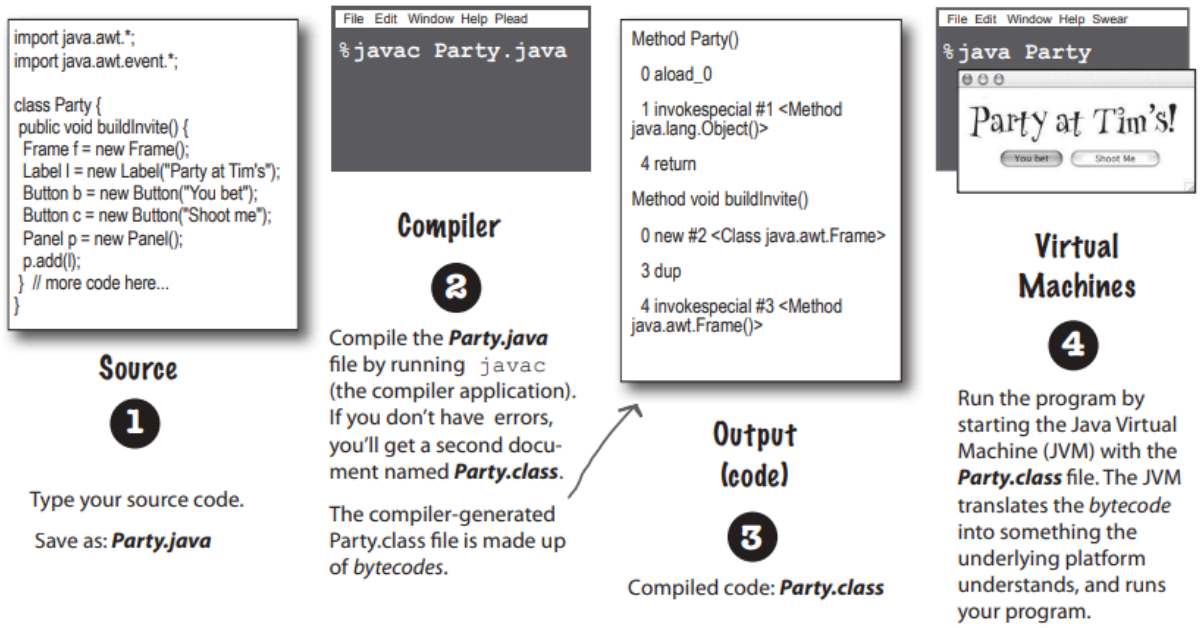
Head First Java Notes

Chapter 1 → Breaking the Surface

The way Java works

You'll have a source code file, compile it using the javac compiler (any device capable of running Java will be able to interpret/translate this file into something it can run. The compiled bytecode is platform independent), and then run the compiled bytecode on a Java virtual machine.



History, Speed and Memory usage

Java was initially released, on January 23, 1996 by James Gosling. Java is famous for its backward compatibility, so old code can run quite happily on new JVMs.

When Java was first released, it was slow. But soon after, the HotSpot VM was created, as were other performance enhancers. While it's true that Java isn't the fastest language out there, it's considered to be a very fast language—almost as

fast as languages like C and Rust, and much faster than most other languages out there.
Java has a magic super-power—the JVM. The Java Virtual Machine can optimize your code while it's running, so it's possible to create very fast applications without having to write specialized high-performance code.
Compared to C and Rust,
Java uses a lot of memory.

The Compiler and JVM battle over the question, "Who's more important?"

JVM

What, are you kidding? HELLO. I am Java. I'm the one who actually makes a program run. The compiler just gives you a file, but the file doesn't do anything unless I'm there to run it.

A programmer could just write bytecode by
hand, and I'd take it. You might be out of a job
soon, buddy.

Compiler

Excuse me, but without me, what exactly would you run? There's a reason Java was designed to use a bytecode compiler. If Java
were a purely interpreted language, where—at
runtime—the virtual machine had to translate
straight-from-a-text-editor source code, a Java program would run at a ludicrously glacial pace.

While it is true but a programmer writing bytecode by hand is like painting pictures of your vacation instead of taking photos—sure, it's an art, but most people prefer to use their time differently.

So, what do you actually do?

Remember that Java is a strongly typed language, and that means I can't allow variables to hold data of the wrong type. This is a crucial safety feature, and I'm able to stop the vast majority of violations before they ever get to you.

But some still get through! I can throw ClassCastExceptions and sometimes I get people trying to put the wrong type of thing in an array that was declared to hold something else.

Yes, there are some datatype exceptions that can emerge at runtime, but some of those have to be allowed to support one of Java's other important features—dynamic binding. At runtime, a Java program can include new objects that weren't even known to the original programmer, so I have to allow a certain amount of flexibility. But my job is to top anything that would never—could never—succeed at runtime.

OK. Sure. But what about security?

Excuse me, but I am the first line of defense, as they say. The datatype violations I previously described could wreak havoc in a program if they were allowed to manifest. I am also the one who prevents access violations, such as code trying to invoke a private method, or change a method that—for

Whatever. I have to do that same stuff too, though, just to make sure nobody snuck in after you and changed the bytecode before running it.

security reasons—must never be changed. I stop people from touching code they're not meant to see, including code trying to access another class critical data.

Of course, but as I indicated previously, if I didn't prevent what amounts to perhaps 99% of the potential problems, you would grind to a halt.

Chapter 2 → A Trip to Objectville

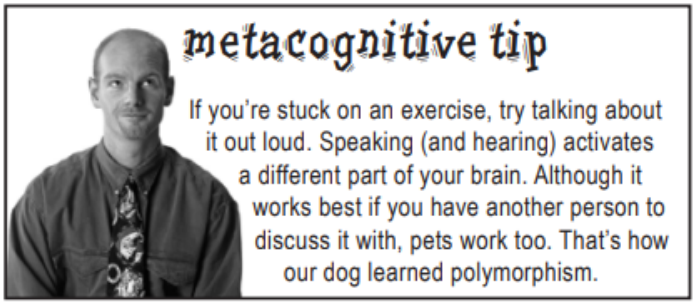
What do you like about OO?

"Object-oriented programming lets you extend a program without having to touch previously tested, working code. (not messing around with code I've already tested, just to add a new feature)"

"It helps me design in a more natural way. Things have a way of evolving."

"I like that the data and the methods that operate on that data are together in one class."

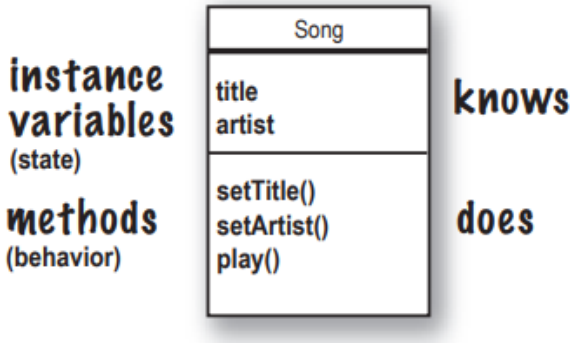
"Reusing code in other applications. When I write a new class, I can make it flexible enough to be used in something new, later."



When you design a class, think about the objects that will be created from that class type. Think about:

- things the object knows
- things the object does

Things an object knows about itself are called instance variables. They represent an object's state (the data) and can have unique values for each object of that type.
Things an object can do are called methods. When you design a class, you think about the data an object will need to know about itself, and you also design the methods that operate on that data.
So objects have instance variables and methods, but those instance variables and methods are designed as part of the class.



What's the difference between a class and an object?

💡 **A class is not an object (but it's used to construct them)**

A class is a blueprint for an object. It tells the virtual machine how to make an object of that particular type. Each object made from that class can have its own values for the instance variables of that class.