References

Arrays

An array is declared like a variable in the following way:

```
Class[] name = new Class[size];
```

This is actually an abbreviation of

```
Class[] name;
name = new Class[size];
```

Something like this actually happens when you instantiate an object.

Object instantiation

To instantiate an object,

```
Dog lucy = new Dog();
```

This is actually an abbreviation of

```
Dog lucy;
lucy = new Dog();
```

Reference variables

- `Dog lucy;` — This creates a *reference variable* for a `Dog` object. Its purpose is to store the address of a `Dog` object.
- `lucy = new Dog();` — This instantiates an actual Dog object and its address is stored in `lucy`.

Thus, `lucy` itself is not a `Dog` object. It is a *reference* that points to a `Dog` object.

Parameters and arguments

Many Java methods have parameters. For example, consider:

```
public void moo(int a, int b) {
  ...
}
```

When calling this method, a caller must pass two `int` values for parameters `a` and `b`.

Parameters and arguments

A variable inside a method whose value is to be specified by a caller is called a *parameter*.

A value being passed by a caller to a parameter is called an *argument*.

A parameter is always a *variable*.

An argument is always a *value*.

Parameters and arguments

Suppose that `Coo` is a class. Consider:

```
public void moo(Coo a, Coo b) {
  ...
}
```

When calling this method with

```
c.moo(x, y);
```

What are being passed under `x` and `y`?

Parameters and arguments

Suppose that `Coo` is a class. Consider:

```
public void moo(Coo a, Coo b) {
  ...
}
```

When calling this method with

```
c.moo(x, y);
```

the value of `x` and `y` are passed, where these values are addresses of `Coo` objects.

Arrays

To declare an array of 100 `Dog` objects named `dogArray`, put

```
Dog[] dogArray = new Dog[100];
```

This is an abbreviation of

```
Dog[] dogArray;
dogArray = new Dog[100];
```

Array references

- `Dog[] dogArray;` — This creates a *reference variable* for a `Dog` array. Its purpose is to store the address of a `Dog` array.
- `dogArray = new Dog[100];` — This instantiates an actual Dog array and its address is stored in `dogArray`.

Thus, `dogArray` itself is not an array. It is a *reference* that points to an array.

Parameters and arguments

A parameter for an array requires `[]` after its type. For example, consider:

```
public void noo(Dog[] A) {
  ...
}
```

When calling this method, a caller must pass an array *reference* for `A`. For example,

```
c.noo(dogArray);
```


Parameters and arguments

To return an array, we must return it by its reference. The return type of a method requires `[]`:

```
public Dog[] noo() {
  Dog[] D
  D = new Dog[100];
  ...
  return D;
}
```

Variables

Example. What does this output?

```
int x = 0;
int y = 1;
x = y;
x = 2;
System.out.print(x);
System.out.print(y);
```

Output: `2 1`

Object references

Example. What does this output?

```
Dog lucy;
Dog rusty;
lucy = new Dog("Golden Retriever", ...);
rusty = lucy;
rusty.setBreed("Akita");
System.out.print(lucy.getBreed());
System.out.print(rusty.getBreed());
```

Output: `Akita Akita`

Array references

Example. What does this output?

```
int[] A;
int[] B;
A = {0, 1, 2, 3, 4, 5, 6, 7};
B = A;
B[0] = 10;
System.out.print(A[0]);
System.out.print(B[0]);
```

Output: `10 10`