Individual Objects vs. Arrays of Objects

Working with an Individual Object

3 steps:

** DECLARE THE OBJECT

** INSTANTIATE THE OBJECT BY CALLING new

** USE METHODS OF THE OBJECT

Working with an Individual Object

3 steps:

** DECLARE THE OBJECT

```
    private Puppy myNewDog;
```

** INSTANTIATE THE OBJECT BY CALLING new

```
    myNewDog = new Puppy("Max", "Pug", 12);
```

** USE METHODS OF THE OBJECT

```
    myNewDog.setWeight(myNewDog.getWeight()+5);
```

Problem with Individual Objects

If we have a large number of objects of the same type, and we want to call the same method to process all of the objects, it is repetitive to use individually-named objects (too much copying/pasting!).

```
firstDog.addOneToAge();
secondDog.addOneToAge();
        :
nineHundredFifthDog.addOneToAge();
```

A better approach: use an Array of Objects

An array is a group of objects all of the same type that are stored together in memory.

The array has a name, and individual members of the array are identified by an index, or subscript, number

Important: First item in the array is always index number 0

Last item in the array has an index number 1 LESS than the number of elements in the array

An element in an array is just an object with a two-part name....

To process an object that is stored in an array, we need to use the name of the object, just like we would for any other object

An object in an array is identified by two things:

- the name of the array
- the index number of the object in the array, inside square brackets

Example: if we have an array of **`Puppy`** objects called **`litter`**, the third object in the array is

```
litter[2]
```

What's so great about this?

The key idea that makes arrays so powerful for programming is that the index number can be represented by a variable, and we can change the value of the variable with a loop

Here's the example from a previous slide, using an array of Puppy objects:

```
litter[0].addOneToAge();      // first Puppy
litter[1].addOneToAge();      // second Puppy
litter[2].addOneToAge();      // third Puppy
        :
litter[904].addOneToAge();    // 905th Puppy
```

Use a loop to avoid all the duplication

The code on the previous slide was all the same, except for the value in the square brackets. We can generate those values with a loop!

```
for (Integer x = 0; x < 905; x++) {
    litter[x].addOneToAge();
}
```

Notice inside the loop body, we use a variable inside the square brackets, not a specific number

A better loop heading

To process all of the elements in an array, the test for loop continuation should use this notation:

```
for (Integer x = 0; x < litter.length; x++) {
    litter[x].addOneToAge();
}
```

The name of the array followed by .length returns the number of elements in the array; they are numbered 0 to (litter.length – 1)

Working with Arrays of Objects

4 steps:

** DECLARE THE ARRAY

** INSTANTIATE THE ARRAY (this specifies the size of the array)

** INSTANTIATE EACH OBJECT IN THE ARRAY

** USE METHODS OF THE OBJECTS IN THE ARRAY

Working with Arrays of Objects

4 steps:

** DECLARE THE ARRAY

```
    private Puppy[] litter;
```

** INSTANTIATE THE ARRAY (this specifies the size of the array)

```
    litter = new Puppy[905];
```

** INSTANTIATE EACH OBJECT IN THE ARRAY

```
    litter[0] = new Puppy("Max", "Pug", 12);
    litter[1] = new Puppy("BonBon", "Poodle", 15);
    litter[2] = new Puppy("Sunshine", "Great Dane", 120);
          :
```

Working with Arrays of Objects

4 steps (continued):

** USE METHODS OF THE OBJECTS IN THE ARRAY (with a loop!)

```
for (Integer index = 0; index < litter.length; index++) {
    System.out.println( litter[index].toString() );
}
```

Individual Objects
vs.
Arrays of Objects