Week 4.1 — Collections continued

Vector

A Vector is a bit like an array which you can grow and shrink.

`VecDemo.java`

Iterators

Iterators are a more flexible way to move through a collection than a foreach loop.
An iterator refers to a position/item in a collection without needing to give it a number.
Most collections support `.iterator()`.
`VecDemo2.java`
Items can be removed from collections via iterators.
`VecDemo3.java`
Modifying one iterator and then trying to use an older iterator fails fast

Lists

- Walk along list
- Insert an item anywhere in the list
- Remove an item anywhere in the list
- Is item in the list?
- No fixed size limit

Options

So: `List<String>`? Yes.
`=new List<String>`? No.
List is an interface not a class (can't be instantiated).
Some possible classes:

- LinkedList — better for ops which modify the middle of the list.
- ArrayList — better for random access
- Vector

`ListDemo.java`

Sets

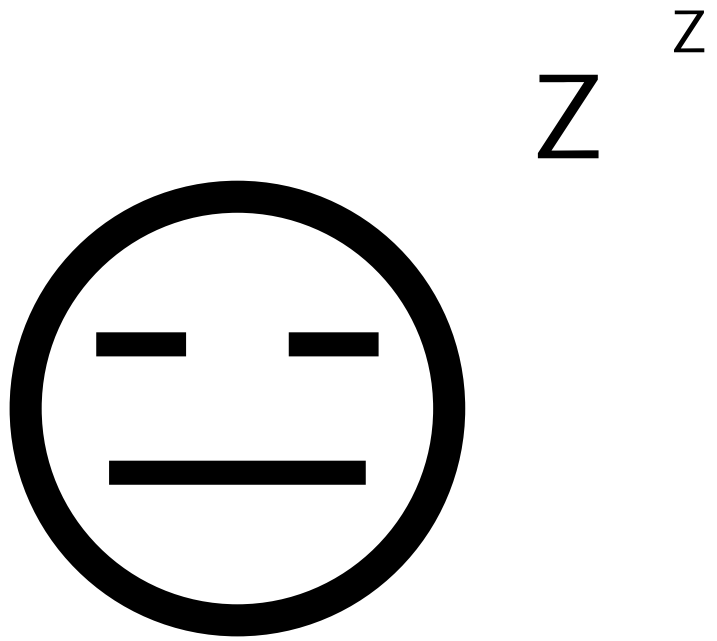
- Iterate over the set (don't assume ordering)
- Add item to set
- Remove item from set
- Is item in set?

Sets do not store duplicates.

Options

- `TreeSet<E>` — `E` needs to implement `Comparable`
- `HashSet<E>` — `E` to have sensible `hashCode()` and `equals()`.

`SetDemo.java`



Map

A map stores key:value pairs. You need to specify a type for each. `Map<Integer, String>` would have `Integer` keys and `String` values.

`MapDemo.java`

Note the use of `.entrySet()` to iterate over map contents.

Sets with custom classes

(This is largly common with Maps, but Sets are simpler).
`CustSet1.java, ...`
It is not enough that interface functions exist, they need to be consistant with what the code expects.
See the documentation for Comparable and Map for example.
In summary:

$$
\begin{aligned}
\text{x.equals(y)} &\Leftrightarrow \text{y.equals(x)} \\
\text{x.equals(y)} &\Rightarrow \text{x.hashCode()==y.hashCode()} \\
\text{x.compareTo(y)} < 0 &\Leftrightarrow \text{y.compareTo(x)} > 0
\end{aligned}
$$

Some things not to do

- **If** you use mutable objects as keys, do not change them once they are in a Set or Map.
- Do not add a Map as a value inside itself.

These touch on the idea of contract programming (later).