ORACLE

Inheritance & Polymorphism

ITVDN
IT VIDEO DEVELOPERS NETWORK

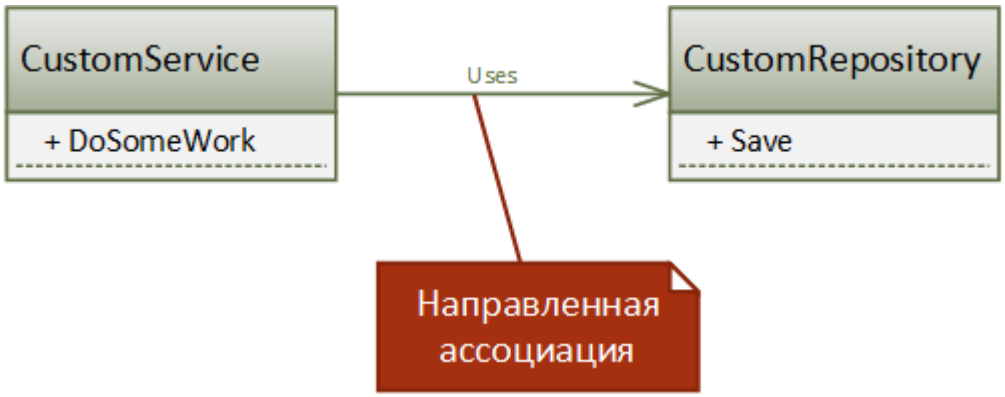
Code recycle

Возможность повторного использования кода принадлежит к числу важнейших преимуществ *Java*.

Решение этой проблемы в *Java*, как и многое другое, строится на концепции класса.

Вместо того чтобы создавать новый класс «с чистого листа», вы берете за основу уже существующий класс, который кто-то уже создал и проверил на работоспособность.

Composition • Aggregation • Inheritance



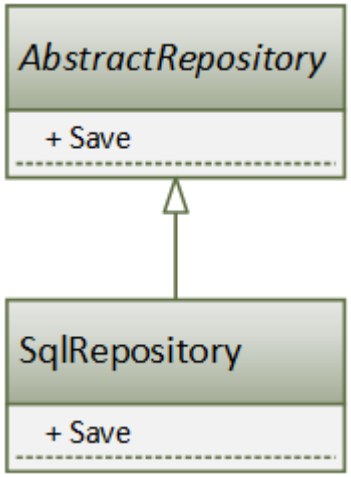
Между двумя классами/объектами существует разные типы отношений.

Самым базовым типом отношений является **ассоциация (association),** это означает, что два класса как-то связаны между собой, и мы пока не знаем точно, в чем эта связь выражена и собираемся уточнить ее в будущем.

Обычно это отношение используется на ранних этапах дизайна, чтобы показать, что зависимость между классами существует, и двигаться дальше.

Composition • Aggregation • Inheritance

Inheritance, “IS A” Relationship



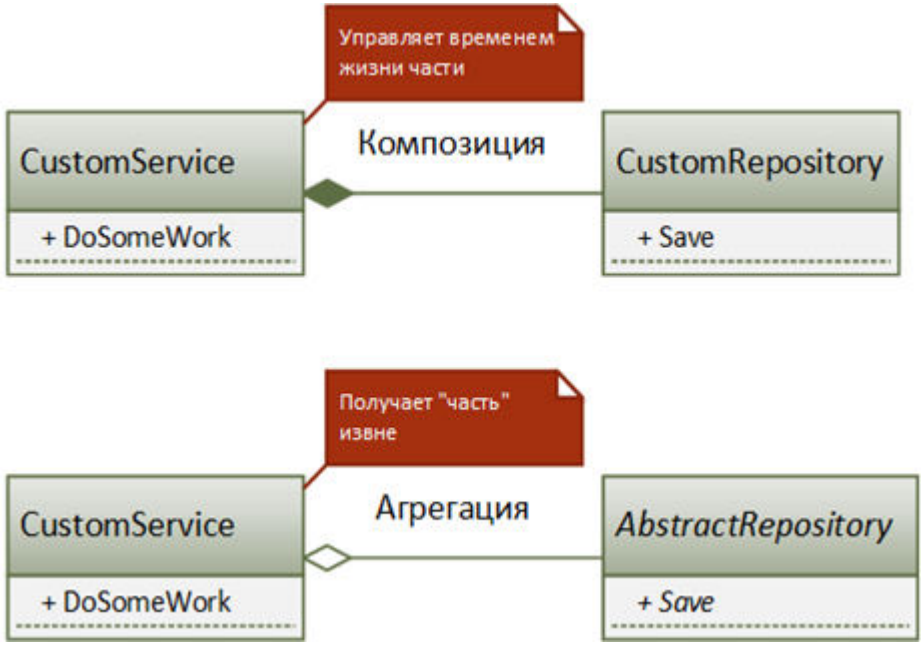
Более точным типом отношений является отношение открытого наследования (отношение «является», IS A Relationship), которое говорит, что все, что справедливо для базового класса справедливо и для его наследника.

Именно с его помощью мы получаем полиморфное поведение, абстрагируемся от конкретной реализации классов, имея дело лишь с абстракциями (интерфейсами или базовыми классами) и не обращаем внимание на детали реализации.

Наследование является самой сильной связью между двумя классами, которую невозможно разорвать во время исполнения (это отношение является статическим и, в строготипизированных языках определяется во время компиляции).

Composition • Aggregation • Inheritance

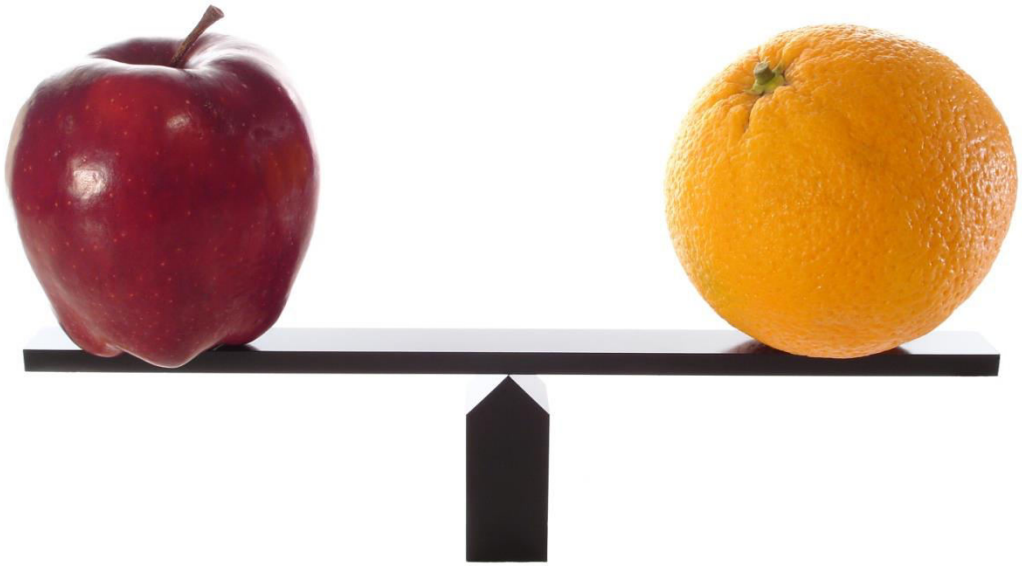
Composition & Aggregation, “HAS A” Relationship



В этом случае нам на помощь приходит другая пара отношений: **композиция** (**composition)** и **агрегация(aggregation).**

Оба они моделируют отношение «является частью» (HAS-A Relationship) и обычно выражаются в том, что класс целого содержит поля (или свойства) своих составных частей. Грань между ними достаточно тонкая, но важная, особенно в контексте управления зависимостями.

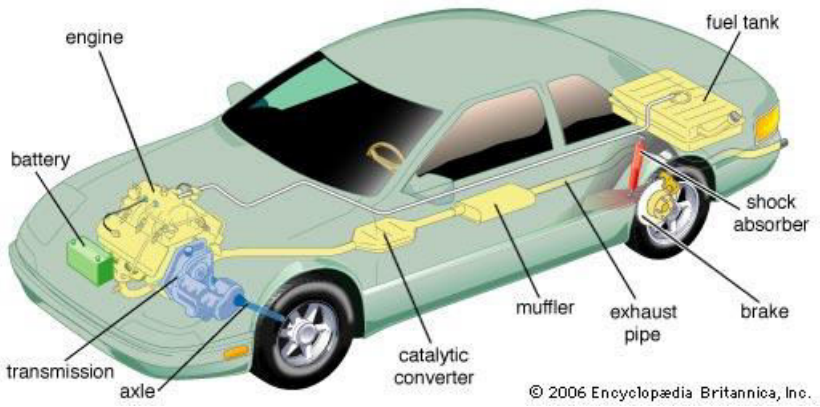
Difference between composition and aggregation



Разница между композицией и агрегацией заключается в том, что **в случае композиции целое явно контролирует время жизни своей составной части** (часть не существует без целого), а **в случае агрегации целое хоть и содержит свою составную часть, время их жизни не связано** (например, составная часть передается через параметры конструктора).

Composition • Aggregation • Inheritance

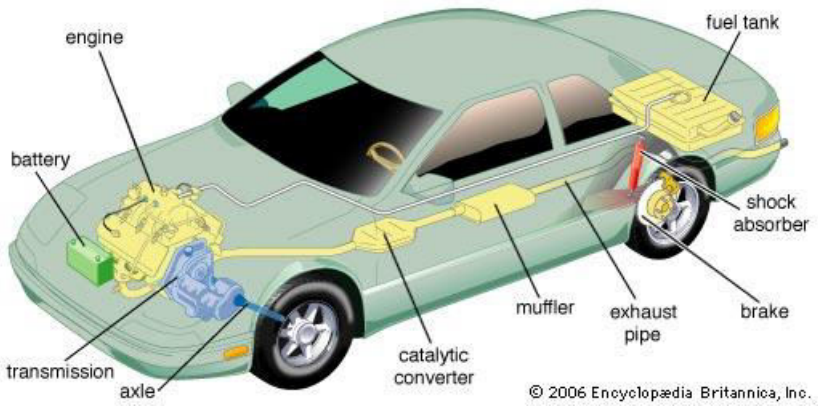
Composition code example



```
class Car {
   private String model;
   private String color;
   private Engine engine = new Engine();

   // constrcutor, getters, setters…

   public void run() {
      engine.on();
   }
}

class Engine {
   private String model;
   private String color;

   // constrcutor, getters, setters…

   public void on() {
      sout("Br…");
   }
}
```

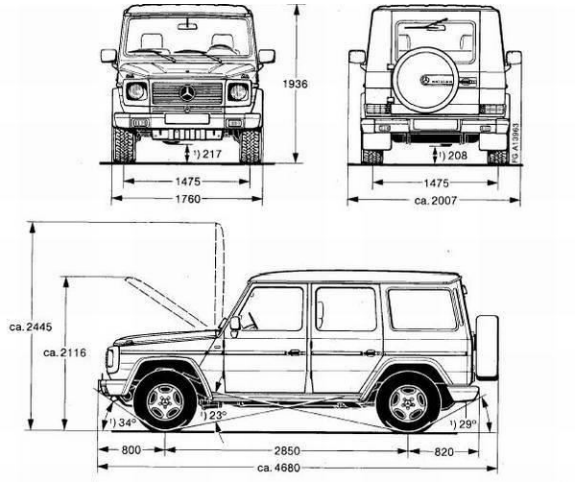
Composition • Aggregation • Inheritance

Aggregation code example




```
class Car {
   private  String  model;
   private  String  color;
   private  Freshener freshener ;

   public Car(Freshener freshener ) {
      this. freshener = freshener;
   }

   //  getters, setters…
}

class Freshener  {
   private  String  model;
   private  String  smell;

   // constrcutor, getters, setters…

}
```

Inheritance code example




```
class Car {
   private  String  model;
   private  String  color;

   // constrcutor, getters, setters…
}

class MercedesBenz  extends Car  {
   private  String  specialField;

   // constrcutor, getters, setters…

}
```

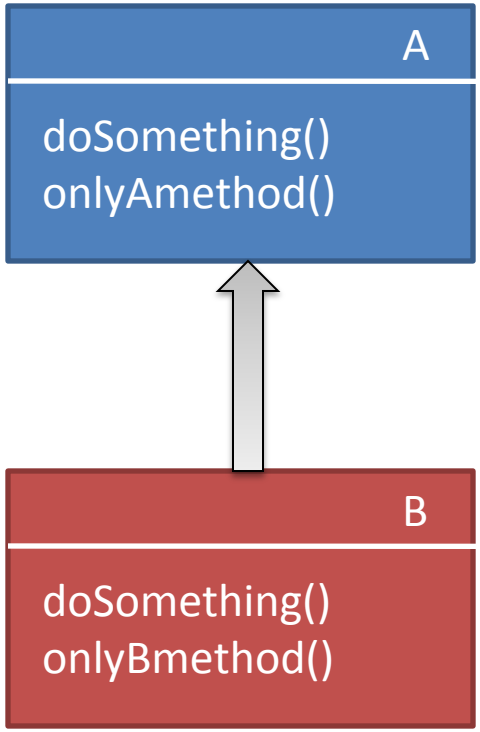
Polymorphism

Полиморфизм – возможность объектов с одинаковой спецификацией иметь различную реализацию.

Полиморфизм – это один интерфейс со множеством реализаций.

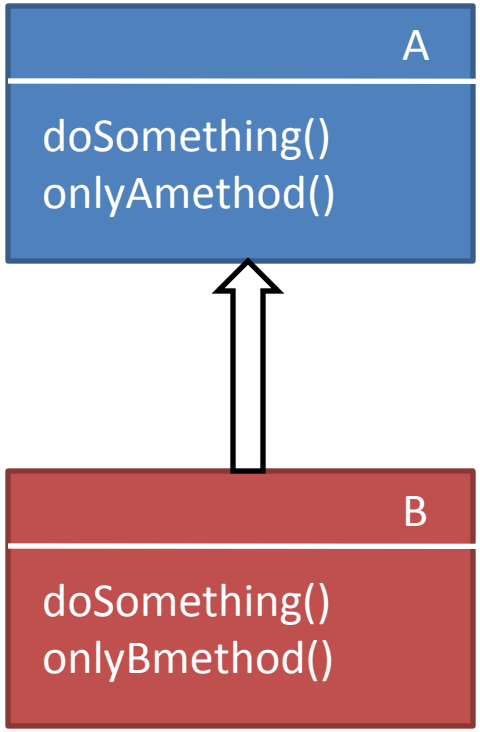
Полиморфизм – принцип ООПб с помощью которого конкретный объект определяется на этапе выполнения программы(runtime).

Overwrite



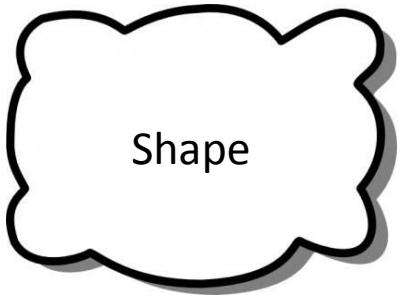

```
A a1 = new A();
A a2 = new B();
B b1 = new B();
```

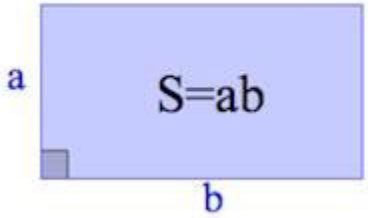
Early(static) and late(dynamic) binding

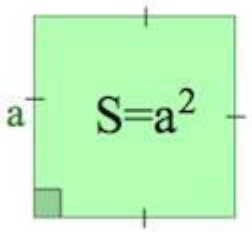


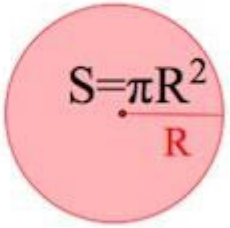

```
A a1 = new A();
a1.doSomething();

A a2 = new B();
a2.doSomething();

B b1 = new B();
b1.doSomething();
```

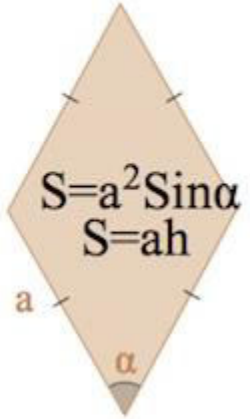
Task

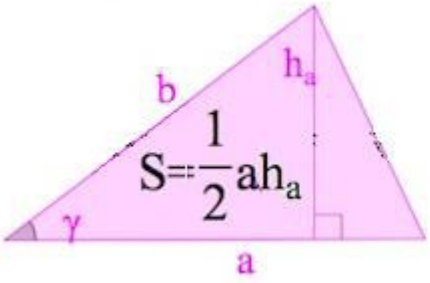

Shape

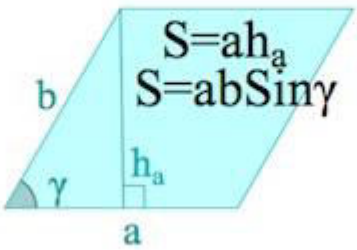

a
S=ab
b

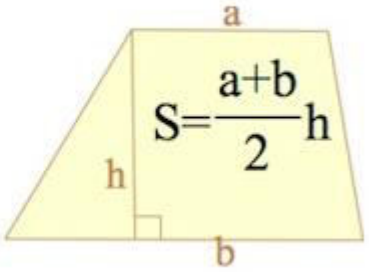

a
S=a²


S=πR²
R


S=a²Sinα
S=ah
a
α


b
h_a
S=½ah_a
γ
a


S=ah_a
S=abSinγ
b
h_a
γ
a


a
S=(a+b)/2 h
h
b


ITVDN
IT VIDEO DEVELOPERS NETWORK