accenture

Strategy | Consulting | Digital | Technology | Operations

Object Oriented Programming

Accenture Java Pre-Bootcamp

High performance. Delivered.

Plan

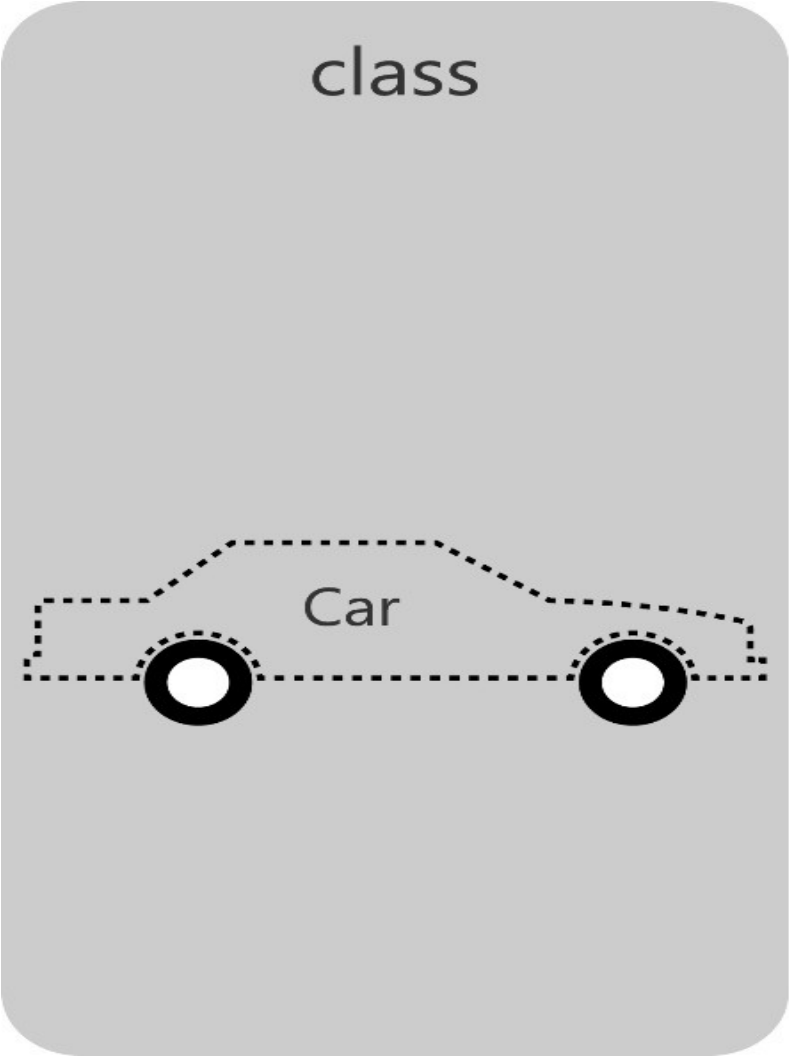
- Object Oriented programming
  - What is Object Oriented programming?
  - Class VS Object
- Object Oriented programming principles
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism

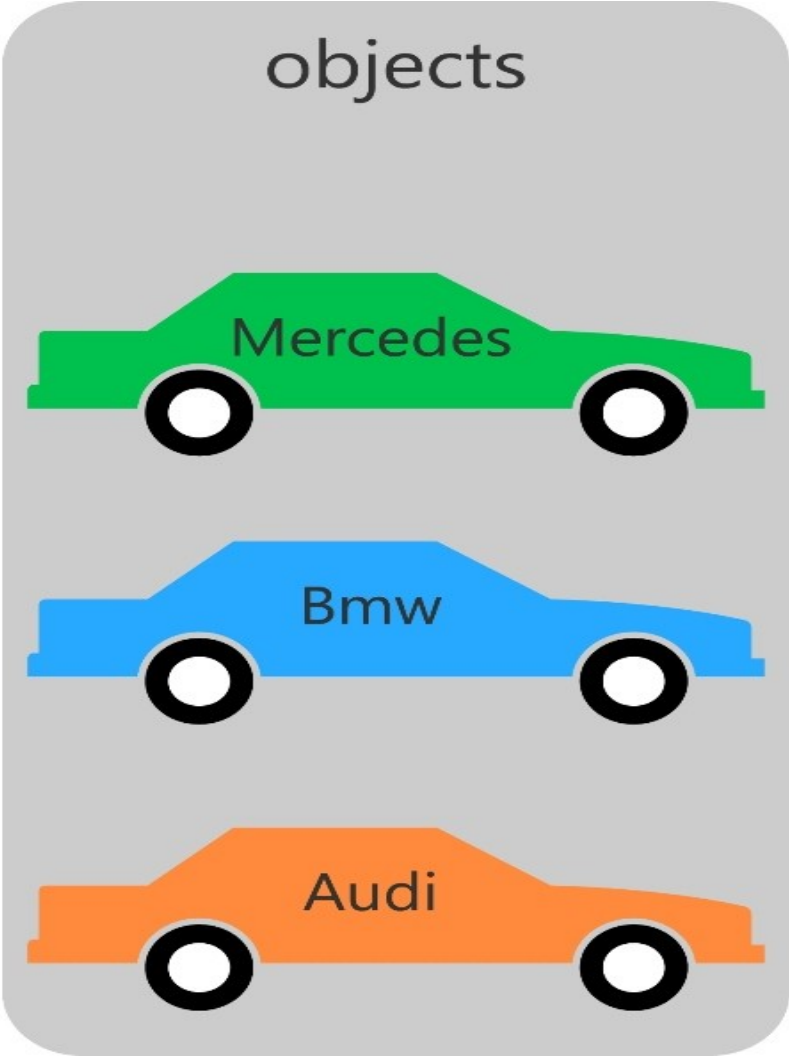
What is Object Oriented Programming?

- The key idea of OOP is that the world can be accurately described as a collection of objects that interact.
- OOP Basic Terminology:
  - **Object:** A representation of a person, a place or a thing (**noun**).
  - **Method**: An action to be performed by an object (**verb**).
  - **Property**/**Attribute**/**Member Field**/**Member Variable**: Characteristics or state of an object.
  - **Class**: Blueprint of an Object.

•

Class vs Object





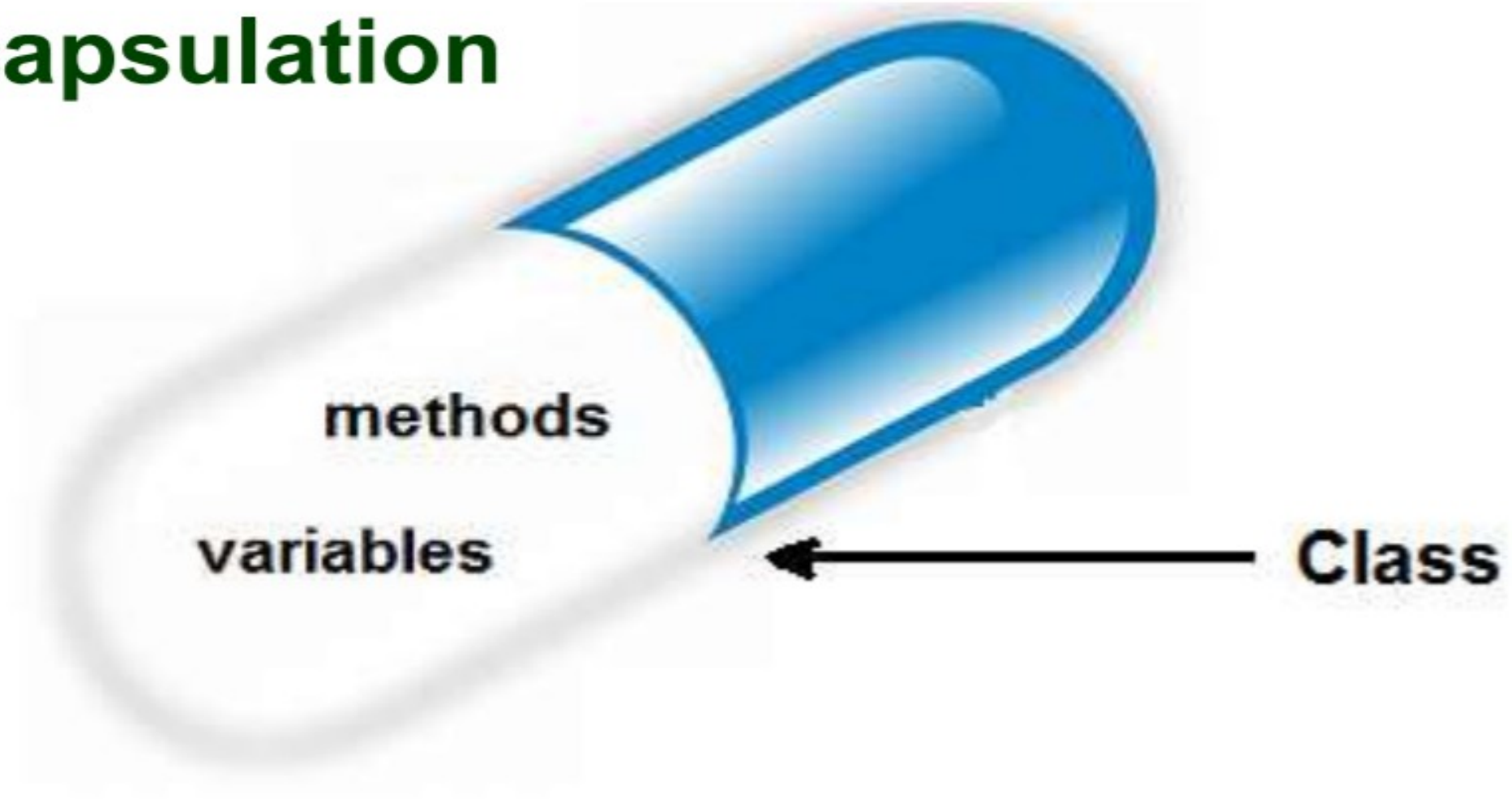
Classes vs Objects

- An Object is a **copy** of a particular class
- Sometime we call it an *instance* of that class.
- Thus, the act of creating a new instance/copy /object of a class is called *instantiation*.

Class vs Object

CLASS	OBJECT
Class is a data type	Object is an instance of Class.
It generates OBJECTS	It gives life to CLASS
Does not occupy memory location	It occupies memory location.
It cannot be manipulated because it is not available in memory *(except static class)*	It can be manipulated.

Encapsulation



Encapsulation

- *Encapsulation* is bundling data and functions that work on that data within one unit, i.e. a class.

- *Encapsulation* hides the internal representation, or state, of an object from the outside.
- *Encapsulation* controls access to the internal state of the object.

Encapsulation In Java

- *Encapsulation* in Java is achieved through **access modifiers** and **accessor methods**.

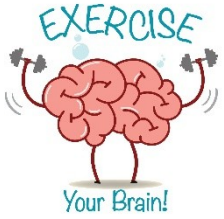
- Access modifiers (*private, protected, public, default*) hide data.
- Accessor methods(*getters, setters*) control access to the internal state of the object.

Encapsulation In Java

Accessibility Matrix

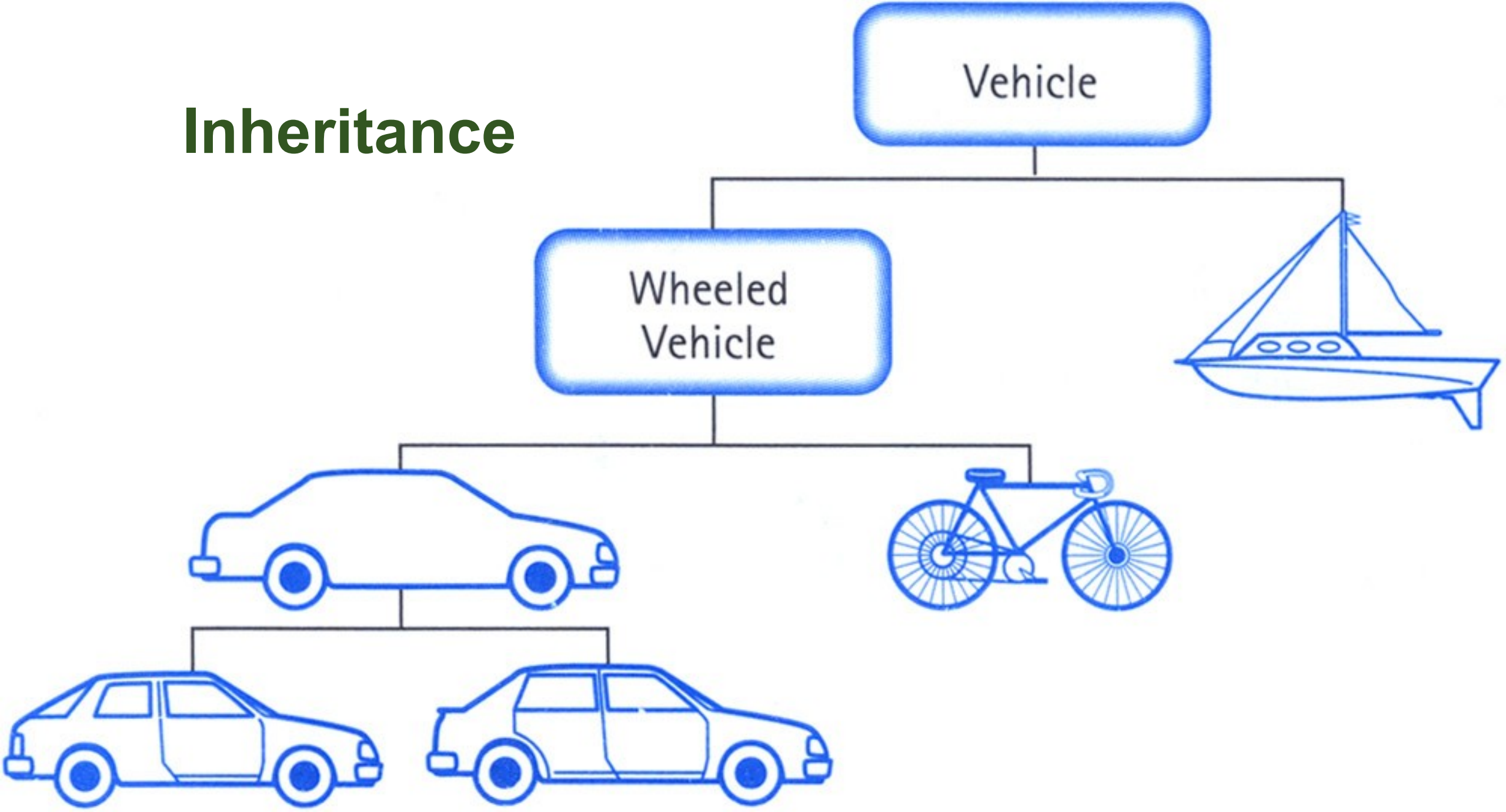
Modifier	Class	Package	Subclass	Other Classes
private	yes	no	no	no
default	yes	yes	no	no
protected	yes	yes	yes	no
public	yes	yes	yes	yes

Abstraction

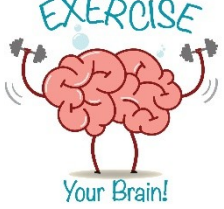


- *Abstraction* hides the implementation complexity of the member methods and exposes a clear contract to the user.
- This contract is called **API**(Application Programming Interface).
- The API of a class is the set of public methods (**public interface**) it exposes to the client.
- *Abstraction* simplifies the class for the client by hiding the unnecessary implementation details.
- *Abstractions* allows the internal implementation to evolve without impacting the client using the API.

Inheritance



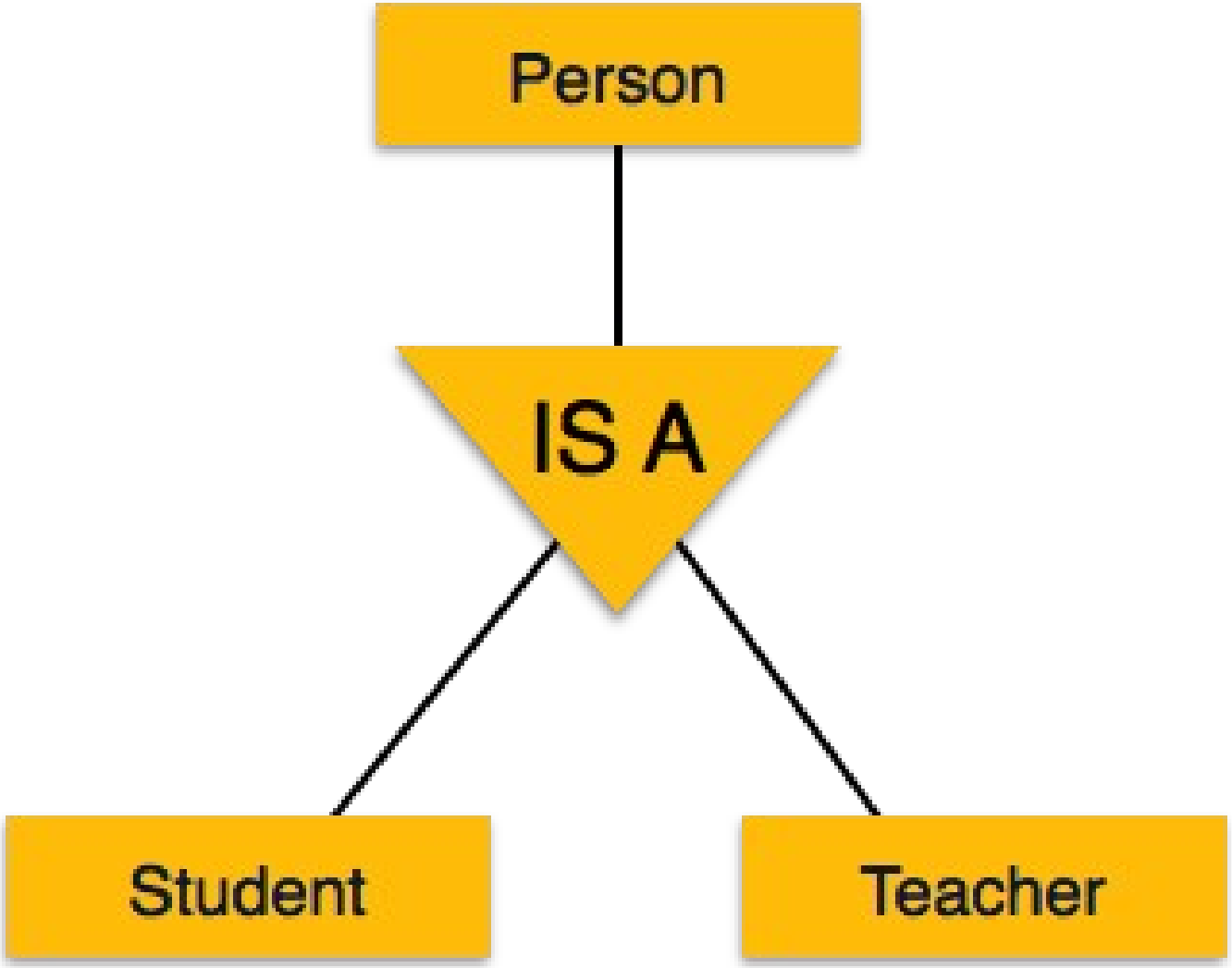
Inheritance



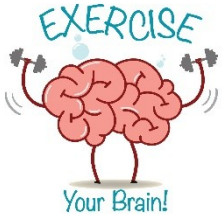
- *Inheritance* is a way of organizing classes.
- Classes with properties in common can be grouped so that their common properties are only defined once in parent class.
- **Superclass/ parent class** holds all the common attributes & methods.
- **Subclass/ child class** – inherits all its superclass attributes & methods besides having its own unique attributes & methods.

Inheritance

- *Inheritance* define an ***IS A*** relationship between parent & child class.

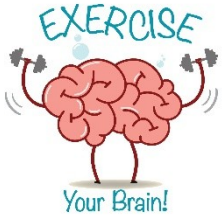


Inheritance in Java



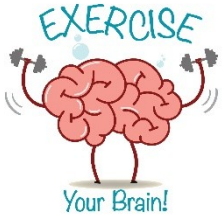
- *Inheritance* is achieved in 2 ways:
  - Extending a class/interface using *extends* keyword.
  - Implementing an interface using *implements* keyword
- Java doesn't allow multi-inheritance between objects.
- Instead an object can inherit different behaviors (methods) through implementing an interface.
- When implementing an interface your class must implement **ALL** its methods.
- An interface can *extend* another interface.

Inheritance in Java

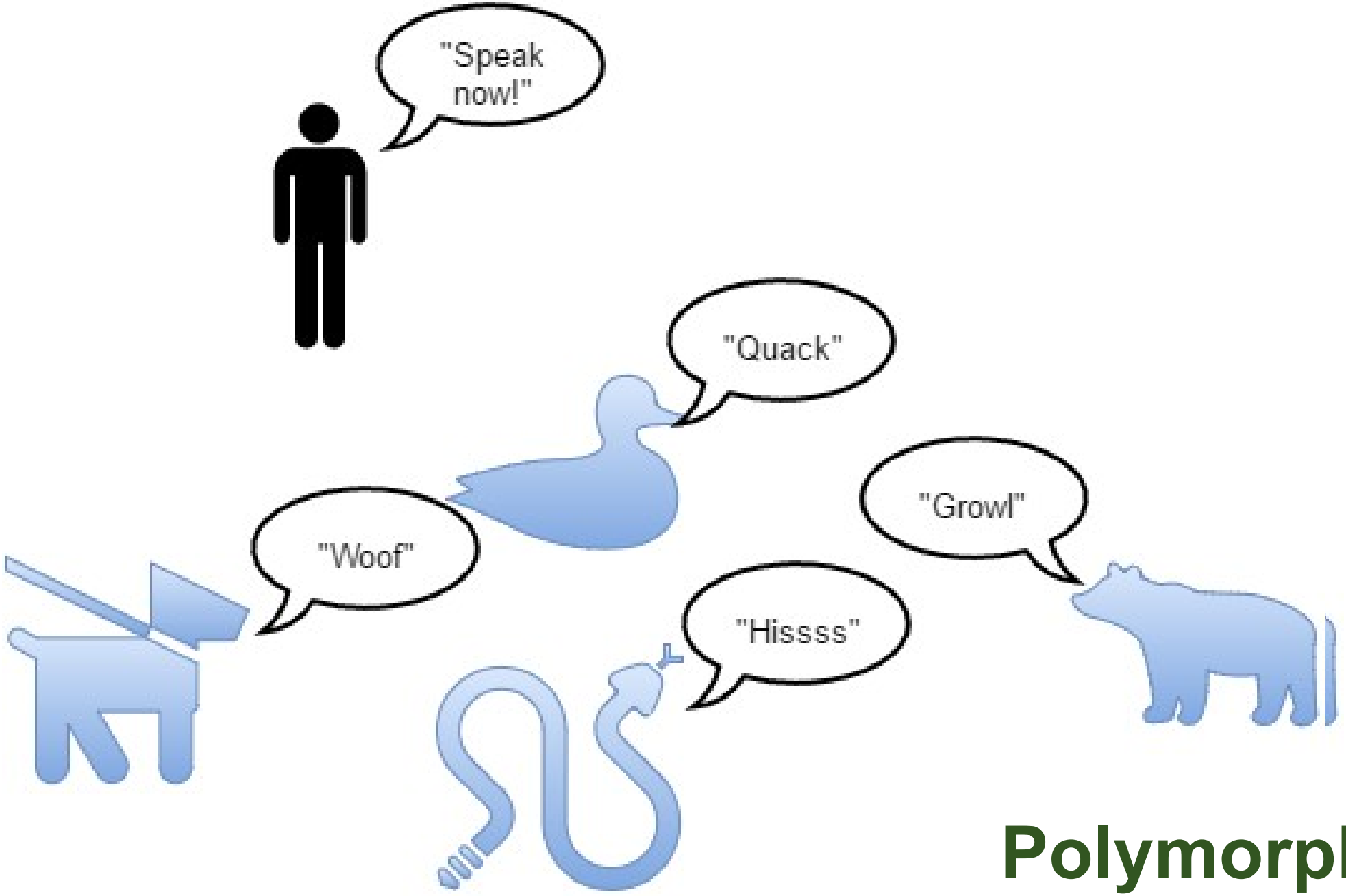


- When you extend a class, your subclass can access all it's protected and public methods and fields and you also *override* them to have a more specific behavior for the child class.
- You can prevent a method in the parent class from being overridden by using the *final* keyword.

Inheritance in Java



- You can force any subclass to implement a superclass's method by marking that method as *abstract*
- This will make the superclass *abstract* which means that you can not instantiate that class.
- A subclass inherits the *type* of the superclass, but not the other way around.



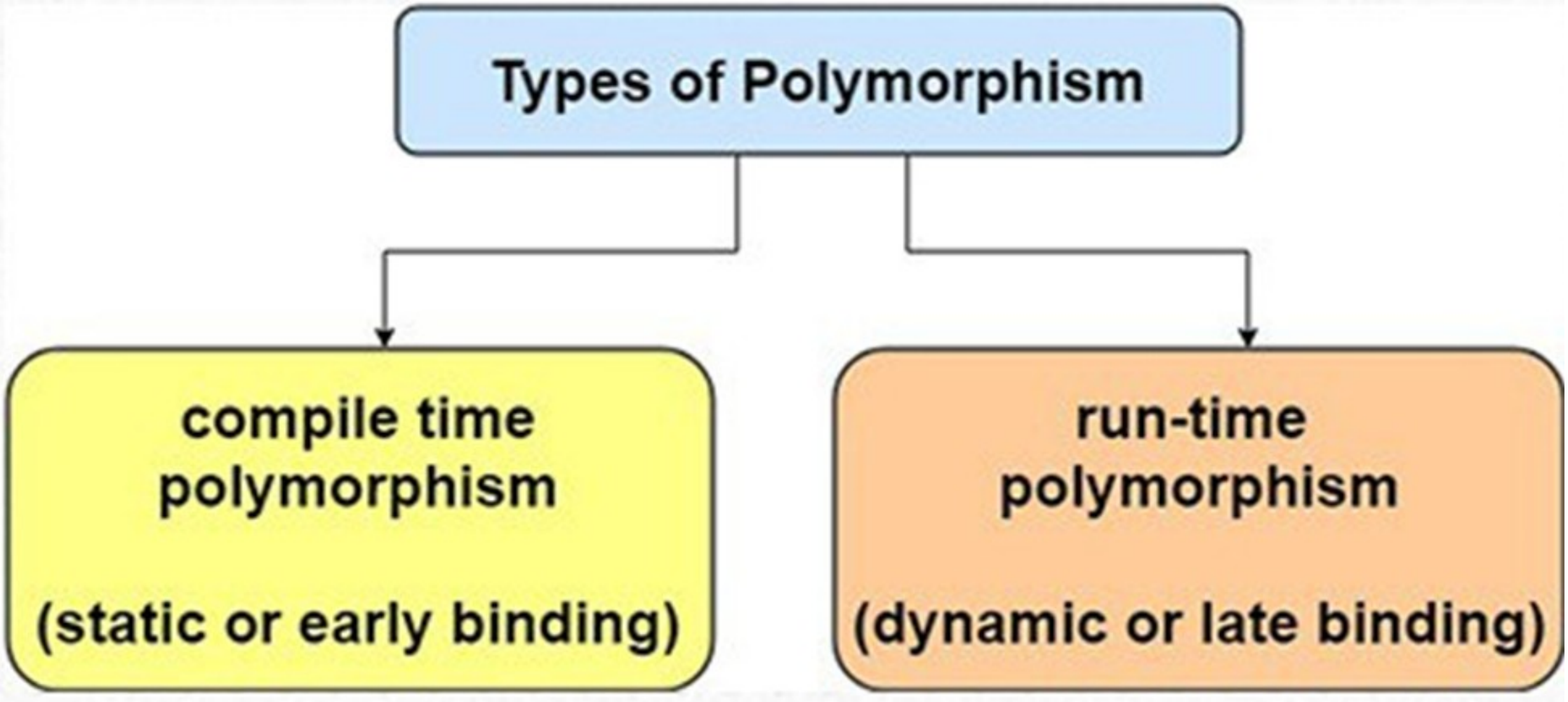
Polymorphism

Polymorphism

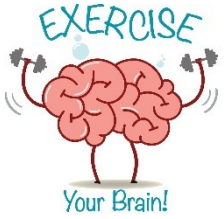
- *Polymorphisms* is a generic term that means 'many forms'.
- In OOP *Polymorphisms* means the ability of the same method to be performed differently based on the type of the calling object.
- In Java, *Polymorphisms* is achieved by the following techniques:
  - method overloading
  - operator overloading

  } **static or compile-time**
  - method overriding. → **dynamic**

Polymorphism



Polymorphism in Java

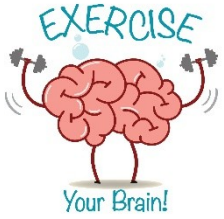


- In **Method overriding** the child class can override a method of its parent class.
- This allows using a single method differently depending on whether it's invoked by an object of the parent class or an object of the child class.
- An overridden method can be invoked by:
  - Superclass referenced as a superclass
  - Subclass referenced as a subclass
  - Subclass referenced as a superclass

Static/compile-time/early binding

Dynamic/runtime/late binding

Polymorphism in Java



- In **Method overloading** a single method performs different functions depending on the context in which it's called.
- That is, a single method name might work in different ways depending on what arguments are passed to it.
- Overloaded methods need to:
  - have a different number of parameters.
  - Or different parameters' type.
  - Or different parameters' order.

Polymorphism

Overriding

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}
class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}
```

Same Method Name,
Same parameter

Overloading

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

Same Method Name,
Different Parameter

Polymorphism

Static Binding	Dynamic Binding
It is a binding that happens at compile time.	It is a binding that happens at run time.
Actual object is not used for binding.	Actual object is used for binding.
It is also called early binding because binding happens during compilation.	It is also called late binding because binding happens at run time.
Method overloading is the best example of static binding.	Method overriding is the best example of dynamic binding.
Private, static and final methods show static binding. Because, they can not be overridden.	Other than private, static and final methods show dynamic binding. Because, they can be overridden.

Q&A