

# What is var?

---

var is a special contextual keyword in Java that lets our code take advantage of Local Variable Type Inference.

By using var as the type, we're telling Java to figure out the compile-time type for us.

# Local Variable Type Inference (LVTI)

---

Local Variable Type Inference was introduced in Java 10.

One of the benefits is to help with readability of code. Another is to reduce boilerplate code.

It's called Local Variable Type Inference for a reason, because:

- It can't be used in field declarations on a class.
- It can't be used in method signatures, either as a parameter type or a return type.
- It can't be used without an assignment because the type can't be inferred in that case.
- It can't be assigned a null literal, again, because a type can't be inferred in that case.

# Run Time vs. Compile Time Typing

---

Are you still confused about the difference between run time and compile time typing?

You can think of the compile time type as the **declared** type.

This type is declared as a variable reference, a method return type, or a method parameter, for example.

In the case of Local Variable Type Inference (LVTI), we don't explicitly declare a type for the compiled reference type. Instead, it gets inferred by the compiler, but the byte code generated is the same as if we had declared the type.

# Run Time vs. Compile Time Typing

---

In many cases, the compile time type is the declared type to the left of the assignment operator.

What is returned on the right side of the assignment operator from whatever expression or method is executed, sometimes can only be determined at runtime, when the code is executing conditionally through the statements in the code.

You can assign a runtime instance to a different compile time type, only if certain rules are followed.

In this course, up to now, we've looked at only one rule that applies, and that's the inheritance rule.

We can assign an instance to a variable of the same type, or a parent type, or a parent's parent type, including `java.lang.Object`, the ultimate base class.

# Run Time vs. Compile Time Typing

---

Why are runtime types different than compile time types?

Because of polymorphism.

Polymorphism lets us write code once, in a more generic fashion, like the code we started this lecture with.

We saw that those two lines of code, using a single compile time type of Movie, actually supported four different runtime types.

Each type was able to execute behavior unique to the class.