# Introduction

In Java, everything has an identifier, every class, every interface, every method, every object, every variable and constant. The identifier has two purposes. First, it allows the programmer to be able to read the code. Second, it allows the compiler to know what thing it is working on. The compiler uses the identifier, a.k.a symbolic name, to track where data is stored, both data in the compiler and in memory for a running program.

# Legal Identifiers

There are rules on what character strings can be used as identifiers.

Rules:

* Must start with a letter, an underscore, or a dollar sign [A-Z, a-z, \_, $]
  + Cannot start with a digit [0-9]
* The rest of the identifier can contain letters, digits, underscores, and dollar signs.
  + This means that an identifier cannot contain a dot (.), dash (-), and other characters that are used as operators.
* Cannot be a Java Keyword such as ***int***, ***class***, ***interface***, ***void***, ***public***, ***if***, ***else***, ***for***.
* Identifiers are case-sensitive.
  + For example, myVariable and myvariable are two different identifiers.
* There is no length limit to the number of characters in an identifier.

Guidelines for naming Identifiers:

* Make sure the identifier is self-documenting.
  + For example, counter instead i, wheel\_speed instead of ws.
  + Make sure the identifier is meaningful.
* Use a consistent naming scheme:
  + Camel’s Case:
    - Identifiers consist of compound words or phrases which each word starting with a upper case letter.
    - Variation: the first word of an identifier starts with a lower case letter.
    - Examples: wheelSpeed, gearTurnRatio, ScarobMotorControl
  + Alternative to Camel’s Case:
    - Identifiers consist of a set of words or phrases separated by underscores.
    - Examples: wheel\_speed, gear\_turn\_ratio, scarob\_motor\_control
  + Type Specifications:
    - Uses Camel’s Case or alternative.
    - First character denotes the type of thing being identified.
      * Classes, Interfaces start with a upper case letter.
        + Things that only exist in the compiler.
      * Variables, objects, methods start with a lower case letter.
        + Things that exist in memory, exist while a program is running.

# Scope

One of the major issues with identifiers is when two identifiers have the same name also called a naming conflict. It is up to the compiler to determine which one it should use for any operation. Luckily, there is a set of rules that the compiler will use when there are multiple identical identifiers. This is called Scope. Scope is based on containers.

The first level of scope is the block. A block is the code within a set of curly brackets {}. An identifier defined within a set of curly brackets is called a local variable. Local variables hide any other identifier that has the same name.

The next level of scope is the method. An identifier defined within a method hides identical identifier for outer scopes. A local variable defined within curly brackets in a method will hide the method defined identifier.

Next is the class scope. Any identifier defined in a class hides identical identifiers that are defined outside the class. Note, a class can declare a class inside itself, which is known as a nested class. An identifier in the nested class hides an identical identifier of its parent class.

The outer most scope is the package. Classes and interfaces belong it one package.

## Package

Consider a package as a collection of classes and interfaces. A package is the programming equivalent of a library. Often a package is stored in a Jar file and imported into the running program as a library.

Every package name needs to be unique, unique everywhere even in other programs. That is because the package name is the final resolution in naming conflicts. There is no organization that assigns package names nor make sure they are unique world wide. Therefore a convention is used based on domain names and URLs.

The convention is to use the URL for the package in reverse order, where each term is separated by dots (.). So if a package is developed by a group with the URL of scarob.starcityrobotics.com, all of its package names would start with ***com.starcityrobotics.scarob.<package name>***. The <package name> consists of one or more identifiers. So, a package name could be something like:

***com.starcityrobotics.scarob.frc2025.driveTrain***

## Overriding Scope

There are times when it is necessary to override the name conflict resolution that the compiler would use. This is done by adding scope to the identifier in the form of:

***<scope>.identifier***

where <scope> is a list of scopes or explicit variable names.

### Local Variables

There is no way to override local variables. In other words, it the same identifier is used in a method and a pair of curly brackets in that method, there is no way to force the compiler to use the outer identifier. In this case, the only solution is to never use the same identifier for two different variables or objects with the same method.

### Method

There are several ways of overriding a local variable in order to use a variable defined within the class but outside the scope of curly brackets. There are two ways of doing this, each way depends on whether the method is declared as static or not.

For non-static methods, a hidden argument is passed the method when the method is called. This argument is called **this** and it is reference to the object that the method belongs to. In this case, therefore format of the identifier is:

***this.<identifier>***

This technique does not work for static methods because one method is used by all objects of that class and no ***this*** argument is passed to the method. In this case, the class name is used to override the name conflict resolution.

***<className>.<identifier>***

There is a limitation here. A static method can only access static data members defined in the class.

### Class

For classes, the may be the need to make sure an object from outside the instance of the class or a static identifier defined by the class. This requires the addition of package. This can be the package the class is defined in or an external package being used as a library.

Using our SCAROB package from before, the following full identifiers can be used

* ***com.starcityrobotics.scarob.frc2025.drivetrain.<identifier>***
* ***frc2025.drivetrain.<identifier>***
* ***java.util.<class Identifier>***

Note that the full package name does not have be used. Just the parts of the package name that allows the identifier to be uniquely identified.

# Conclusion

Identifiers and scope are the first techniques used to uniquely identify an object. The use of the ***import*** and ***from <package> import*** allow additional techniques for uniquely identifying objects. That topic is outside of the scope of this document.