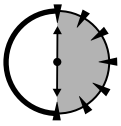
SESSION

Object Programming

Session Checklist

- ✔ Identifying objects in the real world
- ✔ Classifying objects into classes
- ✔ Comparing the object-oriented approach to the functional approach to programming
- ✔ Making nachos



30 Min. To Go

Examples of object programming can be found in everyday life. In fact, objects abound. Right in front of me is a chair, a table, a computer and a half-eaten bagel. Object programming applies these concepts to the world of programming.

Abstraction and Microwave Ovens

Sometimes when my son and I watch football, I whip up a terribly unhealthy batch of nachos. I dump some chips on a plate, throw on some beans, cheese, and lots of jalapeños, and nuke the whole mess in the microwave oven for five minutes.

To use the microwave, I open the door, throw the stuff in, and punch a few buttons on the front. After a few minutes, the nachos are done.

This doesn't sound very profound, but think for a minute about all the things I don't do to use my microwave:

- I don't look inside the case of my microwave; I don't look at the listings of the code that goes into the central processor; and I don't study the wiring diagram.
- I don't rewire or change anything inside the microwave to get it to work. The microwave has an interface that lets me do everything I need to do—the front panel with all the buttons and the little timer display.
- I don't reprogram the software used to drive the little processor inside my microwave, even if I cooked a different dish the last time I used it.
- Even if I were a microwave designer and knew all about the inner workings of a microwave, including its software, I wouldn't think about that stuff when I use it.

These are not profound observations. We can think about only so much at one time. To reduce the number of things we must deal with, we work at a certain level of detail.

In object-oriented (OO) terms, the level of detail at which we are working is called the *level of abstraction*. When I'm working on nachos, I view my microwave oven as a box. As long as I use the microwave only through its interface (the keypad), there should be nothing that I can do that will cause the microwave to

1. Enter an inconsistent state and crash;
2. Worse, turn my nachos into a blackened, flaming mass; or
3. Worst of all, catch on fire and burn down the house.

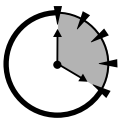
Functional nachos

Suppose I were to ask my son to write an algorithm for how Dad makes nachos. After he understood what I wanted, he would probably write "open a can of beans, grate some cheese, cut the jalapeños," and so on. When it came to the part about microwaving the concoction, he would write something similar to "cook in the microwave for five minutes."

That description is straightforward and complete. But it's not how a functional programmer would code a program to make nachos. Functional programmers live in a world devoid of objects such as microwave ovens and other appliances. They tend

to worry about flowcharts with their myriad functional paths. In a functional solution to the nachos problem, the flow of control would pass through my finger to the front panel and then to the internals of the microwave. Soon, the flow would be wriggling through complex logic paths about how long to turn on the microwave energy and whether to sound the "come and get it" tone.

In a world like this, it's difficult to think in terms of levels of abstraction. There are no objects, no abstractions behind which to hide inherent complexity.



20 Min. To Go

Object-oriented nachos

In an object-oriented approach to making nachos, we would first identify the types of objects in the problem: chips, beans, cheese, and an oven. Then we would begin the task of modeling these objects in software, without regard to the details of how they will be used in the final program.

While we are writing object-level code, we are said to be working (and thinking) at the level of abstraction of the basic objects. We need to think about making a useful oven, but we don't have to think about the logical process of making nachos yet. After all, the microwave designers didn't think about the specific problem of my making a snack. Rather, they set about the problem of designing and building a useful microwave.

After we have successfully coded and tested the objects we need, we can ratchet up to the next level of abstraction. We can start thinking at the nacho-making level, rather than at the microwave-making level. At this point, we can pretty much translate my son's instructions directly into C++ code.

Actually we could keep going up the chain. The next level up might be to get up, go to work, return home, eat, rest, and sleep with nacho consumption falling somewhere in the eat-rest phases.

Classification and Microwave Ovens

Critical to the concept of abstraction is that of classification. If I were to ask my son, "What's a microwave?" he would probably say, "It's an oven that. . . ." If I then asked, "What's an oven?" he might reply, "It's a kitchen appliance that. . . ." I could keep asking this question until we eventually ended up at "It's a thing," which is another way of saying, "It's an object."

My son understands that our particular microwave is an example of the type of things called microwave ovens. In addition, he sees microwave ovens as just a special type of oven, which is, in turn, a special type of kitchen appliance.

The technical way of saying this is that my microwave is an *instance* of the class *microwave*. The class *microwave* is a *subclass* of the class *oven* and the class *oven* is a *superclass* of the class *microwave*.

Humans classify. Everything about our world is ordered in taxonomies. We do this to reduce the number of things that we have to remember. Consider, for example, the first time that you saw the new Ford-based Jaguar (or the new Neon for the rest of us). The advertisement called the Jaguar "revolutionary, a new type of car." But you and I know that that just isn't so. I like the looks of the Jaguar—I like them a lot—but it is just a car. As such, it shares all of (or at least most of) the properties of other cars. It has a steering wheel, seats, a motor, brakes, and so on. I bet I could even drive one without help.

I don't have to clutter my limited storage with all the things that a Jaguar has in common with other cars. All I have to remember is that "a Jaguar is a car that . . ." and tack on those few things that are unique to a Jaguar. Cars are a subclass of wheeled vehicles, of which there are other members, such as trucks and pickups. Maybe wheeled vehicles are a subclass of vehicles, which include boats and planes. And on and on and on.

Why Build Objects This Way?

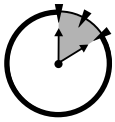
It might seem easier to design and build a microwave oven specifically for our one problem, rather than to build a separate, more generic oven object. Suppose, for example, that I wanted to build a microwave to cook nachos and nachos only. There would be no need to put a front panel on it, other than a START button. You always cook nachos the same amount of time. We could dispense with all that DEFROST and TEMP COOK nonsense. It could be tiny. It would only need to hold one flat little plate. Three cubic feet of space would be wasted on nachos.

For that matter, let's just dispense with the concept of "microwave oven" altogether. All we really need is the guts of the oven. Then, in the recipe, we put the instructions to make it work: "Put nachos in the box. Connect the red wire to the black wire. Notice a slight hum." Stuff like that.

Nevertheless, the functional approach does have some problems:

- **Too complex.** We don't want the details of oven building mixed in with the details of nacho building. If we can't define the objects and pull them out of the morass of details to deal with separately, we must deal with all the complexities of the problem at the same time.
- **Not flexible.** If we need to replace the microwave oven with some other type of oven, we should be able to do so as long as the interface to the new oven is the same as the old one. Without a simple and clearly delineated interface, it becomes impossible to cleanly remove an object type and replace it with another.
- **Not reusable.** Ovens are used to make many different dishes. We don't want to create a new oven each time we encounter a new recipe. Having solved a problem once, it would be nice to reuse the solution in future programs.

It does cost more to write a generic object. It would be cheaper to build a microwave made specifically for nachos. We could dispense with expensive timers, buttons, and the like, that are not needed to make nachos. After we have used a generic object in more than one application, however, the costs of a slightly more expensive class more than outweigh the repeated costs of building cheaper, less flexible classes for every new application.



10 Min. To Go

Self-Contained Classes

Let's reflect on what we have learned. In an object-oriented approach to programming

- The programmer identifies the classes necessary to solve the problem. I knew right off that I was going to need an oven to make decent nachos.
- The programmer creates self-contained classes that fit the requirements of the problem, and doesn't worry about the details of the overall application.
- The programmer writes the application using the classes just created without thinking about how they work internally.

An integral part of this programming model is that each class is responsible for itself. A class should be in a defined state at all times. It should not be possible to crash the program by calling a class with illegal data or with an illegal sequence of correct data.

***Paradigm* is another word for programming model.**

Many of the features of C++ that are shown in subsequent chapters deal with giving the class the capability to protect itself from errant programs just waiting to trip it up.

REVIEW

In this Session, you saw the fundamental concepts of object-oriented programming.

- Object-oriented programs consist of a loosely coupled set of classes.
- Each class represents a real-world concept.
- Object-oriented classes are written to be somewhat independent of the programs that use them.
- Object-oriented classes are responsible for their own well-being.

QUIZ YOURSELF

1. What does the term *level of abstraction* mean? (See "Abstractions and Microwave Ovens.")
2. What is meant by the term *classification*? (See "Classification and Microwave Ovens.")
3. What are three problems with the functional approach to programming that object programming attempts to solve? (See "Why Build Objects This Way?")
4. How do you make nachos? (See "Abstractions and Microwave Ovens.")