Useless Fact of the Day

- Crows in Japan will wait for the traffic lights to turn red at a busy intersection, then place walnuts on the traffic-free crosswalks. Then, when the traffic lights turn green, the cars run over the walnuts, cracking them open. When the lights change and it's time to cross again, the crows descend and eat the walnuts.

Collection Motivation

- Why use collections? So you don't have 100 similar things with different names, like this:

```java
private Sprite enemyAlien1, enemyAlien2, enemyAlien3, ..., enemyAlien100;

public void startGame()
{
    enemyAlien1 = new EllipseSprite(1,1);
    enemyAlien2 = new EllipseSprite(1,1);
    enemyAlien3 = new EllipseSprite(1,1);
    // ...
    enemyAlien100 = new EllipseSprite(1,1);
    // ...
}
```

- Collections enable you to:
  - Easily declare a group of any number of variables (of the same type)
  - Refer to each variable within the collection
  - Group similar variables under one name
  - Group similar code that acts on all the variables in a collection
  - Change the number of variables easily

What are They Good For?
A Small **array**

- Here’s an example of a small collection of integers which stores the first 6 numbers of a certain sequence:

```java
int numbers[6];
numbers = new int[6];
numbers[0] = 0;
numbers[1] = 1;
numbers[2] = 1;
numbers[3] = 2;
numbers[4] = 3;
numbers[5] = 5;
```

- This particular type of collection is called an **array**.

---

**How Arrays Work**

- An array of length $n$ is **indexed** from 0 to $n-1$. For example, our array “numbers,” of length 6:

```java
numbers[0]
nnumbers[1]
nnumbers[2]
nnumbers[3]
nnumbers[4]
nnumbers[5]
```

---

**Available Collections**

- **Arrays**
  - in the `java.util.Collection` package:
    - **ArrayList**
    - **LinkedList**
    - **HashSet**
    - **LinkedHashSet**
  - in the `java.util.Map` package:
    - **HashMap**
    - **TreeMap**

---

**How Arrays Work**

- So:
  - `numbers` is an **array**
  - `numbers[i]` is an **int** (where “$i$” is some index)

- So, if we had an array of Sprites called `enemyAliens`, we could **not** say `enemyAliens.setColor(Color.BLUE)` (because that’s an array of Sprites), but we **could** say `enemyAliens[3].setColor(Color.BLUE)` (because that’s an individual Sprite)
An array...

- has a size which is immutable once created!
- contains a length field
- stores primitives or particular Objects
- is itself an Object
- is indexed 0 to length-1
- can generate an ArrayIndexOutOfBoundsException
- are declared/created/indexed with braces []

An ArrayList...

- size is dynamic!
- has a size() method to give the size (instead of an array’s “length” field)
- stores only Objects, not primitives (but a lot of primitives have object counterparts: for example, “int” has the counterpart “Integer”)
- is itself an Object
- is indexed 0 to length-1
- add something to it with listName.add(theThing)
- get the “i”th thing in it with listName.get(i)

How ArrayLists Work

- An array and its equivalent ArrayList:
  ```java
  int numbers[];
  numbers = new int[6];
  numbers[0] = 0;
  numbers[1] = 1;
  numbers[2] = 1;
  numbers[3] = 2;
  numbers[4] = 3;
  numbers[5] = 5;
  ```

- An ArrayList:
  ```java
  ArrayList<Integer> numbers;
  numbers = new ArrayList<Integer>();
  numbers.add(0);
  numbers.add(1);
  numbers.add(1);
  numbers.add(2);
  numbers.add(3);
  numbers.add(5);
  ```

- We access the contents of the i-th slot of an array with numbers[i], and an ArrayList with numbers.get(i)

  ```java
  numbers.get(3) -> 2
  ```

Special for Loop

- Works for any kind of collection -- but can only be used after the items in the collection have been initialized
- Simple way to access each (and every) variable in a collection in sequence:

  ```java
  int numbers[] = new int[20];
  int sum = 0;
  for (int i = 0; i < numbers.length; i++)
  {
      numbers[i] = i * 2; // store multiples of 2
  }
  ```

  ```java
  // this will take each number in the array "numbers",
  // temporarily call it "x", and add its value to the
  // variable "sum"
  for (int x: numbers)
  {
      sum = sum + x;
  }
  ```