Three kinds of places to store values

handout for CS 302 by Will Benton (willb@cs)

local variable

declaration

```
int x;
```

The example above declares an `int` variable called `x`. You may also specify an initial value:

```
int x = 5;
String f = "foo";
```

example uses

```
x = 2;
```

```
int y = x;
```

```
return x;
```

scope, lifetime, and notes

Local variables belong to a particular method invocation. A local variable is in scope from the point of declaration until the end of the enclosing code block.

A value in a local variable is live from the point it is given to that local until some other value is given to that local (or until the local goes out of scope). Remember that instances are live as long as at there is at least one live reference to them.

parameter variable

declaration

```
void foo(int z) {
  /* ... */
}
```

In the example above, `z` is a parameter variable.

example uses

```
int y = z;
```

```
return x;
```

scope, lifetime, and notes

Parameter variables also belong to a particular method invocation. A parameter variable is in scope inside the entire body of the method that takes it as a parameter. Liveness rules for values are the same as for local variables.

Something to consider: Does assigning to a parameter variable have any effect outside of the method? Why or why not?

instance field

declaration

```
public class C {
   private int q;
   /* ... */
}
```

In the example above, `q` is an instance field.

example uses

```
this.q = 5;
```

```
int x = q;
```

```
this.q = c.q;
```

scope, lifetime, and notes

Instance fields belong to a particular object. A `private` instance field is in scope inside any instance method declared in the same class. (There are also `public` instance fields, but it's considered gauche to discuss them in polite company.) The value contained in a field is live as long as the object containing the field is live.

Note the special `this` reference. `this` is a reserved word and corresponds to an "*implicit parameter*" to every instance method. Basically, it will always refer to the object you're acting on. Why is this a useful feature? (Hint: look in your textbook for "shadowing.")