CCS0023/L

Object Oriented Programming (Java)

Encapsulation and Inheritance

06

LESSON #6

Encapsulation and Inheritance

Encapsulation

➢The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

- declare class variables/attributes as private (only accessible within the same class)
- provide public setter and getter methods to access and update the value of a private variable

Get and Set

- ➢Private variables can only be accessed within the same class (an outside class has no access to it)
- ➢However, it is possible to access them if we provide public **getter (accessor**) and **setter (mutator)** methods.
- ➢The **get** method returns the variable value, and the **set** method sets the value.

Get and Set

```
public class Person {
      private String name; // private = restricted access

      // Getter
      public String getName() {
        return name;
      }

      // Setter
      public void setName(String newName) {
        this.name = newName;
      }
    }
```

The get method returns the value of the variable name.

The set method takes a parameter (newName) and assigns it to the name variable.

The this keyword is used to refer to the current object.

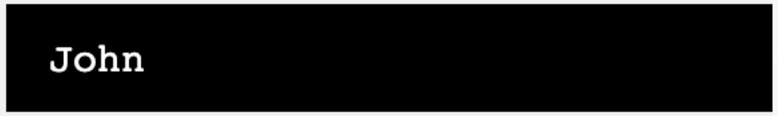
Get and Set

```
public class MyClass {
    public static void main(String[] args) {
      Person myObj = new Person();
      myObj.name = "John";  // error
      System.out.println(myObj.name); // error
    }
  }
```

However, as the name variable is declared as private,
we cannot access it from outside this class.

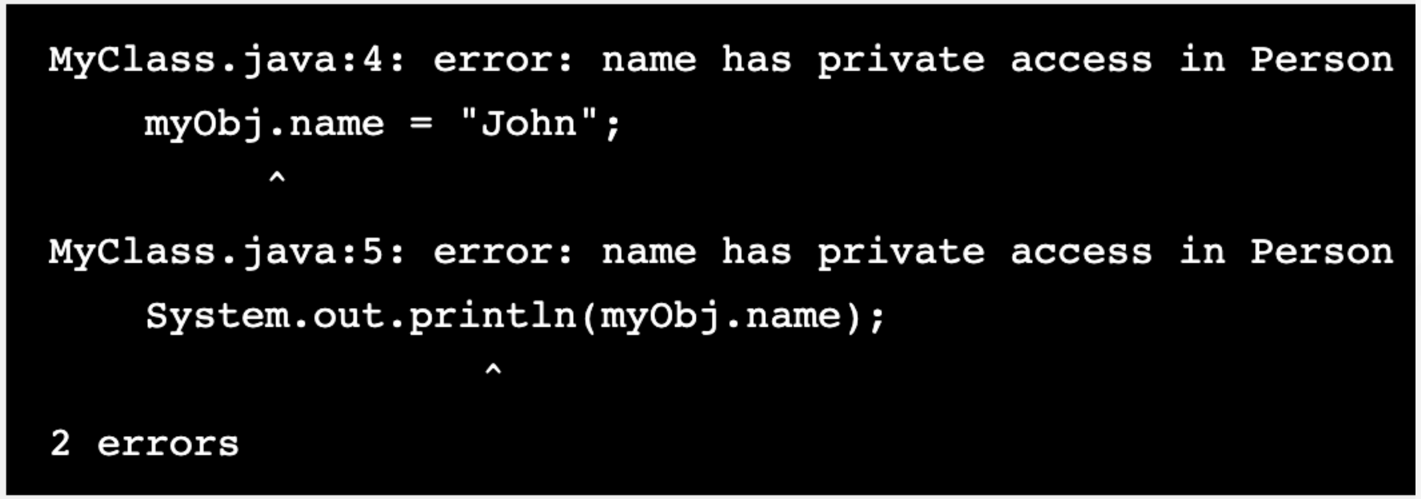
Get and Set

If the variable was declared as public, we would expect the following output:



```
John
```

However, as we try to access a private variable, we get an error:



```
MyClass.java:4: error: name has private access in Person
    myObj.name = "John";
         ^
MyClass.java:5: error: name has private access in Person
    System.out.println(myObj.name);
                            ^
2 errors
```

Get and Set

```
public class MyClass {
    public static void main(String[] args) {
      Person myObj = new Person();
      myObj.setName("John"); // Set the value of the name variable to "John"
      System.out.println(myObj.getName());
    }
  }

  // Outputs "John"
```

Instead, we use the getName() and setName() methods to acccess and update the variable.

Why Encapsulation?

- Better control of class attributes and methods
- Class variables can be made **read-only** (if you omit the <u>set</u> method), or **write-only** (if you omit the <u>get</u> method)
- Flexible: the programmer can change one part of the code without affecting other parts
- Increased security of data

Inheritance

➢The technique of deriving new class definitions from an existing class definition.

Inheritance

➢The following are the benefits of using class inheritance in OOP:

- Re-use of predefined and well-tested classes
- Standardization of behaviors across a group of classes
- Ability to use members of a family of classes interchangeably in methods

Inheritance

Superclasses and Subclasses

➢Superclass is the class from which another class inherits properties. This is a class that is on top of a hierarchy.

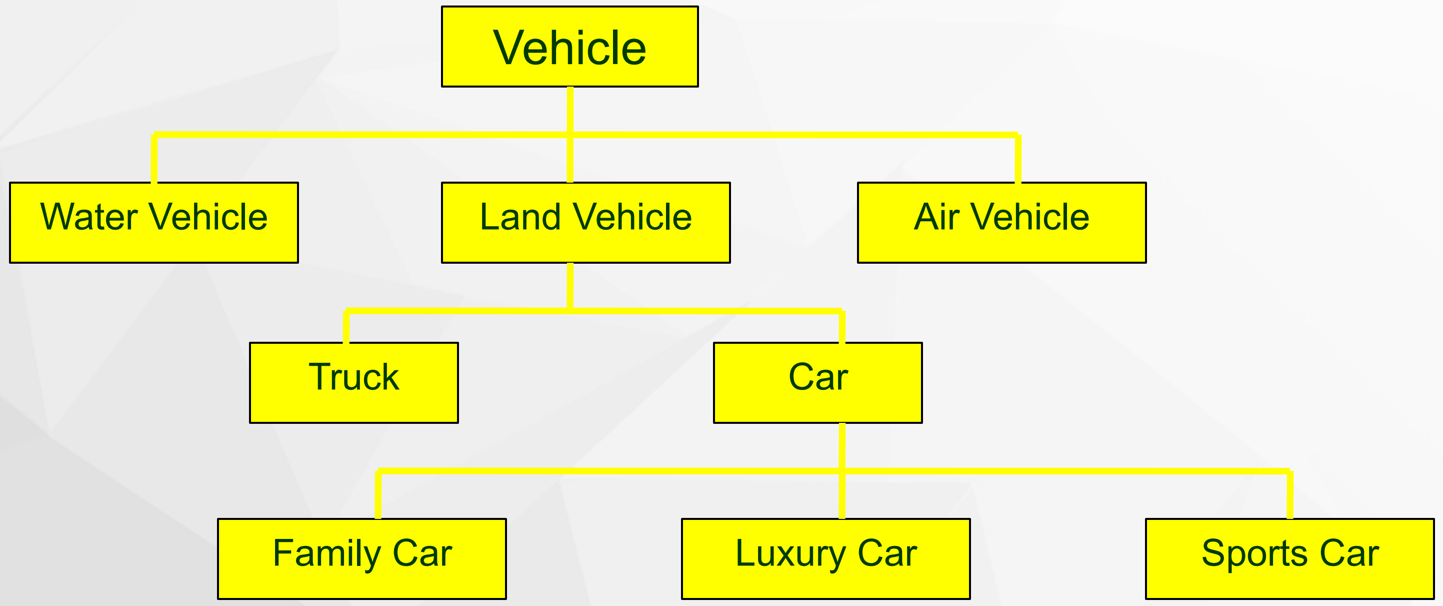
Inheritance

Superclasses and Subclasses

➢Subclass is a class that inherits all the non-private attributes and methods, except constructors from a superclass. This class has the ability to override methods of the superclass.

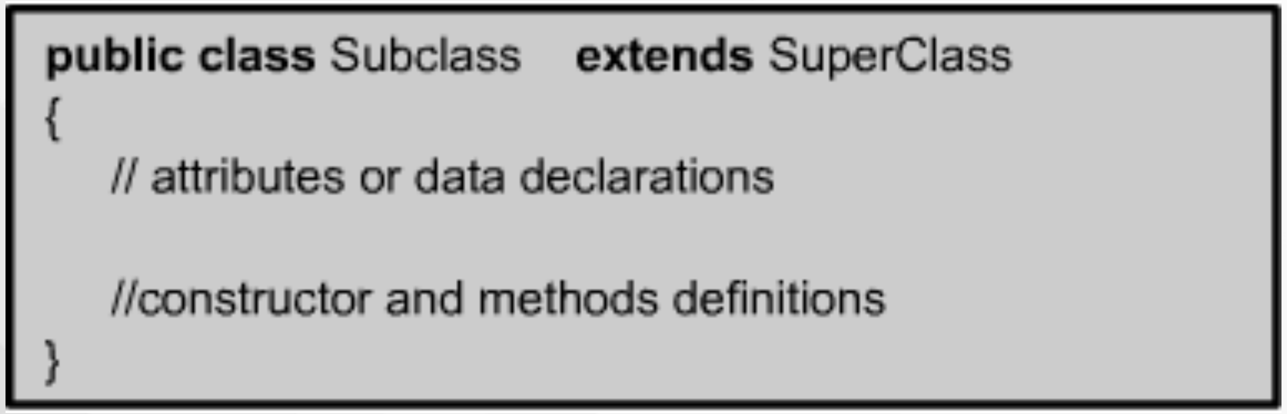
Inheritance

Vehicle Class Hierarchy



Inheritance

Syntax for Implementing Inheritance:



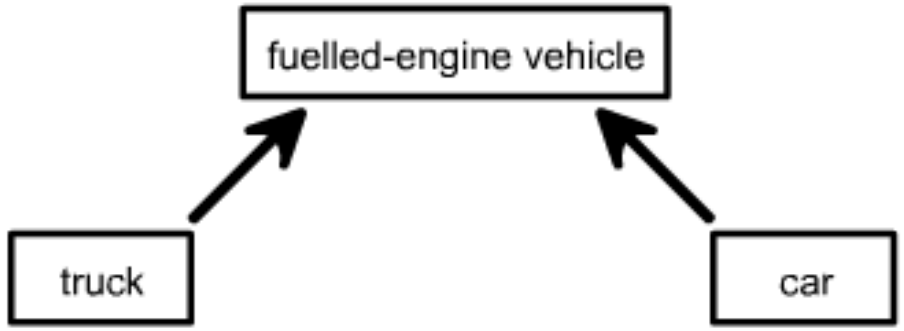
```
public class Subclass    extends SuperClass
{
     // attributes or data declarations

     //constructor and methods definitions
}
```

Inheritance

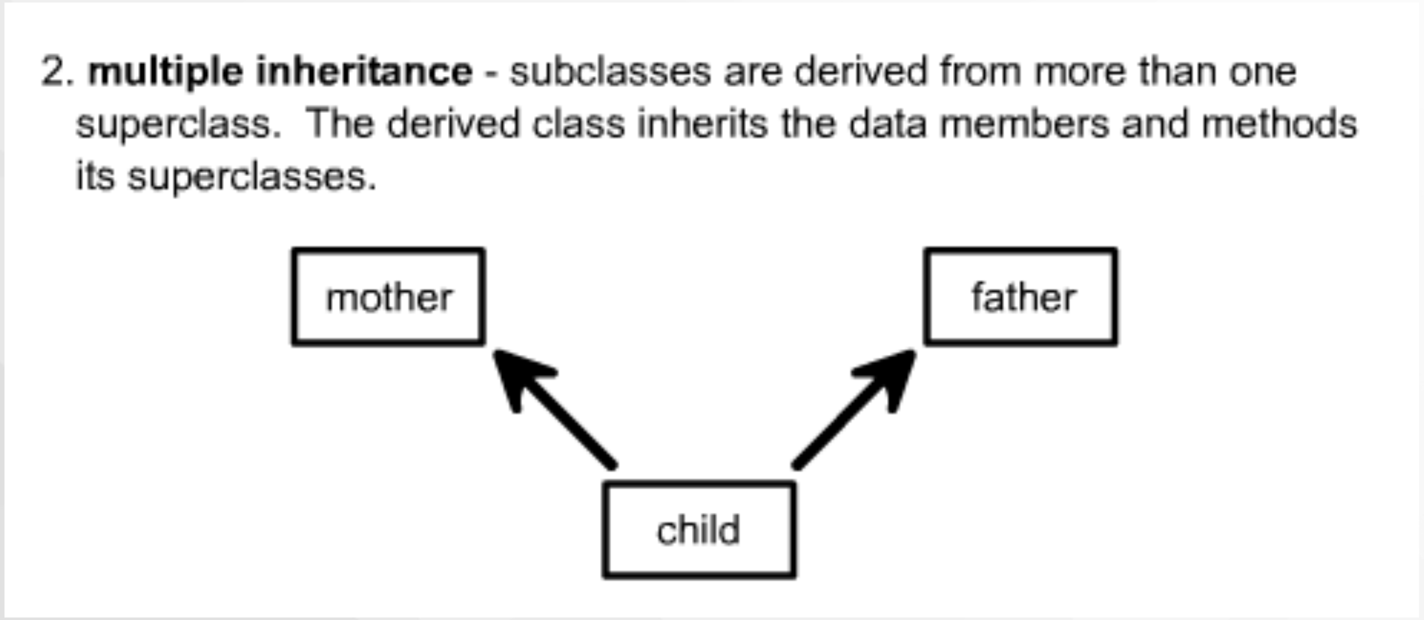
Types of Inheritance:

1. **single inheritance** - subclasses are derived from a single superclass. The derived class inherits the data members and methods of the superclass.



Inheritance

Types of Inheritance:



2. **multiple inheritance** - subclasses are derived from more than one superclass. The derived class inherits the data members and methods its superclasses.

$$\begin{tikzcd} \boxed{\text{mother}} & & \boxed{\text{father}} \\ & \boxed{\text{child}} \arrow[ul] \arrow[ur] & \end{tikzcd}$$

Inheritance

Take Note:

Java does not support multiple inheritance, however, Java provides same effects and benefits of multiple inheritance through the use of interface.

Inheritance

The this and the super Keywords:

➢this – contains a reference to the current object being constructed. It represents an instance of the class in which it appears. It can be used to access class variables and methods.

Inheritance

The this and super Keywords:

- **super –** contains a reference to the parent class (superclass) object. It is used to access members of a class inherited by the class in which it appears. It explicitly call a constructor of its immediate superclass. A super constructor call in the constructor of a subclass will result in the execution of relevant constructor from the superclass, based on the arguments passed.

Inheritance

Defining Superclasses

```
public class Person
{
    protected String name;
    protected String address;
/* Default constructor*/
    public Person(){
    System.out.println(“Inside person:Constructor”);
     name = "";
    address = "";
}
```

Inheritance

```
/*Constructor with 2 parameters*/
public Person( String name, String address ){
this.name = name;
this.address = address;
}
```

Inheritance

```
/*Accessor and Mutator methods*/
public String getName(){
return name;
}
public String getAddress(){
return address;
}
public void setName( String name ){
this.name = name;
}
public void setAddress( String add ){
this.address = add;
}}
```

Inheritance

Defining Subclasses

```
public class Student extends Person
{
public Student(){
System.out.println(“Inside Student:Constructor”);
//some code here
}
// some code here
}
```

Inheritance

Defining Subclasses

When a Student object is instantiated, the default constructor of its superclass is invoked implicitly to do the necessary initializations. After that, the statements inside the subclass are executed.

```
public static void main( String[] args ){
Student anna = new Student();
}
```

Inheritance

Defining Subclasses

In the code, we create an object of class Student. The output of the program is,

```
Inside Person:Constructor
Inside Student:Constructor
```

Inheritance

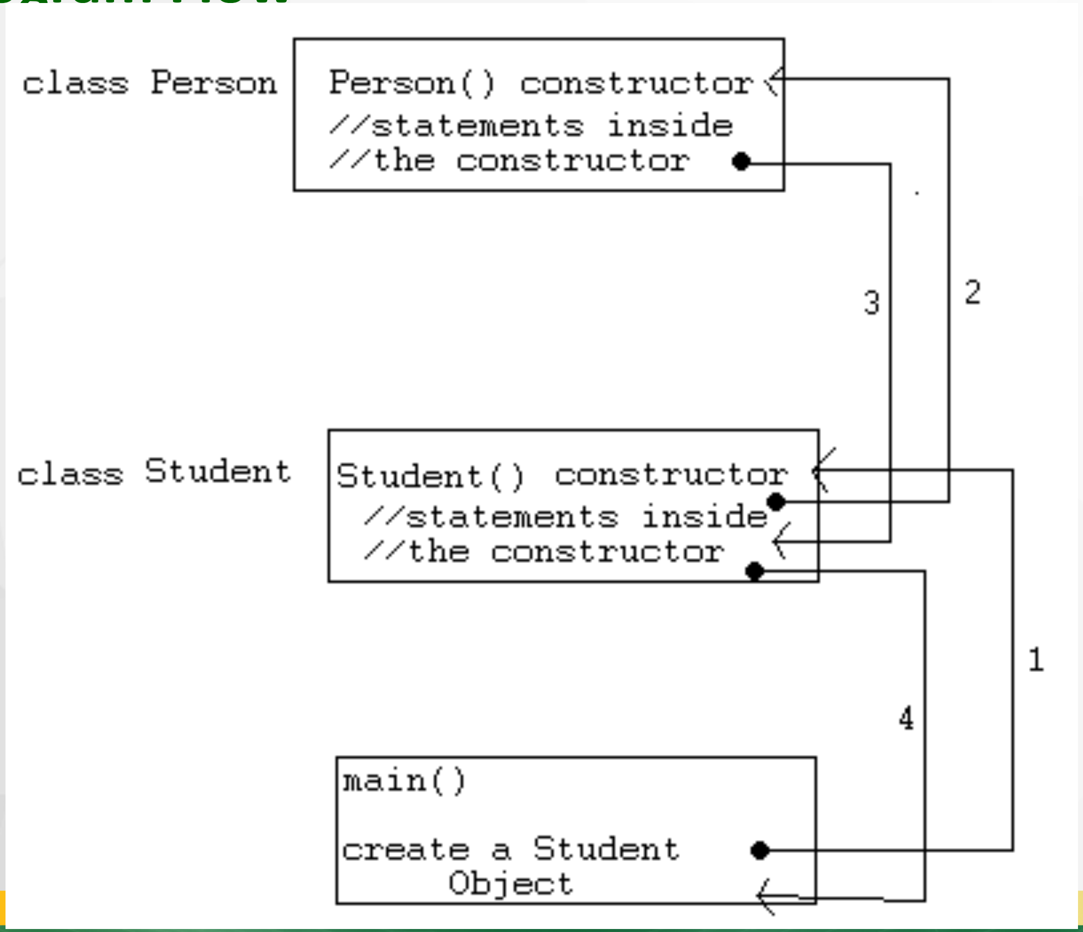
Defining Subclasses

In the code, we create an object of class Student. The output of the program is,

```
Inside Person:Constructor
Inside Student:Constructor
```

Inheritance

Program Flow



Inheritance

super keyword

```
public Student(){
super( "SomeName", "SomeAddress" );
System.out.println("Inside Student:Constructor");
}
```

This code calls the second constructor of its immediate superclass (which is Person) and executes it.

Inheritance

super keyword

Another sample code shown below,

```
public Student(){
super();
System.out.println("Inside Student:Constructor");
}
```

This code calls the default constructor of its immediate superclass (which is Person) and executes it.

Inheritance

There are a few things to remember when using the super constructor call:

1. The super() call MUST OCCUR THE FIRST STATEMENT IN A CONSTRUCTOR.
2. The super() call can only be used in a constructor definition.
3. This implies that the this() construct and the super() calls CANNOT BOTH OCCUR IN

THE SAME CONSTRUCTOR.

Inheritance

Another use of super is to refer to members of the superclass (just like the this reference).

For example,

```
public Student()
{
super.name = “somename”;
super.address = “some address”;
}
```

Overriding Methods

A subclass can override a method defined in its superclass by providing a new implementation for that method.

Suppose we have the following implementation for the getName method in the Person superclass,

```
public class Person
{
        :
        :
        public String getName(){
        System.out.println("Parent: getName");
        return name;
        }
        :
}
```

Overriding Methods

To override, the getName method in the subclass Student, we write,

```
public class Student extends Person
{
        :
        :
        public String getName(){
        System.out.println("Student: getName");
        return name;
        }
        :
}
```

Overriding Methods

So, when we invoke the getName method of an object of class Student, the overridden method would be called, and the output would be,

```
Student: getName
```

Final Methods and Final Classes

In Java, it is also possible to declare classes that can no longer be subclassed. These classes are called **final classes.**

To declare a class to be final, we just add the final

keyword in the class declaration. For example, if we want the class Person to be declared

final, we write,

```
public final class Person
{
     //some code here
}
```

Final Methods and Final Classes

It is also possible in Java to create methods that cannot be overridden. These methods are what we call final methods. To declare a method to be final, we just add the final keyword in the method declaration.

For example, if we want the getName method in class Person to be declared final, we write,

```
public final String getName(){
    return name;
}
```

Static methods are automatically final. This means that you cannot override them.

THANK YOU