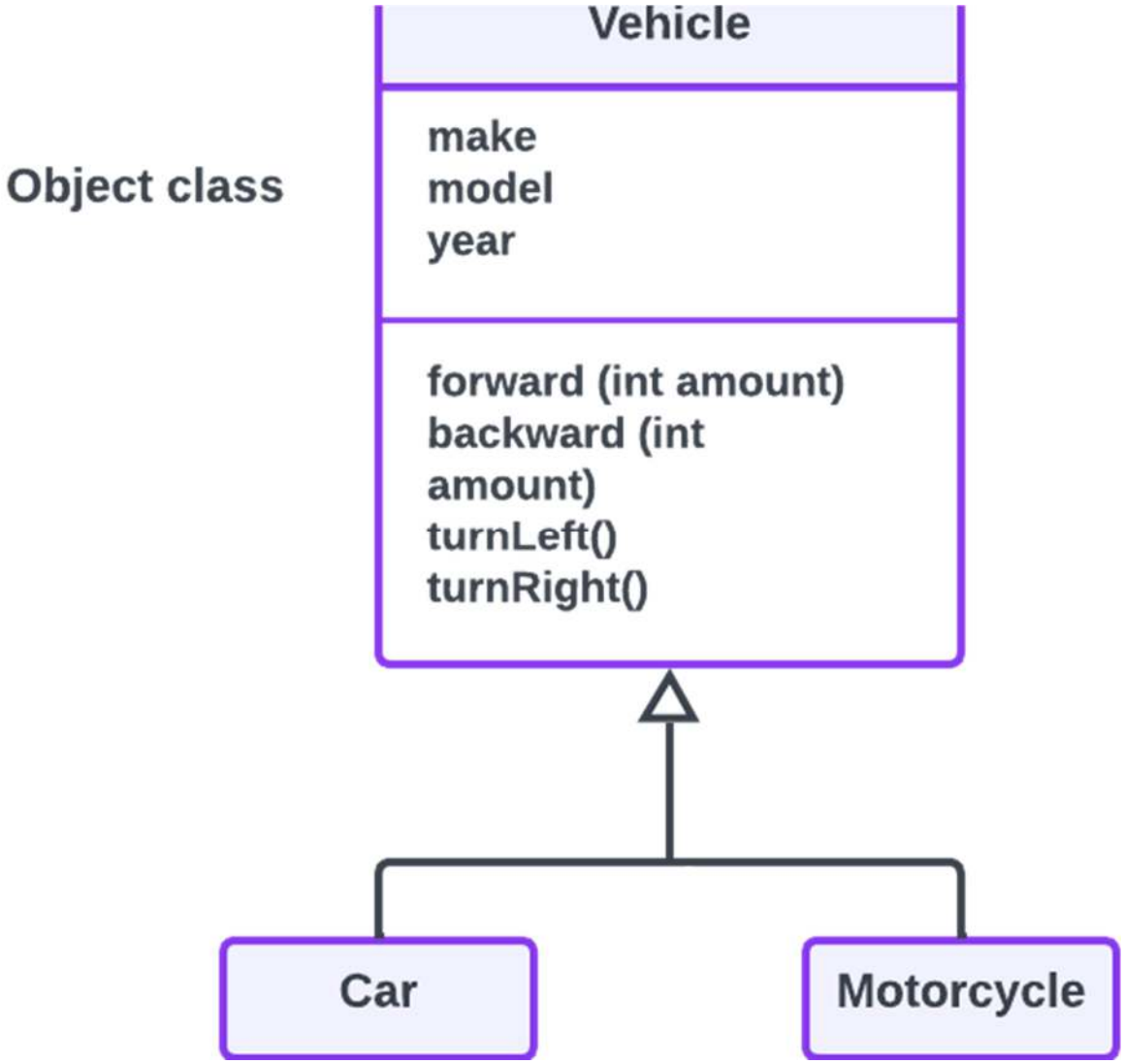
Inheritance

One of the really useful features of Object-Oriented programming is **inheritance**. You may have heard of someone receiving an inheritance, which often means they were left something from a relative who died. Or, you might hear someone say that they have inherited their musical ability from a parent. In Java, all classes can **inherit** attributes (instance variables) and behaviours (methods) from another class. The class being inherited from is called the **parent class** or **superclass**. The class that is inheriting is called the **child class** or **subclass**.

When one class inherits from another, we can say that it is the *same kind of thing* as the **parent class** (the class it inherits from). For example, a car is a kind of vehicle. This is sometimes called the *is-a* relationship, but more accurately it's a *is-a-kind-of* relationship. A motorcycle is another kind of vehicle. All vehicles have a make, model, and year that they were created. All vehicles can go forward, backward, turn left and turn right.



A UML Class Diagram Showing Inheritance

A **UML (Unified Modeling Language) class diagram** shows classes and the relationships between the classes as seen in Figure 1. An open triangle points to the parent class. The parent class for `Car` and `Motorcycle` is `Vehicle`. The `Vehicle` class has two child classes or subclasses: `Car` and `Motorcycle`.

#Subclass extends Superclass

To make a subclass inherit from a superclass, use the Java keyword **extends** with the superclass name when creating a new subclass, as shown below.

CopyC#C++ClojureCSSDartGoHaskellHTMLJavaJavaScriptJSONJSXKotlinMarkdown PascalPerlPHPPlain TextPythonRRubyRustSchemeShellSQLSwiftTypescriptVB.NETVBScriptXMLYAML

```
1
2
public class Car extends Vehicle
public class Motorcycle extends Vehicle
```

InfoWarningTip

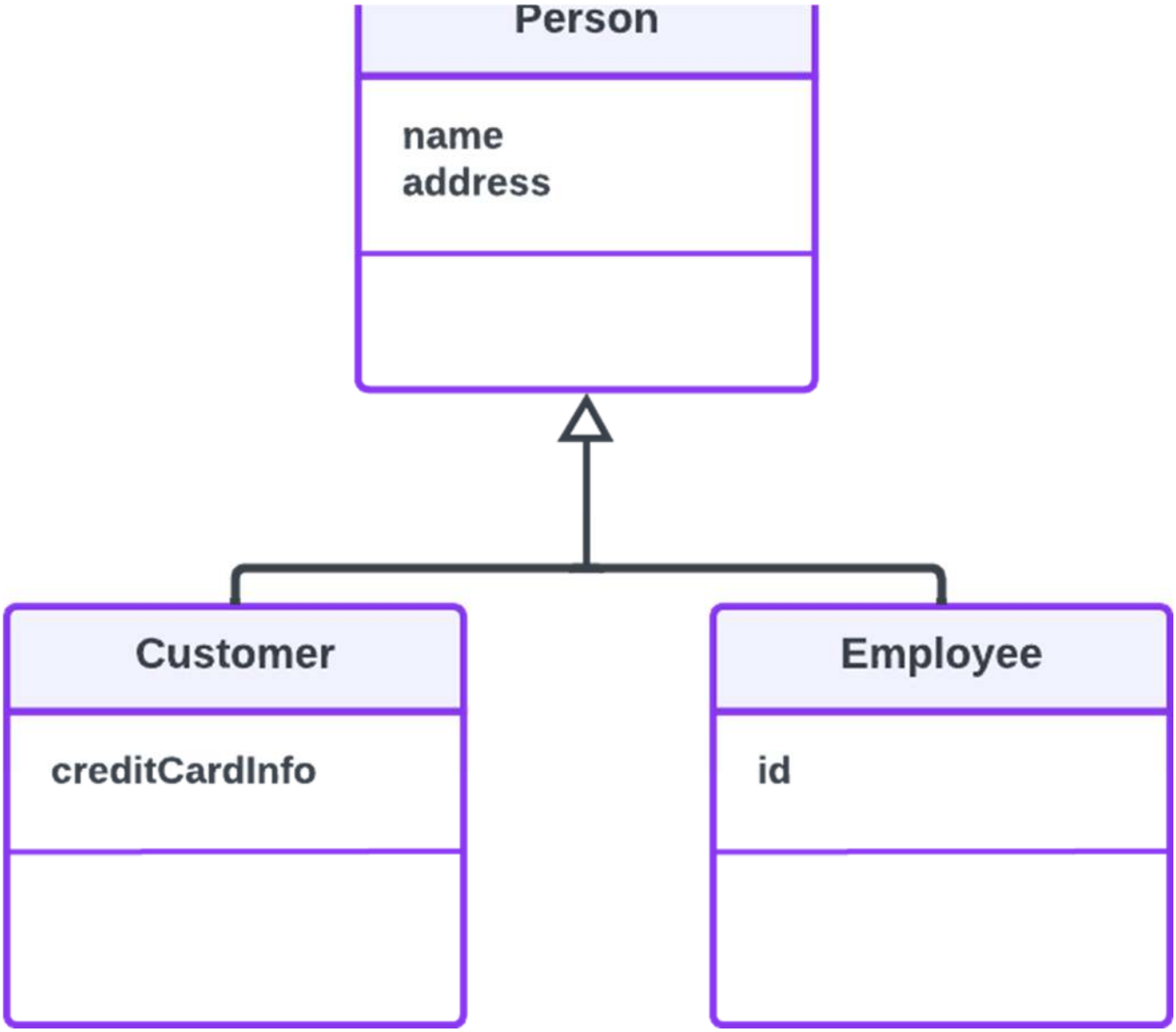
While a person can have two parents, a Java class can only inherit from one parent class. If you leave off the **extends** keyword when you declare a class then the class will inherit from the built-in `Object` class that is already defined in Java. In fact, all Java classes extend the Object class by default so you can leave off the **extends** keyword if the default behaviour suits your goal.

#Why Use Inheritance?

Inheritance allows you to reuse data and behaviour from the parent class. When we design classes, it's a good idea to make them contain only the information they need to contain, and/or to build the attributes and behaviours they share with other classes into one superclass. Ideally, we want to write as few lines of code as possible to achieve our goals, both for the sake of simplicity and for the sake of keeping the code intelligible to our future selves and other developers.

If you notice that several classes share the same data and/or behaviours, you should pull those similarities out into a parent class. This is called **generalization**. For example, `Customers` and `Employees` are both types of people, so it makes sense use the general `Person` class as seen below.

Inheritance is also useful for **specialization** which is when you want most of the behaviour of a parent class, but want to do at least one thing differently and/or add more data. The example below can also be seen as specialization. An employee is a person but also has a unique id. A customer is a person, but also has an associated credit card.



A UML Class Diagram Showing Inheritance

#Check your understanding

If you don’t specify the parent class in a class declaration which of the following is true?

It doesn't have a parent class.

It inherits from the Object class.

It inherits from the Default class.

It inherits from the Parent class.

If the class Vehicle has the instance fields `make` and `model` and the class Car inherits from the class Vehicle, will a car object have a make and model?

Yes

No

In Java, how many parents can a class have?

0

1

2

infinite

#Coding Exercise

The `Student` class can also inherit from the class `Person` just like `Employee` and `Customer` because a `Student` is a type of `Person`.

What do you need to add to the Student class declaration below to make it inherit from type `Person`? When you fix the code, the **instanceof** operator will return `true` that `Student` is an instance of both the `Student` and the `Person` class. What other private instance variables could you add to `Person` and `Student`? In which class would you put an `address` attribute? Where would you put `gpa`?

2

Run

Student.java Person.java

```
// How can we make the Student class inherit from class Person?
public class Student extends Person{
   private int id;
  public static void main(String[] args) {
      Student s = new Student();
      System.out.println(s instanceof Student);
      System.out.println(s instanceof Person);
  }
}
```