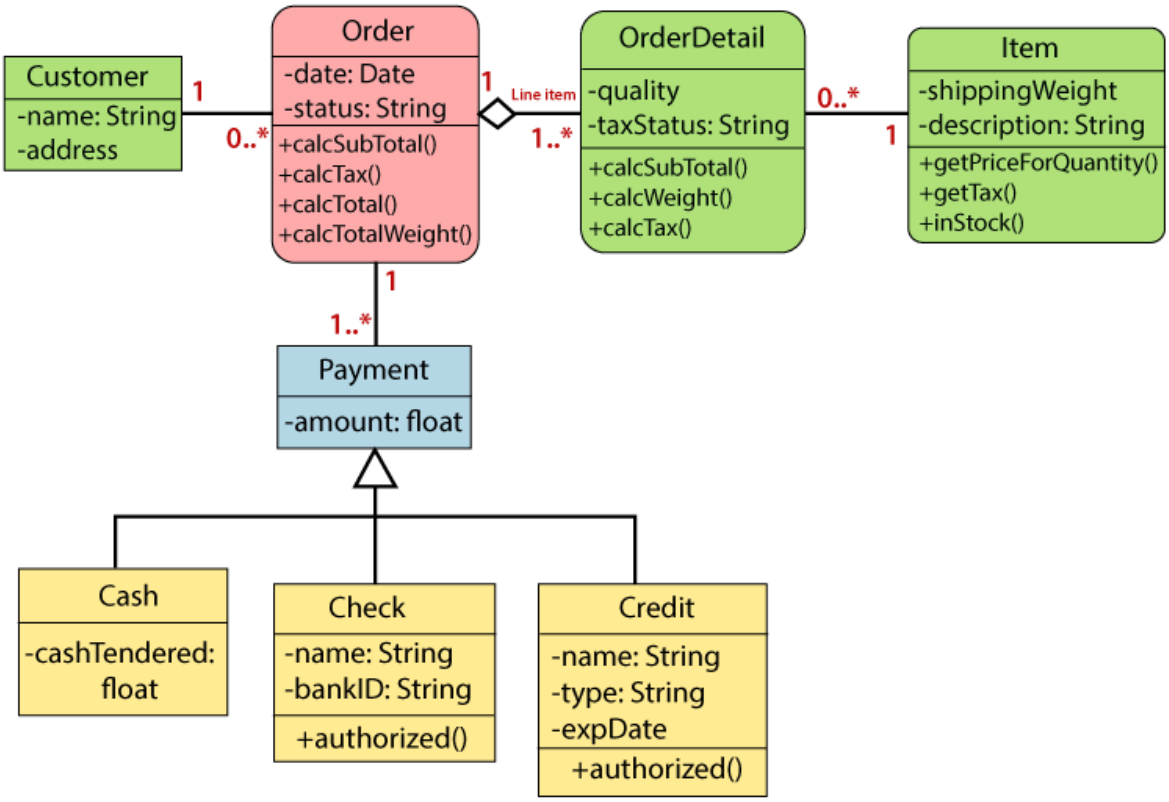
Diagram 1



Cus
an ArrayList of the **Order** class.

Order class has an **Aggregation** relationship with the **OrderDetail** class, which is also **One to Many**. So, the **Order** class will contain an ArrayList of the **OrderDetail** class.

OrderDetail class has a **Many to One Association** with the **Item** class. Each **OrderDetail** is associated with exactly one **Item**, so the **OrderDetail** class will have an **Item** reference instead of an ArrayList.

Item class exists independently, and **OrderDetail** references an **Item** instead of the other way around.

Order class has a **One to Many Association** with the **Payment** class. That's why it will contain an ArrayList of **Payment** objects.

Payment class is an **abstract parent class** of **Cash, Check, and Credit**, so it will be written accordingly using **inheritance**. The **Payment** class will also contain an **abstract method** (authorized()), which each subclass will override with its own implementation.

In the exam, you will be given the diagram only and you will need to find out the relationships on your own and have to write the whole code.

```
import java.util.*;

class Customer {
   String name;
   String address;
   List<Order> orders = new ArrayList<>();

   Customer(String name, String address) {
      this.name = name;
      this.address = address;
   }

   void addOrder(Order order) {
      orders.add(order);
   }
}

class Order {
   Date date;
   String status;
   List<OrderDetail> orderDetails = new ArrayList<>();
   List<Payment> payments = new ArrayList<>();

   Order(Date date, String status) {
      this.date = date;
      this.status = status;
   }

   void addOrderDetail(OrderDetail detail) {
      orderDetails.add(detail);
   }

   void addPayment(Payment payment) {
      payments.add(payment);
   }

   float calcSubTotal() {
      float subtotal = 0;
      for (OrderDetail detail : orderDetails) {
         subtotal += detail.calcSubTotal();
      }
      return subtotal;
```

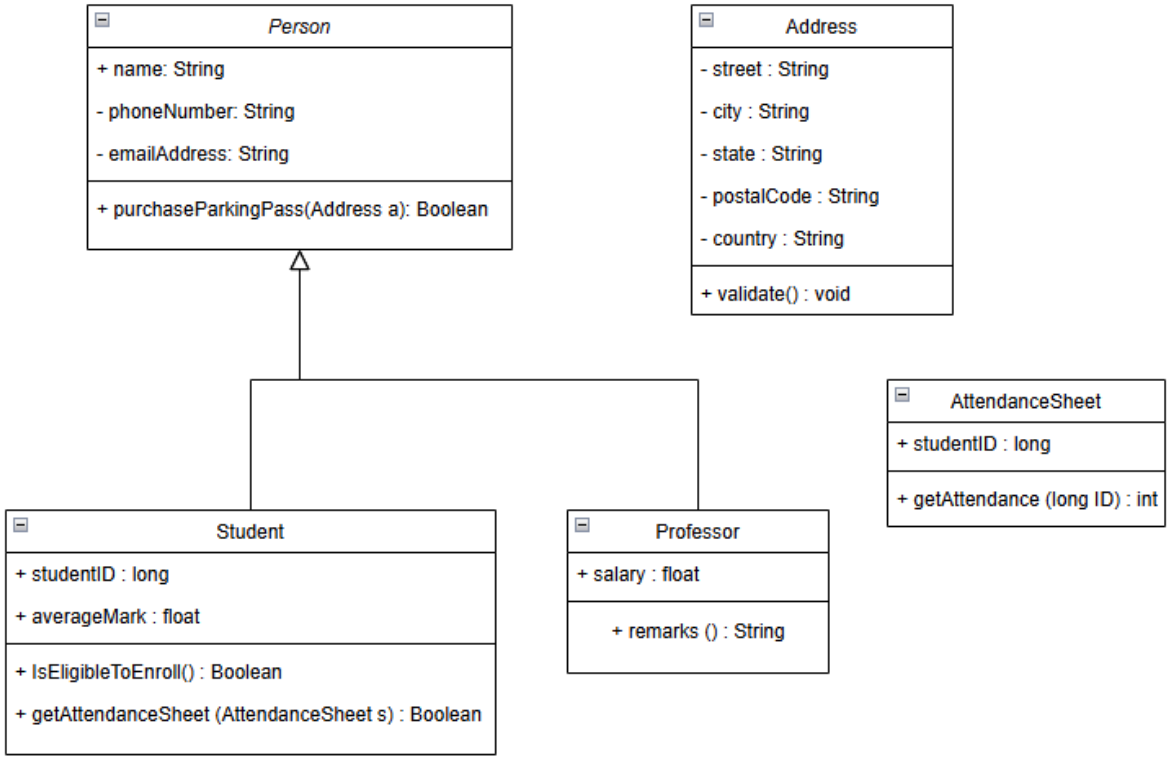
```
    }

    float calcTax() {
        float tax = 0;
        for (OrderDetail detail : orderDetails) {
            tax += detail.calcTax();
        }
        return tax;
    }

    float calcTotal() {
        return calcSubTotal() + calcTax();
    }

    float calcTotalWeight() {
        float weight = 0;
        for (OrderDetail detail : orderDetails) {
            weight += detail.calcWeight();
        }
        return weight;
    }
}

class OrderDetail {
    int quantity;
    String taxStatus;
    Item item;

    OrderDetail(int quantity, String taxStatus, Item item) {
        this.quantity = quantity;
        this.taxStatus = taxStatus;
        this.item = item;
    }

    float calcSubTotal() {
        return item.getPriceForQuantity(quantity);
    }

    float calcTax() {
        return item.getTax() * quantity;
    }
```

```
    float calcWeight() {
        return item.shippingWeight * quantity;
    }
}

class Item {
    float shippingWeight;
    String description;

    Item(float shippingWeight, String description) {
        this.shippingWeight = shippingWeight;
        this.description = description;
    }

    float getPriceForQuantity(int quantity) {
        return 10.0f * quantity;
    }

    float getTax() {
        return 2.0f;
    }

    boolean inStock() {
        return true;
    }
}

abstract class Payment {
    float amount;

    Payment(float amount) {
        this.amount = amount;
    }

    abstract boolean authorized();
}

class Cash extends Payment {
    float cashTendered;

    Cash(float amount, float cashTendered) {
        super(amount);
```

```
        this.cashTendered = cashTendered;
    }

    @Override
    boolean authorized() {
        return cashTendered >= amount;
    }
}

class Check extends Payment {
    String name;
    String bankID;

    Check(float amount, String name, String bankID) {
        super(amount);
        this.name = name;
        this.bankID = bankID;
    }

    @Override
    boolean authorized() {
        return true;
    }
}

class Credit extends Payment {
    String name;
    String type;
    String expDate;

    Credit(float amount, String name, String type, String expDate) {
        super(amount);
        this.name = name;
        this.type = type;
        this.expDate = expDate;
    }

    @Override
    boolean authorized() {
        return true;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Customer customer = new Customer("John Doe", "123 Main St");
        Order order = new Order(new Date(), "Processing");
        Item item1 = new Item(2.5f, "Laptop");
        Item item2 = new Item(1.0f, "Mouse");
        OrderDetail detail1 = new OrderDetail(1, "Taxable", item1);
        OrderDetail detail2 = new OrderDetail(2, "Taxable", item2);
        order.addOrderDetail(detail1);
        order.addOrderDetail(detail2);
        customer.addOrder(order);
        Payment payment = new Credit(1000.0f, "John Doe", "Visa", "12/25");
        order.addPayment(payment);
        System.out.println("Order Total: " + order.calcTotal());
    }
}
```

Diagram 2



Understanding the Class Relationships

1. **Person to Student and Professor (Inheritance - Generalization/Specialization)**
    - Person is a **superclass** with common attributes (name, phoneNumber, emailAddress).
    - Student and Professor **inherit** from Person.
2. **Person to Address (Association)**
    - A Person interacts with Address through the method purchaseParkingPass(Address a).
    - Address has attributes such as street, city, state, postalCode, and country.

3. **Student to AttendanceSheet (Association)**

    - The Student class has a method getAttendanceSheet(AttendanceSheet s), indicating a **relationship** with AttendanceSheet.

    - AttendanceSheet stores studentID and has a method getAttendance(long ID).

4. **Professor has Salary and Remarks Method**

    - The Professor class has a salary attribute and a remarks() method, indicating some **specific behavior** for professors.

```
class Person {
   String name;
   String phoneNumber;
   String emailAddress;
   Person(String name, String phoneNumber, String emailAddress) {
      this.name = name;
      this.phoneNumber = phoneNumber;
      this.emailAddress = emailAddress;
   }
   boolean purchaseParkingPass(Address a) {
      return a.validate();
   }
}
```

```
class Address {
    String street;
    String city;
    String state;
    String postalCode;
    String country;
    Address(String street, String city, String state, String postalCode, String country) {
        this.street = street;
        this.city = city;
        this.state = state;
        this.postalCode = postalCode;
        this.country = country;
    }
    boolean validate() {
        return street != null && city != null && state != null && postalCode != null && country != null;
    }
}

class Student extends Person {
    long studentID;
    float averageMark;

    Student(String name, String phoneNumber, String emailAddress, long studentID, float averageMark) {
        super(name, phoneNumber, emailAddress);
        this.studentID = studentID;
```

```
        this.averageMark = averageMark;
    }
    boolean isEligibleToEnroll() {
        return averageMark >= 50.0;
    }
    boolean getAttendanceSheet(AttendanceSheet s) {
        return s.getAttendance(studentID) > 75;
    }
}

class Professor extends Person {
    float salary;
    Professor(String name, String phoneNumber, String emailAddress, float salary) {
        super(name, phoneNumber, emailAddress);
        this.salary = salary;
    }
    String remarks() {
        return "Keep up the good work!";
    }
}

class AttendanceSheet {
    long studentID;
    AttendanceSheet(long studentID) {
        this.studentID = studentID;
```

```
    }
    int getAttendance(long ID) {
        if (this.studentID == ID) {
            return 80; // Example attendance percentage
        }
        return 0;
    }
}
public class Main {
    public static void main(String[] args) {
        Address addr = new Address("123 Street", "New York", "NY", "10001", "USA");
        Student student = new Student("John Doe", "1234567890", "john@example.com", 101, 85.5f);
        Professor professor = new Professor("Dr. Smith", "0987654321", "smith@example.com", 5000.0f);
        AttendanceSheet attendanceSheet = new AttendanceSheet(101);
        System.out.println("Student Eligible to Enroll: " + student.isEligibleToEnroll());
        System.out.println("Student Attendance Status: " + student.getAttendanceSheet(attendanceSheet));
        System.out.println("Professor Remarks: " + professor.remarks());
        System.out.println("Parking Pass Purchased: " + student.purchaseParkingPass(addr));
    }
}
```

Things to keep in Mind

- Address class and AttendanceSheet class is not showing their relationships with a association line, but you need to find out the relationships from other classes where the methods have the parameters where you can see the class types.
- In exam, you can get a clear diagram like the diagram 1, or you can get a diagram like diagram 2 and you need to find out on your own about it.