Inheritance

Shirley B. Chu

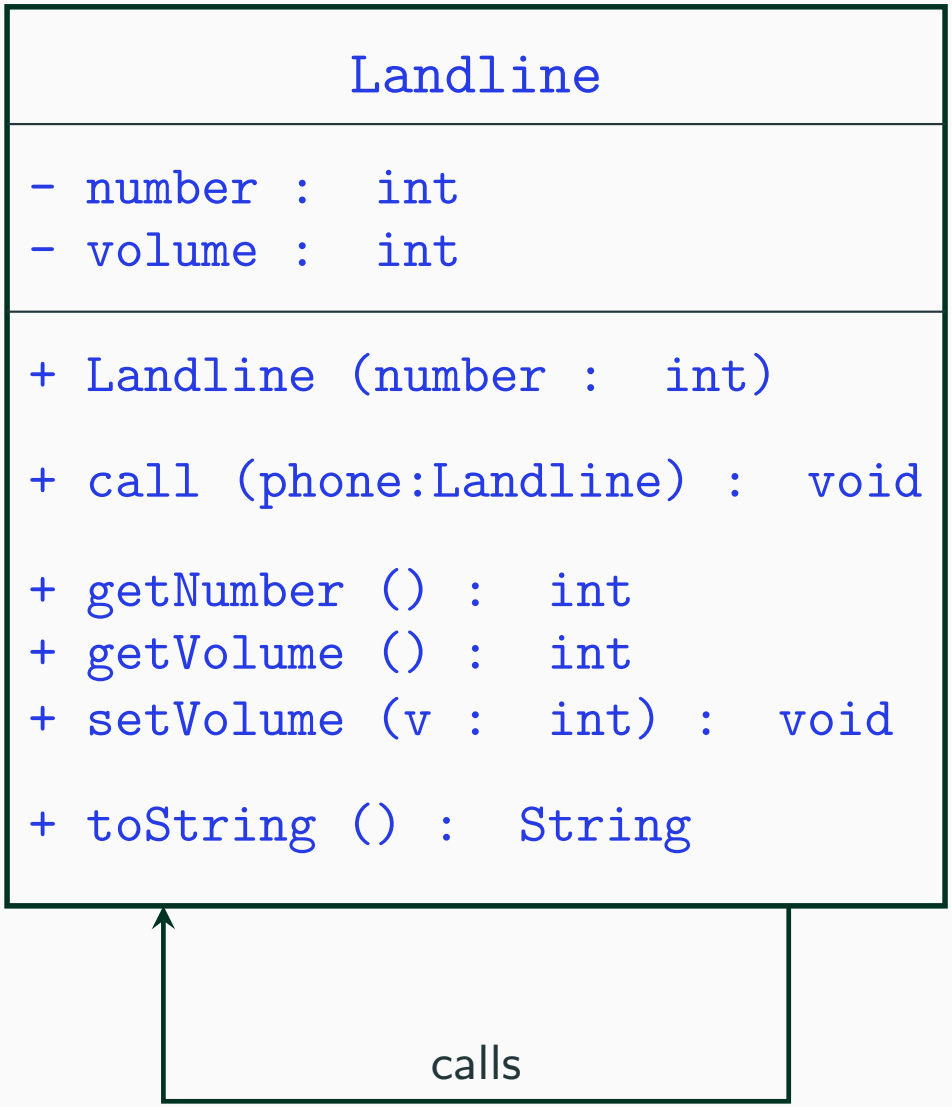
June 22, 2020

De La Salle University
College of Computer Studies

Stage 1

Tiam-Lee Telecoms is a manufacturer of landline phones. A phone has unique seven-digit number, and a ringing volume (integer value from 1 to 3). A phone can make a call to another phone.

Landline


Landline
- number : int
- volume : int
+ Landline (number : int)
+ call (phone:Landline) : void
+ getNumber () : int
+ getVolume () : int
+ setVolume (v : int) : void
+ toString () : String
calls

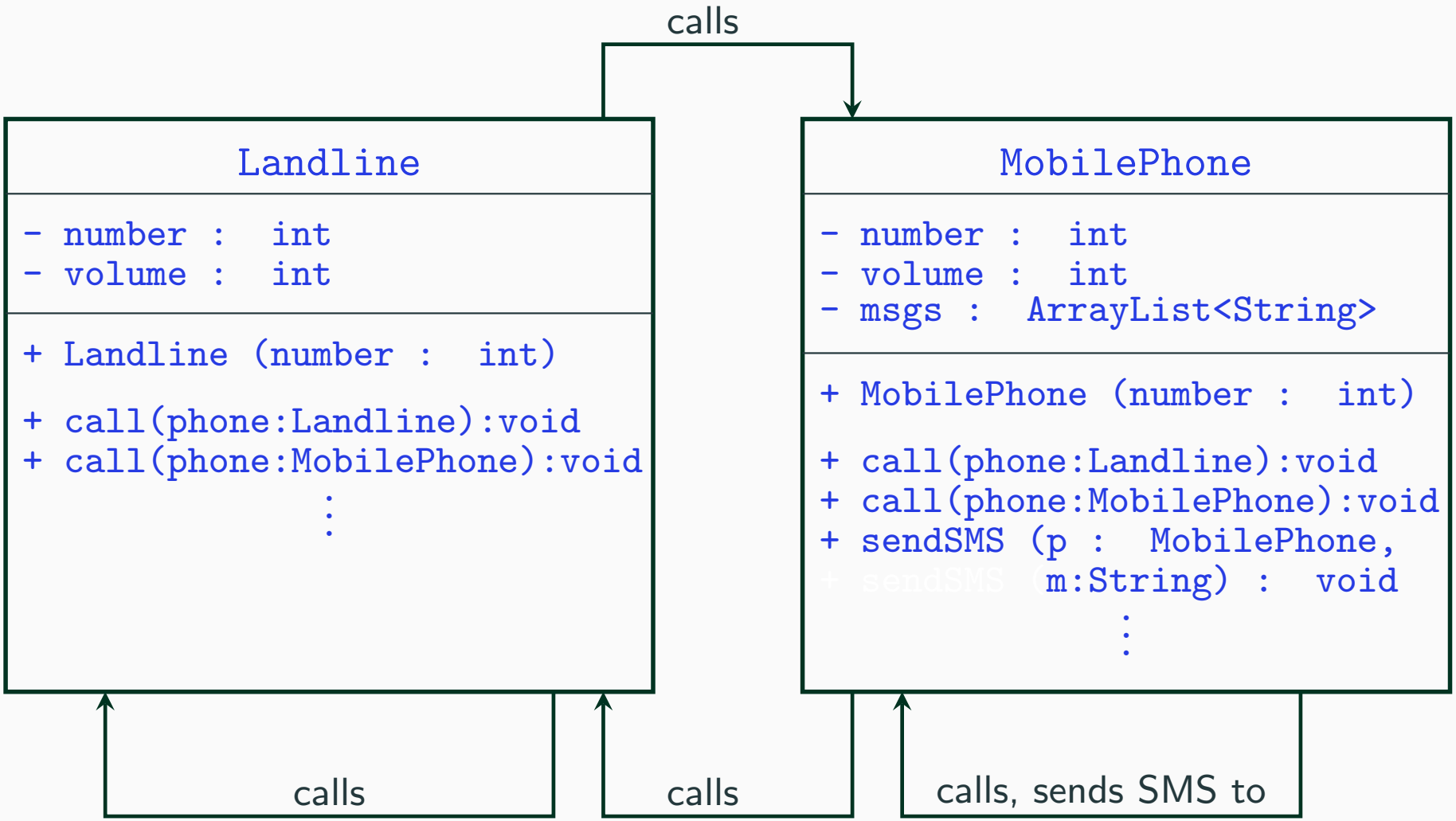
Stage 2

Some time later, Tiam-Lee Telecoms introduced mobile phones, in addition to their landline phones. A mobile phone has an 11-digit phone number, and a ringing volume.

Mobile phones can make calls to another mobile phone, aside from making calls to landline phones. Landline phones can also call mobile phones.

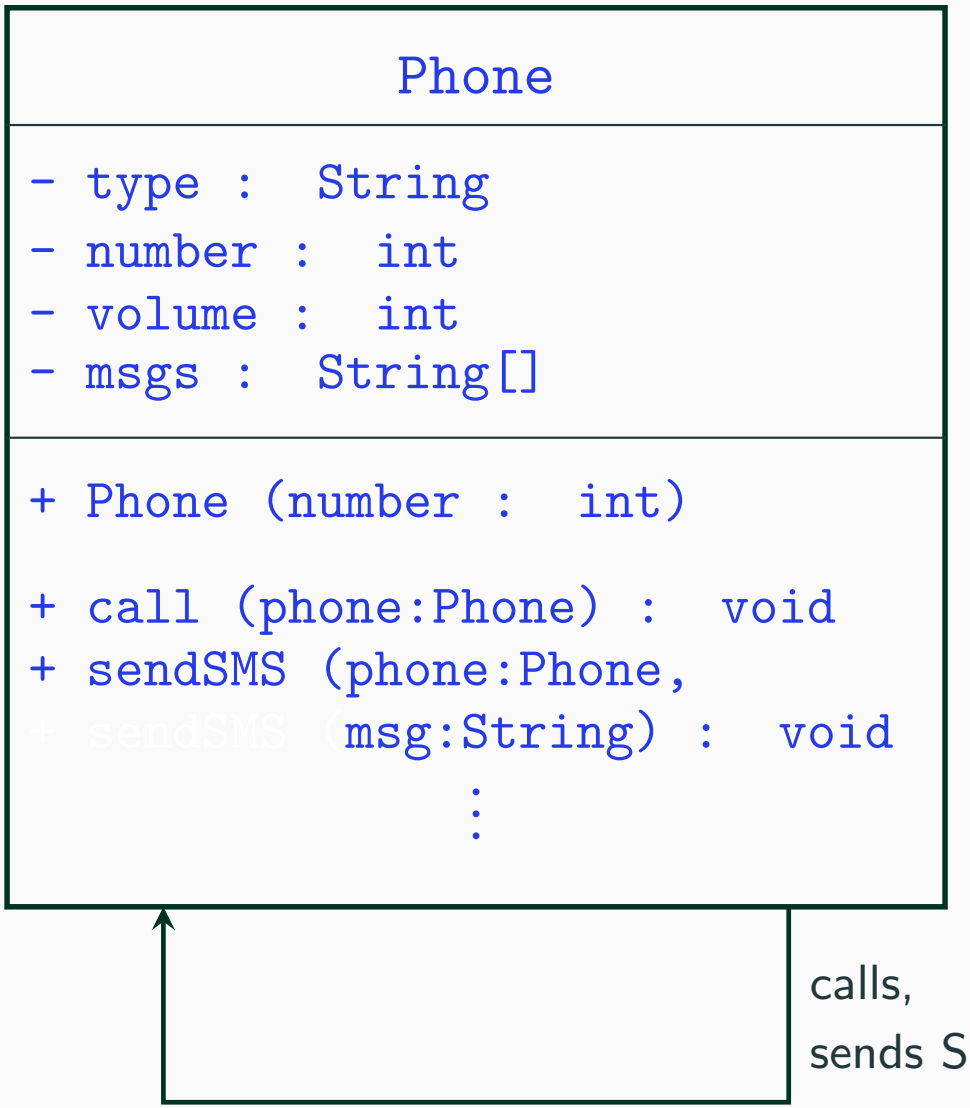
Mobile phones can send text messages (a `String`) to another mobile phone. It can store messages that it receives.

Landline and MobilePhone

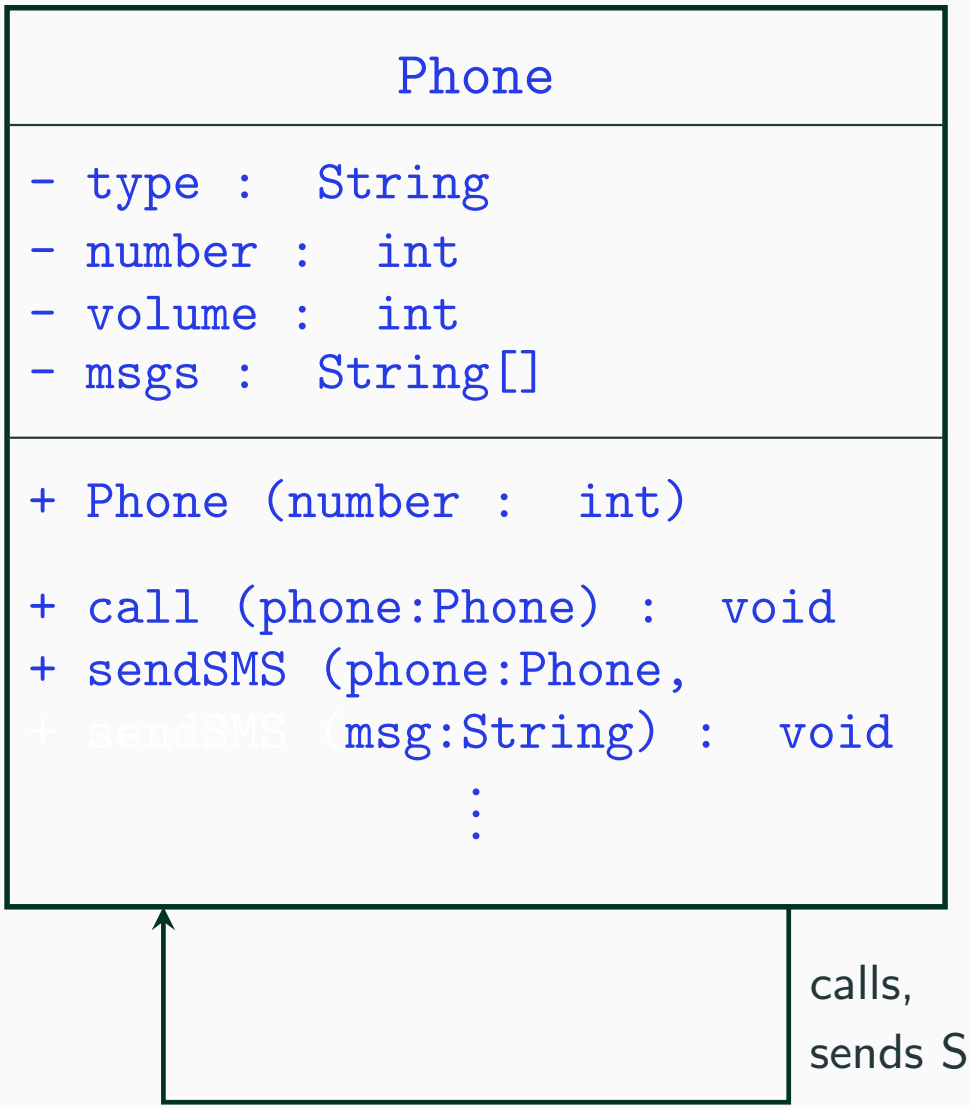


Landline and MobilePhone

Phone


Phone
- type : String
- number : int
- volume : int
- msgs : String[]
+ Phone (number : int)
+ call (phone:Phone) : void
+ sendSMS (phone:Phone,
+ sendSMS (msg:String) : void
:
calls,
sends SMS to (if type = “mobile”)

Phone


Phone
- type : String
- number : int
- volume : int
- msgs : String[]
+ Phone (number : int)
+ call (phone:Phone) : void
+ sendSMS (phone:Phone,
+ sendSMS (msg:String) : void
:
calls,
sends SMS to (if type = “mobile”)

Inheritance

Inheritance

the process by which a class acquires attributes and behaviors of another class

Inheritance

- Classes can be derived from another class.

Inheritance

- Classes can be derived from another class.
- The derived class is the *subclass* (a.k.a child class).

Inheritance

- Classes can be derived from another class.
- The derived class is the *subclass* (a.k.a child class).
- The class from which it's derived is the *superclass* (a.k.a base class, parent class).

Inheritance

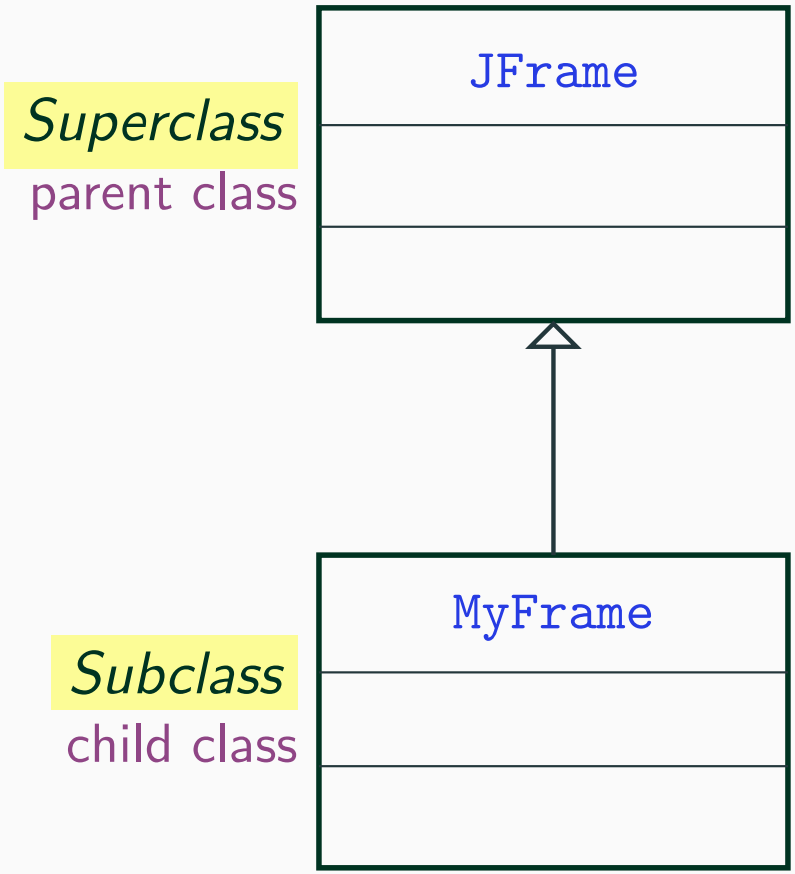
- Classes can be derived from another class.
- The derived class is the *subclass* (a.k.a child class).
- The class from which it's derived is the *superclass* (a.k.a base class, parent class).
- All reference types in Java are also `Object`s, i.e. their superclass is `Object`.

`extends`

In Java, the keyword `extends` is used to denote that a class inherits from another class.

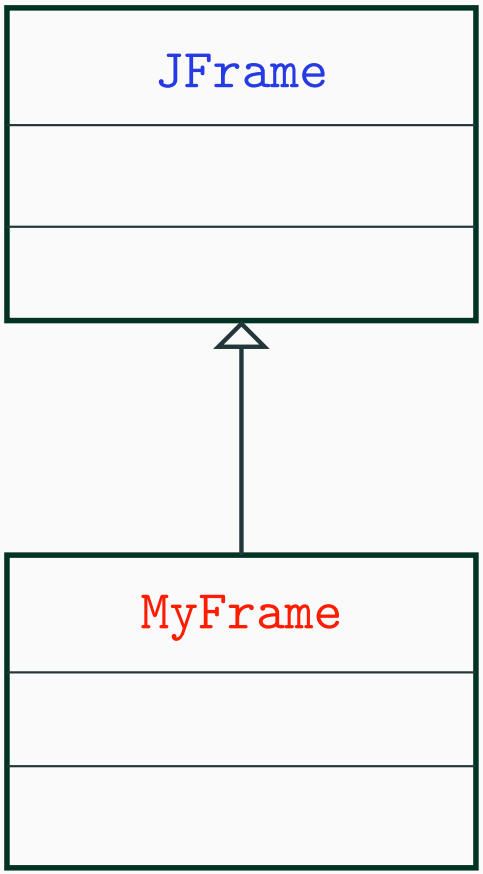
extends

In Java, the keyword `extends` is used to denote that a class inherits from another class.



extends

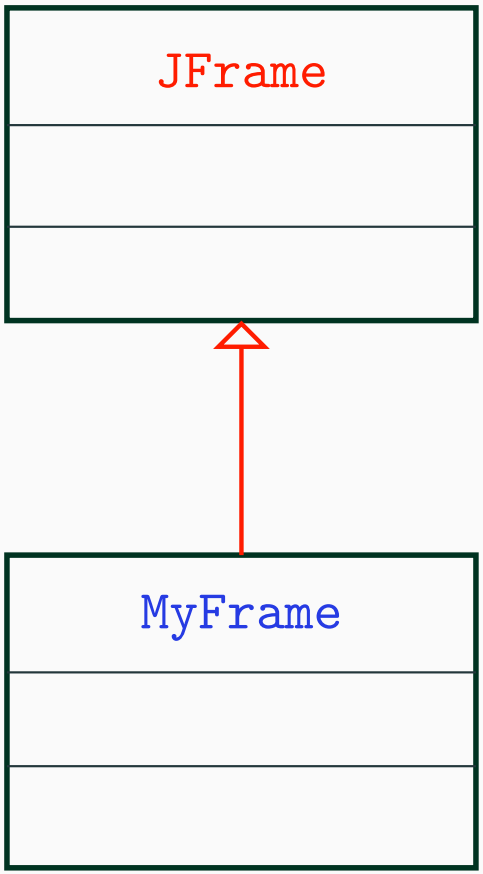
In Java, the keyword `extends` is used to denote that a class inherits from another class.




```
public class MyFrame
{
        ⋮
}
```

extends

In Java, the keyword `extends` is used to denote that a class inherits from another class.



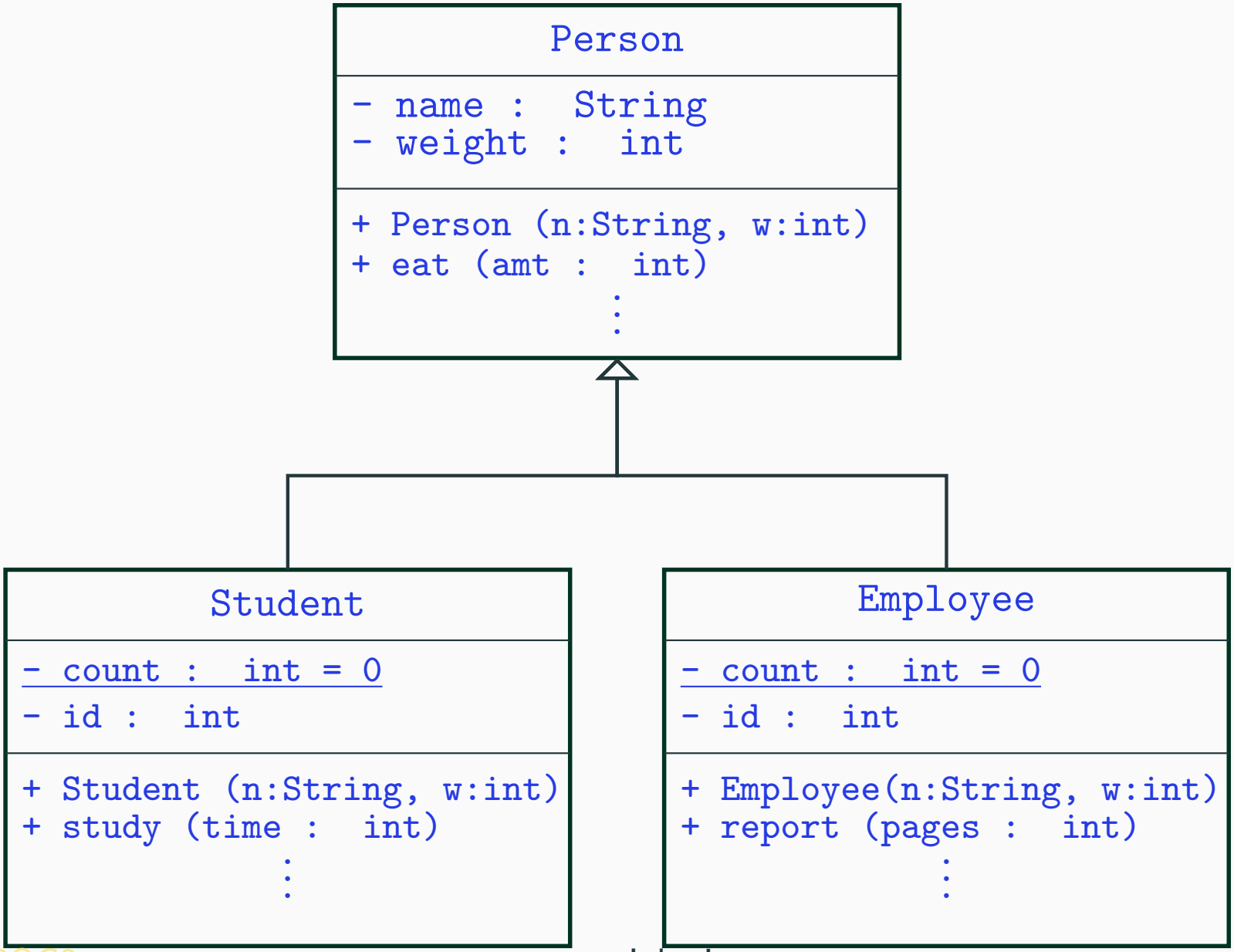

```
public class MyFrame extends JFrame
{
        ⋮
}
```

`extends`

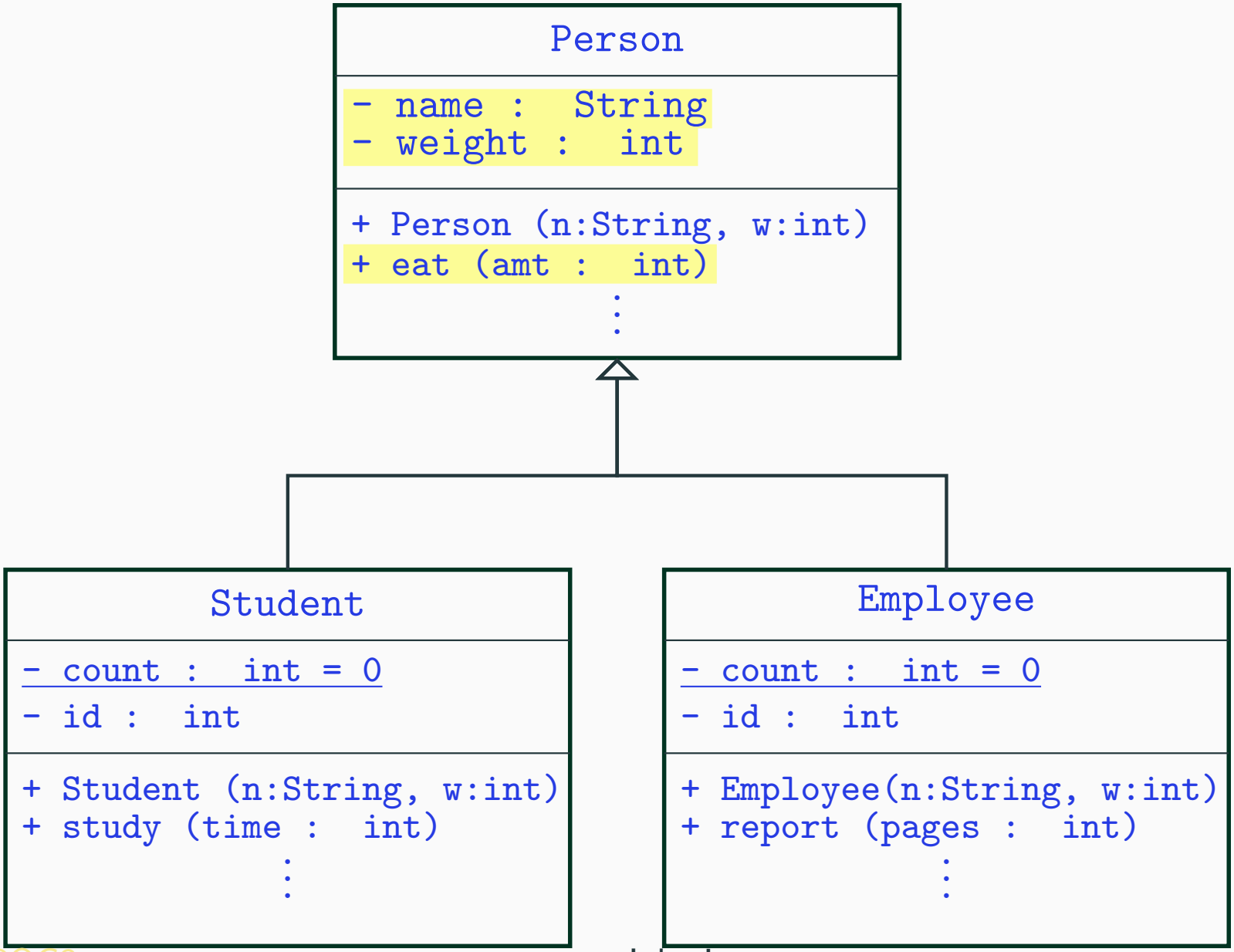
By inheriting from a class, **all attributes and methods** of the superclass will automatically be **part of the subclass** as well.

In Java, classes may only **extend one** superclass.

Example


Person
- name : String
- weight : int
+ Person (n:String, w:int)
+ eat (amt : int)
:
Student
- count : int = 0
- id : int
+ Student (n:String, w:int)
+ study (time : int)
:
Employee
- count : int = 0
- id : int
+ Employee(n:String, w:int)
+ report (pages : int)
:

Example


Person
- name : String
- weight : int
+ Person (n:String, w:int)
+ eat (amt : int)
:
Student
- count : int = 0
- id : int
+ Student (n:String, w:int)
+ study (time : int)
:
Employee
- count : int = 0
- id : int
+ Employee(n:String, w:int)
+ report (pages : int)
:

Inheritance

- All attributes and methods of the parent class (`Person`) are inherited by the child class (`Student`, and `Employee`)

Inheritance

- All attributes and methods of the parent class (`Person`) are inherited by the child class (`Student`, and `Employee`)
- *Constructors are not inherited!*

Inheritance

- All attributes and methods of the parent class (`Person`) are inherited by the child class (`Student`, and `Employee`)
- *Constructors are not inherited!*
- Attributes and methods declared as `private` in the parent class cannot be accessed directly by the child class.

Inheritance

- All attributes and methods of the parent class (`Person`) are inherited by the child class (`Student`, and `Employee`)
- *Constructors are not inherited!*
- Attributes and methods declared as `private` in the parent class cannot be accessed directly by the child class.
- Note that `Student` objects, and `Employee` objects are `Person` objects.

Inheritance

- All attributes and methods of the parent class (`Person`) are inherited by the child class (`Student`, and `Employee`)
- *Constructors are not inherited!*
- Attributes and methods declared as `private` in the parent class cannot be accessed directly by the child class.
- Note that `Student` objects, and `Employee` objects are `Person` objects.
- While, `Person` objects are not necessarily `Student`, nor `Employee` objects.

The `insanceof` operator

`instanceof`

may be used to check if an object is of a certain type.

```
Person ann = new Person ("Ann", 40);
Person ben = new Student ("Ben", 60);
Student ten = new Employee ("Ten", 10);

if (ann instanceof Person)
    System.out.println (ann.getName () + " is a person.");
if (ann instanceof Student)
    System.out.println (ann.getName () + " is a student.");
if (ann instanceof Employee)
    System.out.println (ann.getName () + " is an employee.");
```

Why?

- **code reuse** Specialized classes may be defined by adding new details.

Why?

- **code reuse** Specialized classes may be defined by adding new details.
- **reduce code redundancy**

Why?

- **code reuse** Specialized classes may be defined by adding new details.
- **reduce code redundancy**
- **reliability** Base classes have already been tested and debugged.

Why?

- **code reuse** Specialized classes may be defined by adding new details.
- **reduce code redundancy**
- **reliability** Base classes have already been tested and debugged.
- **allows polymorphism**

Polymorphism

Polymorphism

- “many forms”

Polymorphism

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

Polymorphism

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

```
Person[] people = new Person[3];
```

Polymorphism

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

```
Person[] people = new Person[3];
people[0] = new Person ("Em", 50);
```

Polymorphism

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

```
Person[] people = new Person[3];
people[0] = new Person ("Em", 50);
people[1] = new Student ("Eff", 70);
```

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

```
Person[] people = new Person[3];
people[0] = new Person ("Em", 50);
people[1] = new Student ("Eff", 70);
people[2] = new Employee ("Gi", 40);
```

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

```
Person[] people = new Person[3];
people[0] = new Person ("Em", 50);
people[1] = new Student ("Eff", 70);
people[2] = new Employee ("Gi", 40);

for (int j = 0; j < people.length; j++)
    people[j].eat ();
```

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

```
Person[] people = new Person[3];
people[0] = new Person ("Em", 50);
people[1] = new Student ("Eff", 70);
people[2] = new Employee ("Gi", 40);

for (int j = 0; j < people.length; j++)
    people[j].eat ();
```

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

```
Person[] people = new Person[3];
people[0] = new Person ("Em", 50);
people[1] = new Student ("Eff", 70);
people[2] = new Employee ("Gi", 40);

for (int j = 0; j < people.length; j++)
    people[j].eat ();
```

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

```
Person[] people = new Person[3];
people[0] = new Person ("Em", 50);
people[1] = new Student ("Eff", 70);
people[2] = new Employee ("Gi", 40);

for (int j = 0; j < people.length; j++)
    people[j].eat ();

((Student) people[1]).study (60);
```

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

```
Person[] people = new Person[3];
people[0] = new Person ("Em", 50);
people[1] = new Student ("Eff", 70);
people[2] = new Employee ("Gi", 40);

for (int j = 0; j < people.length; j++)
    people[j].eat ();

((Student) people[1]).study (60);
```

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

```
Person[] people = new Person[3];
people[0] = new Person ("Em", 50);
people[1] = new Student ("Eff", 70);
people[2] = new Employee ("Gi", 40);

for (int j = 0; j < people.length; j++)
    people[j].eat ();

((Student) people[1]).study (60);
```

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

```
Person[] people = new Person[3];
people[0] = new Person ("Em", 50);
people[1] = new Student ("Eff", 70);
people[2] = new Employee ("Gi", 40);

for (int j = 0; j < people.length; j++)
    people[j].eat ();

((Student) people[1]).study (60);
```

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

```
Person[] people = new Person[3];
people[0] = new Person ("Em", 50);
people[1] = new Student ("Eff", 70);
people[2] = new Employee ("Gi", 40);

for (int j = 0; j < people.length; j++)
    people[j].eat ();

((Student) people[1]).study (60);

((Employee) people[2]).report (5);
```

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

```
Person[] people = new Person[3];
people[0] = new Person ("Em", 50);
people[1] = new Student ("Eff", 70);
people[2] = new Employee ("Gi", 40);

for (int j = 0; j < people.length; j++)
    people[j].eat ();

((Student) people[1]).study (60);

((Employee) people[2]).report (5);
```

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

```
Person[] people = new Person[3];
people[0] = new Person ("Em", 50);
people[1] = new Student ("Eff", 70);
people[2] = new Employee ("Gi", 40);

for (int j = 0; j < people.length; j++)
    people[j].eat ();

((Student) people[1]).study (60);

((Employee) people[2]).report (5);
```

Polymorphism

- "many forms"
- allows a single action to be performed in different ways

```
Person[] people = new Person[3];
people[0] = new Person ("Em", 50);
people[1] = new Student ("Eff", 70);
people[2] = new Employee ("Gi", 40);

for (int j = 0; j < people.length; j++)
    people[j].eat ();

((Student) people[1]).study (60);

((Employee) people[2]).report (5);
```

☺ **Thank you!** ☺