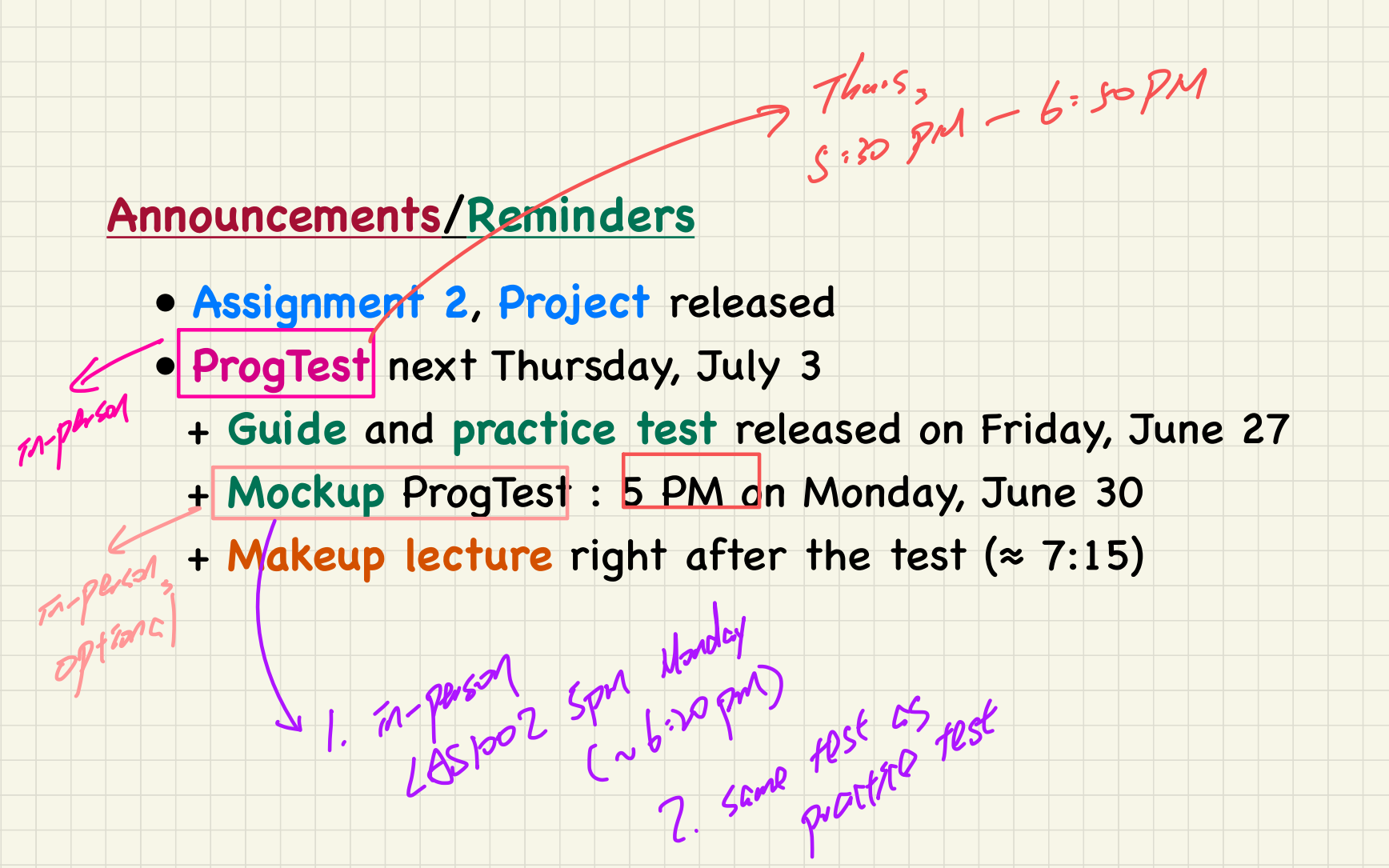
Lecture 14 - June 26

Design Patterns: Composite, Visitor

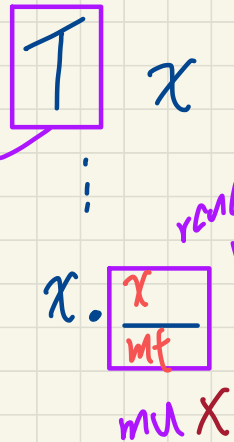
Static vs. Dynamic Types, Dynamic Binding
Design Attempts for Recursive System
Composite Design Pattern

Announcements/Reminders

- Assignment 2, Project released
- ProgTest next Thursday, July 3
  + Guide and practice test released on Friday, June 27
  + Mockup ProgTest : 5 PM on Monday, June 30
  + Makeup lecture right after the test (≈ 7:15)

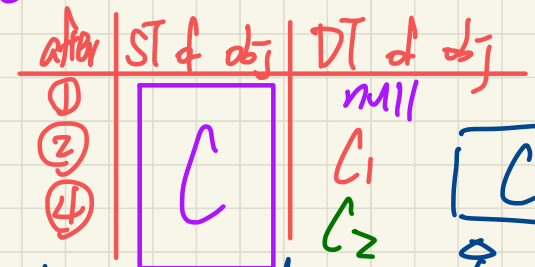


- **Static Type**: declared type

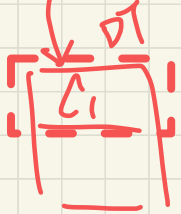


- **Dynamic Binding**: version of method invoked depends on the DT of rcv.

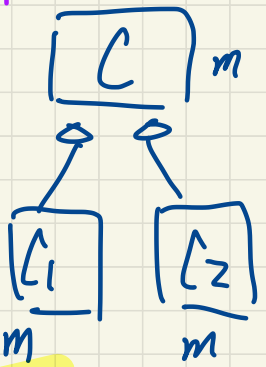
```
class T {
  int x;
  void mt() {... }
}
```

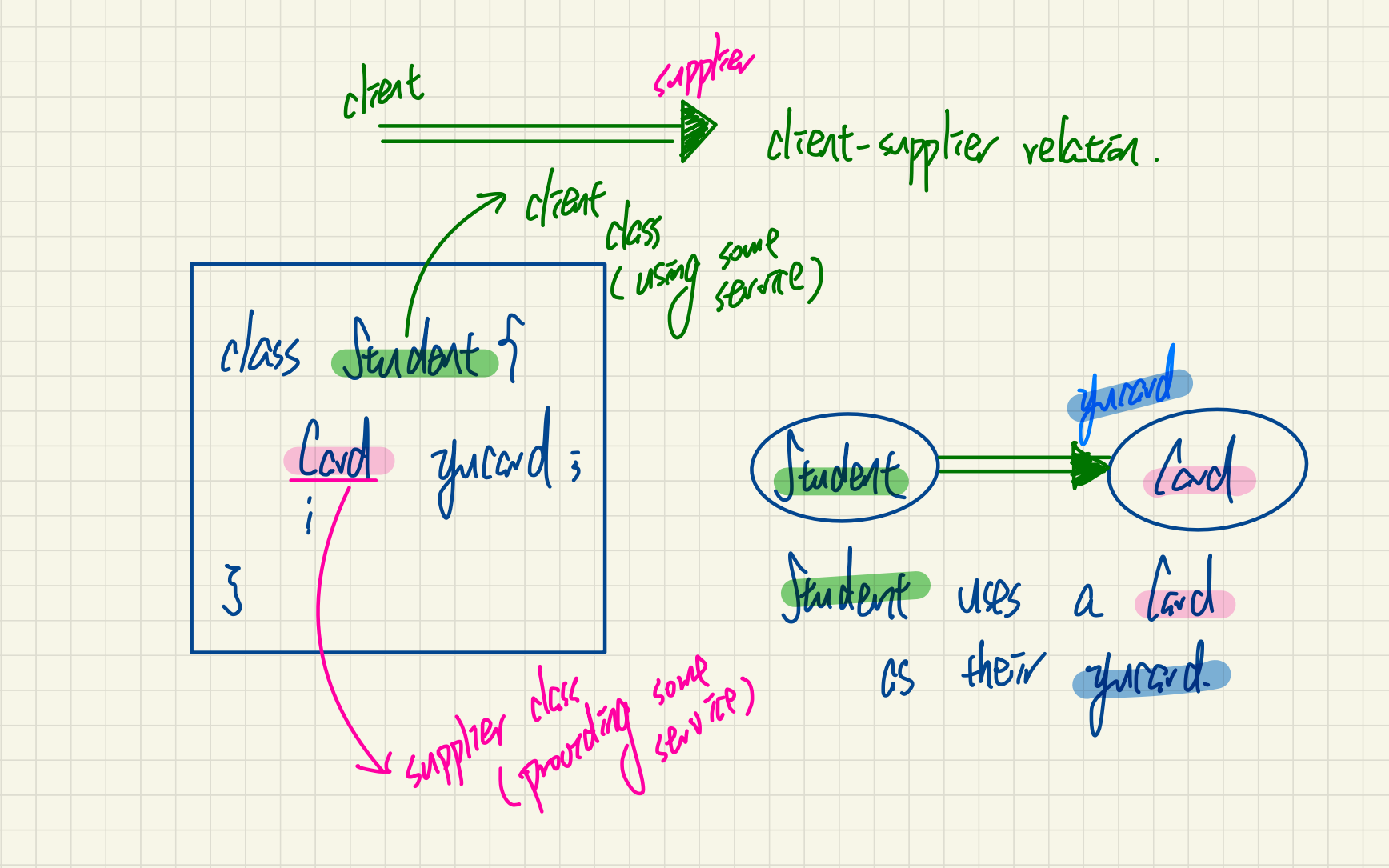


- **Dynamic Type**: type of actual object being pointed to.



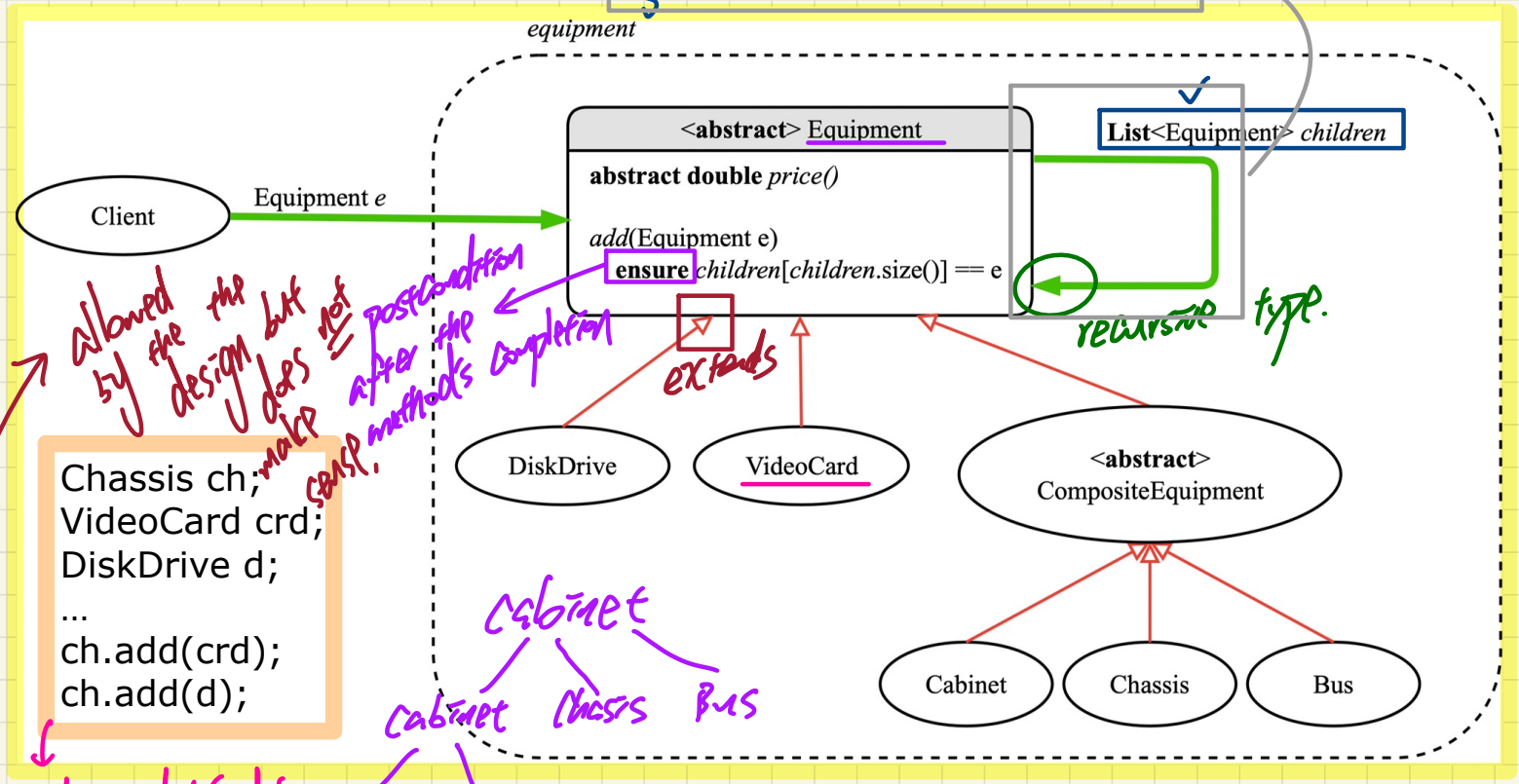

```
①  C obj ;
②  obj = new C1();
③  obj.m(); → call m from C1
④  obj = new C2();
⑤  obj.m(); → call m from C2
```



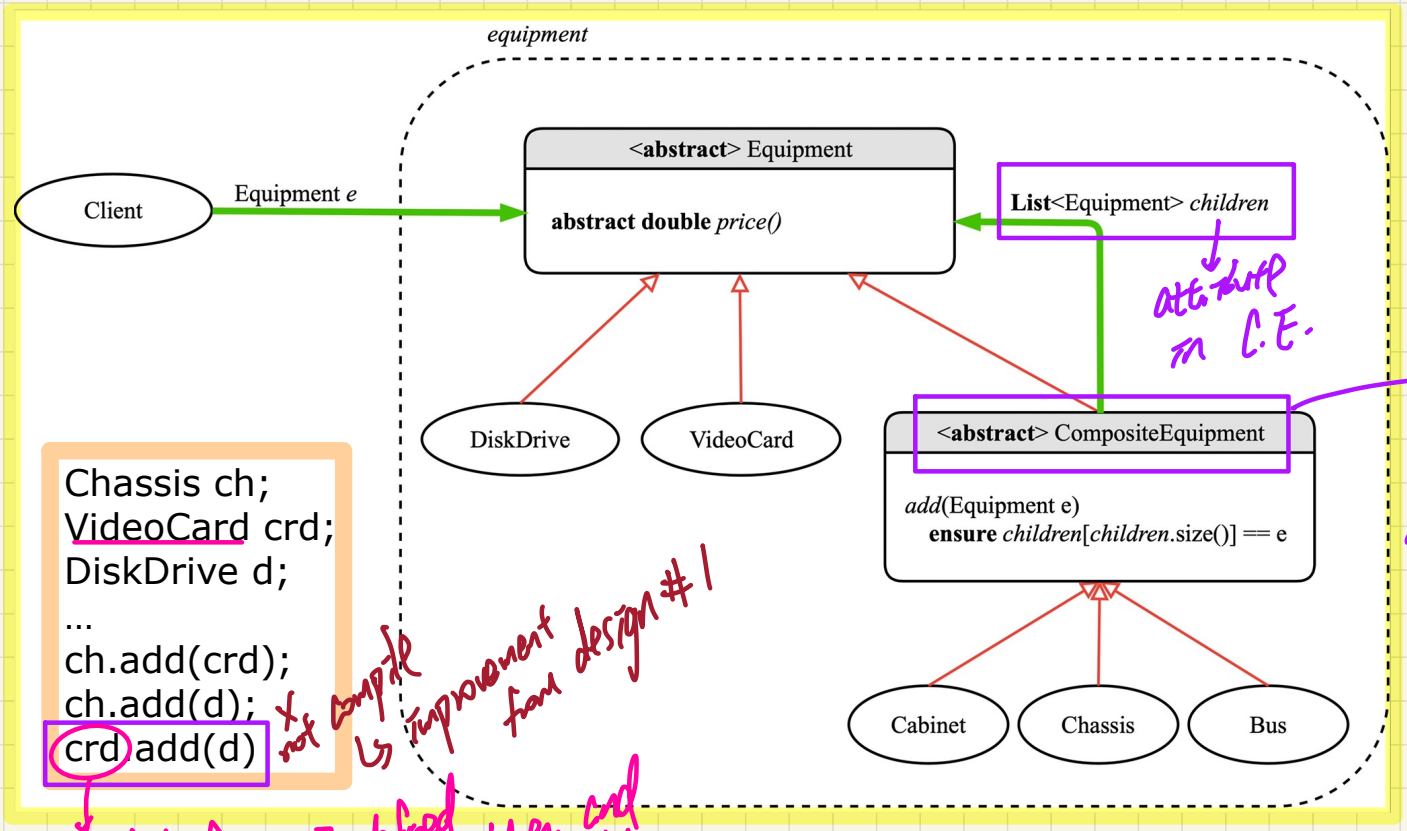

client
supplier
client-supplier relation.
client
class
(using some service)
class Student {
Card yourcard;
}
supplier class
(providing some service)
yourcard
Student
Card
Student uses a Card
as their yourcard.

First Design Attempt

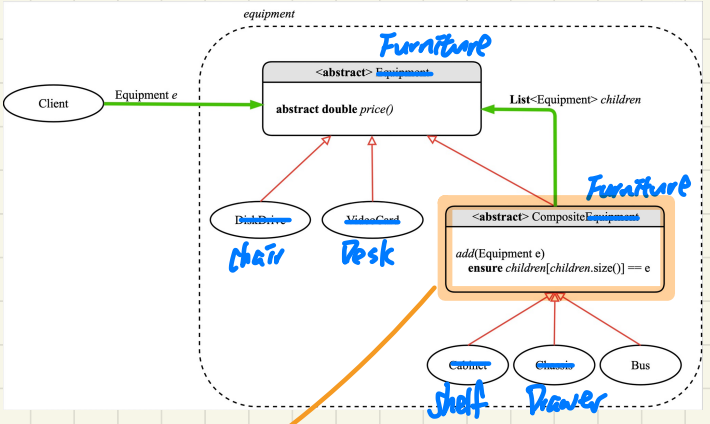
```
class Equipment {
    List<Equipment> children;
}
```

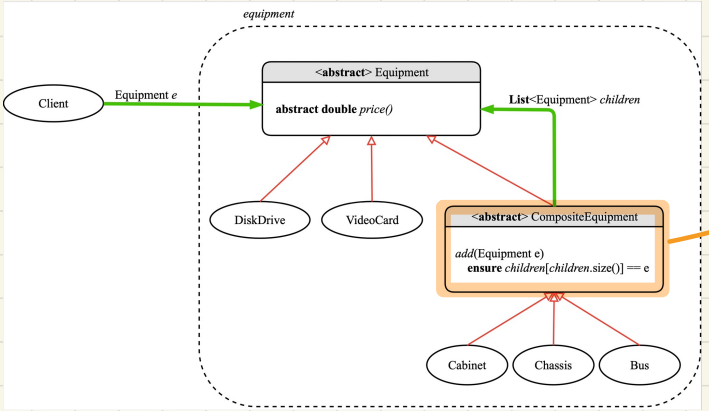


Second Design Attempt



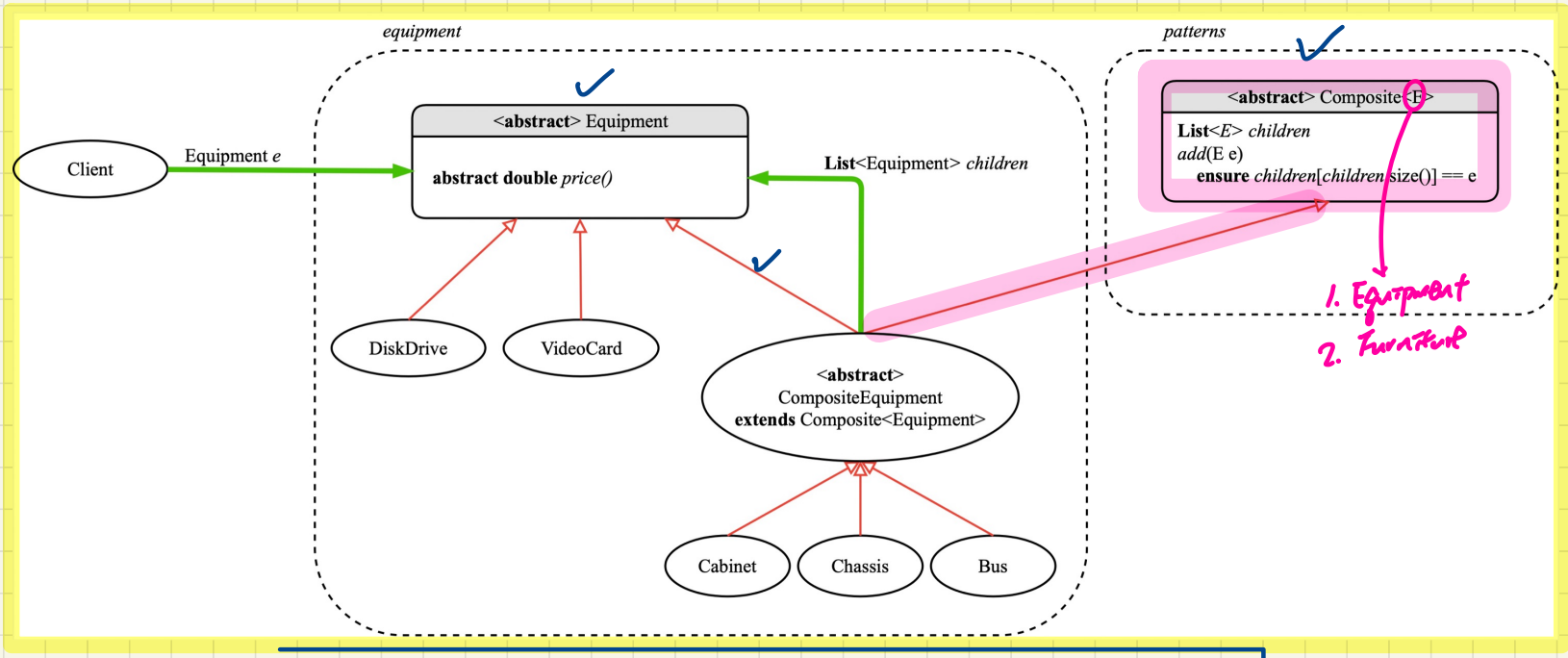
Project





Code duplicates
↳ design smells!
↳ to fix, move common code to a single place

Third Design Attempt



class CE extends Equip., Comp. {

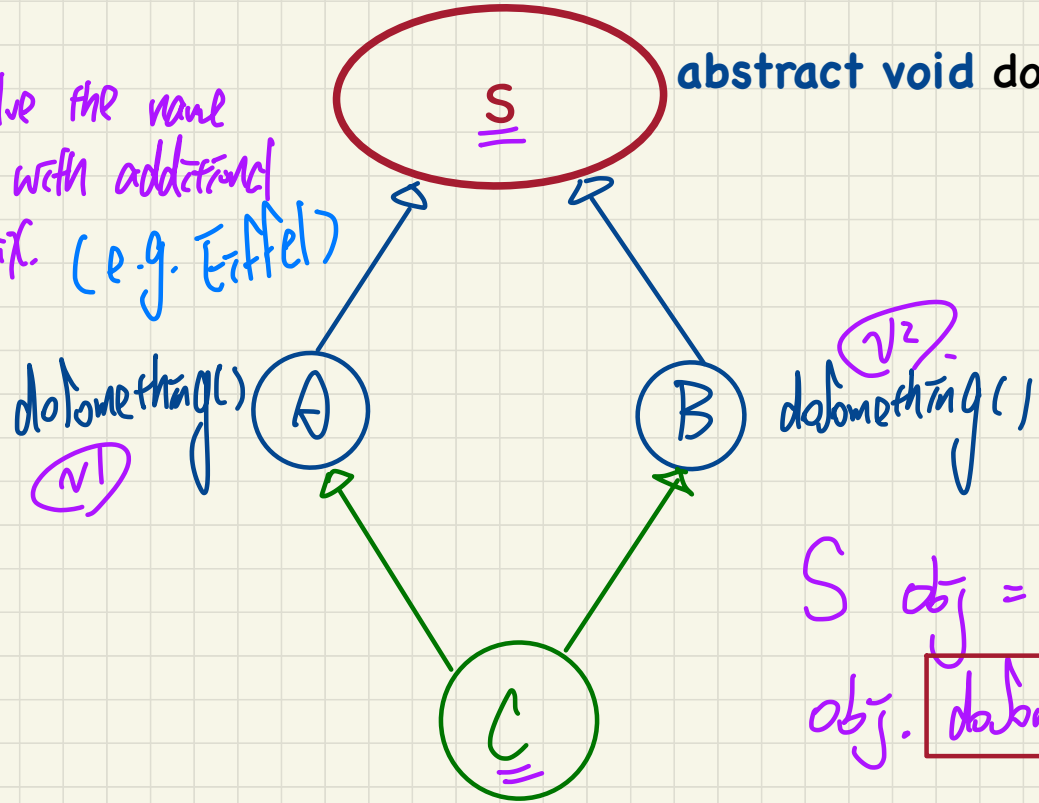
}

↳ Java does not allow extending multiple classes.

Multiple Inheritance in Java: Diamond Problem

Fix

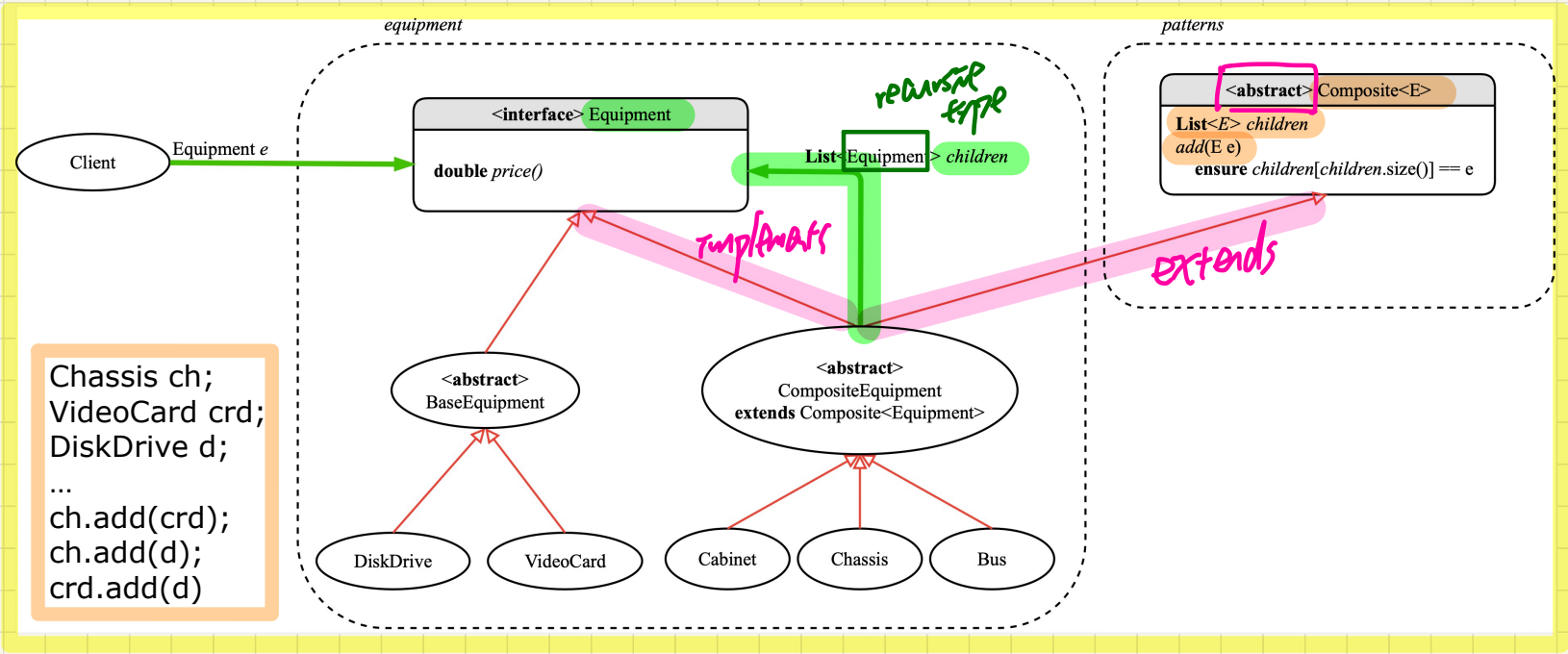
↳ resolve the name clash with additional syntax. (e.g. Eiffel)



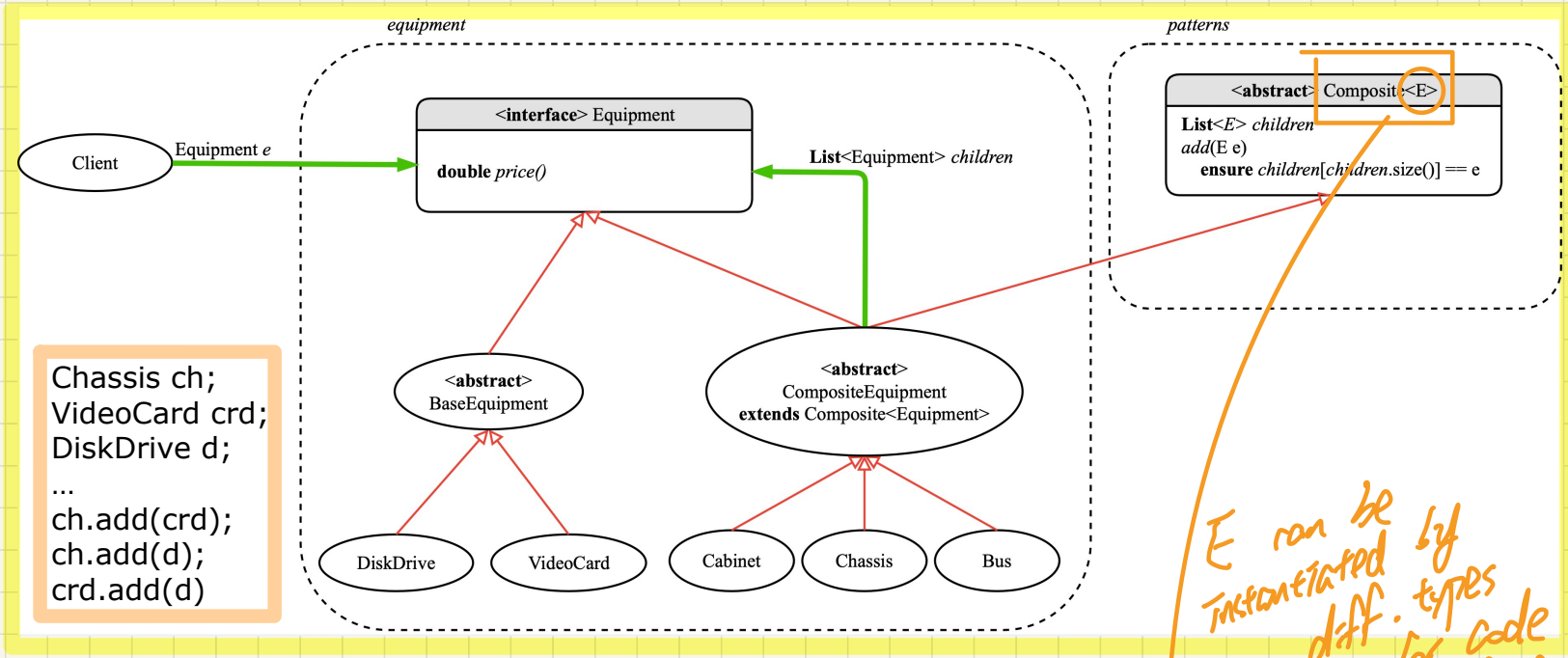

```
S obj = new C();
obj.doSomething();
```

↳ ambiguous.

Composite Pattern: Architecture



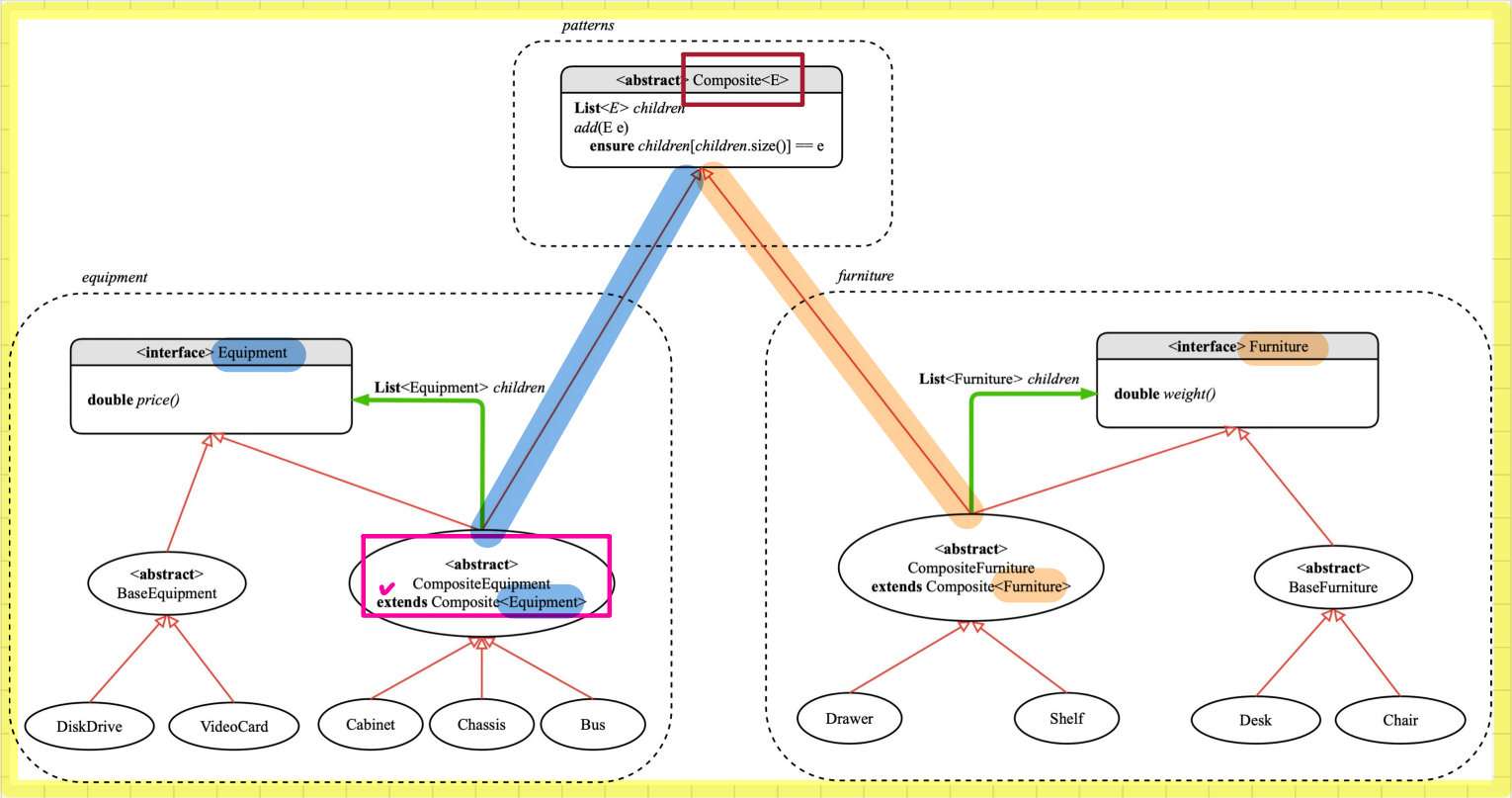
Composite Pattern: Architecture



Why is **Composite** a separate, generic class?

Composite Pattern: Architecture

Composite class is **reusable** by instances of the **composite** pattern.



Composite Pattern: Implementation

```java
public interface Equipment {
  public String name();
  public double price(); /* uniform access */
}
```

① base
② composite

```java
public abstract class Composite<E> {
  protected List<E> children;

  public void add(E child) {
    children.add(child); /* polymorphism */
  }
}
```

```java
public abstract class BaseEquipment implements Equipment {
  private String name;
  private double price;
  public BaseEquipment(String name, double price) {
    this.name = name; this.price = price;
  }
  public String name() { return this.name; }
  public double price() { return this.price; }
}
```

```java
public abstract class CompositeEquipment
  extends Composite<Equipment>
  implements Equipment
{
  private String name;
  public CompositeEquipment(String name) {
    this.name = name;
    this.children = new ArrayList<>();
  }
  public String name() { return this.name; }
  public double price() {
    double result = 0.0;
    for(Equipment child : this.children) {
      result = result + child.price(); /* dynamic binding */
    }
    return result;
  }
}
```

```java
public class VideoCard extends BaseEquipment {
  public VideoCard(String name, double price) {
    super(name, price);
  }
}
```

```java
public class Chassis extends CompositeEquipment {
  public Chassis(String name) {
    super(name);
  }
}
```

Composite Pattern: Testing

```java
@Test
public void test_equipment() {
  Equipment card, drive;
  Bus bus;
  Cabinet cabinet;
  Chassis chassis;

  card = new VideoCard("16Mbs Token Ring", 200);
  drive = new DiskDrive("500 GB harddrive", 500);
  bus = new Bus("MCA Bus");
  chassis = new Chassis("PC Chassis");
  cabinet = new Cabinet("PC Cabinet");
  bus.add(card);
  chassis.add(bus);
  chassis.add(drive);
  cabinet.add(chassis);

  assertEquals(700.00, cabinet.price(), 0.1);
}
```

building the tree

```java
public abstract class BaseEquipment implements Equipment {
  private String name;
  private double price;
  public BaseEquipment(String name, double price) {
    this.name = name; this.price = price;
  }
  public String name() { return this.name; }
  public double price() { return this.price; }
}
```

```java
public abstract class CompositeEquipment
  extends Composite<Equipment>
  implements Equipment
{
  private String name;
  public CompositeEquipment(String name) {
    this.name = name;
    this.children = new ArrayList<>();
  }
  public String name() { return this.name; }
  public double price() {
    double result = 0.0;
    for(Equipment child : this.children) {
      result = result + child.price(); /* dynamic binding */
    }
    return result;
  }
}
```