Why use collections?

Consider the code below. What if you wanted 1000 scores? Why is this code not designed well?

```java
int score0, score1, score2, score3, ..., score100;
score0 = input.nextInt();
score1 = input.nextInt();
...
score100 = input.nextInt();
int sum = score0 + score1 + score2 + ... + score100;
double average = sum / 100.0;
```

Collections & Loops

Recall:
- Loops
  - group repeatedly executed code for uniformity
  - make the number of repetitions easily changeable
  - can be combined with selection to make more complex algorithms
Collections Enable

- Easily declaring any number of variables
- Referring to each variable in the collection
- Grouping similar variables under one name
- Grouping similar code that acts on the variables
- Changing the number of variables easily

Why use collections?

The code below uses an array to average the 100 scores. What change would make it do 1000 scores?

```java
int[] scores = new int[100];
double sum = 0;
for (int i = 0; i < scores.length; i++)
{
    scores[i] = input.nextInt();
    sum += scores[i];
}
double average = sum / scores.length;
```

What a Collection looks like

- **scores** is an array
- **scores[i]** is an int
- Arrays are only one way to collect variables

What collections are available?

- **Arrays**
- **java.util.Collection**
  - ArrayList
  - LinkedList
  - HashSet
  - LinkedHashSet
- **java.util.Map**
  - HashMap
  - TreeMap
Arrays

- Store primitives or particular Objects
- Size is immutable
- Contain length field
- Is an Object
- Indexed 0 to length-1
- Can generate ArrayIndexOutOfBoundsException

ArrayLists

- Generic, so must specify what kind of thing to hold
- Size is typically dynamic
- Has a size() method
- Is an Object
- Indexing varies
- Has toArray(Object[]) method for converting to an array.

Using an ArrayList

- Can hold any number of scores, does not need to be known beforehand:

```java
ArrayList<Integer> scores = new ArrayList<Integer>();
double sum = 0;
for (int i = 0; i < scores.size(); i++)
    { 
      scores.add(input.nextInt()); 
      sum += scores.get(i); 
    }

double average = sum / scores.size();
```

- Note, must hold Integer objects instead of int primitives --- usually not a problem

Enhanced for loop

- Works for any kind of collection
- Simpler syntax for accessing each variable in the collection:

```java
// given array scores, with each value initialized
double sum = 0;
for (int current : scores)
    { 
      sum += current; 
    }

// given ArrayList scores, with each value initialized
sum = 0;
for (Integer current : scores)
    { 
      sum += current; 
    }
```
Practice

- Declare an array of integers
- Initialize the array to be able to hold 10 integers
- Set the values in the array to be the first ten squares (i.e. 1, 4, 9, 16, 25 ...)
- Sum the values
- Output the average
- Alter your code to do the first 100 integers instead

More Practice

- Change the code in pong so that the paddles and walls are stored in a collection instead of individual variables
- Play wackadot with a random number of enemy dots (e.g., from 3 to 10) set at the beginning of each game