Difference between Inheritance and Composition

Asked 15 years ago Modified 1 year, 11 months ago Viewed 269k times

252

Are Composition and Inheritance the same? If I want to implement the composition pattern, how can I do that in Java?

java oop inheritance composition

Share Improve this question Follow

edited Jun 28, 2010 at 15:38
Péter Török
116k ●31 ●277 ●331

asked Mar 8, 2010 at 5:54
gmhk
16k ●29 ●91 ●112

2 Other related question: Is there anything composition cannot accomplish that inheritance can? stackoverflow.com/questions/2238642/… – ewernli Mar 8, 2010 at 8:32

1 Also see is-a-vs-has-a-which-one-is-better – nawfal Oct 3, 2013 at 11:09

2 This article was useful to me and so helped: thoughtworks.com/insights/blog/… – QMaster Nov 23, 2017 at 10:40

check out themightyprogrammer.dev/article/inheritance-composition – The Mighty Programmer Mar 1, 2020 at 4:22

17 Answers

Sorted by: Highest score (default)

381

They are absolutely different. Inheritance is an *"is-a"* relationship. Composition is a *"has-a"*.

You do composition by having an instance of another class `C` as a field of your class, instead of extending `C`. A good example where composition would've been a lot better than inheritance is `java.util.Stack`, which currently extends `java.util.Vector`. This is now considered a blunder. A stack *"is-NOT-a"* vector; you should not be allowed to insert and remove elements arbitrarily. It should've been composition instead.

Unfortunately it's too late to rectify this design mistake, since changing the inheritance hierarchy now would break compatibility with existing code. *Had `Stack` used composition instead of inheritance, it can always be modified to use another data structure without violating the API*.

I highly recommend Josh Bloch's book *Effective Java 2nd Edition*

- Item 16: Favor composition over inheritance
- Item 17: Design and document for inheritance or else prohibit it

Good object-oriented design is not about liberally extending existing classes. Your first instinct should be to compose instead.

See also:

- Composition versus Inheritance: A Comparative Look at Two Fundamental Ways to Relate Classes

Share Improve this answer Follow

edited Mar 8, 2010 at 6:22

answered Mar 8, 2010 at 5:57

polygenelubricants
384k ● 129 ● 568 ● 625

5 Interesting. Why not just create a new class java.util.Stack2 that uses composition? – ged Nov 7, 2014 at 13:37

7 I appreciate this answer; however, I feel as if the answer gets off track and delves more into concerns surrounding the design of the language (and a particular package) than it does answer the asked question regarding composition vs. inheritance. I am a big fan of answering the question on SO, citing resources - not linking to external resources without providing a more in depth summary than a one-line summation. – Thomas Jan 23, 2015 at 14:09

6 Bad example, I had to do extra search to understand what Vector and Stack are. – Sam Ramezanli Mar 25, 2017 at 16:30

1 Good but your example about java stack is not a good candidate because of this inheritance is a sample of bad decision about choosing inheritance over composition as said in this article: thoughtworks.com/insights/blog/… – QMaster Nov 23, 2017 at 10:38

But the GoF design patterns use inheritance a lot. Why is that ? and the code where inheritance is used is more readable and understandable than composition. Everywhere I go I see articles favoring composition over inheritance however, I am yet to see composition is action where it should be actually used. – codingbruh Apr 15, 2023 at 16:34

256

Composition means `HAS A`
Inheritance means `IS A`

`Example`: Car **has a** Engine and Car **is a** Automobile

In programming this is represented as:

```
class Engine {} // The Engine class.

class Automobile {} // Automobile class which is parent to Car class.

class Car extends Automobile { // Car is an Automobile, so Car class extends Automobile class.
  private Engine engine; // Car has an Engine so, Car class has an instance of Engine class as its
member.
}
```

Share Improve this answer Follow

edited Jul 30, 2017 at 22:13
Andrea Bergonzo
5,210 ● 4 ● 25 ● 32

answered Mar 8, 2010 at 6:02
codaddict
456k ● 83 ● 499 ● 536

9 I'd change "automobile" to "powered vehicle" as in many interpratations cars and automobiles are equivalent. – nanofarad Nov 24, 2013 at 22:02

7 @hexafraction I agree that Vehicle would likely be a better choice, but at the same time -codaddict- has illustrated the point just fine for what has been asked. – nckbrz Mar 27, 2014 at 17:46

6 "an Engine" `:-/` – Omar Tariq Dec 14, 2016 at 10:32

nicely expained. +1. further reading javarevisited.blogspot.in/2015/06/… – roottraveller Jul 11, 2017 at 14:55

@AndreyAkhmetov Automobile can have `type` field of Type `Enum` – Ojonugwa Jude Ochalifu Jul 1, 2018 at 10:43

60

How inheritance can be dangerous ?

Lets take an example

```
public class X{
   public void do(){
   }
}
Public Class Y extends X{
   public void work(){
       do();
   }
}
```

1) As clear in above code , Class Y has very strong coupling with class X. If anything changes in superclass X , Y may break dramatically. Suppose In future class X implements a method work with below signature

```
public int work(){
}
```

Change is done in class X but it will make class Y uncompilable. SO this kind of dependency can go up to any level and it can be very dangerous. Every time superclass might not have full visibility to code inside all its subclasses and subclass may be keep noticing what is happening in superclass all the time. So we need to avoid this strong and unnecessary coupling.

How does composition solves this issue?

Lets see by revising the same example

```
public class X{
    public void do(){
    }
}

Public Class Y{
    X x = new X();
    public void work(){
        x.do();
    }
}
```

Here we are creating reference of X class in Y class and invoking method of X class by creating an instance of X class. Now all that strong coupling is gone. Superclass and subclass are highly independent of each other now. Classes can freely make changes which were dangerous in inheritance situation.

2) Second very good advantage of composition in that It provides method calling flexibility, for example :

```
class X implements R
{}
class Y implements R
{}

public class Test{
    R r;
}
```

In Test class using r reference I can invoke methods of X class as well as Y class. This flexibility was never there in inheritance

3) Another great advantage : Unit testing

```
public class X {
    public void do(){
    }
}

Public Class Y {
    X x = new X();
    public void work(){
        x.do();
    }
}
```

In above example, if state of x instance is not known, it can easily be mocked up by using some test data and all methods can be easily tested. This was not possible at all in inheritance as you were heavily dependent on superclass to get the state of instance and execute any method.

4) Another good reason why we should avoid inheritance is that Java does not support multiple inheritance.

Lets take an example to understand this :

```
Public class Transaction {
    Banking b;
    public static void main(String a[])
    {
        b = new Deposit();
        if(b.deposit()){
            b = new Credit();
            c.credit();
        }
    }
}
```

Good to know :

1. composition is easily achieved at runtime while inheritance provides its features at compile time
2. composition is also know as HAS-A relation and inheritance is also known as IS-A relation

So make it a habit of always preferring composition over inheritance for various above reasons.

Share Improve this answer Follow

edited Jan 3, 2019 at 10:57
Samix
79 ● 10

answered Jul 29, 2013 at 5:24
Manoj Kumar Saini
609 ● 5 ● 2

7 agree but given your solution using composition....we still need do a babysit for example now super class X change the method name from do to doing....then sub class Y will also need to be maintained (needs to be changed as well) this is still tight coupling? And how do we get rid of it? – stuckedunderflow Nov 4, 2018 at 16:56

1 @stuckedoverflow Changing contract either by name or signature is anti-pattern. New extensions are, however, are acceptable. (O in SOLID) – Hrishabh Gupta Jun 3, 2021 at 13:01

22

The answer given by @Michael Rodrigues is not correct (I apologize; I'm not able to comment directly), and could lead to some confusion.

Interface implementation is a form of inheritance... when you implement an interface, you're not only inheriting all the constants, you are committing your object to be of the type specified by the interface; it's still an "**is-a**" relationship. If a car implements *Fillable*, the car "**is-a**" *Fillable*, and can be used in your code wherever you would use a *Fillable*.

Composition is fundamentally different from inheritance. *When you use composition, you are (as the other answers note) making a "**has-a**" relationship between two objects, as opposed to the "**is-a**" relationship that you make when you use inheritance*.

So, from the car examples in the other questions, if I wanted to say that a car "**has-a**" gas tank, I would use composition, as follows:

```
public class Car {

private GasTank myCarsGasTank;

}
```

Hopefully that clears up any misunderstanding.

Share Improve this answer Follow

edited Sep 26, 2016 at 16:00
Ravindra babu
39k ● 11 ● 256 ● 220

answered Jun 28, 2010 at 16:05
Kris
566 ● 3 ● 12

This answer elides the issue by moving up a level. GasTank rather than the car is the class that would either extend a base class or implement IFillable, and this doesn't answer which one GasTank should do. – 11 Equals Dec 28, 2023 at 10:14

21

Inheritance brings out **IS-A** relation. **Composition** brings out **HAS-A relation**. Strategy pattern explain that Composition should be used in cases where there are families of algorithms defining a particular behaviour. Classic example being of a duck class which implements a flying behaviour.

```
public interface Flyable{
 public void fly();
}

public class Duck {
 Flyable fly;

 public Duck(){
  fly = new BackwardFlying();
 }
}
```

Thus we can have multiple classes which implement flying eg:

```
public class BackwardFlying implements Flyable{
  public void fly(){
    Systemout.println("Flies backward ");
  }
}
public class FastFlying implements Flyable{
  public void fly(){
    Systemout.println("Flies 100 miles/sec");
  }
}
```

Had it been for inheritance, we would have two different classes of birds which implement the fly function over and over again. So inheritance and composition are completely different.

Share Improve this answer Follow

edited Jan 3, 2019 at 15:01
Samix
79 ● 10

answered Jun 29, 2010 at 5:38
frictionlesspulley
12.4k ● 15 ● 72 ● 122

What are the cons of Duck implementing BackwardFlying or BackwardFlyer? – Hrishabh Gupta Jun 3, 2021 at 12:55

8

Composition is just as it sounds - you create an object by plugging in parts.

EDIT the rest of this answer is erroneously based on the following premise.
This is accomplished with Interfaces.
For example, using the `Car` example above,

```
Car implements iDrivable, iUsesFuel, iProtectsOccupants
Motorbike implements iDrivable, iUsesFuel, iShortcutThroughTraffic
House implements iProtectsOccupants
Generator implements iUsesFuel
```

So with a few standard theoretical components you can build up your object. It's then your job to fill in how a `House` protects its occupants, and how a `Car` protects its occupants.

Inheritance is like the other way around. You start off with a complete (or semi-complete) object and you replace or Override the various bits you want to change.

For example, `MotorVehicle` may come with a `Fuelable` method and `Drive` method. You may leave the Fuel method as it is because it's the same to fill up a motorbike and a car, but you may override the `Drive` method because the Motorbike drives very differently to a `Car`.

With inheritance, some classes are completely implemented already, and others have methods that you are forced to override. With Composition nothing's given to you. (but you can Implement the interfaces by calling methods in other classes if you happen to have something laying around).

Composition is seen as more flexible, because if you have a method such as iUsesFuel, you can have a method somewhere else (another class, another project) that just worries about dealing with objects that can be fueled, regardless of whether it's a car, boat, stove, barbecue, etc. Interfaces mandate that classes that say they implement that interface actually have the methods that that interface is all about. For example,

```
iFuelable Interface:
   void AddSomeFuel()
   void UseSomeFuel()
   int  percentageFull()
```

then you can have a method somewhere else

```
private void FillHerUp(iFuelable : objectToFill) {

   Do while (objectToFill.percentageFull() <= 100)  {

        objectToFill.AddSomeFuel();
   }
```

Strange example, but it's shows that this method doesn't care what it's filling up, because the object implements `iUsesFuel`, it can be filled. End of story.

If you used Inheritance instead, you would need different `FillHerUp` methods to deal with `MotorVehicles` and `Barbecues`, unless you had some rather weird "ObjectThatUsesFuel" base object from which to inherit.

Share Improve this answer Follow

edited Dec 19, 2013 at 22:27
dlamblin
45.4k ● 22 ● 104 ● 144

answered Mar 8, 2010 at 6:32
Michael Rodrigues
5,137 ● 3 ● 27 ● 52

Java conventions state that class and interface names are written in `ThisCase`, not in `camelCase`. Therefore it is best to name your interfaces `IDrivable`, etc. You might not need the "I" if you regroup all your interfaces into a package correctly.
– ThePyroEagle Dec 21, 2015 at 10:15

7

Are Composition and Inheritance the same?

They are not same.

> Composition (From wikipedia): It enables a group of objects have to be treated in the same way as a single instance of an object. The intent of a composite is to "compose" objects into tree structures to ***represent part-whole hierarchies***

> Inheritance (From oracle documentation) : A class inherits fields and methods from all its superclasses, whether direct or indirect. A subclass can override methods that it inherits, or it can hide fields or methods that it inherits.

If I want to implement the composition pattern, how can I do that in Java?

Key Participants: (as per UML diagram shown in Wikipedia link)

Component:

1. Is the abstraction for all components, including composite ones
2. Declares the interface for objects in the *composition*

Leaf:

1. Represents leaf objects in the *composition*
2. Implements all *Component* methods

Composite:

1. Represents a composite Component (component having children)
2. Implements methods to manipulate children
3. Implements all Component methods, generally by delegating them to its children

Code example to understand **Composite** pattern:

```
import java.util.List;
import java.util.ArrayList;

interface Part{
    public double getPrice();
    public String getName();
}
class Engine implements Part{
    String name;
    double price;
    public Engine(String name,double price){
        this.name = name;
        this.price = price;
    }
    public double getPrice(){
        return price;
    }
    public String getName(){
        return name;
    }
}
class Trunk implements Part{
    String name;
    double price;
    public Trunk(String name,double price){
        this.name = name;
        this.price = price;
    }
    public double getPrice(){
        return price;
    }
    public String getName(){
        return name;
    }
}
class Body implements Part{
    String name;
    double price;
```

output:

```
Car name:Innova
Car parts:DiselEngine Trunk Body
Car price:37000.0
```

Explanation:

1. ***Part*** is a leaf
2. ***Car*** contains many Parts
3. Different ***Parts*** of the car have been added to Car
4. The price of ***Car*** = sum of (Price of each **Part**)

Refer to below question for Pros and Cons of Composition and Inheritance.

[Prefer composition over inheritance?]

Share Improve this answer Follow

edited Apr 25, 2023 at 9:47

answered Sep 23, 2016 at 14:53

Ravindra babu

39k ● 11 ● 256 ● 220

Does it make sense to implement parts for car class as well . Will it not be good if you use it as composition – bhalkian Mar 20, 2018 at 4:09

@bhalkian I agree--"Purchasable" would've been a better name. Also, this is a very complicated, technical answer that doesn't actually answer the questions. – 11 Equals Dec 28, 2023 at 10:27

4

as another example, consider a car class, this would be a good use of composition, a car would "have" an engine, a transmission, tires, seats, etc. It would not extend any of those classes.

Share Improve this answer Follow

answered Mar 8, 2010 at 5:58

BlackICE

8,926 ● 3 ● 56 ● 95

This isn't answering the questions: what's the difference between composition and inheritance? And how or why would you use one over the other? If you're going to pick a car as your example, then your answer should address how a car relates to a vehicle. An even better answer would include how those parts you listed relate to a vehicle, as well. – 11 Equals Dec 28, 2023 at 10:30

@11Equals thanks for pointing that out, 14 years later. Sorry but SO answer standards have changed a bit since then, most things this old are left alone for posterity, especially lower voted items. I can understand improving an answer that is highly ranked, but I don't think working on this answer is really worth anyone's time at this point. – BlackICE Jan 5, 2024 at 1:27

4

Composition is where something is made up of distinct parts and it has a strong relationship with those parts. If the main part dies so do the others, they cannot have a life of their own. A rough example is the human body. Take out the heart and all the other parts die away.

Inheritance is where you just take something that already exists and use it. There is no strong relationship. A person could inherit his fathers estate but he can do without it.

I don't know Java so I cannot provide an example but I can provide an explanation of the concepts.

Share Improve this answer Follow

answered Jan 8, 2015 at 18:11
Robert
10.4k ● 20 ● 89 ● 140

This doesn't seem to answer either of the programming questions? Specifically, should Human inherit from Animal, and should Human or Animal contain a heart and other organs? – 11 Equals Dec 28, 2023 at 10:35

3

Inheritance between two classes, where one class extends another class establishes "**IS A**" relationship.

Composition on the other end contains an instance of another class in your class establishes "***Has A***" relationship. *Composition* in java is is useful since it technically facilitates multiple inheritance.

Share Improve this answer Follow

edited Sep 24, 2016 at 18:12
Ravindra babu
39k ● 11 ● 256 ● 220

answered Dec 18, 2012 at 13:42
John Wilson
29 ● 2

3

In Simple Word Aggregation means Has A Relationship ..

Composition is a special case of aggregation. In a more specific manner, a restricted aggregation is called composition. When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition. **Example: A class contains students. A student cannot exist without a class. There exists composition between class and students.**

Why Use Aggregation

Code Reusability

When Use Aggregation

Code reuse is also best achieved by aggregation when there is no is a Relation ship

Inheritance

Inheritance is a Parent Child Relationship Inheritance Means Is A RelationShip

Inheritance in java is a mechanism in which one object acquires all the properties and behaviors of parent object.

Using inheritance in Java 1 Code Reusability. 2 Add Extra Feature in Child Class as well as Method Overriding (so runtime polymorphism can be achieved).

Share Improve this answer Follow

edited Jun 12, 2017 at 12:22

answered Jun 12, 2017 at 12:17
Keshav Gera
11.3k ● 1 ● 77 ● 56

This doesn't answer either of the questions. – 11 Equals Dec 28, 2023 at 10:37

1

Though both Inheritance and Composition provides code reusablility, main difference between Composition and Inheritance in Java is that Composition allows reuse of code without extending it but for Inheritance you must extend the class for any reuse of code or functionality. Another difference which comes from this fact is that by using Composition you can reuse code for even final class which is not extensible but Inheritance cannot reuse code in such cases. Also by using Composition you can reuse code from many classes as they are declared as just a member variable, but with Inheritance you can reuse code form just one class because in Java you can only extend one class, because multiple Inheritance is not supported in Java. You can do this in C++ though because there one class can extend more than one class. BTW, You should always *prefer Composition over Inheritance in Java*, its not just me but even **Joshua Bloch** has suggested in his book

Share Improve this answer Follow

answered Jun 3, 2015 at 10:55
Nitin Pawar
936 ● 15 ● 12

2 Why should I "perfer *Composition over Inheritance*"? They are different concepts and are used for different purposes. Frankly, I don't see how you could even go from one to the other. – Kröw May 22, 2019 at 13:00

@Kröw they're related in the structure of data and the expressiveness of code. Your confusion is a sign of how poor some of these answers are, unfortunately. Other people here used a car as an example without showing inheritance/composition. Specifically, given a Car, should it inherit from Vehicle? If so, why? Should the Car or the Vehicle contain an Engine? What about a Car that contains a Motor instead of an Engine? In most cases, inheritance leads to confusing relationships. Using composition, the car contains IEngine and accepts one in its constructor, and has little use for inheritance. – 11 Equals Dec 28, 2023 at 11:04

1 @11Equals My confusion regards this answer, not the others, and is not a result of the others. I'm confused because, knowing deeply how inheritance and composition work and can be used to express different things in code, it makes little sense to "prefer" one over the other. Inheritance lets you express relationships where one type contains the state and functionality of another, but can add more of each of its own. Composition lets one type *utilize* the functionality offered by another. The answer to each of your question would be this information applied to the use case for the application. – Kröw Dec 28, 2023 at 22:23

1 @11Equals For example, if it would be beneficial to abstract the functionality common between all Vehicles in your application to a separate type, that could then be inherited, then sure, it could be a good idea to have Car inherit from Vehicle. If not, then no. Inheritance and Composition are merely tools to be used for your benefit when and where they can be beneficial. I would not say one should "always prefer" a hammer over a drill. That is terrible advice... – Kröw Dec 28, 2023 at 22:25

1

I think this example explains clearly the differences between **inheritance** and **composition**.

In this exmple, the problem is solved using inheritance and composition. The author pays attention to the fact that ; in **inheritance**, a change in superclass might cause problems in derived class, that inherit it.

There you can also see the difference in representation when you use a UML for inheritance or composition.

http://www.javaworld.com/article/2076814/core-java/inheritance-versus-composition--which-one-should-you-choose-.html

Share Improve this answer Follow

edited Nov 7, 2016 at 16:51
Ravindra babu
39k ● 11 ● 256 ● 220

answered Mar 18, 2014 at 12:00
aunte
11 ● 2

1 Link only answers are discouraged because they go stale. Please include at least the most important relevant information from the link in your answer. – Scott Solmer Mar 18, 2014 at 12:17

Inheritances Vs Composition.

Inheritances and composition both are used to re-usability and extension of class behavior.

Inheritances mainly use in a family algorithm programming model such as IS-A relation type means similar kind of object. Example.

1. Duster is a Car
2. Safari is a Car

These are belongs to Car family.

Composition represents HAS-A relationship Type.It shows the ability of an object such as Duster has Five Gears , Safari has four Gears etc. Whenever we need to extend the ability of an existing class then use composition.**Example** we need to add one more gear in Duster object then we have to create one more gear object and compose it to the duster object.

We should not make the changes in base class until/unless all the derived classes needed those functionality.For this scenario we should use Composition.Such as

class A Derived by Class B

Class A Derived by Class C

Class A Derived by Class D.

When we add any functionality in class A then it is available to all sub classes even when Class C and D don't required those functionality.For this scenario we need to create a separate class for those functionality and compose it to the required class(here is class B).

Below is the example:

```
     // This is a base class
            public abstract class Car
               {
                  //Define prototype
                  public abstract void color();
                  public void Gear() {
                      Console.WriteLine("Car has a four Gear");
                  }
               }


      // Here is the use of inheritence
      // This Desire class have four gears.
     //  But we need to add one more gear that is Neutral gear.

     public class Desire : Car
              {
                  Neutral obj = null;
                  public Desire()
                  {
// Here we are incorporating neutral gear(It is the use of composition).
// Now this class would have five gear.

                      obj = new Neutral();
                      obj.NeutralGear();
                  }

                  public override void color()
                  {
                      Console.WriteLine("This is a white color car");
                  }

              }


        // This Safari class have four gears and it is not required the neutral
        //  gear and hence we don't need to compose here.
```

0

Composition means creating an object to a class which has relation with that particular class. Suppose Student has relation with Accounts;

An Inheritance is, this is the previous class with the extended feature. That means this new class is the Old class with some extended feature. Suppose Student is Student but All Students are Human. So there is a relationship with student and human. This is Inheritance.

Share Improve this answer Follow

answered Jun 9, 2017 at 22:31
HM Nayem
2,427 ● 1 ● 13 ● 10

0

Inheritence means reusing the complete functionality of a class, Here my class have to use all the methods of the super class and my class will be titely coupled with the super class and code will be duplicated in both the classes in case of inheritence.

But we can overcome from all these problem when we use composition to talk with another class . composition is declaring an attribute of another class into my class to which we want to talk. and what functionality we want from that class we can get by using that attribute.

Share Improve this answer Follow

answered Feb 7, 2018 at 9:21
Vikas Kapadiya
1

-1

No , Both are different . Composition follow "HAS-A" relationship and inheritance follow "IS-A" relationship . Best Example for composition was Strategic pattern .

Share Improve this answer Follow

answered Dec 24, 2017 at 17:34
Ranga Reddy
54 ● 1 ● 7

2 Its only comment quality – Mathews Sunny Dec 24, 2017 at 17:51

Did you just comment the same thing the accepted answer said 7 years before you? – Anjil Dhamala Feb 9, 2018 at 14:52

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.

Start asking to get answers

Find the answer to your question by asking.

Ask question

Explore related questions

java oop inheritance composition

See similar questions with these tags.