

Inheritance

Extending Classes



SoftUni Team
Technical Trainers



SoftUni



Software University

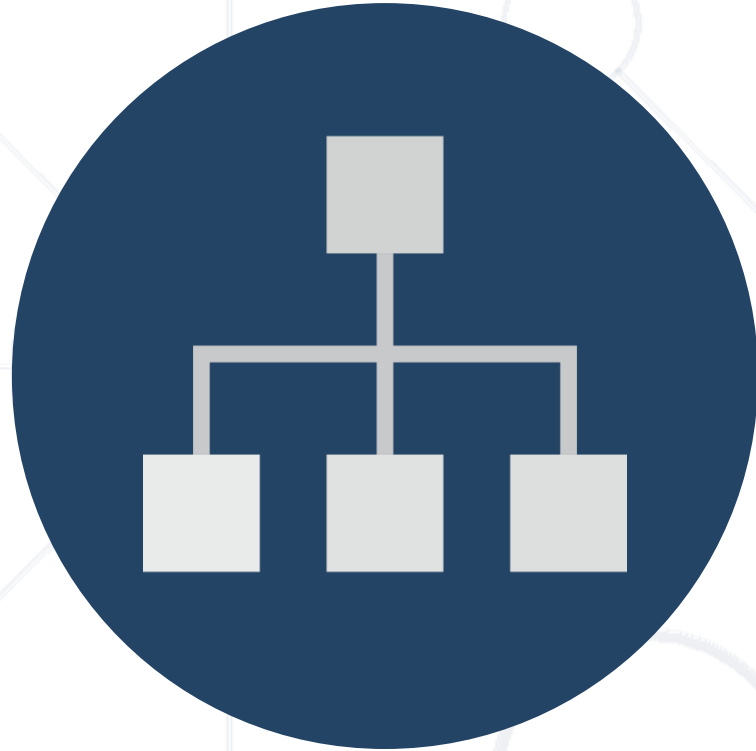
<https://softuni.org>



Channel for Communication

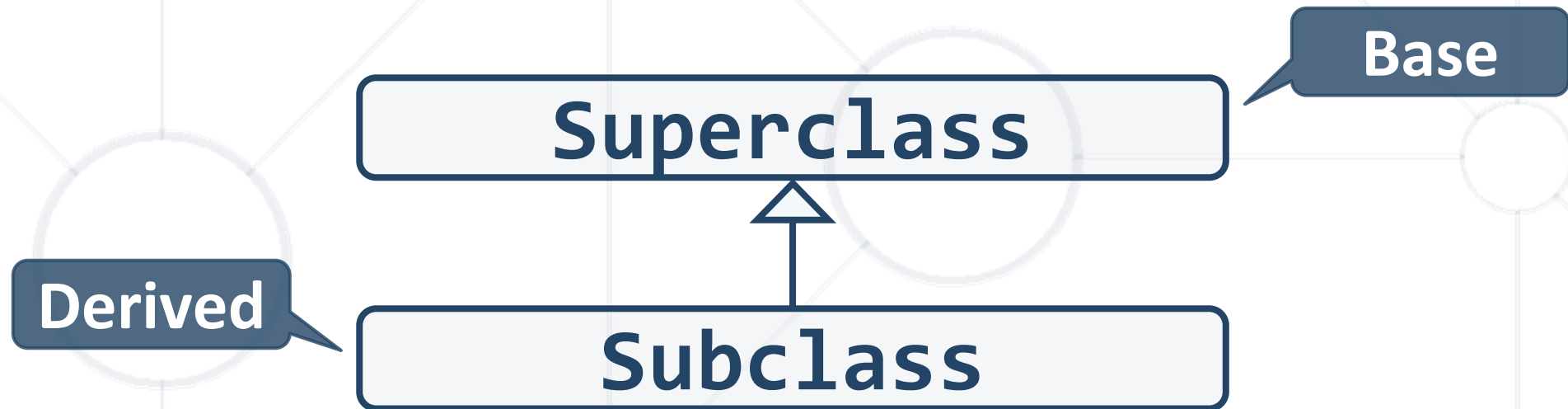
1. Inheritance
2. Class Hierarchies
3. Inheritance in Java
4. Accessing Members of the Base Class
5. Types of Class Reuse
 - Extension, Composition, Delegation
6. When to Use Inheritance



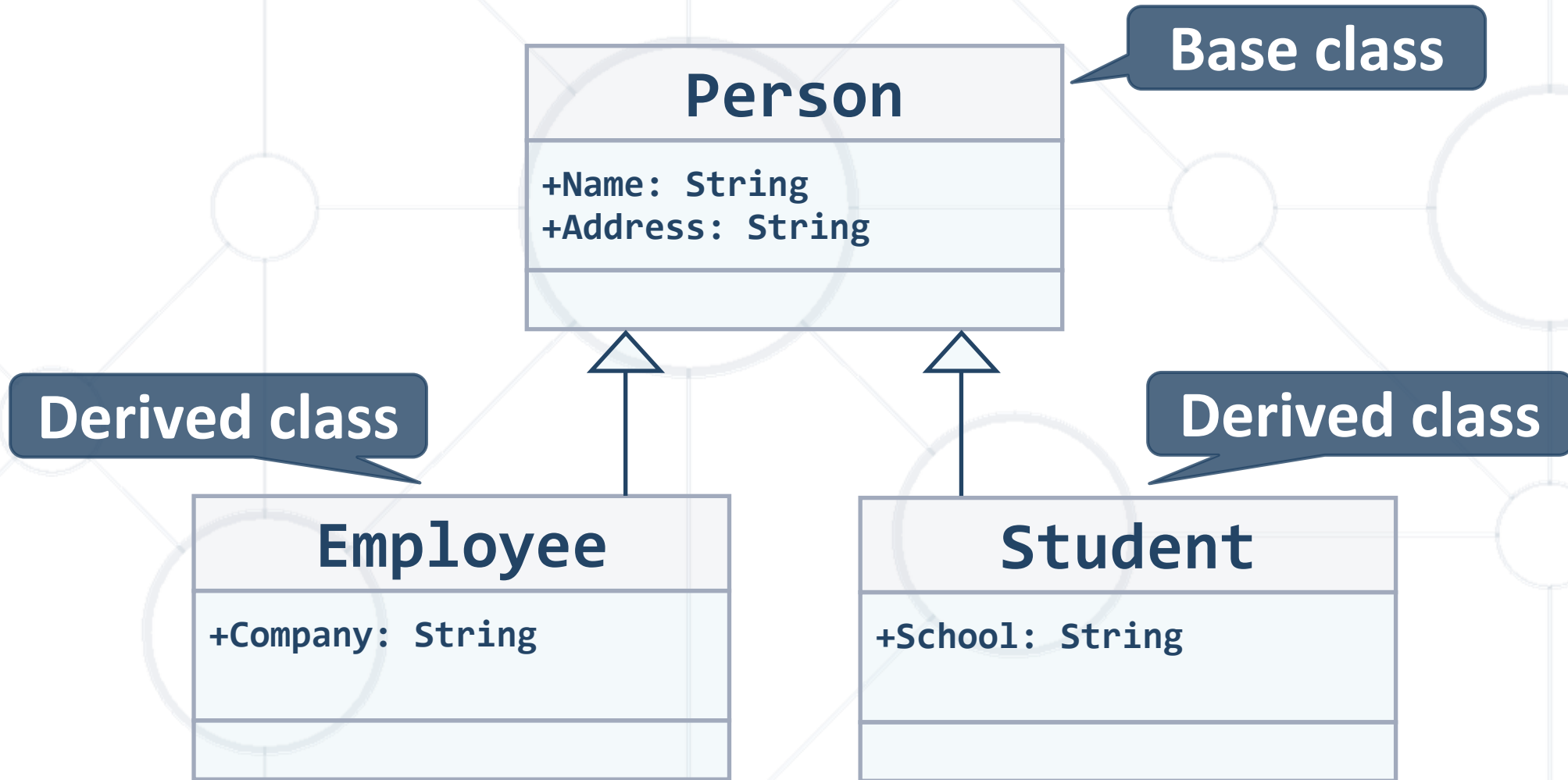


Inheritance

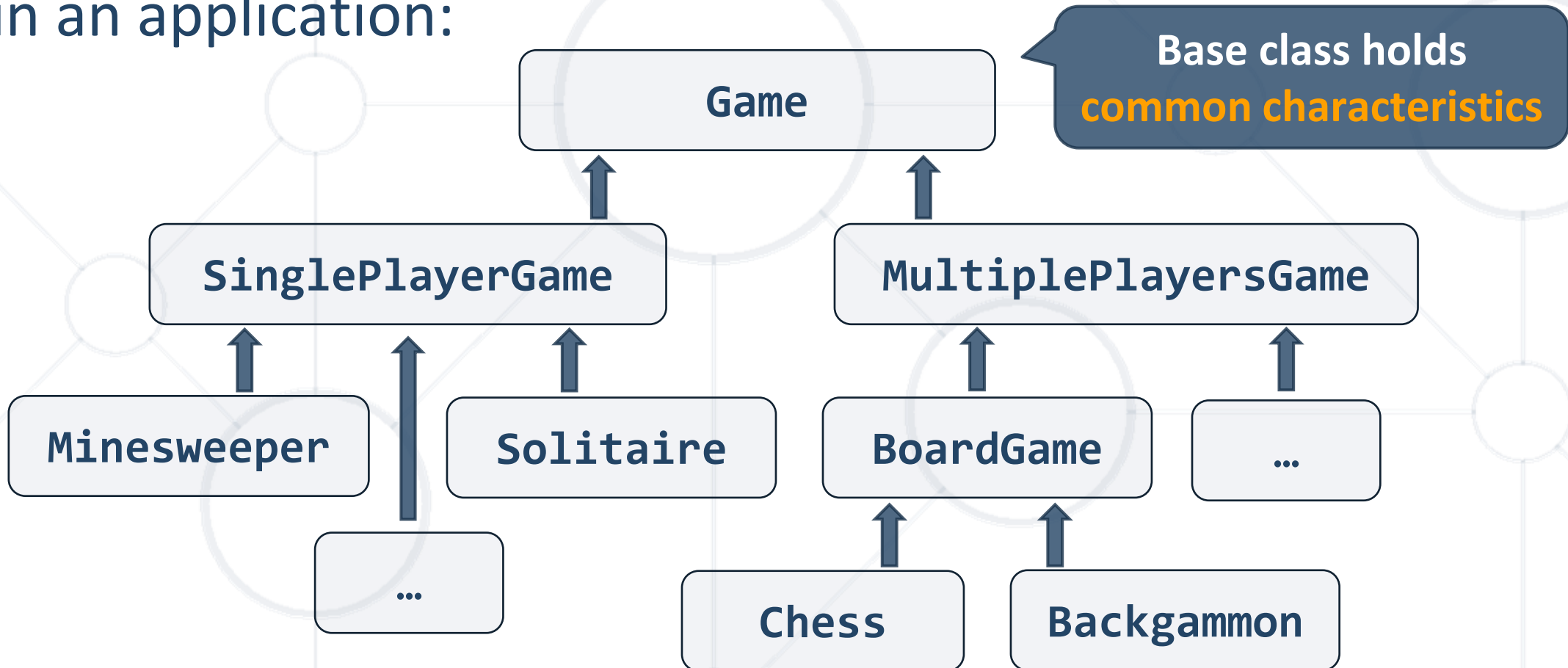
- **Superclass** - Parent class, Base Class
 - The class gives its members to its child class
- **Subclass** - Child class, Derived Class
 - The class taking members from its base class



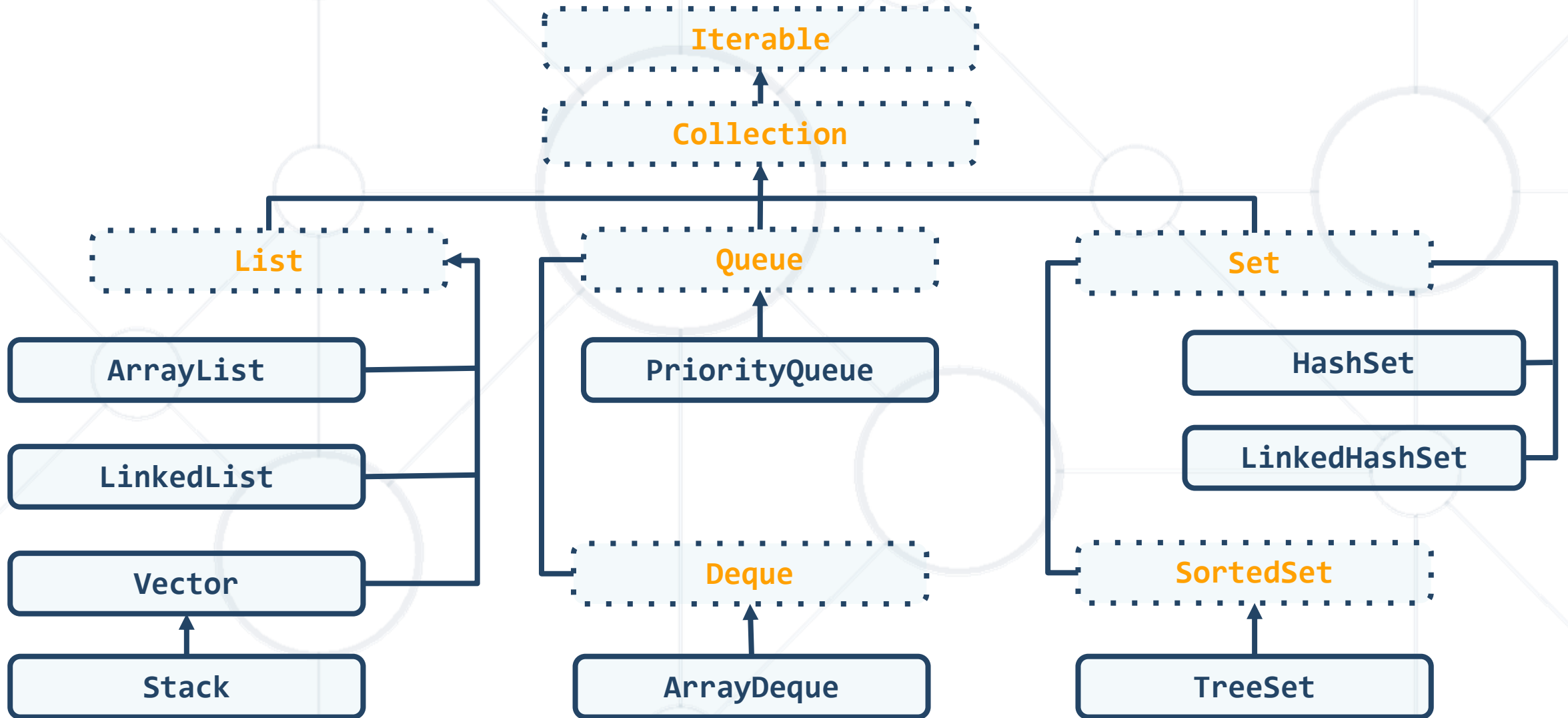
Inheritance – Example



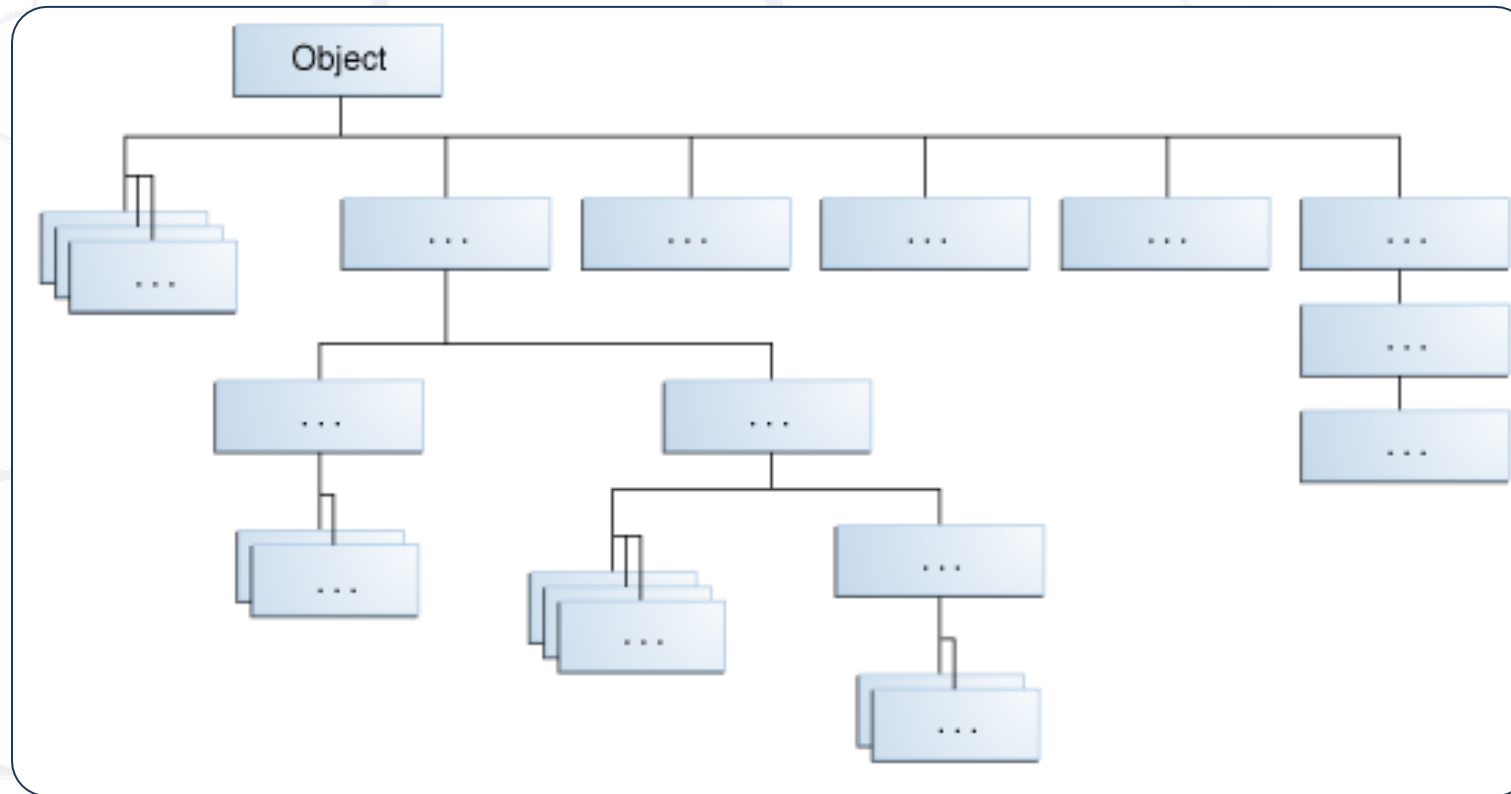
- An **Inheritance** leads to **hierarchies** of classes and/or interfaces in an application:



Class Hierarchies – Java Collection



- The **Object** is at the root of Java Class Hierarchy



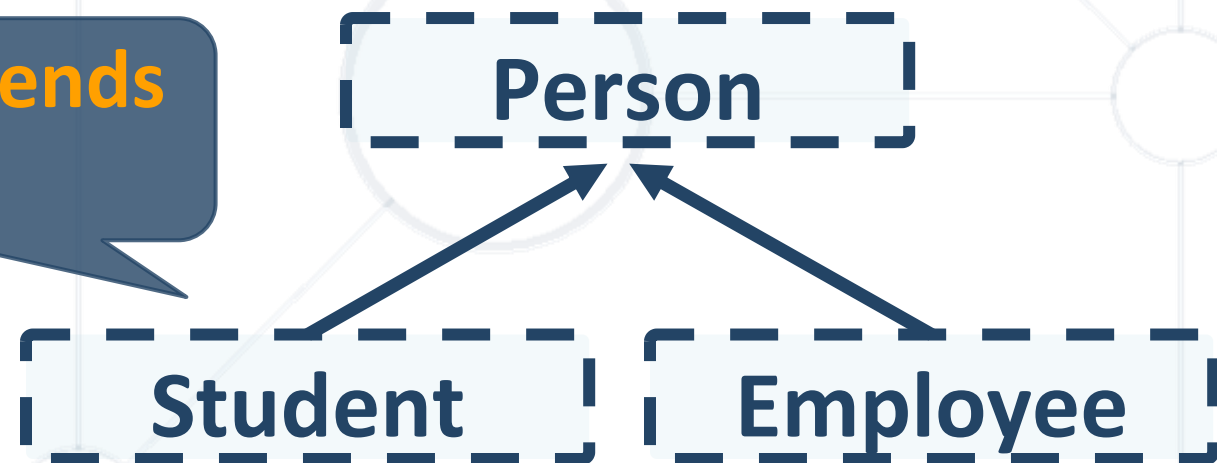
- Java supports inheritance through **extends** keyword

```
class Person { ... }
```

```
class Student extends Person { ... }
```

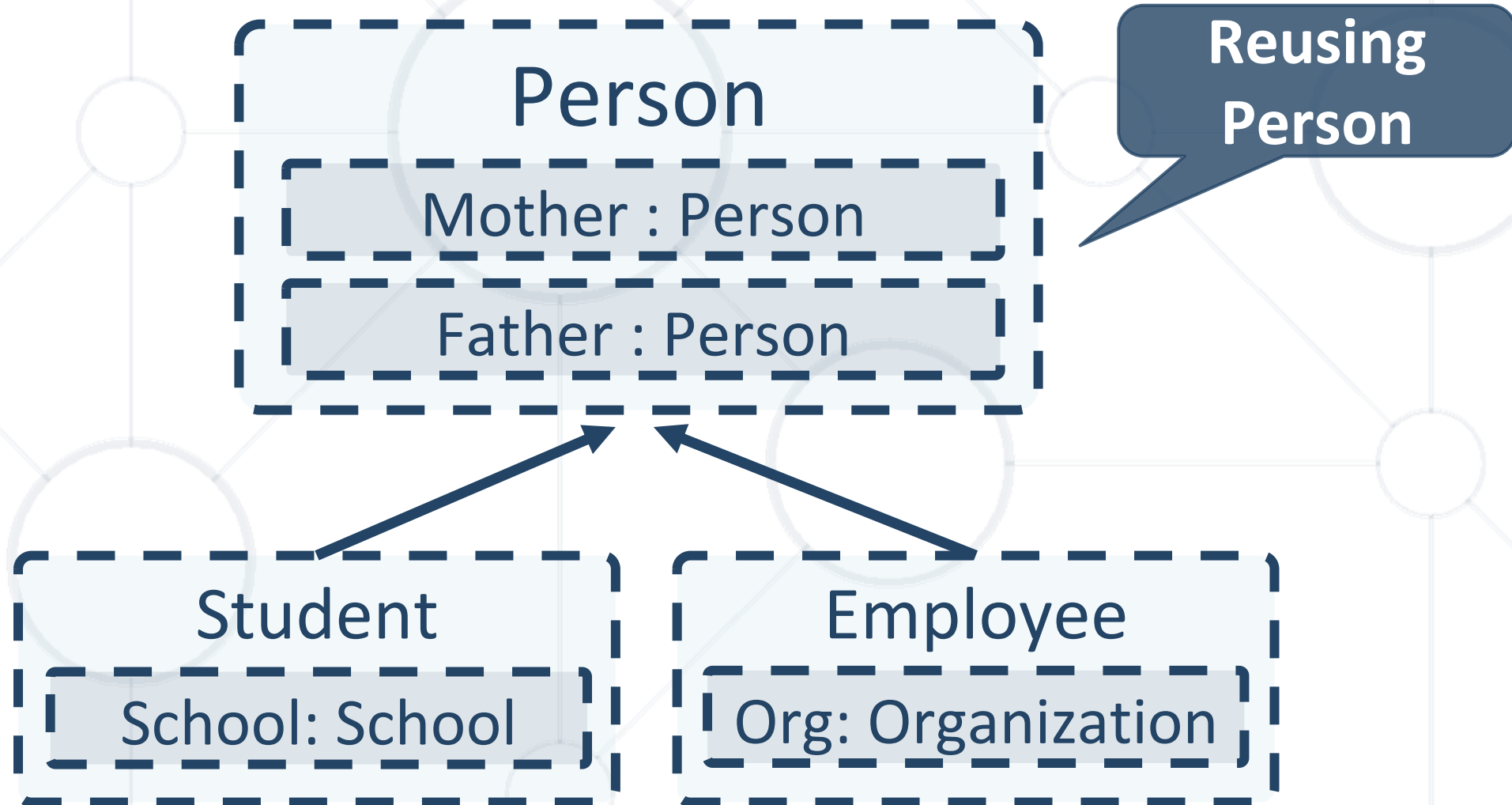
```
class Employee extends Person { ... }
```

Student **extends**
Person



Inheritance – Derived Class

- Class **taking all members** from another class



- You can access inherited members

```
class Person { public void sleep() { ... } }  
class Student extends Person { ... }  
class Employee extends Person { ... }
```

```
Student student = new Student();  
student.sleep();  
Employee employee = new Employee();  
employee.sleep();
```

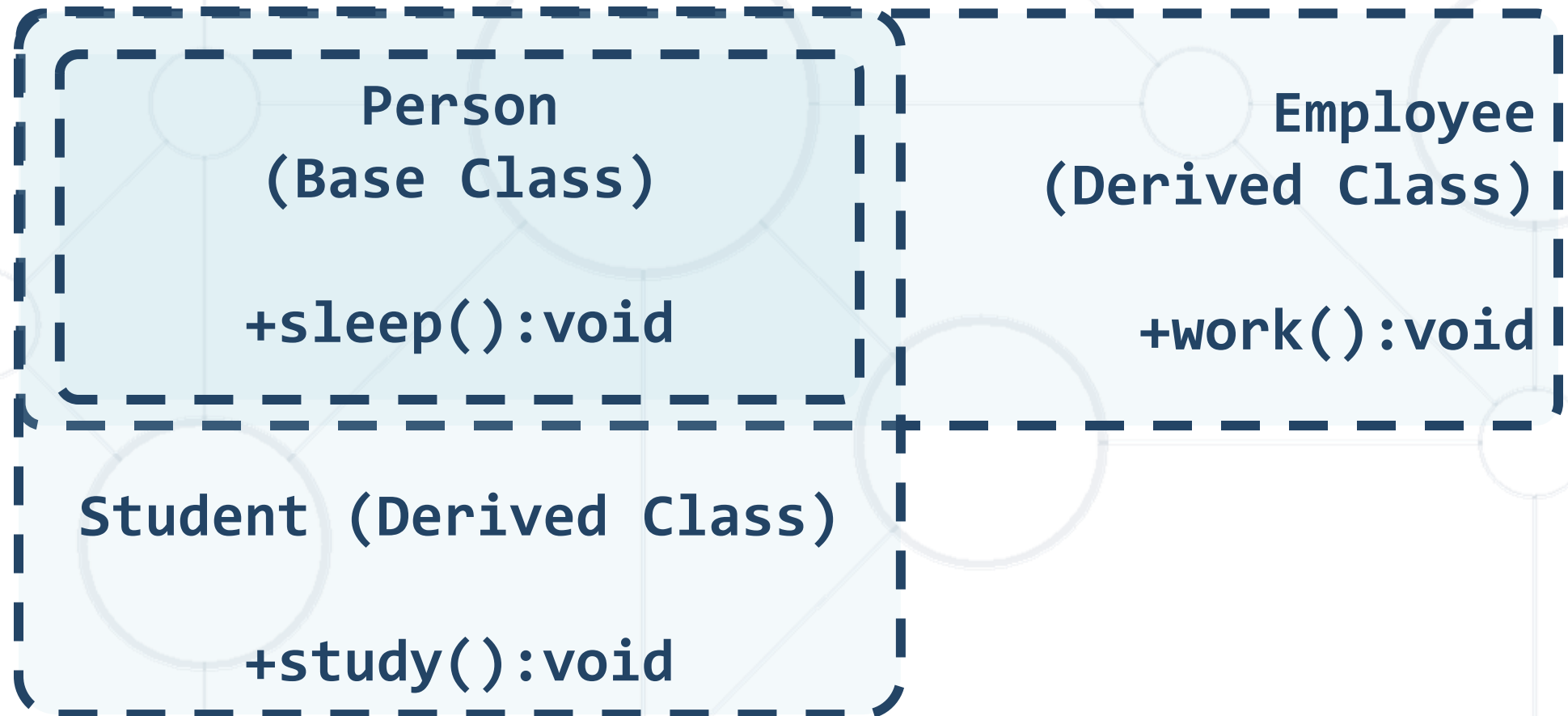
- Constructors are **not inherited**
- Constructors **can be reused** by the child classes

```
class Student extends Person {  
    private School school;  
    public Student(String name, School school) {  
        super(name);  
        this.school = school;  
    }  
}
```

Constructor call
should be first

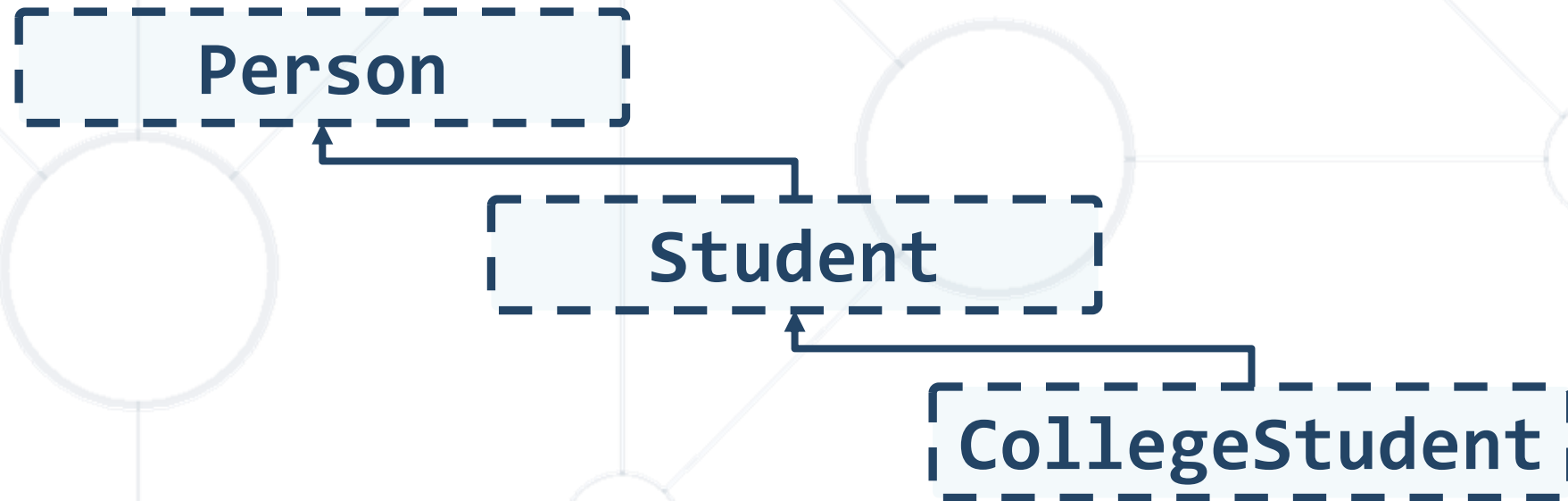
Thinking about Inheritance – Extends

- A derived class instance **contains** an instance of its base class



- Inheritance has a **transitive relation**

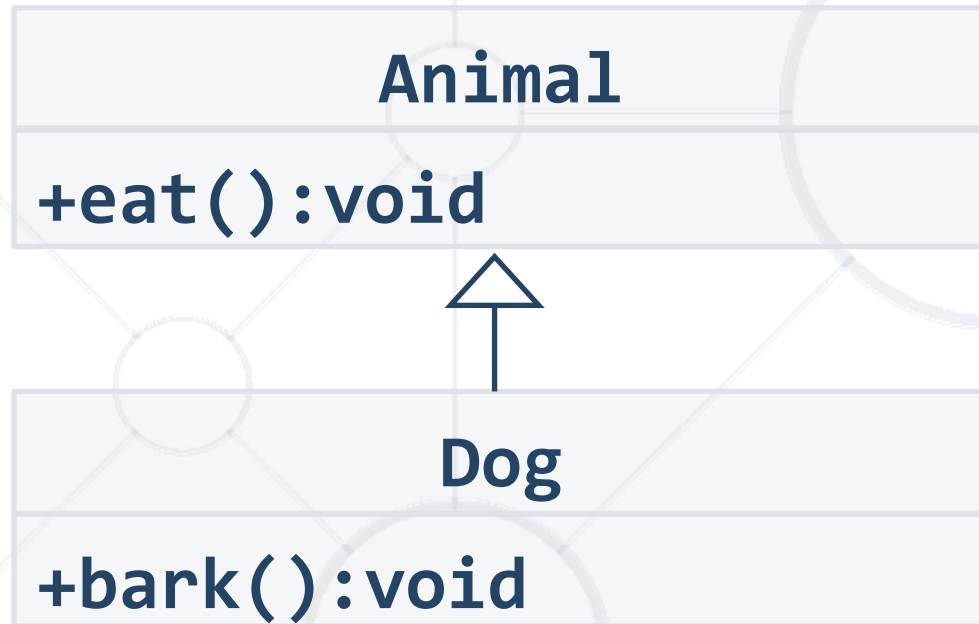
```
class Person { ... }  
class Student extends Person { ... }  
class CollegeStudent extends Student { ... }
```



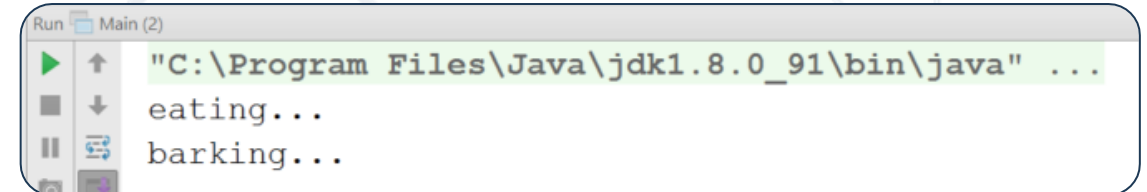
- Use the **super** keyword

```
class Person { ... }  
  
class Employee extends Person {  
    public void fire(String reasons) {  
        System.out.println(  
            super.name +  
            " got fired because " + reasons);  
    }  
}
```


Problem: Single Inheritance

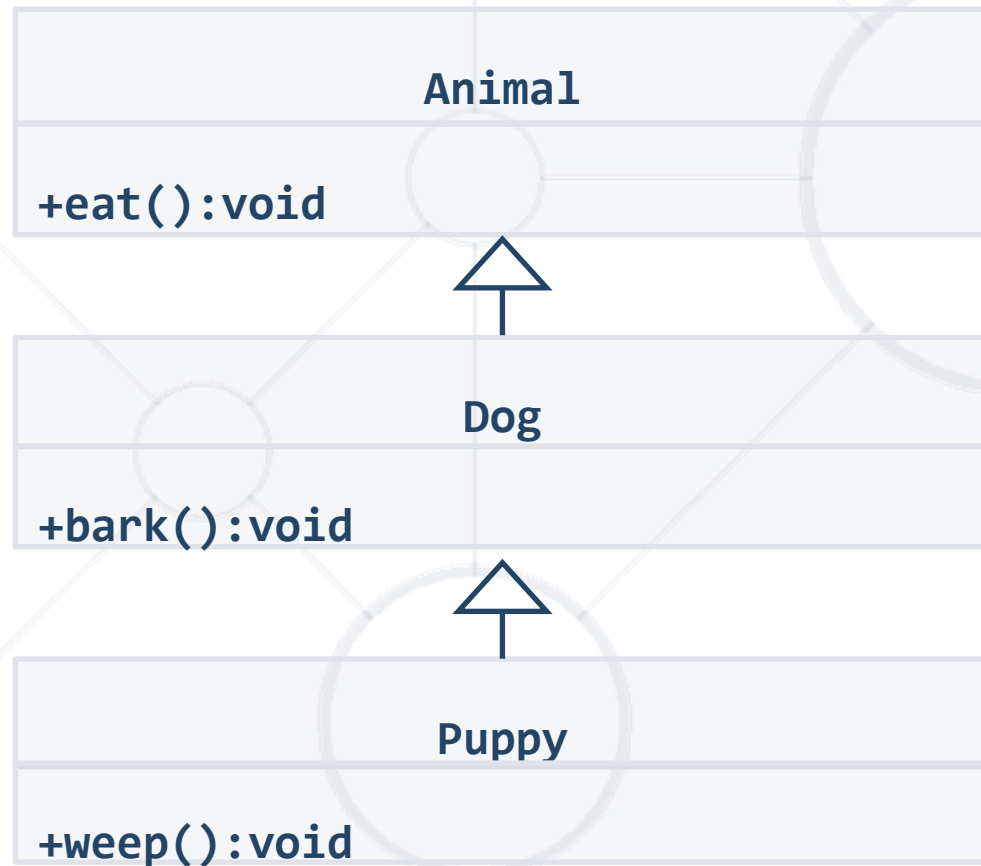


```
public static void main(String[] args) {  
  
    Dog dog = new Dog();  
    dog.eat();  
    dog.bark();  
  
}
```

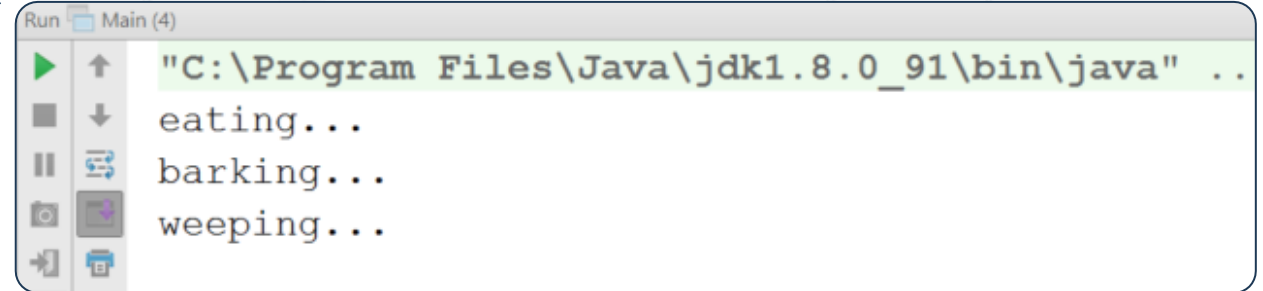


```
Run Main (2)  
"C:\Program Files\Java\jdk1.8.0_91\bin\java" ...  
eating...  
barking...
```

Problem: Multiple Inheritance



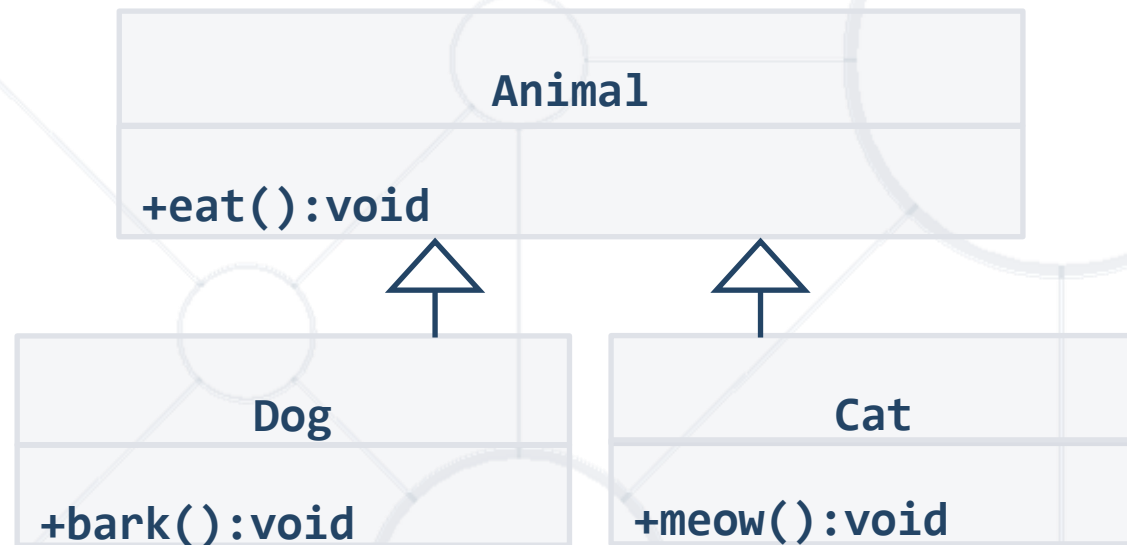
```
Puppy puppy = new Puppy();
puppy.eat();
puppy.bark();
puppy.weep();
```



The screenshot shows a Java IDE window titled "Run Main (4)". The command prompt displays the execution of the Java program, showing the output of the `eat()`, `bark()`, and `weep()` methods.

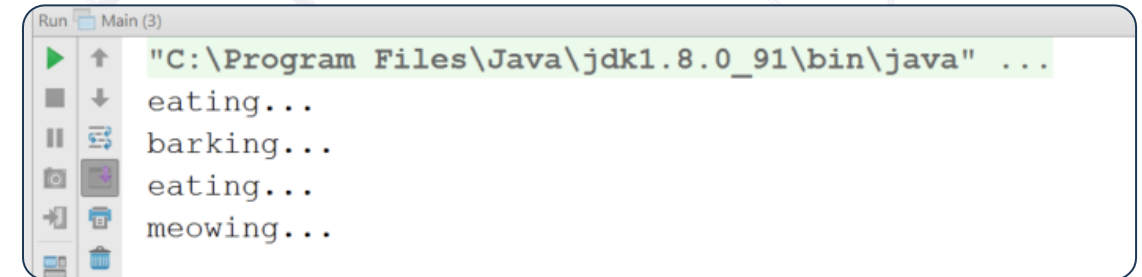
```
"C:\Program Files\Java\jdk1.8.0_91\bin\java" ..
eating...
barking...
weeping...
```

Problem: Hierarchical Inheritance



```
Dog dog = new Dog();
dog.eat();
dog.bark();
```

```
Cat cat = new Cat();
cat.eat();
cat.meow();
```



```
Run Main (3)
"C:\Program Files\Java\jdk1.8.0_91\bin\java" ...
eating...
barking...
eating...
meowing...
```



Reusing Classes

- Derived classes **can access all public** and **protected** members
- Derived classes can access **default** members **if in same package**
- **Private** fields **aren't inherited** in subclasses (can't be accessed)

```
class Person {  
    protected String address;  
    public void sleep();  
    String name;  
    private String id;  
}
```

Can be accessed
through other methods

- Derived classes **can hide** superclass variables

```
class Person { protected int weight; }
```

```
class Patient extends Person {  
    protected float weight;  
    public void method() {  
        double weight = 0.5d;  
    }  
}
```

hides **int weight**

hides both

Shadowing Variables – Access

- Use **super** and **this** to specify member access

```
class Person { protected int weight; }
```

```
class Patient extends Person {  
    protected float weight;  
    public void method() {  
        double weight = 0.5d;  
        this.weight = 0.6f;  
        super.weight = 1;  
    }  
}
```

Local variable

Instance member

Base class member

Overriding Derived Methods

- A **child class** can redefine existing methods

```
public class Person {  
    public void sleep() {  
        System.out.println("Person sleeping");  
    }  
}
```

Method in base class **must not be final**

```
public class Student extends Person {  
    @Override  
    public void sleep(){  
        System.out.println("Student sleeping");  
    }  
}
```

Signature and return
type **should match**

- **final** – defines a method that **can't be overridden**

```
public class Animal {  
    public final void eat() { ... }  
}
```

```
public class Dog extends Animal {  
    @Override  
    public void eat() {} // Error...  
}
```

- Inheriting from final classes is forbidden

```
public final class Animal {  
    ...  
}
```

```
public class Dog extends Animal { }           // Error..  
public class MyString extends String { }      // Error..  
public class MyMath extends Math { }          // Error..
```

- One approach for providing an abstraction

Focus on common
properties

```
Person person = new Person();  
Student student = new Student();
```

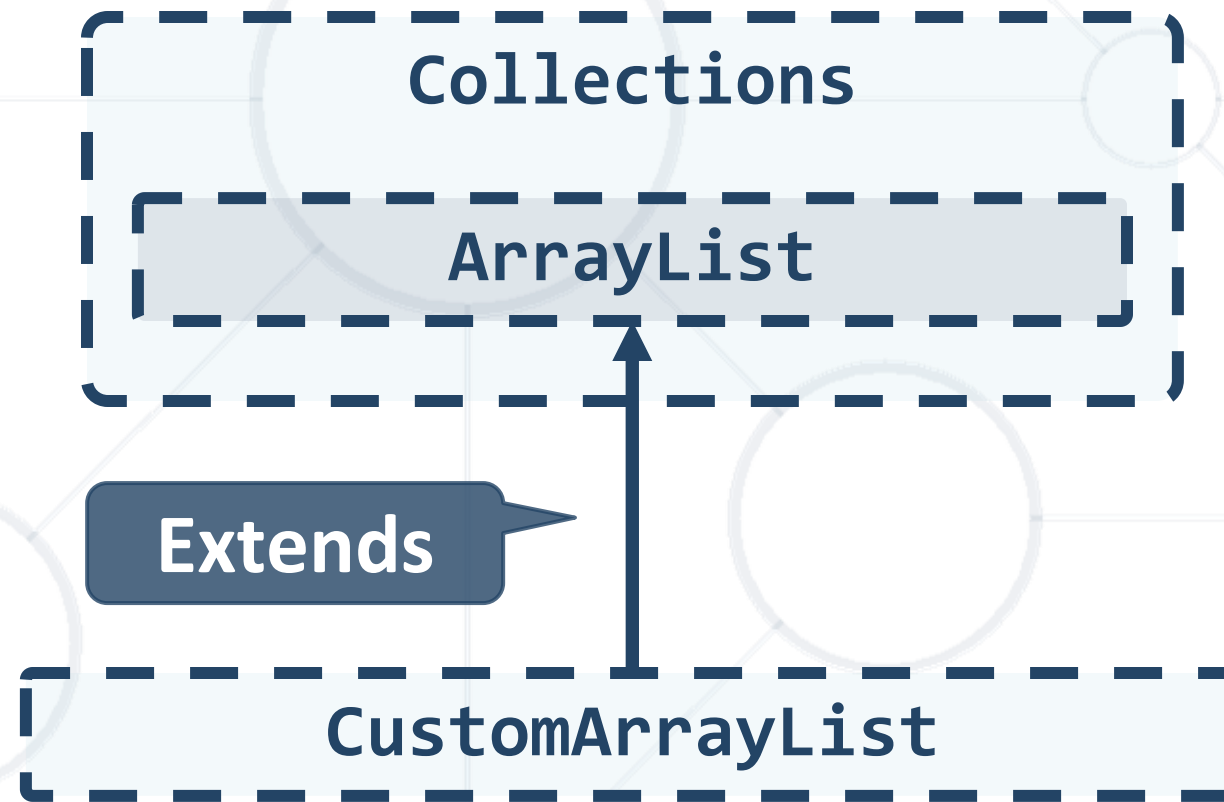
```
List<Person> people = new ArrayList();
```

```
people.add(person);  
people.add(student);
```

Person (Base Class)

Student (Derived Class)

- We can **extend a class** that we **can't otherwise change**



Problem: Random Array List

- Create an array list that has
 - All functionality of an ArrayList
 - Function that returns and removes a random element



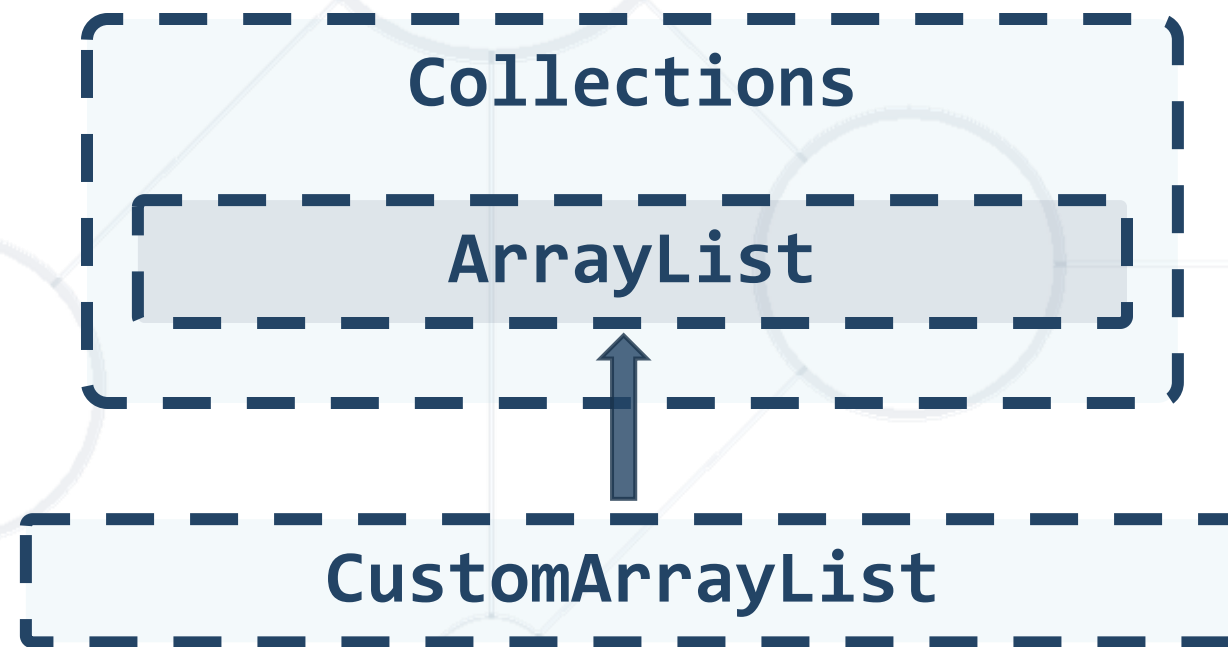
Solution: Random Array List

```
public class RandomArrayList extends ArrayList {  
    private Random rnd; // Initialize this...  
  
    public Object getRandomElement() {  
        int index = this.rnd.nextInt(super.size());  
        Object element = super.get(index);  
        super.remove(index);  
        return element;  
    }  
}
```



Types of Class Reuse

- **Duplicate code** is error prone
- **Reuse classes** through the **extension**
- Sometimes the only way



- Using classes to **define classes**

```
class Laptop {  
    Monitor monitor;  
    Touchpad touchpad;  
    Keyboard keyboard;  
    ...  
}
```

Reusing classes

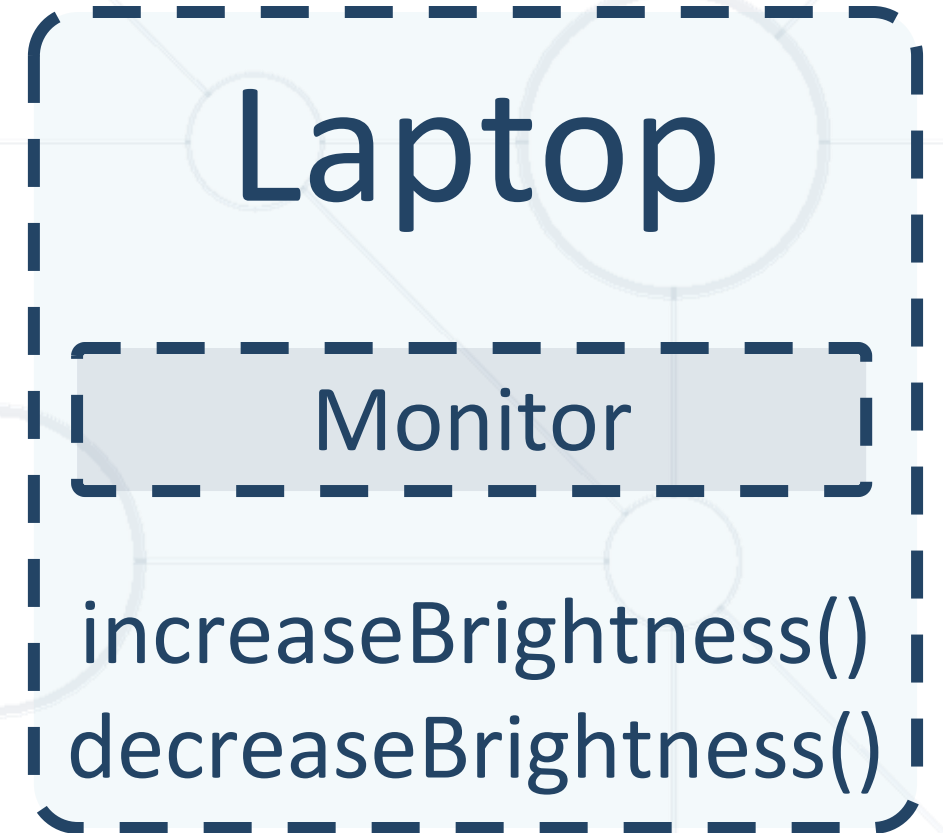
Laptop

Monitor

Touchpad

Keyboard

```
class Laptop {  
    Monitor monitor;  
    void incrBrightness() {  
        monitor.brighten();  
    }  
  
    void decrBrightness() {  
        monitor.dim();  
    }  
}
```



Problem: Stack of Strings

- Create a simple Stack class which can store only strings

StackOfStrings

```
-data: List<String>  
+push(String) :void  
+pop(): String  
+peek(): String  
+isEmpty(): boolean
```


StackOfStrings

ArrayList

```
StackOfStrings sos = new StackOfStrings();  
sos.push("one");  
System.out.println(sos.pop());  
System.out.println(sos.isEmpty());  
System.out.println(sos.peek());
```

Solution: Stack of Strings

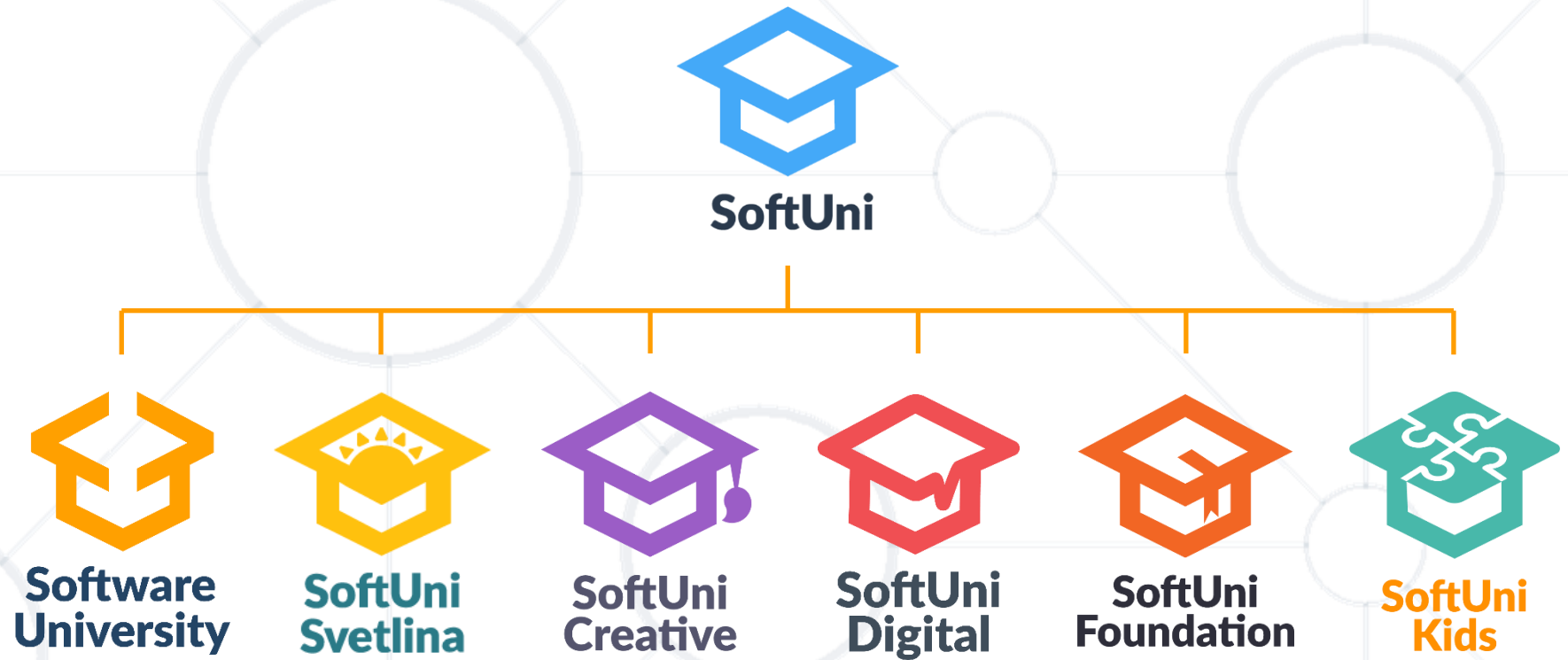
```
public class StackOfStrings {  
    private List<String> container;  
    // TODO: Create a constructor  
    public void push(String item) { this.container.add(item); }  
    public String pop() {  
        // TODO: Validate if list is not empty  
        return this.container.remove(this.container.size() - 1);  
    }  
}
```

- Classes share **IS-A** relationship  Too simplistic
- Derived class **IS-A-SUBSTITUTE** for the base class
- Share the **same role**
- The derived class is the **same as the base class** but adds a **little bit more functionality**

- Inheritance is a powerful tool for **code reuse**
- **Subclass inherits** members from **Superclass**
- Subclass can **override** methods
- Look for classes with the **same role**
- Look for **IS-A** and **IS-A-SUBSTITUTE** for relationship
- Consider **Composition** and **Delegation** instead



Questions?



SoftUni Diamond Partners



SCHWARZ



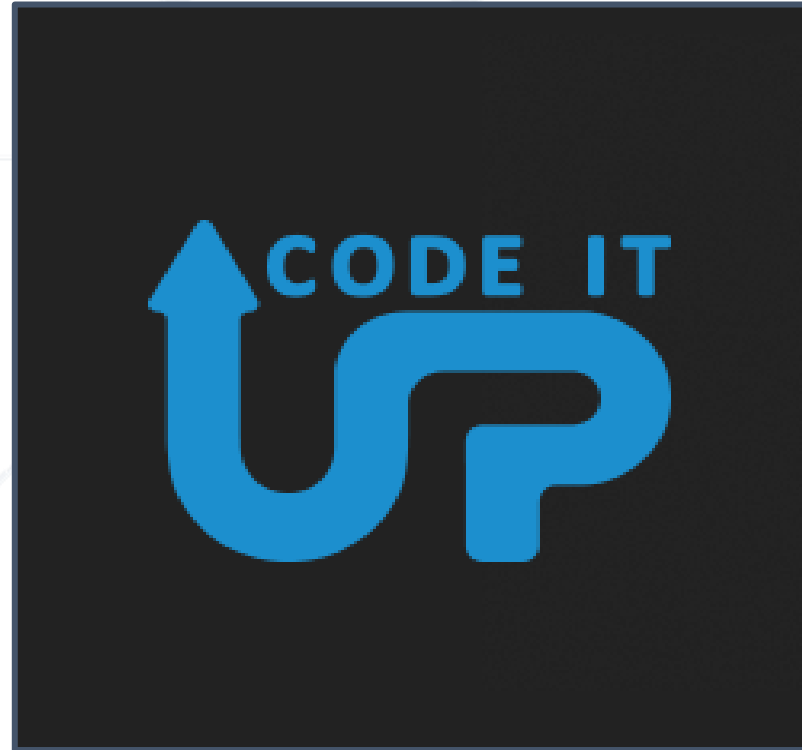
**SUPER
HOSTING
.BG**



INDEAVR
Serving the high achievers

Bosch.io





Trainings @ Software University (SoftUni)



- Software University – High-Quality Education, Profession and Job for Software Developers
 - softuni.bg
- Software University @ Facebook
 - facebook.com/SoftwareUniversity
- SoftUni Global
 - softuni.org



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>
- © SoftUni Global – <https://softuni.org>

