language, not just in libraries)
- Maps are also called symbol tables

Concrete example

- Maps are for you can imagine storing as data with two columns, a key and a value
- In this way you can look up the weight of anyone
- However, the keys **must** be unique
  - Ahmad and Carmen might weigh the same, but Ahmad cannot weight two different values
- There are multimaps in which a single key can be mapped to multiple values
  - But they are used much less often
  - All you really need is a map whose values are lists

Name (Key)	Weight (Value)
Ahmad	210
Bai Li	145
Carmen	105
Deepak	175
Erica	205

JCF Map

- The Java interface for maps is, unsurprisingly, **`Map<K,V>`**
  - **`K`** is the type of the key
  - **`V`** is the type of the value
  - Yes, it's a container with **two** generic types
- Any Java class that implements this interface can do the important things that you need for a map
  - **`get(Object key)`**
  - **`containsKey(Object key)`**
  - **`put(K key, V value)`**

JCF implementation

- Because the Java gods love us, they provided two main implementations of the **`Map`** interface
- **`HashMap<K,V>`**
  - **Hash table** implementation
  - To be useful, type **`K`** must have a meaningful **`hashCode()`** method
- **`TreeMap<K,V>`**
  - **Balanced binary search tree** implementation
  - To work, type **`K`** must implement the **`compareTo()`** method
  - Or you can supply a comparator when you create the **`TreeMap`**

Code example

- Let's see some code to keep track of some people's favorite numbers

```java
Map<String,Integer> favorites = new TreeMap<String,Integer>();

favorites.put("John", 42); // Autoboxes int value
favorites.put("Paul", 101);
favorites.put("George", 13);
favorites.put("Ringo", 7);
if( favorites.containsKey("George") )
     System.out.println(favorites.get("George"));
```

JCF `Set`

- Java also provides an interface for sets
- A set is like a map without values (only keys)
- All we care about is storing an unordered collection of things
- The Java interface for sets is **`Set<E>`**
  - **`E`** is the type of objects being stored
- Any Java class that implements this interface can do the important things that you need for a set
  - **`add(E element)`**
  - **`contains(Object object)`**

JCF implementation

- As with maps, there are two main implementations of the `Set` interface
- `HashSet<E>`
  - **Hash table** implementation
  - To be useful, type `E` must have a meaningful `hashCode()` method
- `TreeSet<E>`
  - **Balanced binary search tree** implementation
  - To work, type `E` must implement the `compareTo()` method
  - Or you can supply a comparator when you create the `TreeSet`

Map practice

- An **anagram** is a word or phrase arrived at by scrambling the letters of another word or phrase
- For example, "silent" is an anagram of "listen"
- We can use a **`HashMap`** to determine if one **`String`** is an anagram of another
- We'll make a **`Map<Character,Integer>`** so that we can store the number of times a letter appears

Map practice continued

- Complete the method below that determines if **`string1`** and **`string2`** are anagrams, using the following algorithm:
- For each character in **`string1`**
  - See if it has an entry in the map
  - If it does, add 1 to the number stored there
  - Otherwise, add an entry with the value 1
- Then, for each character in **`string2`**
  - See if it has an entry in the map
  - If it does, subtract 1 from the number stored there or return false if the value is already 0
  - Otherwise return false
- If the two **`String`** values had the same length and this process completed without going below 0 in the map, return true

Upcoming

Next time…

- Sorting libraries
- Custom comparators

Reminders

- Start Project 4
  - Get your teams figured out immediately!