Inheritance in Java

In this lesson, you'll learn about an important concept of Object Oriented programming known as Inheritance.

We'll cover the following

- What is inheritance?
- Terminology
- Notation
- What does a child have?
- Class access specifiers
- The super keyword
- Understanding the example

What is inheritance?

`Inheritance` provides a way to create a **new** class from an **existing** class. The **new** class is a *specialized* version of the **existing** class such that it **inherits** elements of the existing class.

Terminology

- **Superclass**(Parent Class): This class allows the re-use of its members in another class.
- **Derived Class**(Child Class or Subclass): This class is the one that inherits from the superclass.

Hence, we can see that classes can be built by using previously created classes!

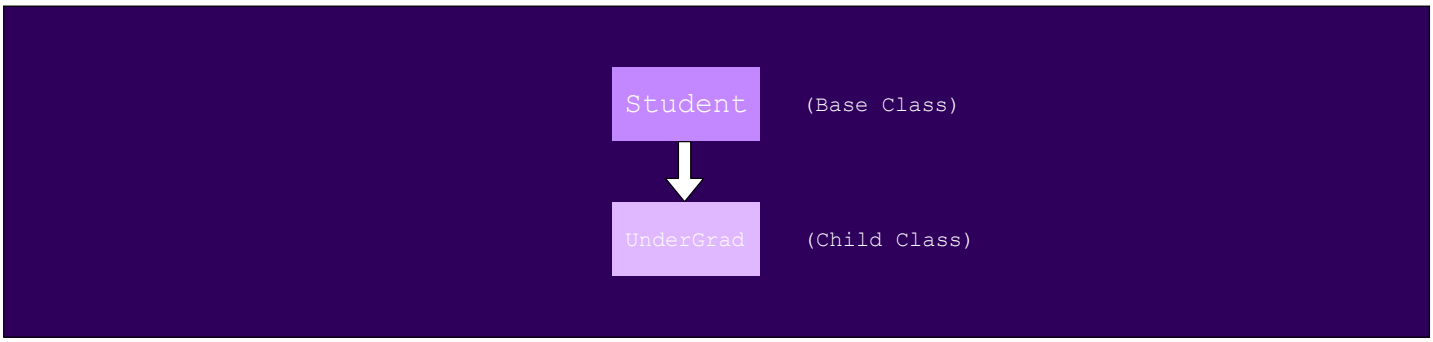
Notation

Let's take a look at the notation necessary for the creation of **subclasses**. This is done by using the keyword `extends`.

```
class Student{
```

```
  public String name;
  public int age;
}


class Undergrad extends Student{
  String major;
  public Undergrad(){
    this.major = "Computer"
    this.name="John Doe";
    this.age=50;
  }
}
```

Note: In line 6, the derived class is declared but is followed by the word "extends" and then the name of the parent class



Inheritance establishes an **" is an"** relationship between *classes*.

An *object* of the *derived class* “is an” object of the *base class* or the *parent class*. For example:

- An `UnderGrad` is a `Student`.

> **Important Note:** A *derived* object has **all** characteristics of the *parent* class.

What does a child have?

An *object* of **child** class has:

- All *members* defined in the **child** class.
- All *members* declared in the **parent** class.

An object of a child class can use:

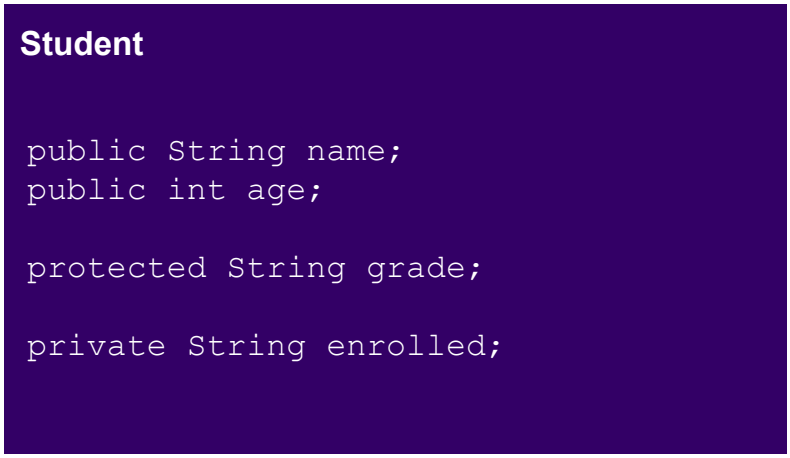
- All `public` members defined in the **child** class.
- All `public` members defined in the **parent** class.

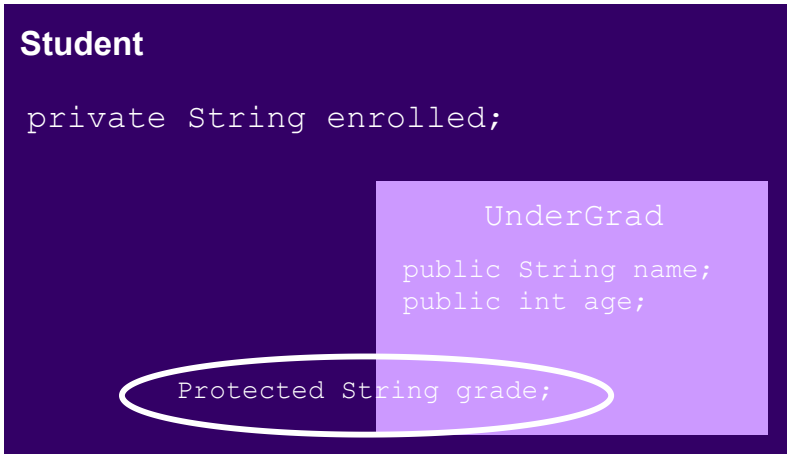
> **Note:** Some classes cannot be inherited. Such classes are defined with the keyword, `final`. An example of such a class is `Integer` - this class cannot have derived classes.

Another thing to be kept in mind is that a class can only inherit from *one class*! Hence, a subclass cannot be created as a child of two classes- this creates ambiguity and would require complicated rules. Hence, **to keep it simple**, inheritance is only applicable if derived from a single class.

Class access specifiers

- `public`: the *object* of the **derived** class can be treated as an object of the **super** class.
- `protected`: more restrictive than `public`, but allows **derived** classes to know details of the *super* class.
- `private`: prevents objects of the **derived** class to be treated as objects of the **super** class.



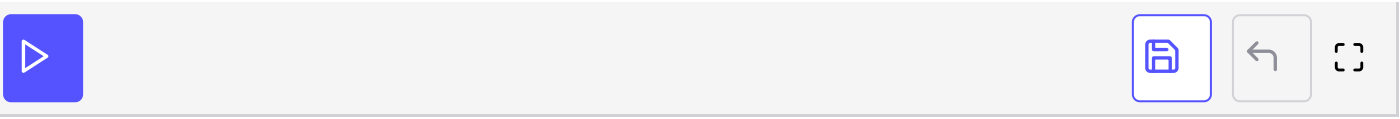


The `super` keyword

This keyword allows the *derived* class to access members of its *superclass* because the `this` keyword is used to access the members of its own class. Confused? Let us look at the example below, and its explanation for a better understanding.

```
class Student {
    public String name;
    public int age;

    public void setAge(int a) {
        age = a;
    }
}

class UnderGrad extends Student {
    public UnderGrad() {
        this.age = 10;
        this.name = "John Doe";
    }
    public void set_age(int a) {
        if (a < 50) {
            age = 0;
        } else {
            super.setAge(a);
        }
    }
}

class example {
    public static void main(String[] args) {
        UnderGrad one = new UnderGrad();
        System.out.println("Age without any method called, only constructor: " + one.age);
        one.set_age(50);
```

```
        System.out.println("Age after set_age(50) is called: " + one.age);
        one.set_age(10);
        System.out.println("Age after set_age(10) is called: " + one.age);

    }
}
```



Understanding the example

Lines 1 - 8:

- The **superclass** is declared in these lines. The name of the **superclass** is set as `Student`.
- The class has two variables, `name` which is of type **String** and `age` which is of type **int**.
- This class also has a **public** method called `setAge()`. This method simply takes in an **integer** as an argument and sets the age to the given input.

Lines 11 - 21:

- The **derived class** is declared in these lines. The derived class is given the name `UnderGrad` and see that on **line 11** it `extends` from the `Student` class- this indicates that it is a **child class** of this particular class.
- This class has a **constructor** which sets the `name` of the `Undergrad` student initially as **John Doe** and the `age` is set to **10**.
- The class also has a **public** method called `set_age()`. This method is interesting as it uses the keyword `super` in its body. The method takes in an **integer** parameter. It then **checks** that if the parameter is **less than 50** then, it sets the `age` as **0**. However, if this is not the case, then it simply calls the `setAge()` method of the *parent class* by using the keyword `super`. We already know what this parent class method does!

Now that we have an understanding of classes and inheritance in Java, let's look at some challenges!