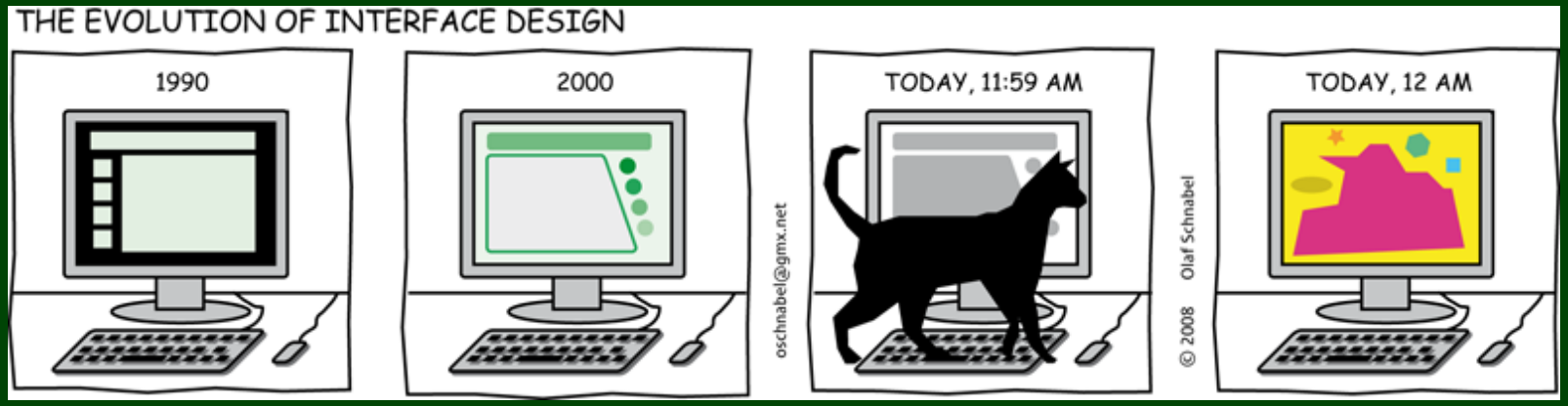
CSE 143

Computer Programming II

More Interfaces & Iterators



Today's Goals

We begin with `ArrayIntList` & `LinkedIntList`.

Our goals are:

- To make an interface that captures the behaviors of an "`IntList`"
- To write a **client** `search` function in both of these classes
- To learn what **iterators** are (and why they might be useful!)
- To re-implement a better version of `search` using iterators

An interface is...

- A promise that you will have certain features
- Giving a name to a group of behaviors

Imagine we were a company making safes (the lock things). We make multiple types of safes. What would they all have in common?

- A way to **lock** the safe
- A way to **unlock** the safe

How about a company making `IntLists`?

- `void add(int value)`
- `int get(int index)`
- `void remove(int index)`
- `void set(int index, int value)`
- `int size()`
- `String toString()`

This basically **is** the interface...

```
1 public interface IntList {
2    void add(int value);
3    int get(int index);
4    void remove(int index);
5    void set(int index, int value);
6    int size();
7    String toString();
8 }
```

Then, to make ArrayIntList and LinkedIntList actually **use** it:

```
1 public class ArrayIntList implements IntList {
2    ...
3 }
4
5 public class LinkedIntList implements IntList {
6    ...
7 }
```

Now, these lines work:

```
1 IntList list = new ArrayIntList();
2 IntList list = new LinkedIntList();
```

Now, let's make a client search function for `IntList`


```
1 /** Returns true if value can be found in list and false otherwise. */
2 public boolean search(IntList list, int value) {
3    for (int i = 0; i < list.size(); i++) {
4       if (list.get(i) == value) {
5          return true;
6       }
7    }
8    return false;
9 }
```

Consider the following:

```
1 IntList arrayList = new ArrayIntList();
2 IntList linkedList = new LinkedIntList();
3 /* Add 1000000 elements to each list... */
4 search(arrayList, 9);
5 search(linkedList, 9);
```

What is the complexity of the two method calls?

- In `ArrayIntList`, `get` is an $\mathcal{O}(1)$ operation
- In `LinkedIntList`, `get` is an $\mathcal{O}(n)$ operation

So, $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$, respectively.

How do `for-each` loops work?

How does Java **KNOW** the ordering?

If you were implementing a for-each `loop` for a type `T`, what would you need to be able to do with the elements in that data structure?

We would need to be able to provide them one after one. . .

Java calls this idea an "`Iterator`".

`Iterator`

The `Iterator` interface allows us to tell Java how to **order** the elements of a data structure:

```
1 public interface Iterator<E> {
2    public boolean hasNext();
3    public E next();
4    public void remove();
5 }
```

This says, "to be an `Iterator`, classes must define `hasNext`, `next`, and `remove`".

This is a lot like how we use a `Scanner`!!

Using a Scanner

```
1 Scanner input = new Scanner(...);
2 while (input.hasNext()) {
3    System.out.println(input.next());
4 }
```

Using an Iterator

```
1 List<Integer> list = new ArrayList<Integer>();
2 ...
3 Iterator<Integer> it = list.iterator();
4 while (it.hasNext()) {
5    System.out.println(it.next());
6 }
```

You've actually been using iterators without knowing it:

Java uses the `iterator()` method to power for-each loops!

You Can't Remove In A foreach Loop!

```
1 Set<String> set = new TreeSet<String>();
2 set.add("hello");
3 set.add("world");
4 for (String s : set) {
5    if (s.startsWith("h")) {
6       set.remove(s);
7    }
8 }
```

OUTPUT

```
>> Exception in thread "main" java.util.ConcurrentModificationException
>>     at java.util.TreeMap$PrivateEntryIterator.nextEntry(TreeMap.java:1115)
>>     at java.util.TreeMap$KeyIterator.next(TreeMap.java:1169)
>>     at Client.main(Client.java:12)
```

ConcurrentModificationException

A ConcurrentModificationException happens when you try to edit a structure that you are looping through in a foreach loop. **You should not try to remove inside a foreach loop! It will fail!**

Using `Iterators` to make search better

```java
/** Returns true if value can be found in list and false otherwise. */
public boolean search(IntList list, int value) {
   Iterator<Integer> it = list.iterator();
   while (it.hasNext()) {
      int next = it.next();
      if (next == value) {
         return true;
      }
   }
   return false;
}
```

Now, they're both $\mathcal{O}(n)$.

Implementing `numberOfOdds` in multiple ways

- As a client...
  - with loops
  - with iterators
- As an implementor...
  - with loops
  - with iterators

See code for solutions.