Vietnam National University of HCMC
International University
School of Computer Science and Engineering

Inheritance

(IT069IU)

Le Duy Tan, Ph.D.

ldtan@hcmiu.edu.vn

leduytanit.com

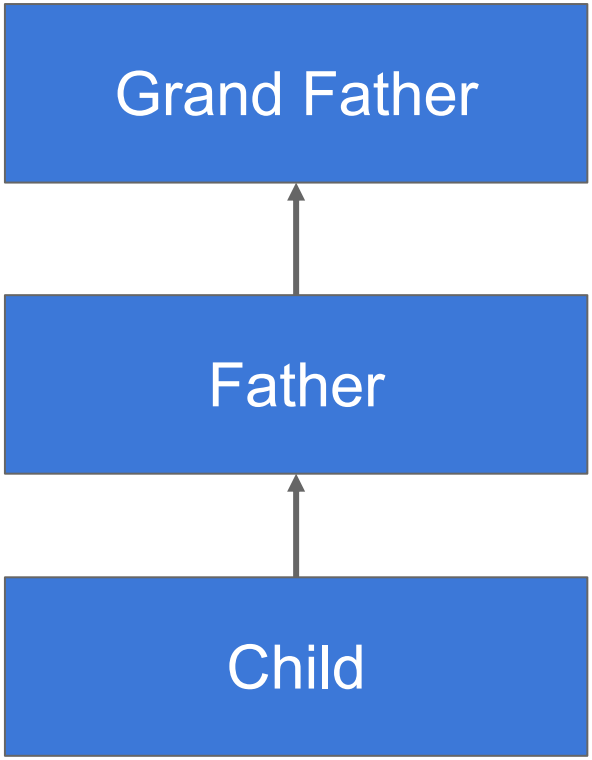
Previously,

- **Increment and Decrement Operators**
- **Scope of Declarations**
  - this keyword
- **Array:**
  - Declare and Create Array
  - Loop through Array
  - Pass Arrays to Methods
    - Pass by Value vs Pass by Reference
  - Multidimensional Arrays
  - Class Arrays for helper methods
- **ArrayList** (Collections Class)
- **Array vs ArrayList**

Agenda

- **Inheritance**:
  - Definition and Examples
  - Types of Inheritance
  - UML Diagram
  - Animal Inheritance Example
    - Without Inheritance
    - With Inheritance
    - Method Overriding
  - Constructors in Subclasses
    - Keyword Super
  - Method Overriding
    - Keyword Super
  - Access Modifier
    - Protected
  - Exercise for University Staff & Lecturer

Inheritance



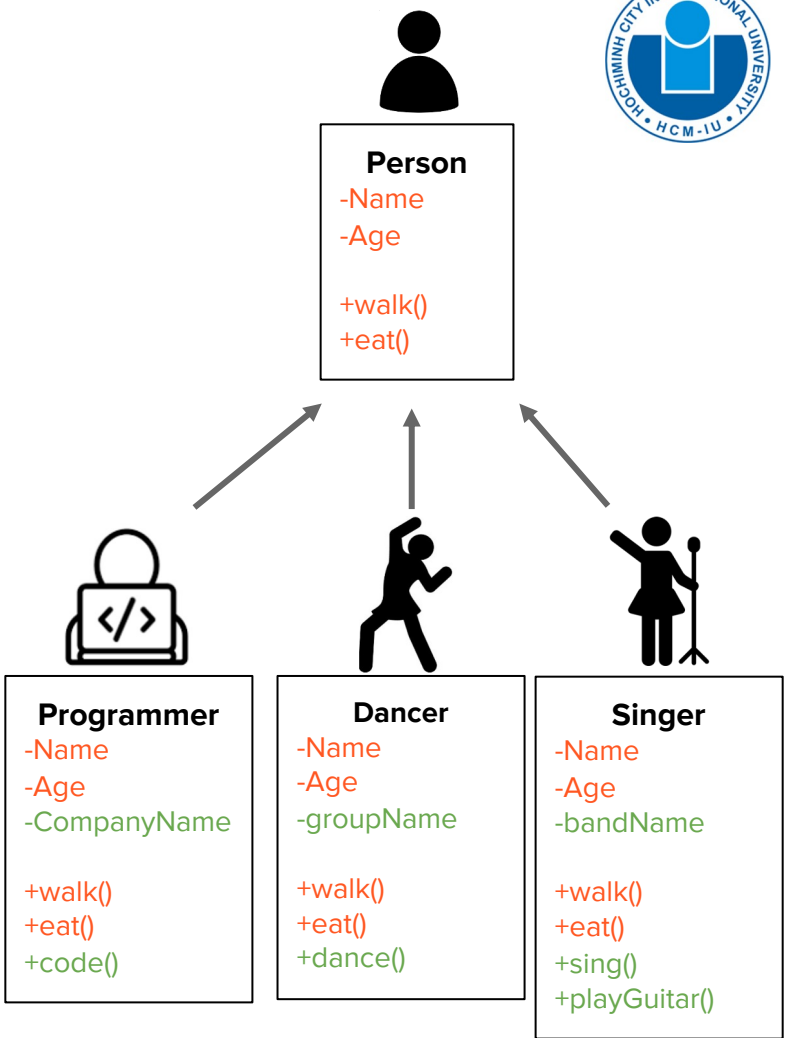
Inheritance

- Inheritance is a way to **reuse classes by expanding them into more specific types of classes.**
- Inheritance allows a **child class** (**subclass**) to **inherit the attributes and the methods** of a **parent class** (**superclass**). So **a child class can do anything that the parent class can do**!

- A **child class**
  - can have **its own attributes and methods.**
  - A child class can customize methods that it inherits from its parent class (**method overriding**)

- Inheritance represents **IS-A relationship** between parent and child objects.

- Inheritance promotes the idea of **code reuse** to **reduces code repetition** as classes can share similar common logic, structure, attributes and methods.

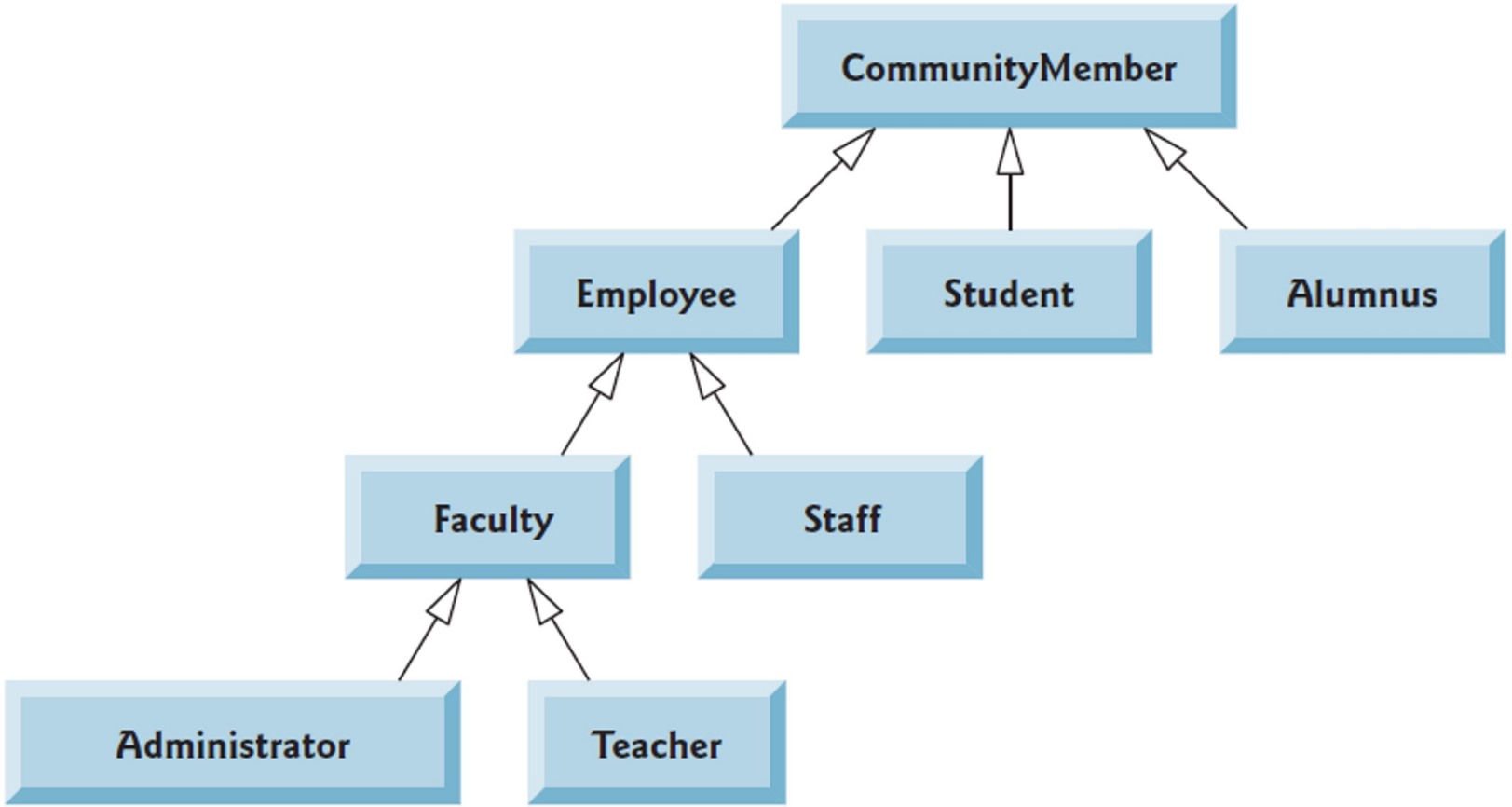


Examples of Superclass, Subclasses

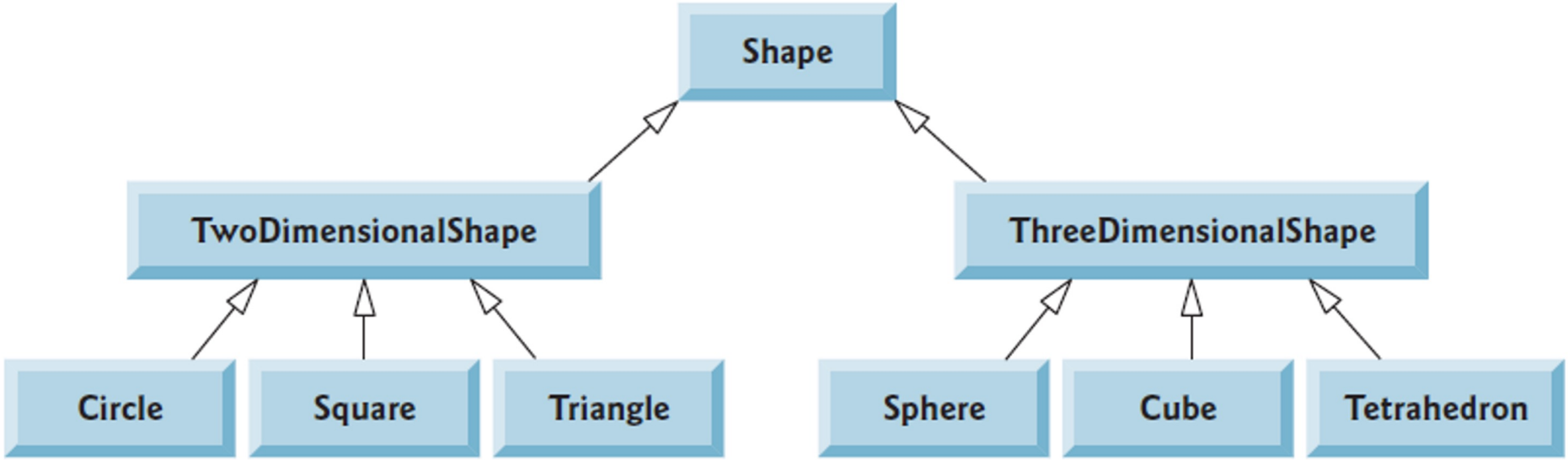
- **Superclass** tend to be “more general” and **Subclass** is more “more specific.”

Parent Class (Superclass)	Child Class (Subclasses)
Vehicles	Car, Truck, Boat, Bicycle
Shape	Circle, Triangle, Rectangle, Cube
UniversityStaff	Lecturer, TeachingAssistant
Animal	Dog, Cat, Spider, Duck

UML Class Diagram for CommunityMembers

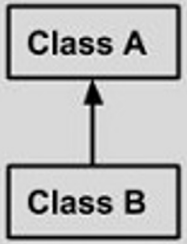


UML Class Diagram for Shape



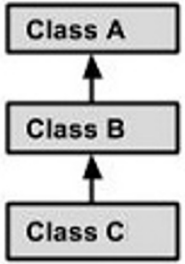
Types of Inheritance & Syntax

Single Inheritance



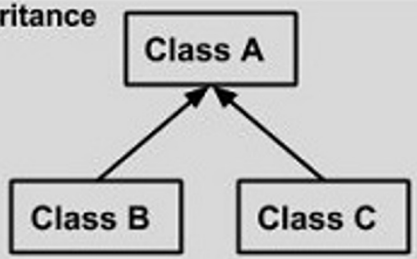

```
public class A {
   .......
}
public class B extends A {
   .........
}
```

Multi Level Inheritance



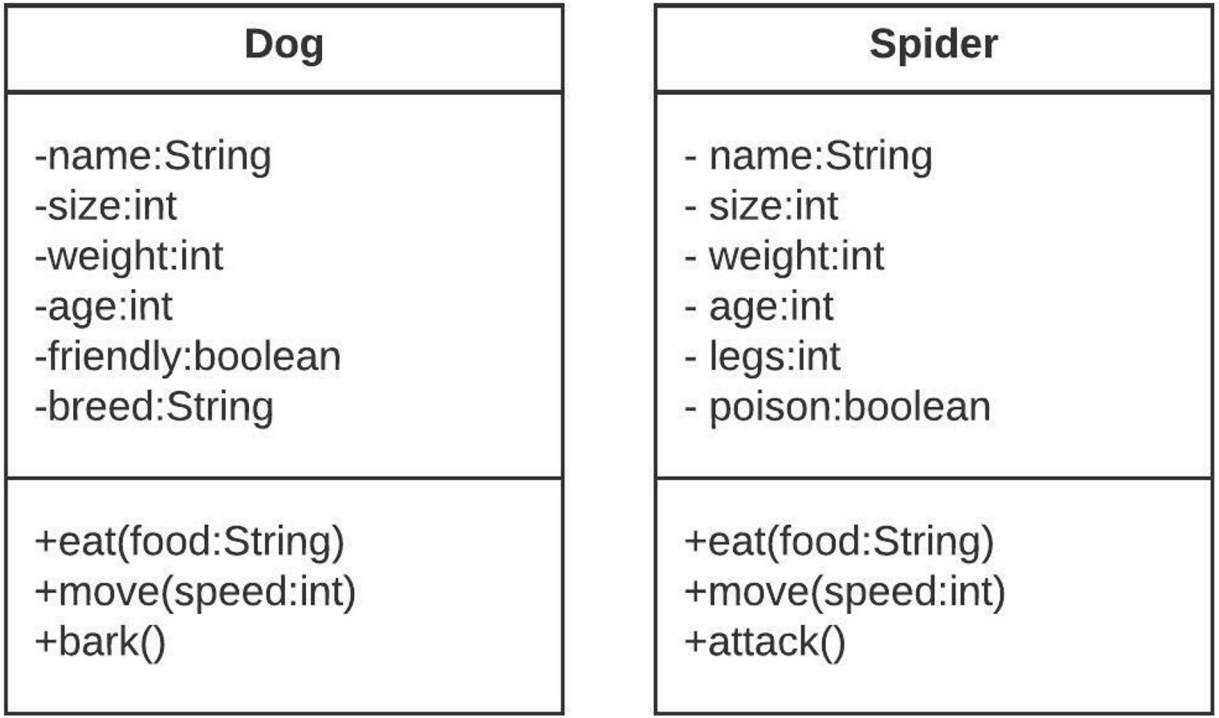

```
public class A { ..................}

public class B extends A {.....................}

public class C extends  B {..................... }
```

Hierarchical Inheritance

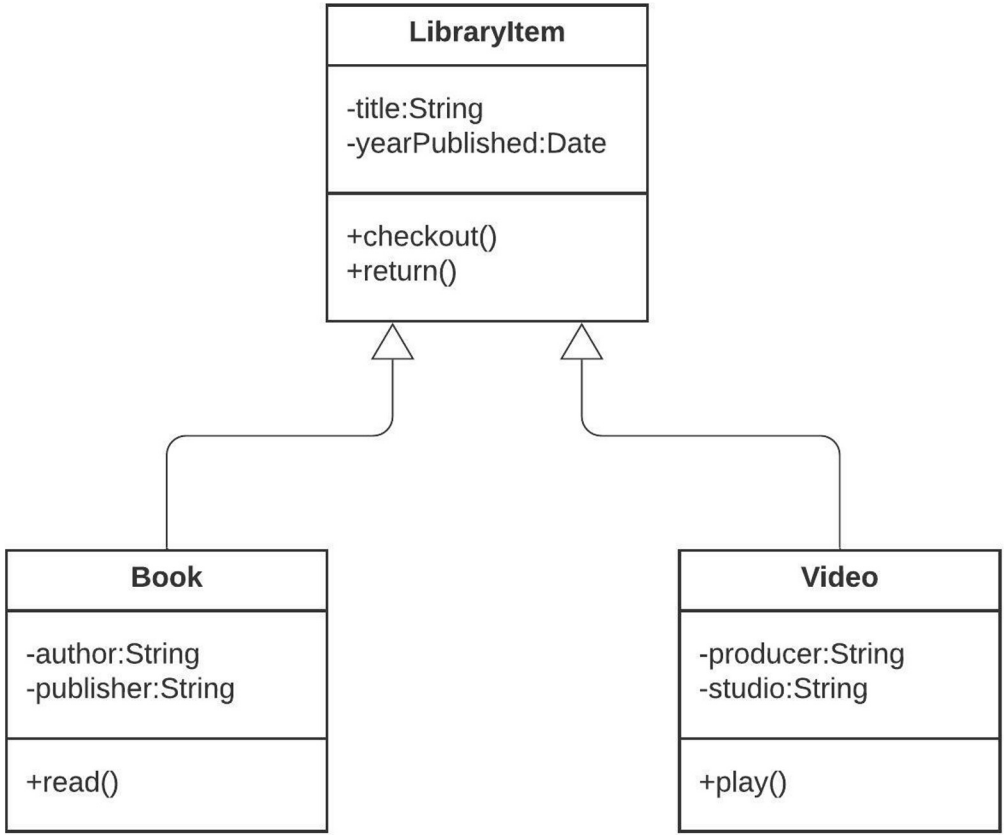



```
public class A { ..................}

public class B extends A {.....................}

public class C extends A {..................... }
```

Without Inheritance

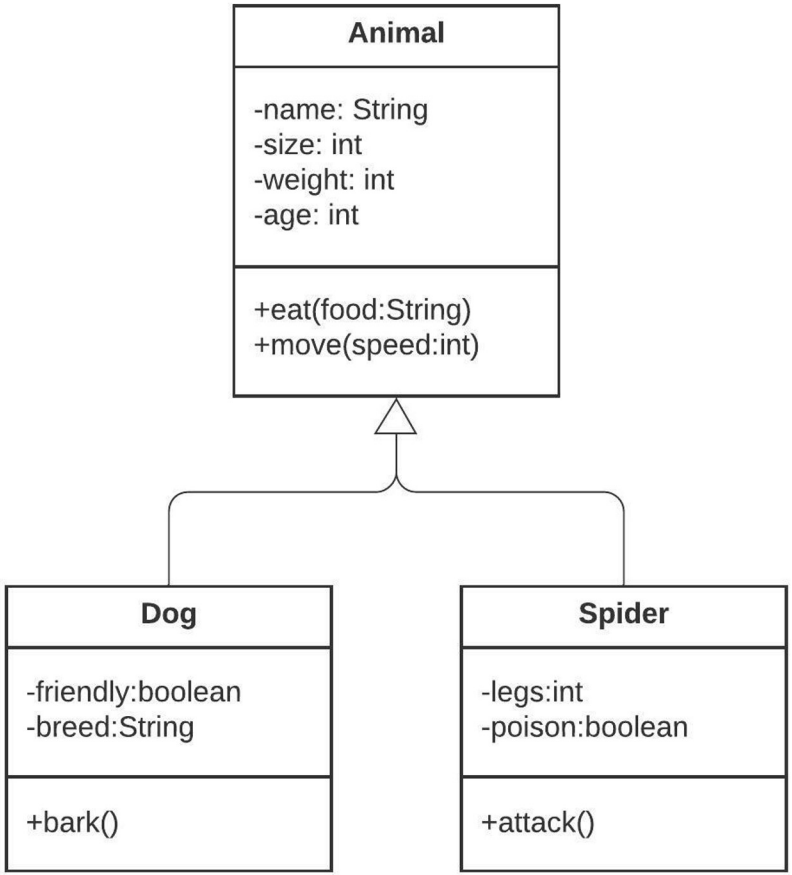


IU Digital Library Example



Animal Inheritance Example

With Inheritance



Let's live code in Java!

Animal Class

[Info] Did you notice the superclass Animal looks like the normal class with attributes and method.

```java
public class Animal {
    private String name;
    private int size;
    private int weight;
    private int age;

    public Animal(String name, int size, int weight, int age) {
        this.name = name;
        this.size = size;
        this.weight = weight;
        this.age = age;
    }

    public void eat(String food){
        System.out.printf("The %s is eating %s!\n", getName(), food);
    }

    public void move(int velocity){
        System.out.printf("The %s is moving %d km/h!\n", getName(), velocity);
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getSize() {
        return size;
    }
    public void setSize(int size) {
        this.size = size;
    }
    public int getWeight() {
        return weight;
    }
    public void setWeight(int weight) {
        this.weight = weight;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

Dog Class

[Question] Can you guess what does the keyword super do in the constructor of Dog Class?

```java
public class Dog extends Animal {
    private boolean friendly;
    private String breed;

    public Dog(String name, int size,
               int weight, int age,
               boolean friendly, String breed) {
        super(name, size, weight, age);
        this.friendly = friendly;
        this.breed = breed;
    }

    public void bark(){
        System.out.println("The dog is barking!");
    }

    public boolean isFriendly() {
        return friendly;
    }

    public void setFriendly(boolean friendly) {
        this.friendly = friendly;
    }

    public String getBreed() {
        return breed;
    }

    public void setBreed(String breed) {
        this.breed = breed;
    }
}
```

Spider Class

[Question] Can you guess what does the keyword super do in the constructor of Spider Class?

```
public class Spider extends Animal {
    private int legs;
    private boolean poison;

    public Spider(String name, int size,
                  int weight, int age,
                  int legs, boolean poison) {
        super(name, size, weight, age);
        this.legs = legs;
        this.poison = poison;
    }

    public void attack(){
        System.out.println("The spider is attacking!");
    }

    public int getLegs() {
        return legs;
    }

    public void setLegs(int legs) {
        this.legs = legs;
    }

    public boolean isPoison() {
        return poison;
    }

    public void setPoison(boolean poison) {
        this.poison = poison;
    }
}
```

Main Class for Testing

```
public class Zoo {

    public static void main(String[] args) {
        Dog myDog = new Dog("Kiki", 200, 10, 5, true, "Corgi");
        Spider mySpider = new Spider("Spider-man", 2, 3, 20, 20, false);

        myDog.eat("sauces");
        myDog.move(20);
        myDog.bark();
        System.out.println(myDog.getAge());

        mySpider.eat("sauces");
        mySpider.move(20);
        mySpider.setName("Venom");
        System.out.println(mySpider.getName());
    }
}
```

```
The Kiki is eating sauces!
The Kiki is moving 20 km/h!
The dog is barking!
5
The Spider-man is eating insects!
The Spider-man is moving 3 km/h!
Venom
```

Keyword super() in a constructor in child class

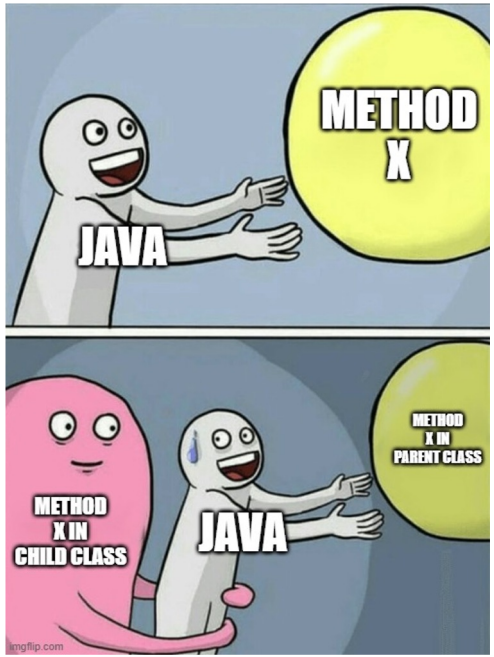
- A child class, all properties, instance variables, and methods are inherited, except for constructors, so **you need to define constructors for child class**.
- The **keyword super in a constructor in a child class** can be **used to call a constructor in the parent class.** It's a way to delegate the responsibility to the parent class.

```
public class Animal {
    private String name;
    private int size;
    private int weight;
    private int age;

    public Animal(String name, int size, int weight, int age) {
        this.name = name;
        this.size = size;
        this.weight = weight;
        this.age = age;
    }

}
```

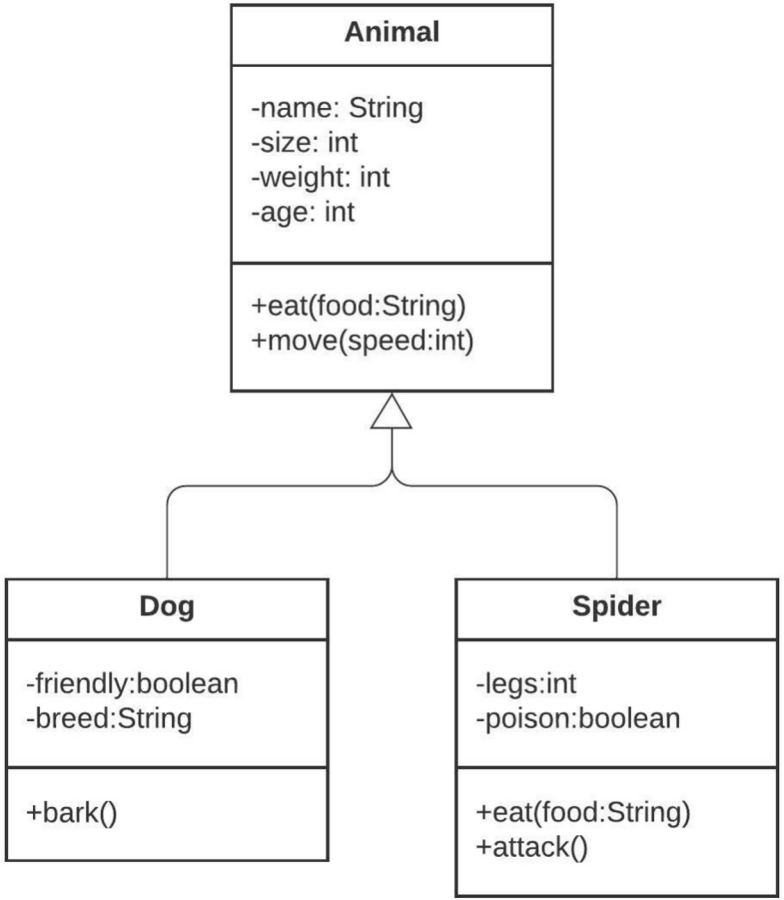
```
public class Spider extends Animal {
    private int legs;
    private boolean poison;

    public Spider(String name, int size,
                  int weight, int age,
                  int legs, boolean poison) {
        super(name, size, weight, age);
        this.legs = legs;
        this.poison = poison;
    }
```

Method Overriding

- If **subclass (child class) has the same method as declared in the superclass (parent class)**, it is known as **method overriding** in Java.
- Method overriding is used to **provide a different implementation of a method which is already provided by its superclass.**
- **Rules** for method overriding:
  - Method must have the **same name** as in the parent class.
  - Method must have **same parameter** as in the parent class.
  - Method overriding must **happens in IS-A relationship (inheritance)**.



Method Overriding Example



[**Question**] Can you find out which method of which class is overridden?

Method Overriding in Spider Class

Spider.java

```
public class Spider extends Animal {
    private int legs;
    private boolean poison;

    public Spider(String name, int size,
                  int weight, int age,
                  int legs, boolean poison) {...}

    @Override
    public void eat(String food){
        System.out.printf("Spider is capturing the %s first before eating it!", food);
    }
```

Zoo.java

```
public class Zoo {
    public static void main(String[] args) {
        Spider mySpider = new Spider("Spider-man", 2, 3, 20, 20, false);
        mySpider.eat("insects");
    }
}
```

```
Spider is capturing the insects first before eating it!
```

[Question] @Override is optional, but why does we want it when we override any class?

Keyword super() in a method in child class

- Also, the **keyword super can be used in a method of child class to call a method of a parent class** even if that method is overridden.

Animal.java

```java
public class Animal {
    private String name;
    private int size;
    private int weight;
    private int age;

    public Animal(String name, int size, int weight, int age) {...}

    public void eat(String food) {
        System.out.printf("The Animal is eating %s!\n", food);
    }
}
```

Spider.java

```java
public class Spider extends Animal {
    private int legs;
    private boolean poison;

    public Spider(String name, int size,
                  int weight, int age,
                  int legs, boolean poison) {...}

    @Override
    public void eat(String food) {
        System.out.printf("Spider is capturing the %s!\n", food);
        super.eat(food);
    }
}
```

Zoo.java

```java
public class Zoo {
    public static void main(String[] args) {
        Spider mySpider = new Spider("Spider-man", 2, 3, 20, 20, false);
        mySpider.eat("human");
    }
}
```

Output:

```
Spider is capturing the human!
The Animal is eating human!
```

Protected Members

- Remember, apart from public and private, we have “protected” to be an access modifier
- **“Protected” access offers an intermediate level of access between public and private.**
- **A superclass’s protected members can be accessed by members of that superclass, by members of its subclasses and by members of other classes in the same package.**
- **In short,** the **“protected”** access modifier means that **anything within the class can use it, as well as anything in any subclass.**

Modifier	Class	Package	Subclass	Global
Public	✓	✓	✓	✓
Protected	✓	✓	✓	✗
Default	✓	✓	✗	✗
Private	✓	✗	✗	✗

The ‘protected’ Access Modifier

In the past, we discussed the private and public access modifiers. To review, remember that **public meant anyone could get access to the member** (variable, method, property, etc.), while **private means that you only have access to it from inside of the class that it belongs to.**

With inheritance we add **another option**: **protected**. If a member of the class uses the protected accessibility level, then **anything inside of the class can use it, as well as any subclass**. It’s a little **broader than private, but still more restrictive than public.**

Issue Without Protected Access

Animal.java

```java
public class Animal {
    private String name;

    public Animal(String name) {
        this.name = name;
    }
}
```

[Info] This will give **an error** as **the instance variable “name” is private in the parent class** because remember that private instance variables can be access within the parent class but not in the child class.

Dog.java

```java
public class Dog extends Animal {
    public Dog(String name, boolean friendly) {
        super(name);
        this.friendly = friendly;
    }
    // This will give an error that name has a private access in Animal class
    public void bark(){
        System.out.printf("The Dog %s is barking!", name);
    }
}
```

Protected Access Solution

Animal.java

```java
public class Animal {
    protected String name;

    public Animal(String name) {
        this.name = name;
    }
}
```

Dog.java

```java
public class Dog extends Animal {
    private boolean friendly;

    public Dog(String name, boolean friendly) {
        super(name);
        this.friendly = friendly;
    }
    // This will give an error that name has a private access in Animal class
    public void bark(){
        System.out.printf("The Dog %s is barking!", name);
    }
}
```

Zoo.java

```java
public class Zoo {
    public static void main(String[] args) {
        Dog myDog = new Dog("Kiki", true);
        myDog.bark();
    }
}
```

With protected access modifier, the **subclass** Dog **can access** the **instance variable "name"** that it inherited from Animal Class.

Output:

```
The Dog Kiki is barking!
```

Getter Methods for Private Attributes

```java
public class Animal {
    private String name;

    public Animal(String name) {
        this.name = name;
    }

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

```java
public class Dog extends Animal {
    private boolean friendly;

    public Dog(String name, boolean friendly) {
        super(name);
        this.friendly = friendly;
    }
    // This will give an error that name has a private access in Animal class
    public void bark(){
        System.out.printf("The Dog %s is barking!", getName());
    }
}
```

```java
public class Zoo {
    public static void main(String[] args) {
        Dog myDog = new Dog("Kiki", true);
        myDog.bark();
    }
}
```

Output:

```
The Dog Kiki is barking!
```

- **This is a better way** to **keep your instance variables to be private** and **still have ways to access or modify their values using getter and setter methods.**

Which access modifier to use?

- For beginners, keep it simple:
  - Class: **public**
  - Attributes/Instance Variables: **private**
  - Methods (including getters and setters): **public**
  - Class Variables (static keyword): **public**
  - Class Methods (static keyword): **public**
  - Constants (final keyword):
    - Class Constant (with static keyword): **public**
    - Normal Constant (without static keyword): **private**

- **The short answer is, you should make everything as restricted as possible** (private over protected, and protected over public) **while still allowing things to get done.**
- When it comes to deciding between public and private, it all comes down to how widely used you expect the class to be. If you think a class is reusable across many projects, then you might as well make it public from the beginning.

Class Object

- Secretly, any class/objects are inherited from the **default Object** class that is the **base class of everything**.
- If you go up the inheritance hierarchy, everything always gets back to the object class eventually.
- You can say that **Class Object is the mother of all objects in Java**.

You can imagine every classes or objects would be an child to the default class Object of Java.

```
public class MyClass extends Object {


}
```

Default Methods of the Class Object

Method	Description
`equals`	This method compares two objects for equality and returns `true` if they're equal and `false` otherwise. The method takes any `Object` as an argument. When objects of a particular class must be compared for equality, the class should override method `equals` to compare the *contents* of the two objects. For the requirements of implementing this method (which include also overriding method `hashCode`), refer to the method's documentation at `docs.oracle.com/javase/7/docs/api/java/lang/Object.html#equals(java.lang.Object)`. The default `equals` implementation uses operator `==` to determine whether two references *refer to the same object* in memory. Section 14.3.3 demonstrates class `String`'s `equals` method and differentiates between comparing `String` objects with `==` and with `equals`.
`hashCode`	Hashcodes are `int` values used for high-speed storage and retrieval of information stored in a data structure that's known as a hashtable (see Section 16.11). This method is also called as part of `Object`'s default `toString` method implementation.

- Since **all objects are inherited from the class Object** then **all objects can use those methods.**
- All classes in Java inherit directly or indirectly from class Object (package java.lang), so its 11 methods (some are overloaded) are inherited by all other classes.

Method	Description
`toString`	This method (introduced in Section 9.4.1) returns a `String` representation of an object. The default implementation of this method returns the package name and class name of the object's class typically followed by a hexadecimal representation of the value returned by the object's `hashCode` method.
`wait`, `notify`, `notifyAll`	Methods `notify`, `notifyAll` and the three overloaded versions of `wait` are related to multithreading, which is discussed in Chapter 23.
`getClass`	Every object in Java knows its own type at execution time. Method `getClass` (used in Sections 10.5 and 12.5) returns an object of class `Class` (package `java.lang`) that contains information about the object's type, such as its class name (returned by `Class` method `getName`).
`finalize`	This `protected` method is called by the garbage collector to perform termination housekeeping on an object just before the garbage collector reclaims the object's memory. Recall from Section 8.10 that it's unclear whether, or when, `finalize` will be called. For this reason, most programmers should avoid method `finalize`.
`clone`	This `protected` method, which takes no arguments and returns an `Object` reference, makes a copy of the object on which it's called. The default implementation performs a so-called shallow copy—instance-variable values in one object are copied into another object of the same type. For reference types, only the references are copied. A typical overridden `clone` method's implementation would perform a deep copy that creates a new object for each reference-type instance variable. *Implementing `clone` correctly is difficult. For this reason, its use is discouraged.* Some industry experts suggest that object serialization should be used instead. We discuss object serialization in Chapter 15. Recall from Chapter 7 that arrays are objects. As a result, like all other objects, arrays inherit the members of class `Object`. Every array has an overridden `clone` method that copies the array. However, if the array stores references to objects, the objects are not copied—a shallow copy is performed.

The default implementation of toString() method of Object Class

```
public class Zoo {
    public static void main(String[] args) {
        Dog myDog = new Dog("Kiki", false);
        System.out.println(myDog);
    }
}
```

Output:

```
Dog@1b28cdfa
```

- When you print out any object, it will automatically call toString() method of the class Object.
- By default, toString() method will print out the Class Name with the location of the object in the memory.
- This is not very useful to us! So let's override it!

Override toString() method

```java
public class Dog extends Animal {
    private boolean friendly;

    public Dog(String name, boolean friendly) {
        super(name);
        this.friendly = friendly;
    }

    // Overriding toString() method of Object Class
    @Override
    public String toString() {
        return String.format("This Dog %s is %s friendly", getName(),
                friendly==false?"not":"");
    }
}
```

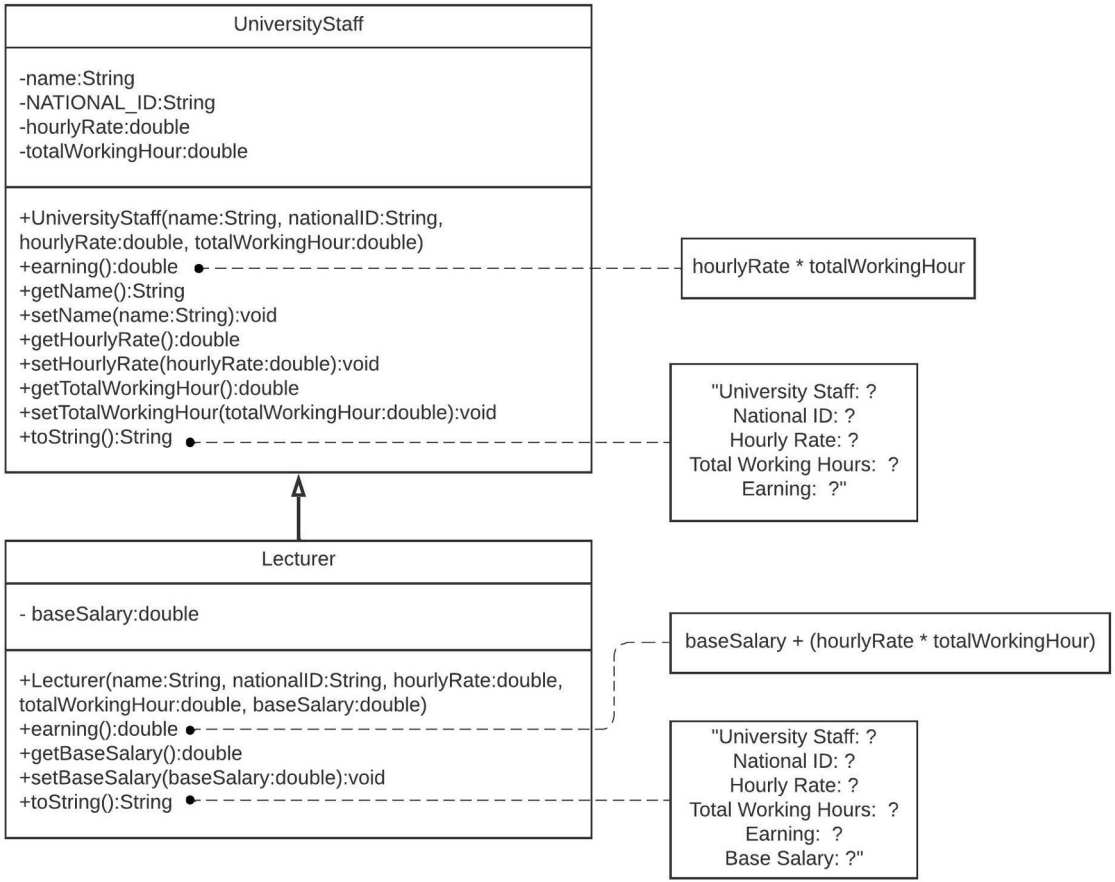
```java
public class Zoo {
    public static void main(String[] args) {
        Dog myDog = new Dog("Kiki", false);
        System.out.println(myDog);
    }
}
```

Now, our new overridden toString() method is customized to print out the string that is interesting and useful to our defined classes by call print() method.

Output:

```
This Dog Kiki is not friendly
```

Exercise for University Staff Inheritance



Sample Output

```
Staff information obtained by get methods:
First name is Khanh
Social security number is C12345
Hourly Rate is 10.00
Total Working Hour is 80.00

Updated staff information obtained by toString::
University Staff: Khanh
National ID: C12345
Hourly Rate: 15.00
Total Working Hours: 80.00
Earning: 1200.00

Lecturer information obtained by get methods:
First name is Tom
Social security number is C43252
Hourly Rate is 25.00
Total Working Hour is 160.00
Earning 4150.00

Updated lecturer information obtained by toString:
University Staff: Tom
National ID: C43252
Hourly Rate: 25.00
Total Working Hours: 160.00
Earning: 4015.00
Base Salary: 15.00
```

Exercise for University Staff Inheritance

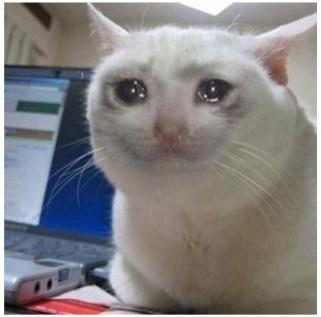
```java
public class UniversityStaffTest {
    public static void main(String[] args) {
        UniversityStaff staff = new UniversityStaff( name: "Khanh", NATIONAL_ID: "C12345",
                hourlyRate: 10,  totalWorkingHour: 80);

        System.out.println("Staff information obtained by get methods:");
        System.out.printf("%s %s \n", "First name is", staff.getName());
        System.out.printf("%s %s \n", "Social security number is", staff.getNationalID());
        System.out.printf("%s %.2f\n", "Hourly Rate is", staff.getHourlyRate());
        System.out.printf("%s %.2f\n", "Total Working Hour is", staff.getTotalWorkingHour());

        staff.setHourlyRate(15);
        System.out.printf("\n%s: \n%s",
                "Updated staff information obtained by toString:", staff);

        Lecturer tom = new Lecturer( name: "Tom", NATIONAL_ID: "C43252",
                hourlyRate: 25,  totalWorkingHour: 160,  baseSalary: 150);

        System.out.println("\n\nLecturer information obtained by get methods:");
        System.out.printf("%s %s \n", "First name is", tom.getName());
        System.out.printf("%s %s \n", "Social security number is", tom.getNationalID());
        System.out.printf("%s %.2f\n", "Hourly Rate is", tom.getHourlyRate());
        System.out.printf("%s %.2f\n", "Total Working Hour is", tom.getTotalWorkingHour());
        System.out.printf("%s %.2f\n", "Earning", tom.earnings());

        tom.setBaseSalary(15);
        System.out.printf("\n%s: \n%s", "Updated lecturer information obtained by toString", tom);
    }
}
```

Sample Output

```
Staff information obtained by get methods:
First name is Khanh
Social security number is C12345
Hourly Rate is 10.00
Total Working Hour is 80.00

Updated staff information obtained by toString::
University Staff: Khanh
National ID: C12345
Hourly Rate: 15.00
Total Working Hours: 80.00
Earning: 1200.00

Lecturer information obtained by get methods:
First name is Tom
Social security number is C43252
Hourly Rate is 25.00
Total Working Hour is 160.00
Earning 4150.00

Updated lecturer information obtained by toString:
University Staff: Tom
National ID: C43252
Hourly Rate: 25.00
Total Working Hours: 160.00
Earning: 4015.00
Base Salary: 15.00
```

Let's live code in Java!

Let's write the Class UniversityStaff & the Class Lecturer together!



Recap

- **Inheritance**:
  - Definition and Examples
  - Types of Inheritance
  - UML Diagram
  - Animal Inheritance Example
    - Without Inheritance
    - With Inheritance
    - Method Overriding
  - Constructors in Subclasses
    - Keyword Super
  - Method Overriding
    - Keyword Super
  - Access Modifier
    - Protected
  - Exercise for University Staff & Lecturer

Thank you for your listening!

“Motivation is what gets you started. Habit is what keeps you going!”

Jim Ryun