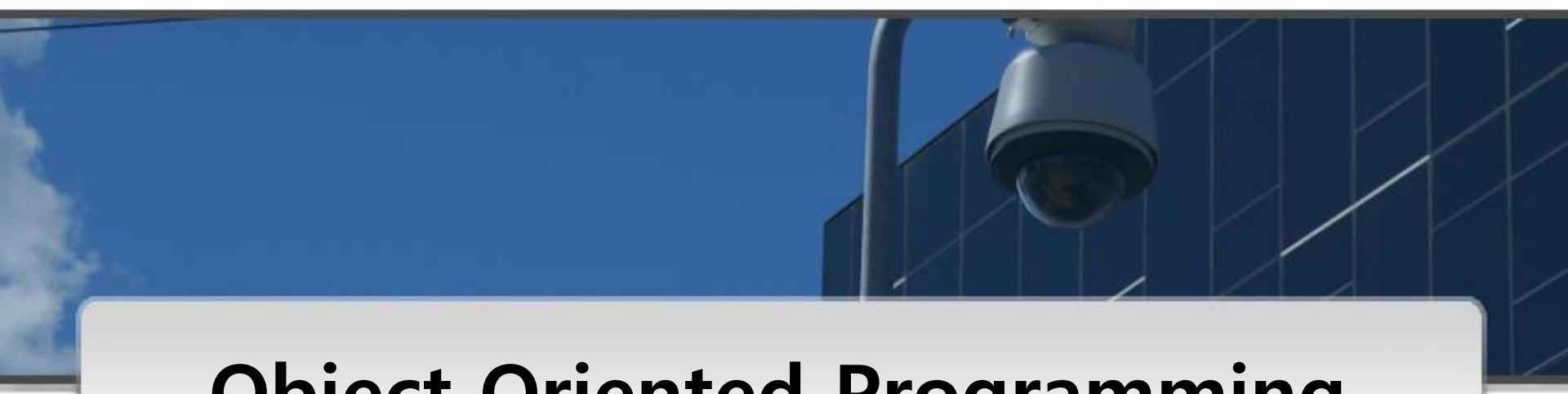


Object Oriented Programming

Inheritance

Lab 7

Content

- Inheritance
- Overriding methods
- Super and this
- Access modifiers
- Object class

Inheritance - Motivation

- **Consider a transportation computer game**
  - Different types of vehicles:
    - Planes
      - Jets, helicopters, space shuttle
    - Automobiles
      - Cars, trucks, motorcycles
    - Trains
      - Diesel, electric, monorail
    - Ships
      - …

- **Let's assume a class is written for each type of vehicle**

Inheritance - Motivation

- **Sample code for the types of planes:**
  - takeOff()
  - fly()
  - setAltitude()
  - land()

- **Note that a lot of this code is common to all types of planes**
  - They have a lot in common!
  - It would be a waste to have to write separate fly() methods for each plane type
    - What if you then have to change one – you would then have to change dozens of methods

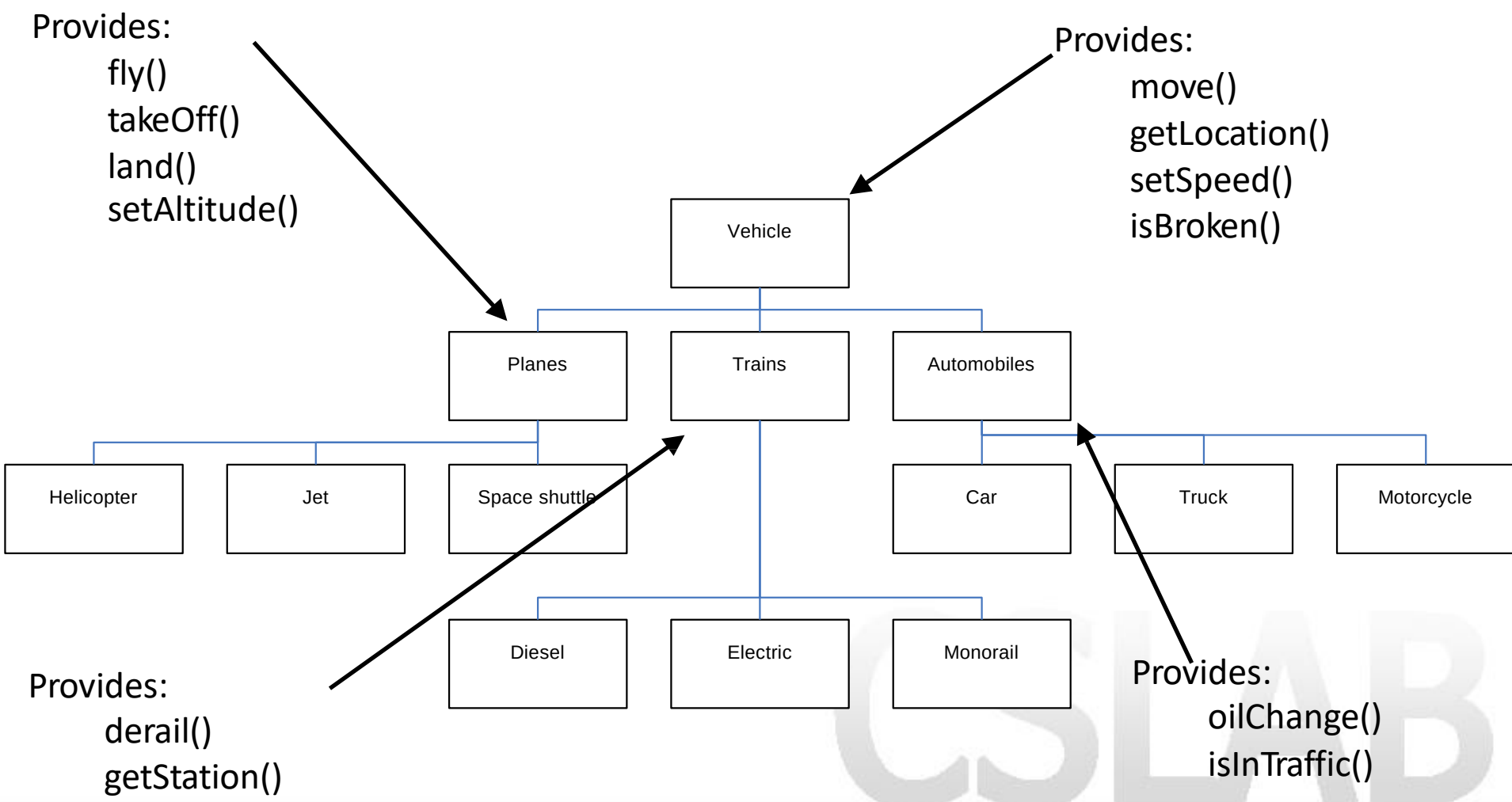
Inheritance - Motivation

- **Indeed, all vehicles will have similar methods:**
  - move()
  - getLocation()
  - setSpeed()
  - isBroken()

- **Again, a lot of this code is common to all types of vehicles**
  - It would be a waste to have to write separate move() methods for each vehicle type
    - What if you then have to change one – you would then have to change dozens of methods

- **What we want is a means to specify one move() method, and have each vehicle type inherit that code**
  - Then, if we have to change it, we only have to change one copy

Inheritance - Motivation



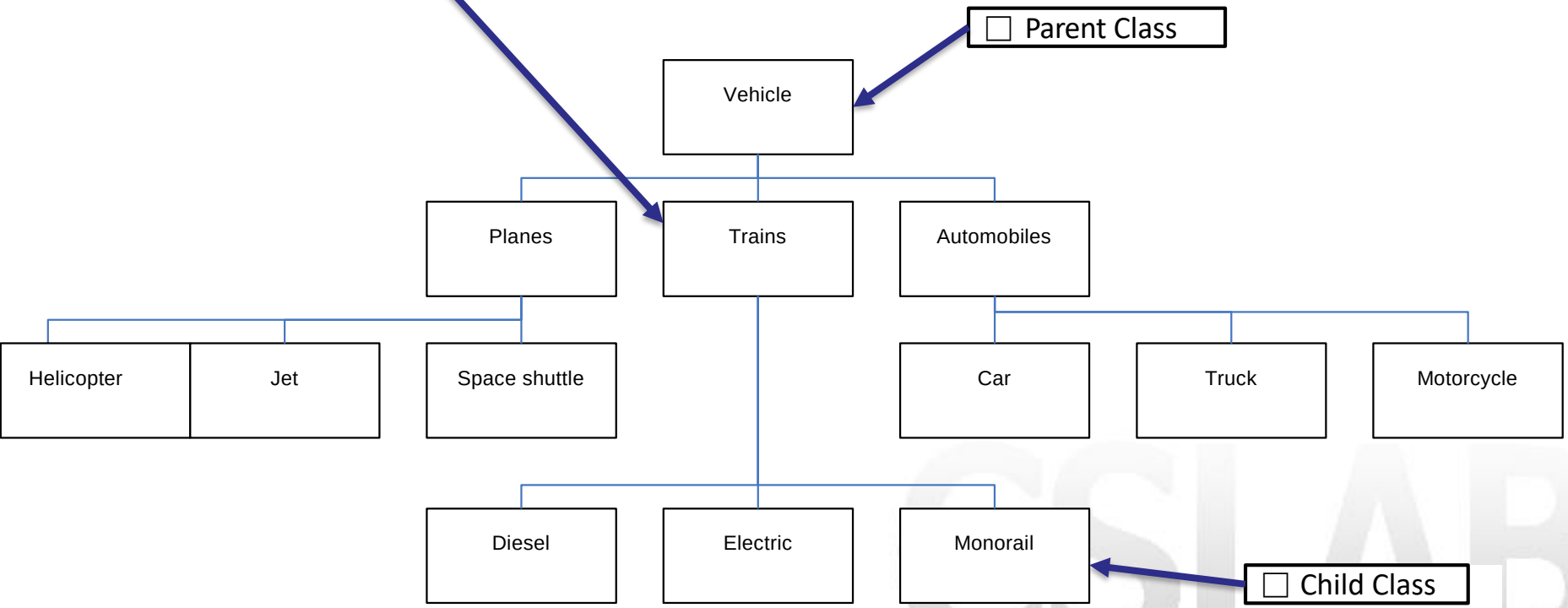
Inheritance - Motivation

- What we will do is create a "parent" class and a "child" class
- The "child" class (or subclass) will inherit the methods (etc.) from the "parent" class (or superclass)
- Note that some classes (such as Train) are both subclasses and super classes

Inheritance - Motivation

☐Child of Vehicle

☐Parent of Diesel, Electric and Monorail



Inheritance

- **Inheritance is the process by which a new class is created from another class**
  - The new class is called a *derived class* or *subclass*
  - The original class is called the *base class*, *parent class* or *superclass*

- **The subclass**
  - Will automatically inherit all methods and instance and static variables from the superclass.
  - Can declare more instance variables and methods

Inheritance code

```
class Vehicle {
    ...
}

class Train extends Vehicles {
  ...
}

class Monorail extends Train {
  ...
}
```

Superclass

Subclass of Vehicle

Subclass of Train

About extends

- **If class A extends class B**
  - Then class A is the subclass of B
  - Class B is the superclass of class A
  - A "is a" B
  - A has (almost) all the methods and variables that B has

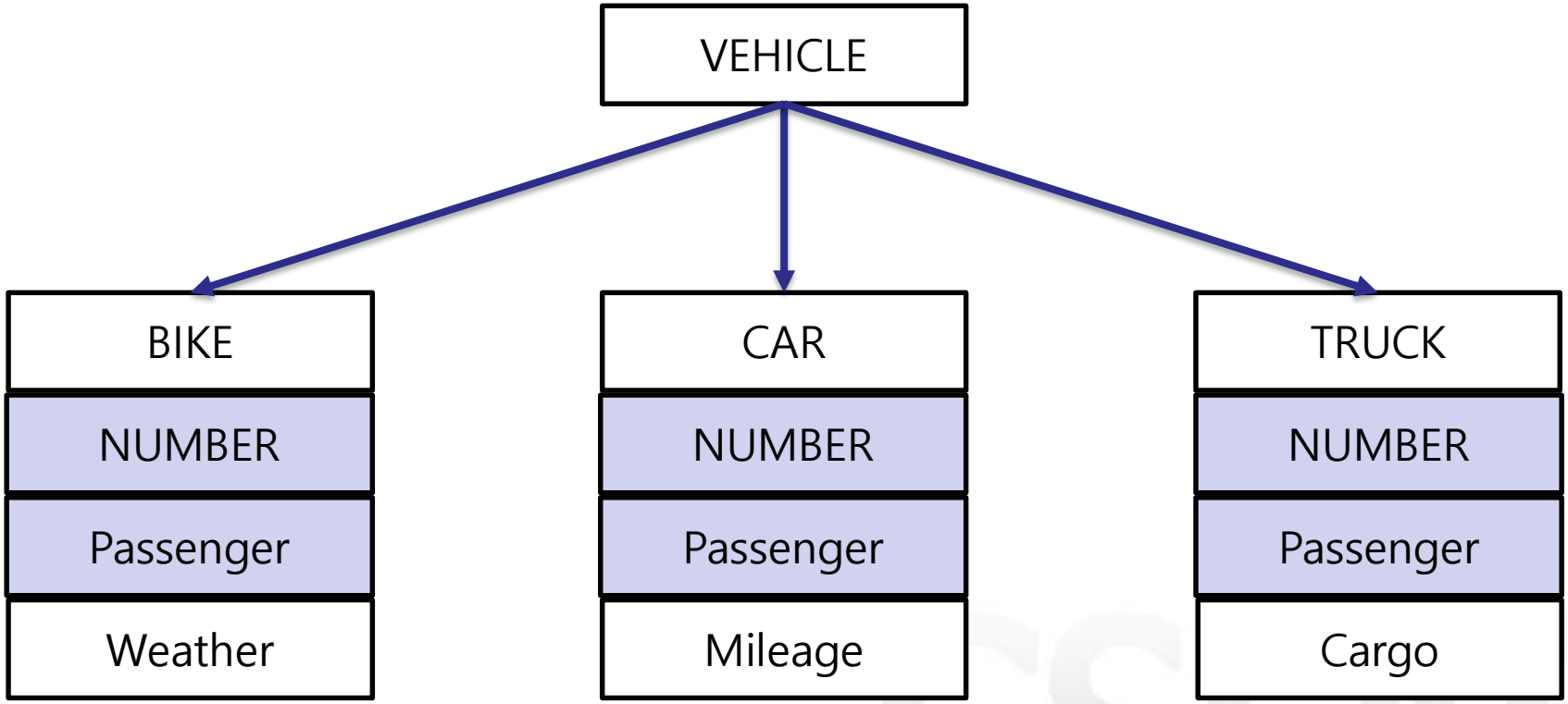
- **If class Train extends class Vehicle**
  - Then class Train is the subclass of Vehicle
  - Class Vehicle is the superclass of class Train
  - Train "is a" Vehicle
  - Train has (almost) all the methods and variables that Vehicle has

Inheritance

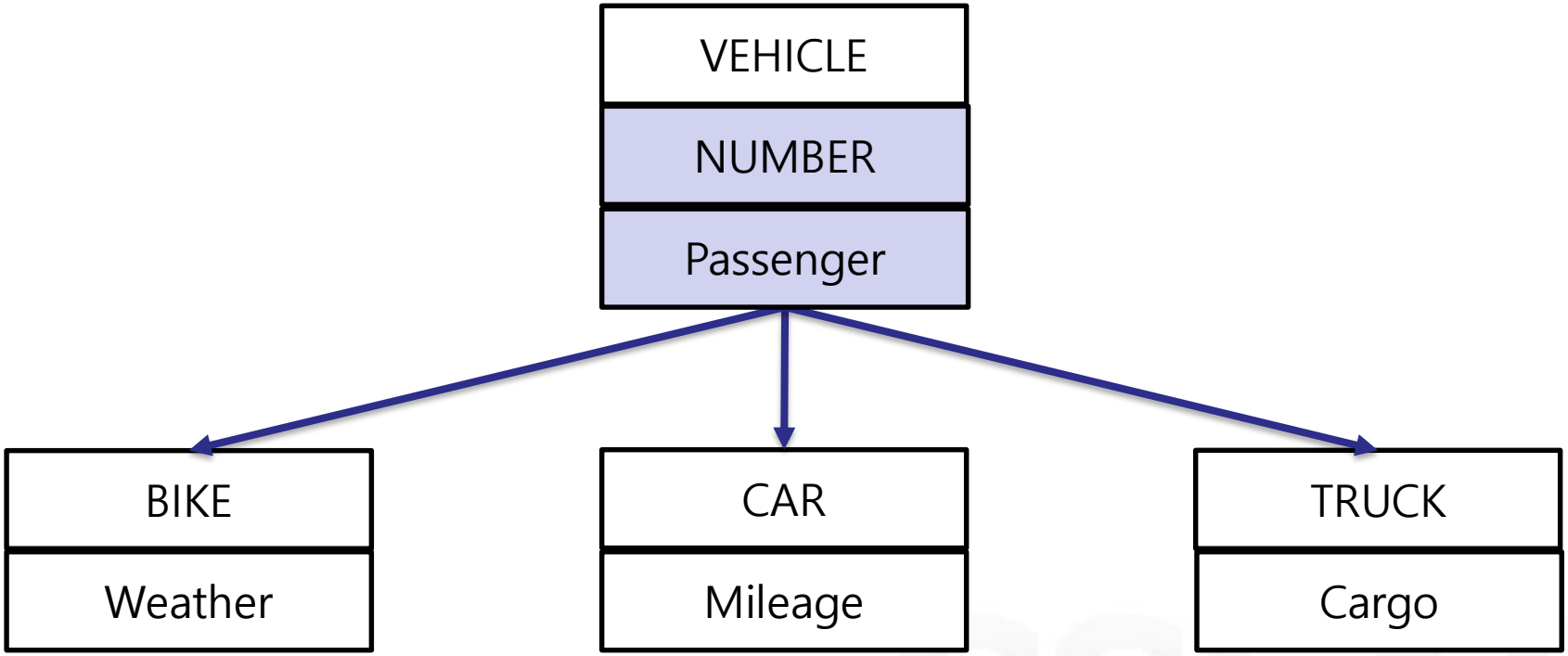
- **Organizes objects in a top-down fashion from most general to least general**

- **Inheritance defines a "is-a" relationship**
  - A mountain bike "is a" kind of bicycle
  - A SUV "is a" kind of automobile
  - A border collie "is a" kind of dog
  - A laptop "is a" kind of computer

Inheritance



Inheritance



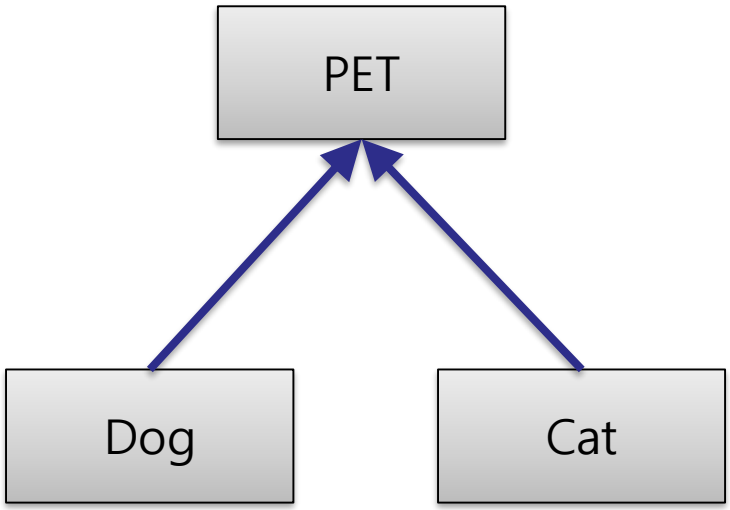
Overriding

- **Overriding is the process of changing the definition of an inherited method**

- **Each time you write a class you may override the methods.**
  - equals(Object obj)
  - toString()

Simple Example

- We can effectively model similar, but different types of objects using inheritance.
- Suppose we want to model dogs and cats. They are different types of pets. We can define the Pet class and the Dog and Cat classes as the subclasses of the Pet class.



The Pet Class

```
class Pet {
        private String name;
        public String getName() {
                return name;
        }
        public void setName(String petName) {
                name = petName;
        }
        public String speak( ) {
                return "I'm your cuddly little pet.";
        }
}
```

Sub-Class of the Pet Class

```
class Cat extends Pet {
    public String speak( ) {
        return "Don't give me orders.\n" +
               "I speak only when I want to.";
    }
}
```

The Cat subclass **overrides** the inherited method speak.

```
class Dog extends Pet {
    public String fetch( ) {
        return "Yes, master. Fetch I will.";
    }
}
```

The Dog subclass adds a new method fetch.

Sample Usage of the Subclasses

```
Dog myDog = new Dog();

System.out.println(myDog.speak());
System.out.println(myDog.fetch());
```

```
I'm your cuddly little pet.
Yes, master. Fetch I will.
```

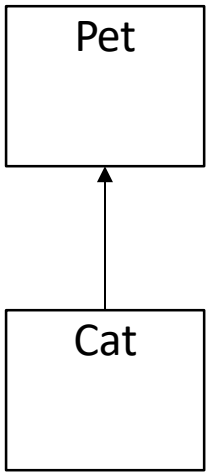
```
Cat myCat = new Cat();

System.out.println(myCat.speak());
System.out.println(myCat.fetch());
```

```
Don't give me orders.
I speak only when I want to.
```

ERROR

Overriding

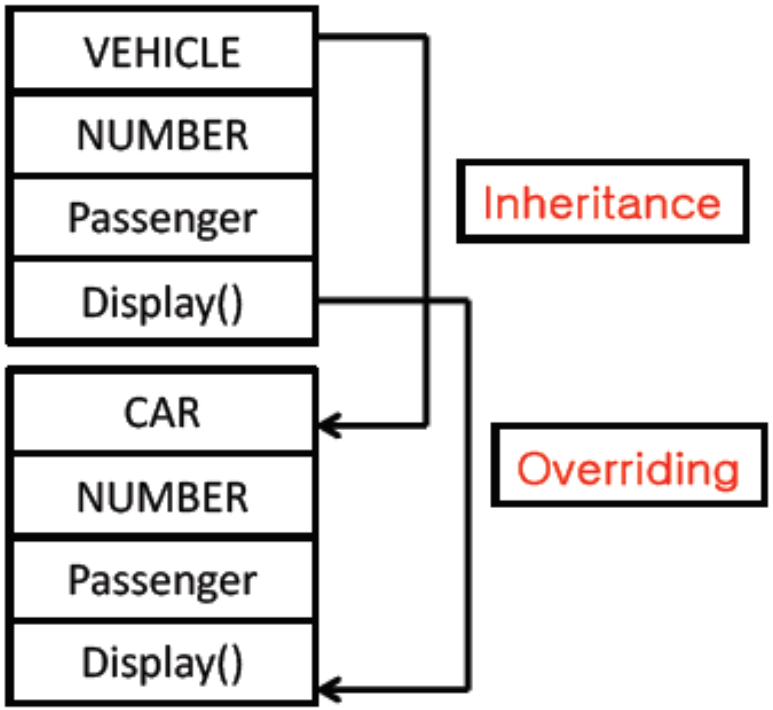
- **To override a method**
  - Create a method in the subclass with the same header as the method in the parent class.



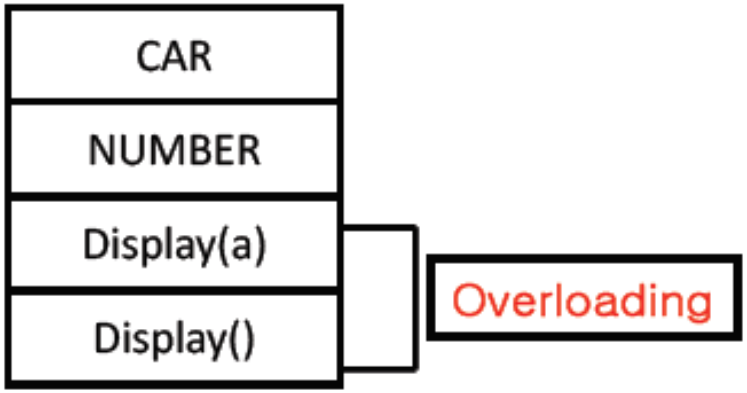

```
public String speak( ) {
    return "I'm your cuddly little pet.";
}

public String speak( ) {
    return "Don't give me orders.\n" +
           "I speak only when I want to.";
}
```

Do not confuse Overriding, Overloading

Overriding



Overloading



The final Modifier

- If the modifier final is placed before the definition of a method, then that method may not be redefined in a derived class.

- If the modifier final is placed before the definition of a class, then that class may not be used as a base class to derive other classes.

- final prevents inheritance.

Self-Test (1)

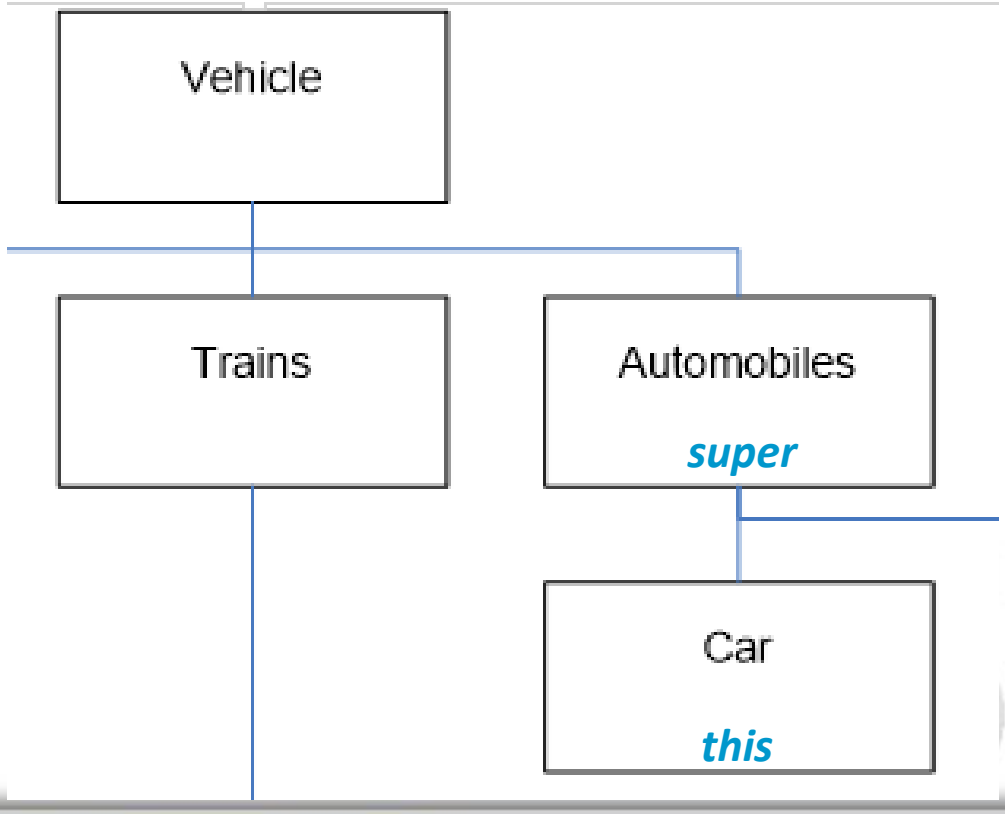
- **Employee, Manager, Engineer Class를 생성한다.**
  - Employee Class에는 3개의 instance 변수가 있다.
    - String name
    - int employeeNum
    - String department
  - department는 초기값은 "No Dept"로 설정 한다.
  - department를 get/set하기 위한 접근자와 변형자를 작성한다. (public)
  - Employee의 name과 employeeNum이 같은경우 true, 다른 경우 false를 반환하는 equals(Object obj) 메소드를 작성한다.
  - name과 employeeNum을 반환하는 toString() 메소드를 작성한다.
    (형식 : Name : [name]₩nEmp# : [employeeNum])
- **Manager Class는 Employee Class를 extend한다.**
  - Manager Class에는 2개의 instance 변수가 있다.
    - int officeNum
    - String team
  - officeNum, team을 인자로 받는 Manager 객체의 생성자를 작성한다.
  - Manager의 name과 employeeNum, location및 하는 일을 반환하는 toString() 메소드를 만든다.
    이때, locatio은 department 및 officeNum이다.
    (형식 : Name : [name]₩nlocation : [department], [officeNum])

Inheritance and Constructors

- Unlike members of a base class, constructors of a base class are not inherited by its subclasses
- You must define a constructor for subclasses
- It is useful to use the constructor of the parent class within the constructor of the child class.
- This can be done through the keyword super.

Inheritance and Constructors

- super like the word this can be used to refer to different objects in the hierarchy.



super Constructor

- **To define a constructor using the superclass's constructor**
  - add the **super constructor** as the first statement of the derived constructor.

```
public Car(){
        super();
        this.number = 0;
}

public Car(Colour col, Number num){
        super(col);
        this.number = num;
}
```

this Constructor

- **The this constructor refers to an overloaded constructor in the same class.**
  - i.e. Within the definition of a constructor for a class, this can be used as a name for invoking a constructor of the same class

```
class Car extends Automobiles {
        private int number;

        public Car(){
                super();
                super.number = 1;
        }

        public Car(Number num){
                this();
                this.number = num;
        }
}
```

If it is necessary to include a call to both **`super`** and **`this`**, the call using **`this`** must be made first, and then the constructor that is called must call **`super`** as its first action

The *this* Constructor

- **Often, a no-argument constructor uses this to invoke an explicit-value constructor**
  - No-argument constructor (invokes explicit-value constructor using this and default arguments):

```
public ClassName() {
        this(argument1, argument2);
}
```

  - Explicit-value constructor (receives default values):

```
public ClassName(type1 param1, type2 param2) {
        . . .
}
```

Access Modifiers

- **Private instance variables in a superclass are not accessible by name in a subclass**
  - They can be accessed using accessors and mutators.

- **Private methods of a superclass are also not directly accessible by name in a subclass**
  - They may be indirectly accessed. If a subclass accesses a public method that accesses that private method. (by proxy)

The Protected Modifier

- **The modifier protected makes a variable or method visible and accessible to**
  - the instances of the class
  - Instances of subclasses of the class.
  - Instances of classes in the same package

- **Public data members and methods are accessible to everyone.**

- **Private data members and methods are accessible only to instances of the class.**

Packages

- Allow definitions to be collected together into a single entity — a package.
- Classes and names in the same package are stored in the same folder.
- Classes in a package go into their own "namespace" and therefore the names in a particular package do not conflict with other names in other packages

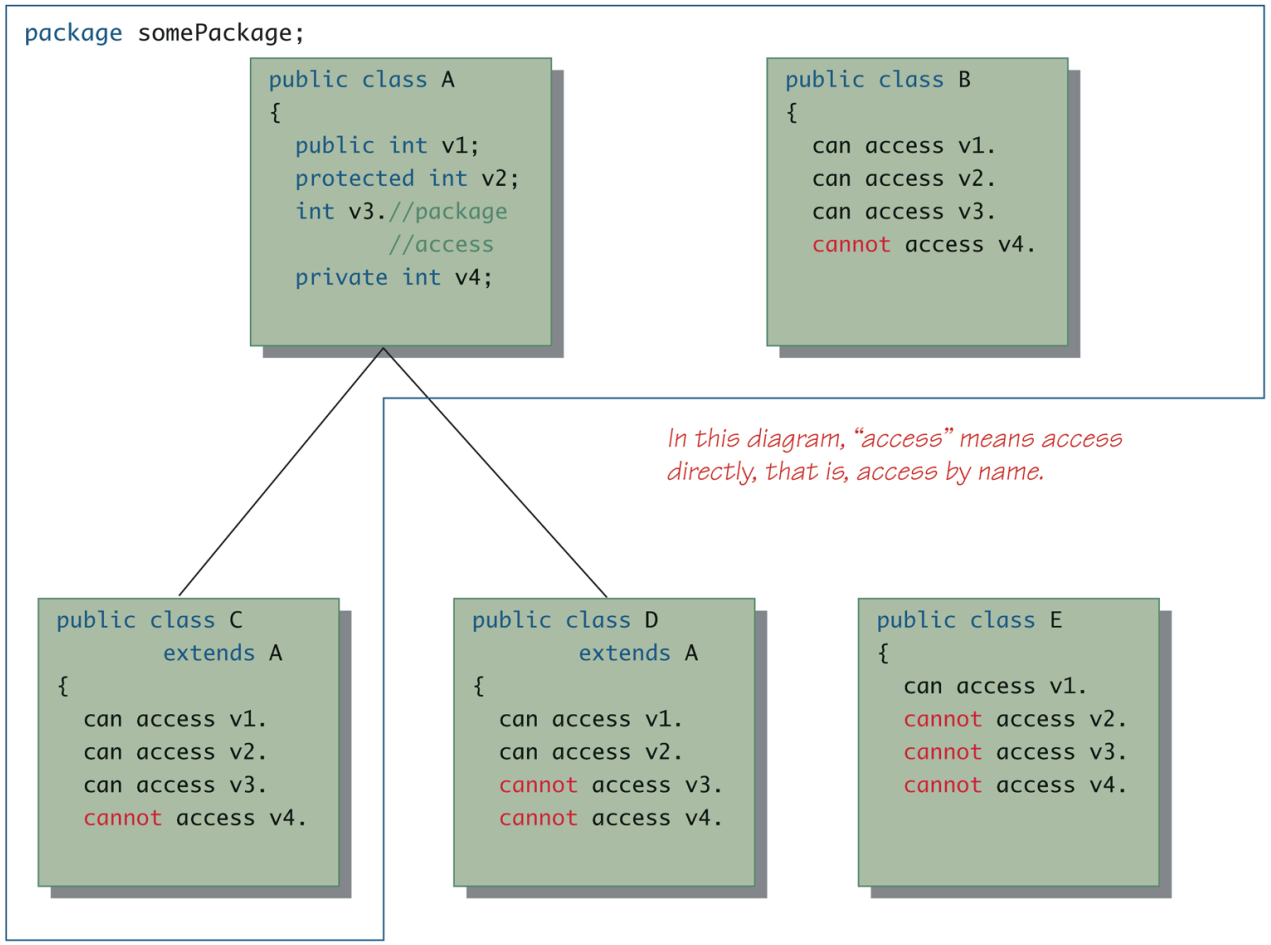
Controlling access

- **Class access rights**

Modifier	Class	Package	Subclass	World
public	O	O	O	O
protected	O	O	O	
no modifier	O	O		
private	O			

Access Modifiers

Display 7.9 Access Modifiers



Changing the Access Permission of an Overridden Method

- The access permission of an overridden method can be changed from private in the base class to public (or some other more permissive access) in the derived class

- However, the access permission of an overridden method cannot be changed from public in the base class to a more restricted access permission in the derived class

Changing the Access Permission of an Overridden Method

- **Given the following method header in a base case:**
  - private void doSomething()
- **The following method header is valid in a derived class:**
  - public void doSomething()
- **However, the opposite is not valid**

- **Given the following method header in a base case:**
  - public void doSomething()
- **The following method header is not valid in a derived class:**
  - private void doSomething()

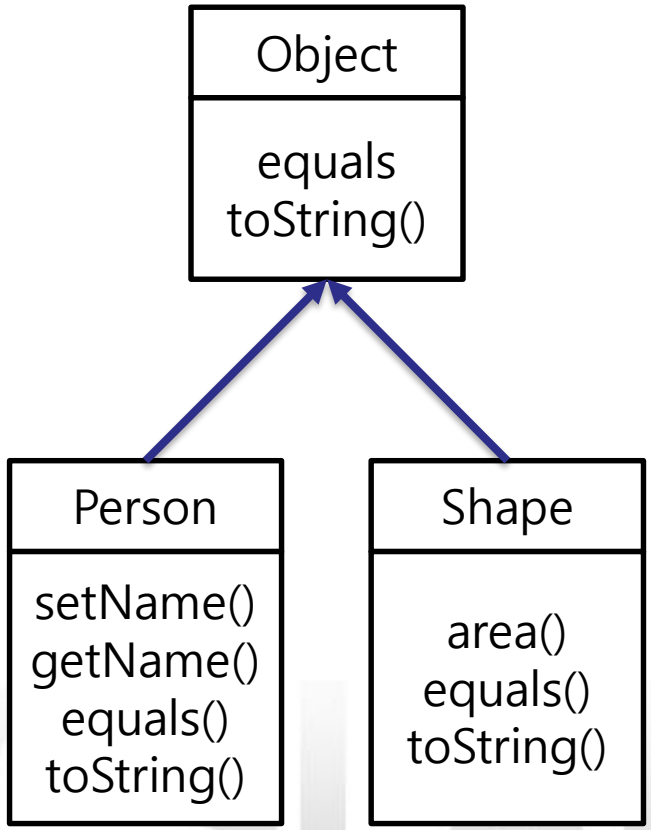
Object class

- Java contains a special class in the java.lang package known as Object
- All classes in java are inherited from the Object class. This means all classes are Objects.
- This general object can be used as a stand-in for any class.
- Some methods may be written to accept parameters of type Object since any class is an Object.

Object class

- **The class Object has some methods that every Java class inherits**
  - equals(Object obj)
  - toString()

- **It is necessary to override these methods in classes written in Java.**



Better equals Method

- The equals method is inherited from the Object class.
- When overriding the equals method it should accept an Object as the parameter.

```
public boolean equals(Object obj) {
        . . .
}
```

- The obj is compare by casting. It to the type of object that you would like to compare to.
- Our equals method should check if obj is not null and is really of the type you are trying to cast it to.

Better equals Method

```
public boolean equals(Object obj) {
    if(obj == null) return false;
    else if(getClass( ) != obj.getClass( )) return false;
    else {
        Employee otherEmp = (Employee)obj;
        return (name.equals(otherEmp.name) && hireDate.equals(otherEmp.hireDate));
     }
}
```

The instanceof Operator

- **The instanceof operator checks if an object is of the type given as its second argument**
- **Object instanceof ClassName**
  - This will return **true** if **Object** is of type **ClassName**, and otherwise return **false**
  - Note that this means it will return **true** if **Object** is the type of ***any descendent*** class of **ClassName**

The *getClass()* Method

- **Every object inherits the same *getClass()* method from the *Object* class**
  - This method is marked final, so it cannot be overridden

- **An invocation of *getClass()* on an object returns a representation only of the class that was used with *new* to create the object**
  - The results of any two such invocations can be compared with == or != to determine whether or not they represent the exact same class

```
(object1.getClass() == object2.getClass())
```

Self-Test (2)

- **Manager Class를 수정한다.**
  - Manager의 name, employeeNum, officeNum, team이 동일할 경우 true, 다른 경우 false를 반환하는 equals(Object obj) 메소드를 작성한다.
    (Super Class의 equals()를 사용한다.)
  - Manager의 생성자가 name과 employeeNum, officeNum, team을 받도록 수정한다.
- **Employee Class를 extends하는 Engineer Class를 작성한다.**
  - Engineer Class는 2개의 instance 변수가 있다.
    - String workZone, String protject
  - name, employeeNum, workZone, project, name을 갖는 생성자를 작성한다.
  - Engineer Class의 name이 동일한 경우 true를 반환하는 equals(Object obj) 메소드를 작성한다.
  - Engineer의 name, employeeNum, officeNum, workZone이 동일하다면 true를, 다르다면 false를 반환하는 equals(Object obj) 메소드를 작성한다.
    (Super Class의 equals()를 사용한다.)
  - Engineer의 name, employeeNum, location, workZone을 반환하는 toString()을 작성한다.
    (형식 : Name : [name]₩nEmp# : [employeeNum]₩nlocation : [department], [workZone])