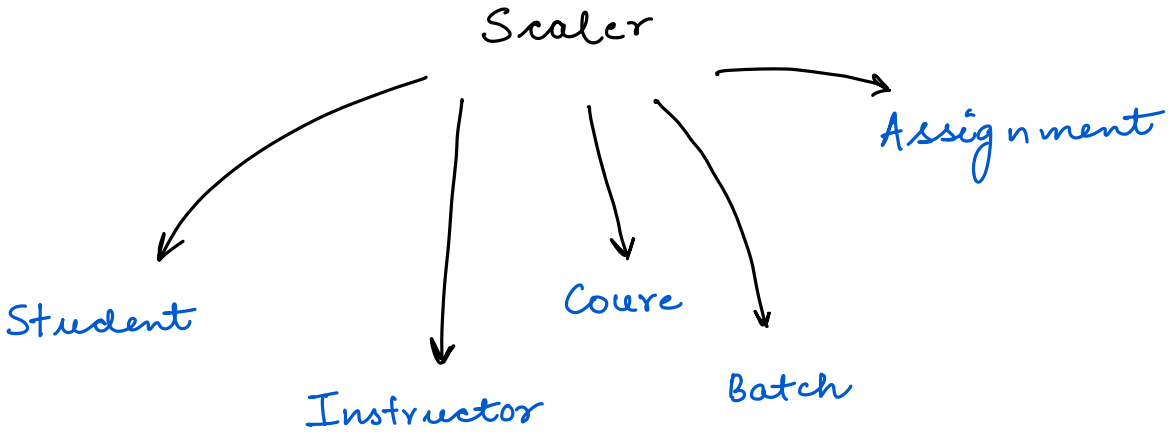
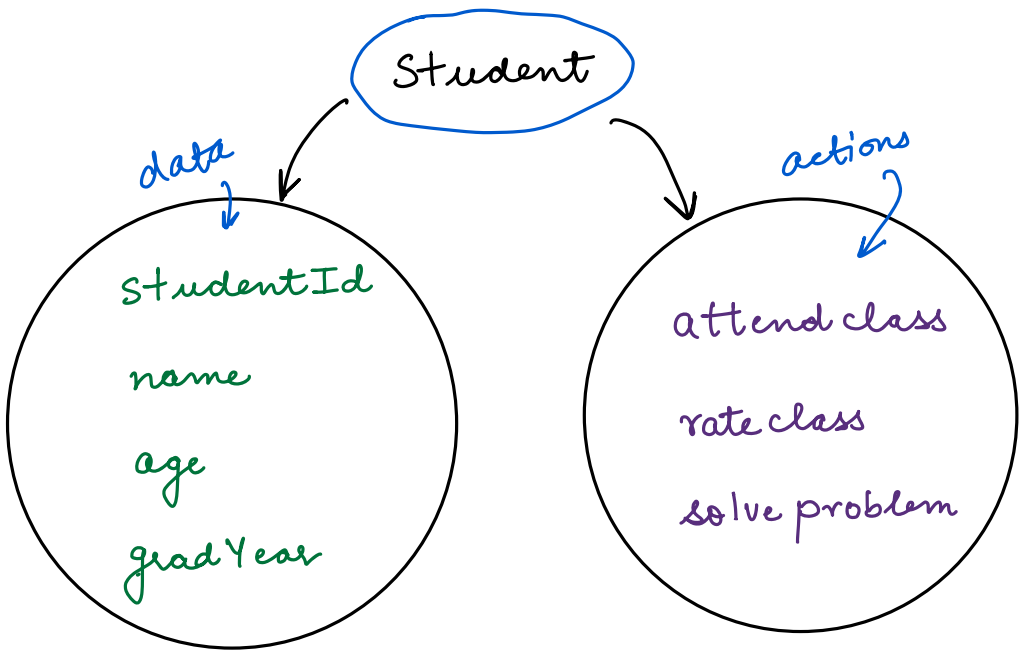
OOPS-1

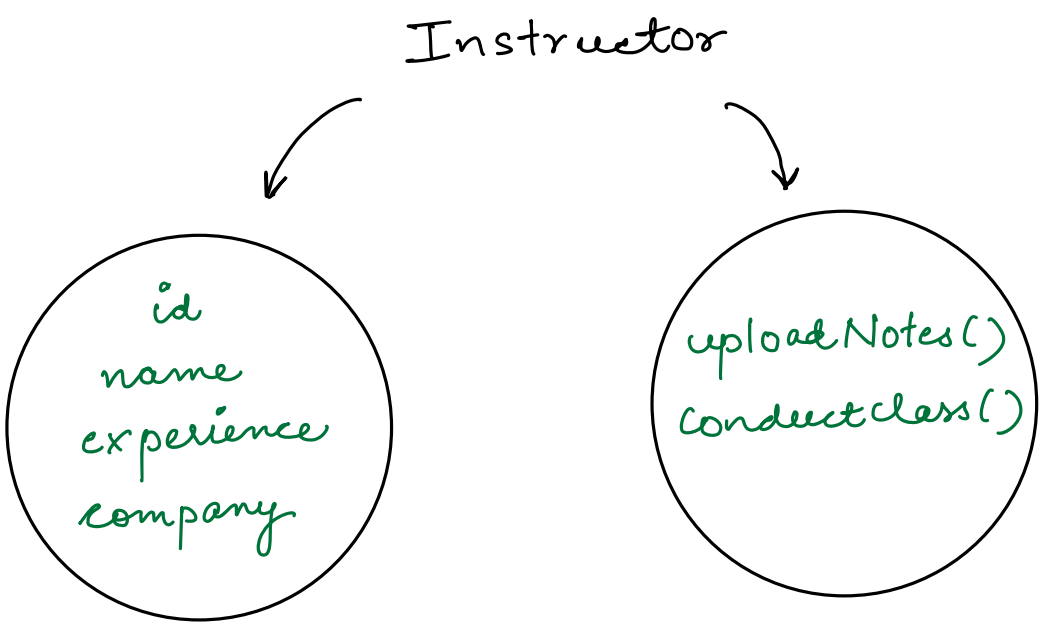
Agenda :

- class
- Object
- Encapsulation
- constructors
- Inheritance.

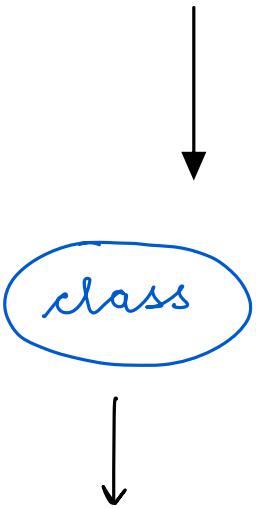


data (propatic)
+
behaviours





We need something which can help us group all of this information together !

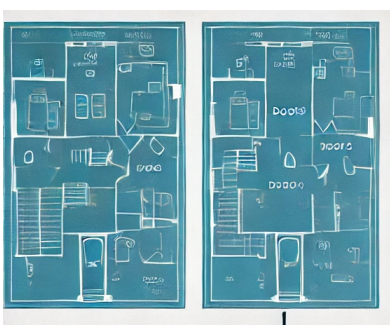


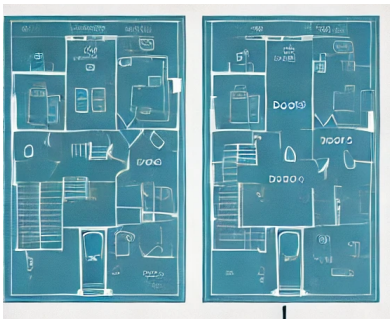
class

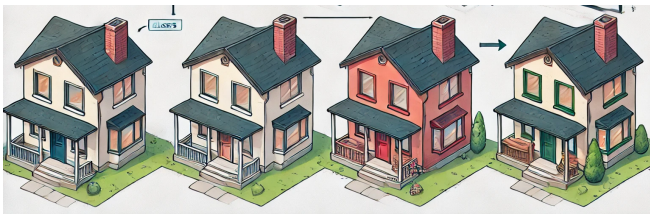
Blueprint of an entity / custom datatype, groups properties & behaviours together.

```java
class Student {  no usages  new *
    // Attributes
    String name;  3 usages
    int id;  1 usage
    String email;  1 usage
    String course;  3 usages

    // Functions
    void registerForCourse(String newCourse) {  no usages  new *
        this.course = newCourse;
        System.out.println(name + " has registered for the course: " + course);
    }

    void submitAssignment(String assignmentTitle) {  no usages  new *
        System.out.println(name + " has submitted the assignment: " + assignmentTitle);
    }

    void displayDetails() {  no usages  new *
        System.out.println("Student Details:");
        System.out.println("Name: " + name);
        System.out.println("ID: " + id);
        System.out.println("Email: " + email);
        System.out.println("Course: " + course);
    }

}
```

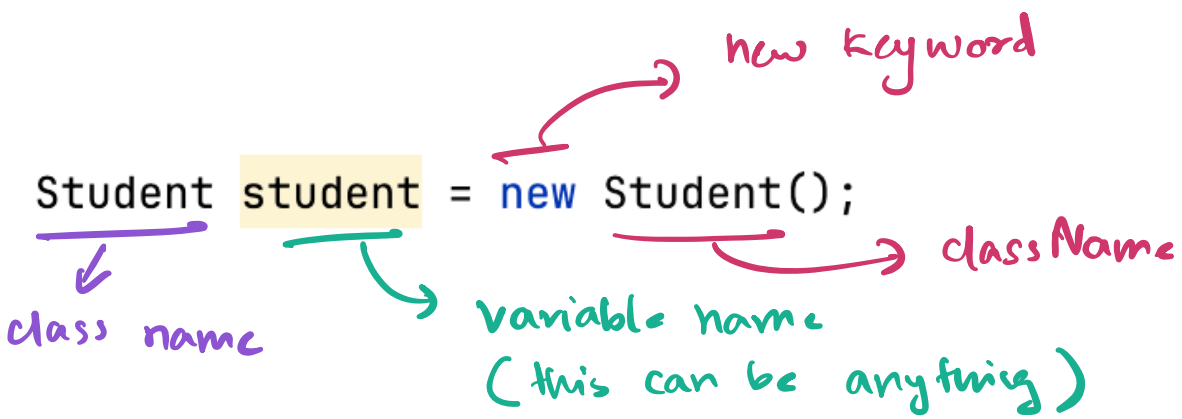
Can we store the data or use the methods just by creating this blueprint ???







class → existence → Object




```
student.name = "Mohit Sharma";
student.id = 1;
student.email = "mohit.sharma@scaler.com";
student.course = "Backend Low Level Design";
```

TASK 💡

- Create a new Java file BankAccount.java and define a simple BankAccount class to represent a bank account.
- Add two fields: balance (double) and ownerName (String).
- Add deposit and withdraw methods with basic validation (e.g., ensuring withdrawals don't result in negative balances).

Solution at 8:05

OOP PILLARS

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

What is Encapsulation ???

↳ capsule

Encapsulation is the bundling of data (attributes) and methods (functions) that operate on that data into a single unit called a class.

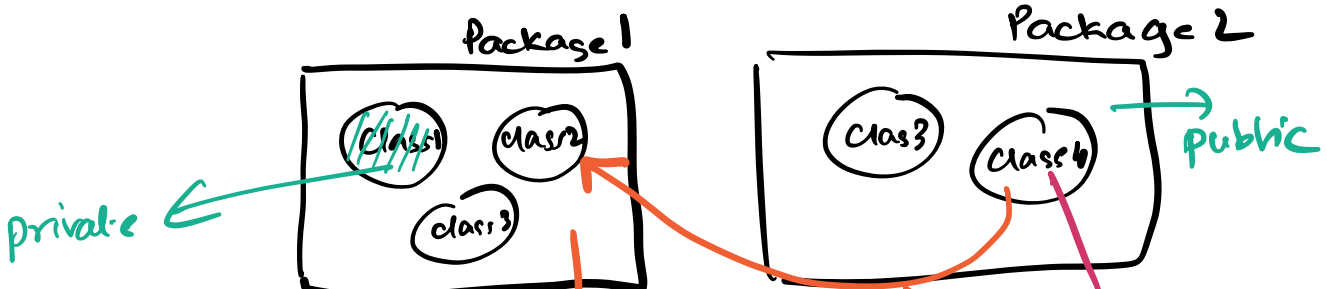
It also involves restricting direct access to some of the object's components for better control and security.

Access Modifiers

Helps us to control the access of our variables and methods.

- private → can only be accessed with in same class
- public → can be accessed anywhere
- protected (inheritance)
- default → can be accessed only with in a package (no access modifier)

Access Modifier	Same class	Same package subclass	Same package non-subclass	Difference Package subclass	Different package non-subclass
Default	Yes ✓	Yes ✓	Yes ✓	No ✗	No ✗
Private	Yes ✓	No ✗	No ✗	No ✗	No ✗
Protected	Yes ✓	Yes ✓	Yes ✓	Yes ✓	No ✗
Public	Yes ✓	Yes ✓	Yes ✓	Yes ✓	Yes ✓



default

inheritance

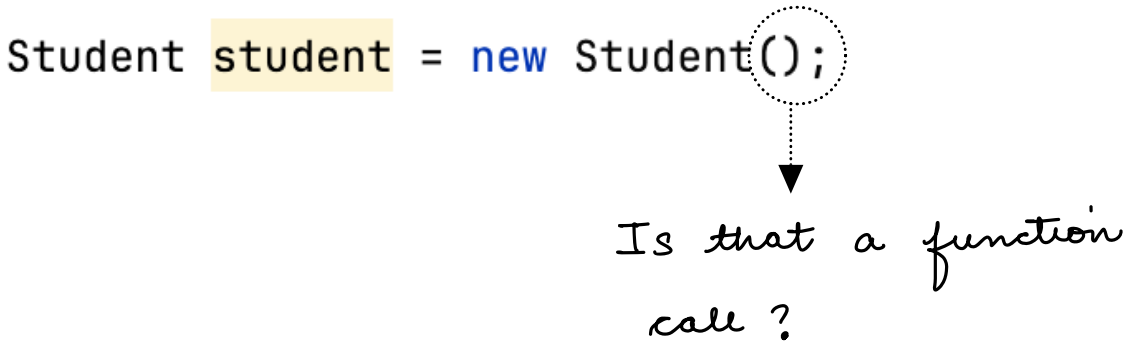
protected

How can we access our private data outside of the class if we want to ??

TASK 💡

- Make the fields private and methods public.
- Add getBalance(), setBalance(double balance), getOwnerName(), and setOwnerName(String name) methods.

Solution → 8:57



A constructor is a special type of method in object-oriented programming used to initialize an object when it is created. It is automatically called when an instance of a class is created.

Key Features of a Constructor:

- **Same Name as Class**: A constructor has the same name as the class it belongs to.
- **No Return Type**: It does not have a return type, not even void.
- **Automatic Invocation**: It is called automatically when an object is instantiated.
- **Initialization**: It is primarily used to set initial values for the object's attributes or perform setup tasks.

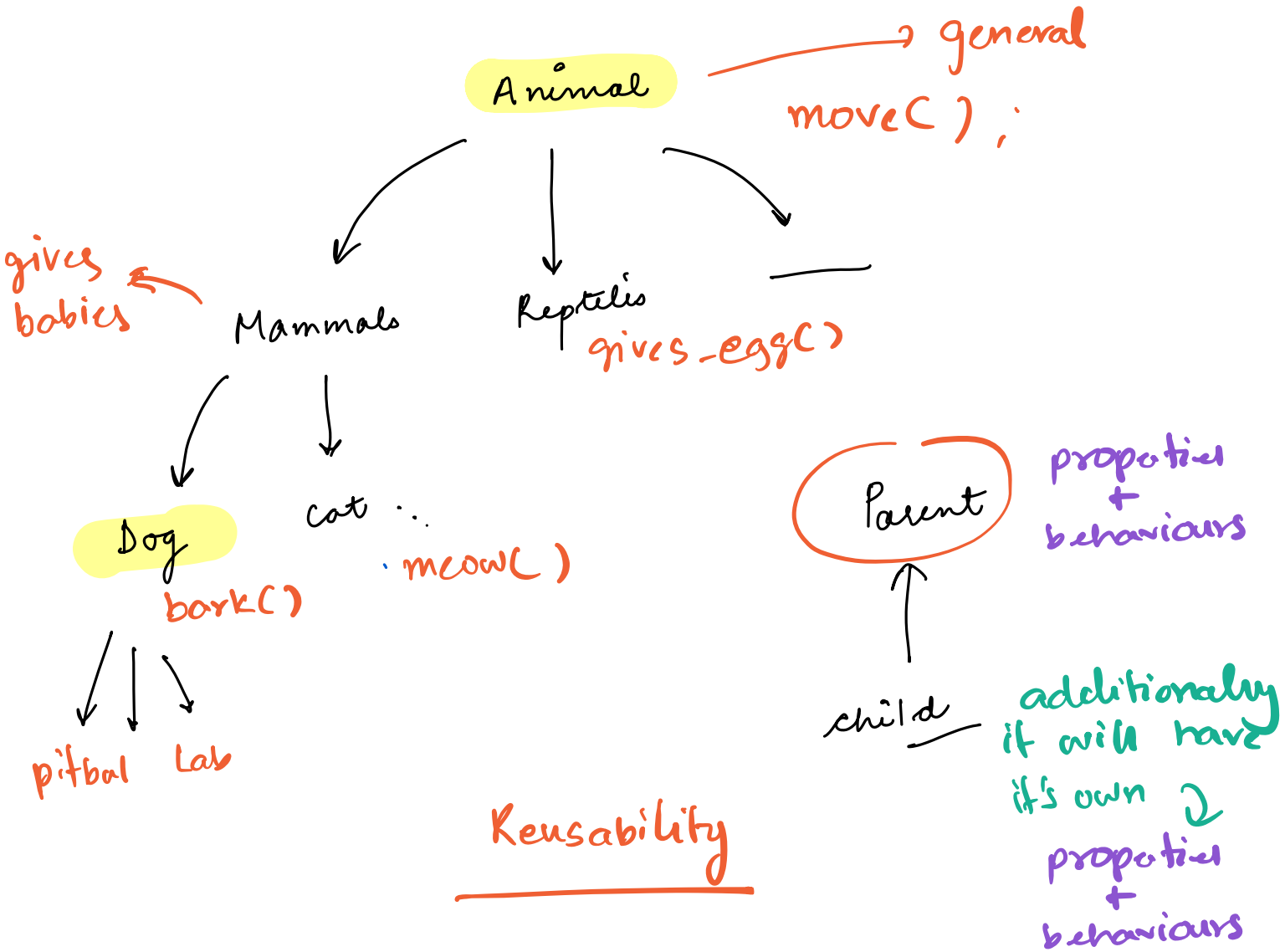
TASK 💡

- Create a default constructor and a parameterized constructor
- Create a simple transaction log in the form of a String array or ArrayList that records deposits and withdrawals.
  - Add a List<String> transactions field to store transaction messages.
  - Update deposit and withdraw to add messages to transactions (e.g., “Deposited $100”, “Withdrew $50”).
  - Add a method printTransactionHistory() to display the history.

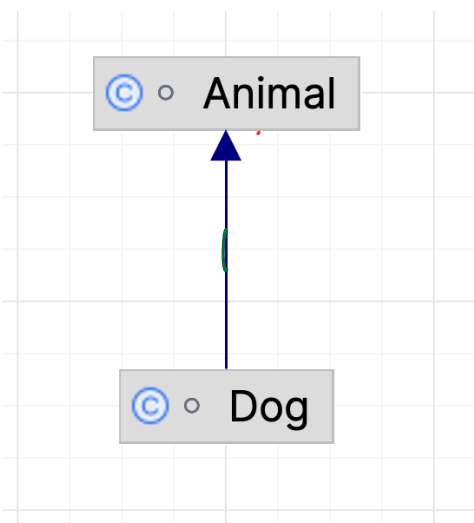
OOP PILLARS

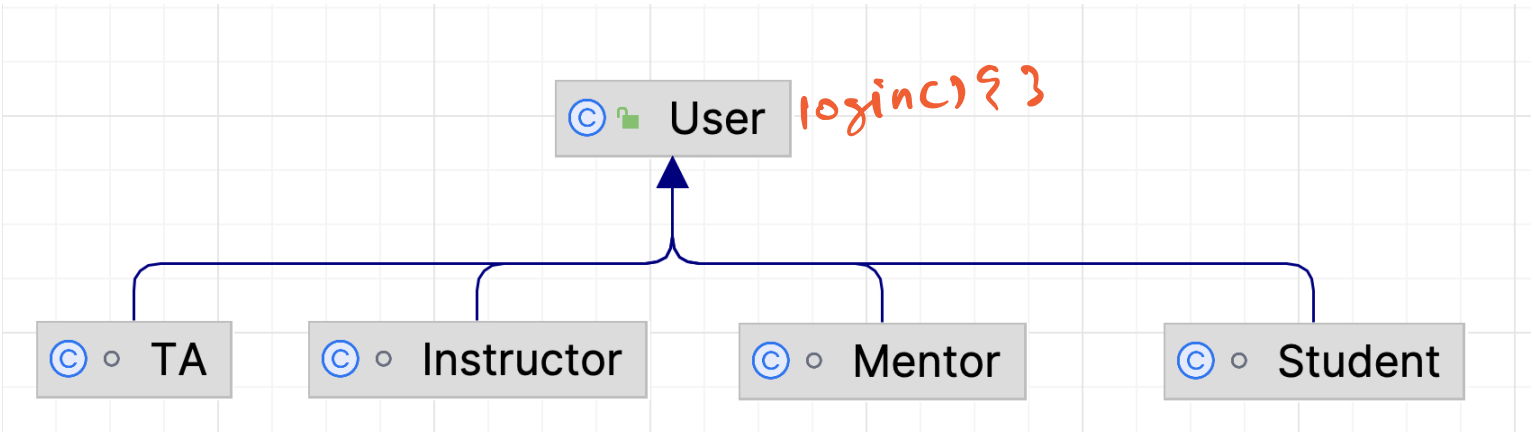
- Encapsulation
- Inheritance
- Polymorphism
- Abstraction

We are Animals!

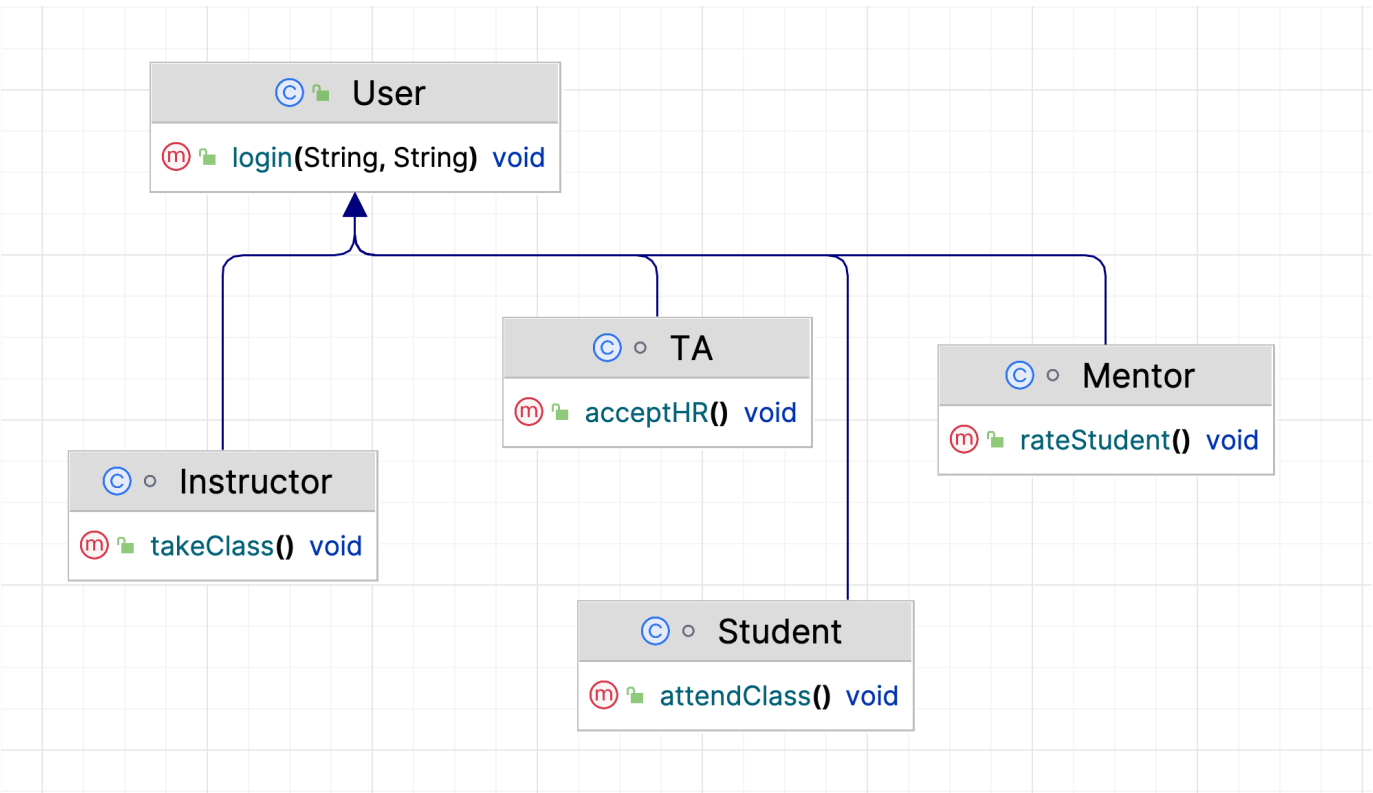



```
class Animal {  1 usage  1 inheritor  new *
}

class Dog extends Animal{  no usages  new *
}
```





login(){} login(){} login(){}



TASK 💡

- Creating Specialized Accounts with Inheritance
  - Create a SavingsAccount class that extends BankAccount.
  - Add a field interestRate, a constructor, and an applyInterest() method that calculates and deposits interest based on the balance.
  - Create CurrentAccount, which might have additional features.