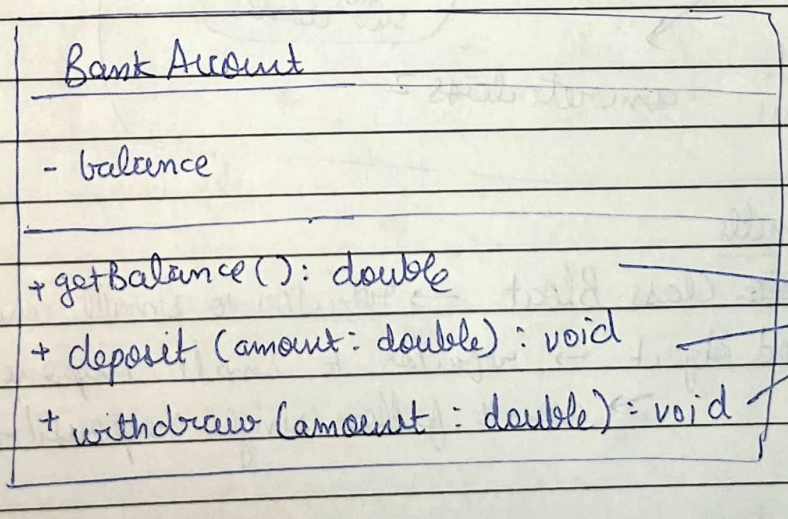
Object Oriented Programming

→ What is OOP?
- programming paradigm
- uses objects & classes to structure code
- modular, reusable & easier to maintain
- 4 core principles → Encapsulation, Abstraction, Inheritance, and Polymorphism

→ Encapsulation
- bundling of data & related methods
- protect internal state from unwanted external changes.
- provides controlled way to access & modify object's state (ACCESS MODIFIERS, GETTER; SETTER)
- promotes data integrity & security

→ • Private : only accessible in same class
• Protected: accessible by subclasses & within same package
• public : accessible everywhere
• default : accessible within same package.

e.g -



→ Abstraction

- hide the implementation details of a class/method
- expose only essential features
- reduces complexity and focuses on "what" an object does rather than "how" it does it?
- Java uses ABSTRACT CLASSES & INTERFACES to achieve this.

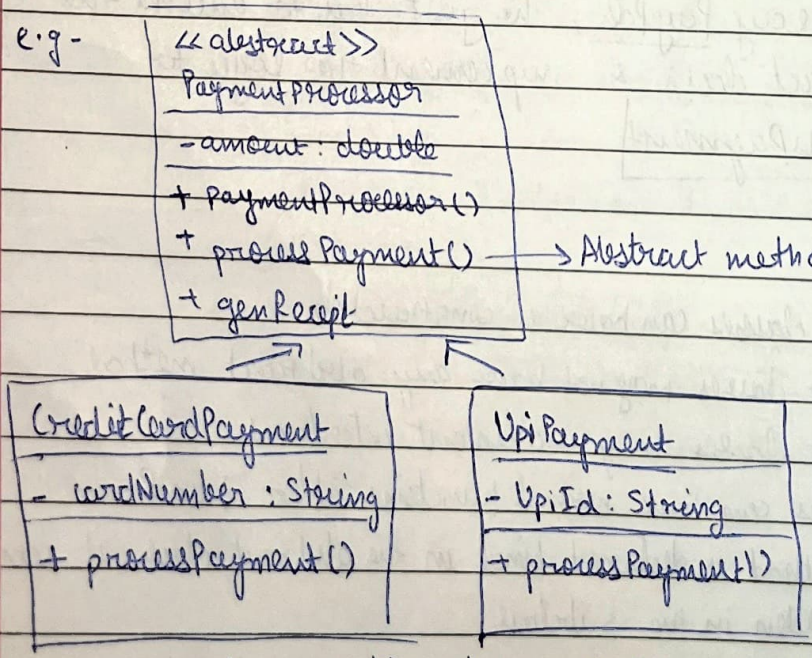
→ Abstract class

- have both abstract methods & concrete methods
- cannot be instantiated
- abstract methods have to be implemented by subclasses
- concrete methods inherited or overridden

→ Interface

- defines contract of methods that a class must implement.
- cannot have any method implementations
- used to represent functionalities that will be implemented separately by its subclasses.

e.g.-



Implementation of processPayment() is defined separately depending upon each payment method.

Main method

```
PaymentProcessor p1 = new CreditCardPayment("J--", 1234-);
p1.processPayment();
p1.generateReceipt();

PaymentProcessor p2 = new UpiPayment(250, "a@upi");
p2.processPayment();
p2.generateRecept();
```

- Abstract class PaymentProcessor defines common interface
- contains abstract method **processPayment** to be implemented by its sub classes.
- concrete classes implement processPayment according to the logic independent to that concrete class.
- Main class uses PaymentProcessor to process payments dynamically ~~at~~ at runtime depending on the user's choice of payment mode.

- Now, if some dev has to add a new payment method, let us say PayPal, he just has to extend the abstract class & implement the logic to **processPayment**

→ Questions

- Abstract classes can have a constructor
- Abstract classes may not have any abstract method.
- Abstract classes may implement interfaces.
- Interfaces can have normal functions after Java 8.
- If a method is declared final in the abstract class, it cannot be overridden in the subclass.
- data members in interfaces are public final static

→ Inheritance

- one class can inherit data members/ methods from another class.
- promotes code re usability
- In JAVA, classes are allowed to inherit from one class only, but it can implement multiple interfaces.
- Keywords - extends / implements / super / @Override

→ • super refers to immediate parent class
  • In constructor, it is used to invoke superclass's constructor
  • In methods, used to invoke superclass's method.

e.g -

```
class Payment {
    double amount;
    Payment (double amount) {
        this.amount = amount;
    }
    void ProcessPayment () {
        print (Processing a payment of amount)
    }

//subclass class CcPayment extends Payment {
        CcPayment (double amount) {
            super(amount); // call the parent constructor
        }

        //override processPayment method.
        @Override
        void ProcessPayment () {
            super.processPayment (); // call parent method
            print (Processing payment via credit card)
        }
    }
```

→ Polymorphism

- enables one method to represent multiple forms.
- allows objects of different classes to be treated as objects of a common superclass.
- 2 main types : Runtime (Overriding) & Compile Time (overload)

→ Method Overloading (compile time)

- multiple methods - same name - different parameter list
- decided at compile time based on fun signature.

e.g. ArrayList list = new ArrayList <> (1,2,3,4);
list.add(5)
list.add(1,6)
} by two implementations of add method.

→ Method overriding (runn time Polymorphism)

- a subclass provides its own implementation of a method already defined in superclass.
- the method to be called is determined at runn time, based on object's actual class, not the reference type.
- same method name & signature
- dynamic method dispatch (fun overriding)

e.g - runtime polymorphism

```
interface Payment {
    void processPayment(double amount); //common method
}

class CcPayment implements Payment {
    @Override
    public void ProcessPayment(double amount) {

    }
}

class BankTransfer implements Payment {
    @Override
    public void processPayment(double amount) {

    }
}
```

Main

```
Payment p1 = new CcPayment();
Payment p2 = new BankTransfer();

//runtime polymorphism
p1.processPayment();
p2.processPayment();
```

Q/A

- Diff b/w public / private / protected access modifiers?
- Diff b/w abstract classes, interfaces?
  - abstract → both abstract & concrete methods
    → can have fields & constructors
    → class can inherit from only 1 abstract class
  - interface → only abstract methods
    → only method signatures & final static vars.
    → class can implement multiple interfaces.
- Can abstract classes have constructors?
  - yes, called when object of subclass is created.
  if default or non-parameterized constructor is not present in abstract class, then we explicitly need to call super(..) in subclass
- Diff b/w extends & implements
- Can a class extend multiple classes in Java?
- What happens when subclass doesn't override a method in the parent class?
  - if method is not abstract, then it will be inherited.
  - if method is abstract, then subclass must override it, unless it is abstract itself.
- Diff b/w method overloading & overriding.
- What is Dynamic method dispatch in JAVA?
  - mechanism in Java, where the call to an overridden method is resolved at runtime. The JVM determines the method to call based on the actual object type (not the reference type) used during method invocation.

- Can polymorphism be achieved without inheritance?
  - yes, via interfaces

- Difference b/w static & instance methods?

static
  - declared using static keyword
  - called without creating object
  - belong to class itself & not the object
  - can't access instance methods directly

instance
  - belong to particular instance / object of class
  - can access both static & instance variables.
  - need an object to be invoked.

- What happens when you try to access a non-static method from a static context?
  - you can't directly access it, as static methods don't have access to instance data. You would need to create an instance of class to access the non-static members.

- Can you override a static method?
  - No, static methods cannot be overridden, they're bound at compile time and belong to the class itself.

- Purpose of static keyword?
  - static variables
  - static methods
  - static blocks → used for initialization of static variables that runs when the class is loaded, can also be used for one-time operations that only runs once when class is loaded.