I generally try and describe Object-Orientated-Programming by using real world examples.

For example, I might say that a class called Vehicle describes the minimum things that a vehicle is. I'll ask the person to tell me what he or she thinks a vehicle is. Sometimes they say things like "Well, like a car or a truck", and I'll nod and agree with them. Then I'll ask what the differences are between a car and a truck. Sometimes they mention size, sometimes the purpose, and other things.

Then I'll ask them to forget about a car, or a truck and just ask them to continue to describe a vehicle:

"Oh, well it moves"

"It has an operator, or a driver"

etc...

Soon, we know what a Vehicle is and I said that in OOP we would define a vehicle, and for the sake of argument say it can move, and give it a driver of sorts. Then I'll ask, ok, so what does a Car have?

"Doors"

"Windows"

And then a truck....

"Doors" "windows" "More Wheels!"

Soon, after lots of discussion, the other person generally has identified:

1) What constitutes a vehicle

2) What constitutes a car

3) What constitutes a truck

4) What constitutes an aeroplane.

All without any technicalities. We've divided up the properties of each in to the right areas. They understand inheritance ("Yeah, a car is a vehicle, a truck is a vehicle, but a car is not a truck, it's SIMPLE, duh!").

They even understand polymorphism, "Sure, they basically do the same, but that might do it slightly different.". We can talk about behavior and where that should live in our tree of objects.

Depending on their education and background, some get it faster than others. But when I compare OOP to real-life objects, most people always get it. In fact, I have found in conversations with non-technical people things I had never thought of. Vehicles don't have to be manned, for example (drones), but would a programmer have thought of the operator of the vehicle as a property of it? I am not saying it is right or wrong to have an operator mentioned, but it causes us to think about what we are modelling and what we are trying to achieve when we develop software.

Now, partial template specialization, on the other hand.... :)

Sitting in a restaurant with my wife, she asked me "What does Object Oriented mean?"

I started bloviating about code reuse and encapsulation and polymorphism, and at some point I realized her eyes were terminally glazed over.

So I grabbed a Splenda packet out of the container. I said, "Here's an object. What are its properties?"

She said, "It's rectangular, it's made of paper, it contains splenda, it's blue, it has printing on it."

I picked up a sugar packet. "What does it have in common with this one?"

She said, "The rectangularness, the paper, that there is printing."

I said, "What about that they both contain something sweet?"

She said, "Sure."

I said, "So these are both instances of what we might call an abstract sweetener packet. A platonic ideal sweetener packet, if you like."

She said, "Sure."

I said, "And each one has properties inherited from the abstract packet, and then variations on that that are specific to its type of thing."

She said, "Right. Oh! And if I wanted to make, like, a Saccharine packet, I'd take the generic one, and set the details about it for the Saccharine, and then I'd have that!"

I said, "Bingo: Object Oriented Programming."

You and I know she just intuited her way to the Factory design pattern. Whatever. It illustrates inheritance, encapsulation, object class identity... Good stuff.

There's two kinds of wizards. There's the guy who makes specific things happen with magic words. He's got a word for summoning fire. He's got a word for making a frozen chicken appear out of thin air. And another word for creating a pot of force (I prefer my force green, glowy, and translucent) full of friolating goodness. With the right application of his words he can produce a fried chicken.

And then there's the OOP wizard. Who just summons an imp who knows how to go the grocery store, buy a chicken (or ingredients for any other food you want him to prepare), cook it, and serve you dinner. OOP Wizard doesn't have to tell his imp how to do it. He just needs to let him know what he wants which in this case is fried chicken. Not only that, the OOP wizard can summon other minions to tell his imp-chef what to do.

So, the incantation guy impresses at parties but the OOP wizard is the one you want when you're going to start a magic restaurant with a bunch of characters (like say, a pissed off unicorn waiter, and a troll floor manager) who all have to work together. If you try to invoke every step of the problem of solving "restaurant" you can easily get tangled up in the details and it's really easy to make mistakes. The OOP wizard pre-trains his minions to sort out the details for him so he can just focus on solving the bigger problem by having his people interact.

Not to mention it's easier to re-use your chef-imp for your grade school cafeteria problem then it is to separate all the junk you might or might not re-use when you're doing it all one step at a time by calling words and words that call other sets of words (which will get more an more numerous as you have to handle a greater variety of problems).

To be fair, with very, very careful application, the incantation wizard can get it all done as fast as the OOP wizard. He can break things down in the right way such that calling the right spells requires no more work on his part than the OOP wizard. But the work is much harder to understand or duplicate and much harder to re-use large portions of because it's all tied together for one specific complex problem.