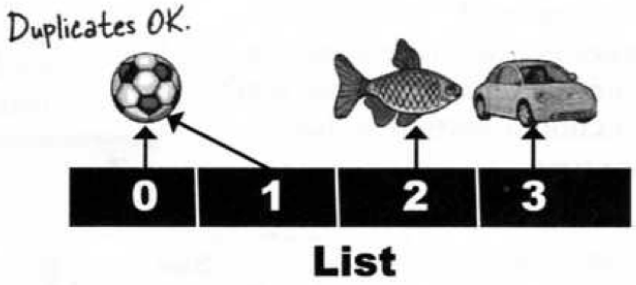
# Collections

- **LIST - when *sequence* matters**

  Collections that know about ***index position.***

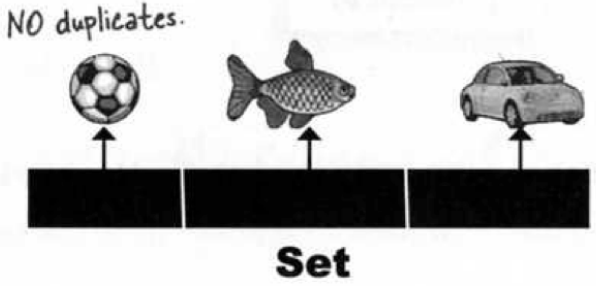
  Lists know where something is in the list. You can have more than one element referencing the same object.




- **SET - when *uniqueness* matters**

  Collections that ***do not allow duplicates.***

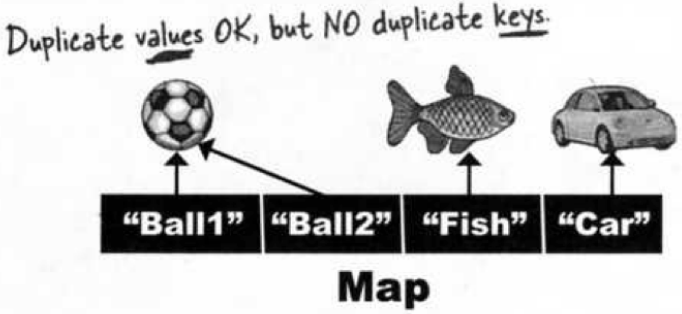
  Sets know whether something is already in the collection. You can never have more than one element referencing the same object (or more than one element referencing two objects that are considered equal—we'll look at what object equality means in a moment).




- **MAP - when *finding something by key* matters**

  Collections that use ***key-value pairs.***

  Maps know the value associated with a given key. You can have two keys that reference the same value, but you cannot have duplicate keys. Although keys are typically String names (so that you can make name/value property lists, for example), a key can be any object.




From
Head First Java p. 557

# ArrayList

- `ArrayList` is an example of a list or sequence (and also array) -like collection
  - No need to decide on collection size at creation time
  - Elements are accessible by index…
    - `add(i, e)` - adds `e` into position `i`, `set(i, e)` - replaces element at position `i` with `e`, `get(i)` - returns element at position `i` (and more)
  - … and in list fashion
    - `add(e)` - adds to end of `ArrayList`
  - … and more
    - `isEmpty()`, `size()`, `contains(o)`, `remove(o)`

(See http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html)

# Type parameters

- ArrayList class is declared like this:

```
public class ArrayList<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
```

- `<E>` is a type parameter - it is instantiated when an `ArrayList` instance is created:

```
ArrayList<Dog> dogs = new ArrayList<Dog>();
```

- Now we have variable `dogs` which refers to an `ArrayList` whose elements are of class `Dog`
- See: http://docs.oracle.com/javase/tutorial/java/generics/why.html and http://docs.oracle.com/javase/tutorial/java/generics/types.html

# Iteration

- It is (of course) possible to iterate through an ArrayList by using indexing:

```
for(int i = 0; i < dogs.size(); i++)
  System.out.print(dogs.get(i));
```

- But we also have a much handier (and faster) enhanced for -loop:

```
for(Dog d: dogs)
  System.out.print(d + " ");
```

- Note: enhanced version works with arrays, too.
- See: http://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html

# Printing objects

- Class `Object` has `toString()` method that gives a `String` representation of the object. Default implementation inherited from `Object` is not very useful but it can be overridden:

```
public String toString() {
  return name + ", " + weight;
}
```

# Sorting

- There are collections that keep elements sorted always (trees, for example - more later). To sort other kind of collections use `Collections` class method `sort()` (or one of its variants).

```
Collections.sort(dogs);
```

- But… how do we compare two instances of class `Dog`? Need to implement `Comparable` interface:

```
public class Dog implements Comparable<Dog> {
…
public int compareTo(Dog d) {
   return this.weight - d.weight;
}
```

# Sorting with other criteria

- One might want to sort a collection using ordering other than the one imposed by collection element class `compareTo()`. Create a new class that implements `Comparator` interface for the element class:

```
public class DogCompare implements Comparator<Dog> {
```

- … and implement `Comparator` interface method

```
  public int compare(Dog d1, Dog d2) {
     return d1.getName().compareTo(d2.getName());
  }
}
```

# Ordered collection - `TreeSet`

- A `TreeSet` is a *set* data structure - one that allows *no duplicate* values - that has been implemented as a *tree* (more on trees in data structures and algorithms course)
- The elements that are inserted into `TreeSet` must be comparable (there must be a way to tell if an object is larger, smaller, or equal to another object)
  - Elements must be of a class that implements `Comparable`, or a `Comparator` is provided in the `TreeSet` constructor

# HashMap

- When are two objects considered equal, again?
  1. Reference equality: `aCar == bCar` - variables refer to the same object instance in memory
  2. Object equality, two conditions must be met:
     1. `aCar.hashCode() == bCar.hashCode()`, and
     2. `aCar.equals(bCar)`
- `hashCode()` method computes an integer value for an object - if two objects are equal, their hash values are equal. (Note that hash value equality does not imply equality.)

# Reading list

- Head First Java, pages 529 - 579
- Oracle tutorial trails:
  - https://docs.oracle.com/javase/tutorial/collections/intro/index.html
  - https://docs.oracle.com/javase/tutorial/collections/interfaces/index.html
  - https://docs.oracle.com/javase/tutorial/collections/implementations/index.html
  - (https://docs.oracle.com/javase/tutorial/collections/streams/index.html) - read this if you want to take a look at something new in Java 8