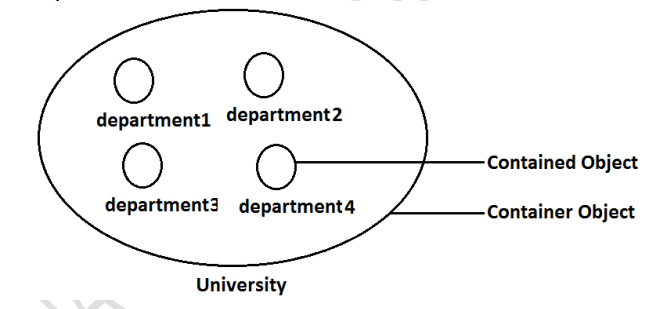
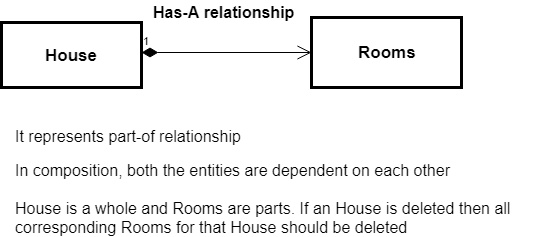
**Composition**is an association that represents a part of a whole relationship where a part cannot exist without a whole.

**Example:** University consists of several departments whenever university object destroys automatically all the department objects will be destroyed is without existing university object there is no chance of existing dependent object hence these are strongly associated and this relationship is called **composition**.

****

**2. Intent/Definition**

**Composition**is an association that represents a part of a whole relationship where a part cannot exist without a whole. If a whole is deleted then all parts are deleted. It has a stronger relationship.

****

**Key Points**

* It represents a part-of-relationship.
* In composition, both entities are dependent on each other.
* When there is a composition between two entities, the composed object cannot exist without the other entity. For example, if order HAS-A line-items, then an order is a whole, and line items are parts. If an order is deleted then all corresponding line items for that order should be deleted.
* Favor Composition over Inheritance.

**3. Implementation with Example**

Let's take the example of **Placing an Order**. If order *HAS-A* line-items, then an order is a whole, and line items are parts. If an order is deleted then all corresponding line items for that order should be deleted.

Let's implement this example step by step.

**Step 1:**Create a *Product* class.

class Product {

private int id;

private String name;

private String description;

public Product(int id, String name, String description) {

super();

this.id = id;

this.name = name;

this.description = description;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

@Override

public String toString() {

return "Product [id=" + id + ", name=" + name + ", description=" + description + "]";

}

}

**Step 2:**This is *LineItem* class, which *HAS-A* aggregation associated with the *Product* class.

class LineItem {

private int id;

private int quantity;

private Product p;

public LineItem(int id, int quantity, Product p) {

super();

this.id = id;

this.quantity = quantity;

this.p = p;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public Product getP() {

return p;

}

public void setP(Product p) {

this.p = p;

}

@Override

public String toString() {

return "LineItem [id=" + id + ", quantity=" + quantity + ", p=" + p + "]";

}

}

**Step 3:** This is the *Order* class, which *HAS-A* composition association with *LineItem* class. That means if you delete *Order*, then associated all *LineItem* must be deleted.

class Order {

private int id;

private String name;

private List<LineItem> lineItems;

public Order(int id, String name) {

super();

this.id = id;

this.name = name;

this.lineItems = new ArrayList<LineItem>();

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "Order [id=" + id + ", name=" + name + ", lineItems=" + lineItems + "]";

}

// Add line item to order

public void addItem(int id, int quantity, Product p) {

this.lineItems.add(new LineItem(id, quantity, p));

}

// Remove line item from order for given item id

public void removeItemById(int itemId) {

// TODO - Not implemented yet

}

}

**Step 4:** Let's write a test *CompositionDemo* class to test the above implementation:

public class CompositionDemo {

public static void main(String[] args) {

// Create Products

Product p1 = new Product(1, "Pen", "This is red pen");

Product p2 = new Product(2, "Pencil", "This is pencil");

Product p3 = new Product(3, "ColorBox", "This is color box");

// Create Order and Add Line Items

Order o = new Order(1, "ORD#1");

o.addItem(1, 2, p1); // Ordered of 2 quantity for p1 product

o.addItem(2, 1, p2); // Ordered of 1 quantity for p2 product

o.addItem(3, 5, p3); // Ordered of 5 quantity for p3 product

// Print Order detail before deleting

System.out.println("Order ---");

System.out.println(o);

// Deleting order would also delete associated LineItems

o = null;

}

}