The ’.’ operator (revisited), also the *this* object reference

How come a dot is not used when we are referring to a field or a method within a class definition? For example when we refer to balance in a dynamic method, e.g., `getBalance`, in the `Account` class in `Account.java`?

It is required when we access a field or a method outside the `Account` class definition. For example, when we refer to balance within the main of `UseAccount` class, we would have to use `jims.balance` assuming that balance is declared public. If private, we would do `jims.getBalance()` instead. In either case, the question is still valid.

That is confusing, isn’t it?

Well, there actually is an implicit ’.’ operator within the `Account` class when you refer to balance. In fact, you can replace balance inside a method in `Account` by `this.balance`.

Hmm... What is this ’this’ then?

Suppose you are accessing balance inside the `getBalance` method in `Account`.

The call to `getBalance` would be possible because we created an `Account` object like this:

```java
Account jims = new Account(400);
```

in the main of `UseAccount.java`. The call would be in this form: `jims.getBalance()`. The name `jims` makes sense within the main since it is a local variable in that main, but not inside `getBalance` since it is not a variable visible in it. By the time the body of `getBalance` is being executed, we need to access the `jims` object in main inside the `getBalance` method in the `Account` class. So, we carry the `jims` object reference *'in our pocket'* when we call `getBalance` and use a generic name (alias) spelled *'this'* inside the `getBalance` method. *'this'* there is an alias of `jims` in `main`. We use that *'this'* object reference inside `getBalance` and access the balance field inside `jims` using `this.balance`.

So, you can explicitly add *'this.'* to any field or method call that is being used within a method in a class. So, the definition of `setBalance` method in `Account.java` (repeated here):

```java
public void setBalance (int newBal) {
    balance = newBal;
}
```

can be rewritten by adding the implicit *'this'*. object reference without changing the meaning of the method as follows:

```java
public void setBalance (int newBal) {
    this.balance = newBal;
}
```

You might ask, why not use *'jims.'* here instead of *'this.'*? Well, `jims` is a name that is not in scope. It is a variable defined somewhere else, e.g., in the main of `UseAccount.java`. That object with the name `jims` in main of `UseAccount` is being referred to as *'this'* here in this method.

What if we wrote `setBalance` this way?

```java
public void setBalance (int balance) {
    balance = balance;
}
```
What would it mean? Which `balance` is the `balance` declared as a field named `balance` and which `balance` is the `balance` declared as a parameter named `balance` in the method? Because of the scoping rules in Java, both `balance`'s in the body of this method refer to the closest definition of `balance` which is the parameter definition. That is, both `balance`'s are referring to the parameter `balance`.

Then, how do we make it so that the first `balance` in the body refers to the field `balance`, and the second `balance` refers to the parameter `balance`? That is exactly what we want them to mean. We do it by rewriting it this way:

```java
public void setBalance (int balance) {
    this.balance = balance;
}
```

This way, the first `balance` is the one ‘owned’ by ‘this’ object reference, which is the field. Of course, you can avoid all this confusion by carefully choosing distinctive names, but this example illustrates many subtle points.

Then, you might ask: let’s carefully choose our names and avoid using ‘this’ altogether. Well, not possible! There are situations where you would have to use ‘this’ because there is no other way of saying without using it. See the body of `deposit` in `Account`. We want to pass the current object to the `notifyManager` method which requires an `Account` object. To pass the current `Account` object as an argument to that method, we would have to use ‘this’ as follows:

```java
Bank.notifyManager(this);
```

In summary, (1) there are situations where using ‘this’ is purely optional (redundant), (2) there are situations where we can avoid using it if we choose our names carefully, and (3) there are situations where we must use it because there is no other way of expressing what we want to say.