Why to use Polymorphism?

Asked 9 years, 3 months ago Active 2 years, 8 months ago Viewed 82k times

36

23

I have the following code in which I have a parent class and its child. I am trying to determine how the code benefits from using polymorphism.

```
class FlyingMachines {
    public void fly() {
        System.out.println("No implementation");
    }
}

class Jet extends FlyingMachines {
    public void fly() {
        System.out.println("Start, Taxi, Fly");
    }

    public void bombardment() {
        System.out.println("Throw Missile");
    }
}

public class PolymorphicTest {
    public static void main(String[] args) {
        FlyingMachines flm = new Jet();
        flm.fly();

        Jet j = new Jet();
        j.bombardment();
        j.fly();
    }
}
```

What is the advantage of polymorphism when both `flm.fly()` and `j.fly()` give me the same answer?

java polymorphism

Share Edit Follow

edited Jun 19 '12 at 3:26

matt5784

2,990 ● 2 ● 22 ● 42

asked Jun 16 '12 at 14:47

baig772

3,212 ● 11 ● 37 ● 81

1 I think he wants to know Is there any Difference Between flm.fly() and j.fly()? If yes? than What? – khan Jun 16 '12 at 15:06

There's a distinction to be made here between *compile-time* and *run-time*... The Java Compiler will see `flm.fly();` evaluate to: `System.out.println("No`

`implementation");` ... The Java Virtual Machine, at run-time will see `flm.fly();` evaluate to: `System.out.println("Start, Taxi, Fly");` ... If you changed line 20 from `flm.fly();` to, `flm.bombardment();`, you would get a compilation error, as the `bombardment()` message is not part of `FlyingMachines` protocol. – Jack_Hu Nov 20 '19 at 1:25

13 Answers

Active | Oldest | Votes

49

In your example, the use of polymorphism isn't incredibly helpful since you only have one subclass of `FlyingMachine`. Polymorphism becomes helpful if you have multiple kinds of `FlyingMachine`. Then you could have a method that accepts any kind of `FlyingMachine` and uses its `fly()` method. An example might be `testMaxAltitude(FlyingMachine)`.

Another feature that is only available with polymorphism is the ability to have a `List<FlyingMachine>` and use it to store `Jet`, `Kite`, or `VerySmallPebbles`.

One of the best cases one can make for using polymorphism is the ability to refer to interfaces rather than implementations.

For example, it's better to have a method that returns as `List<FlyingMachine>` rather than an `ArrayList<FlyingMachine>`. That way, I can change my implementation within the method to a `LinkedList` or a `Stack` without breaking any code that uses my method.

Share Edit Follow

edited Jun 16 '12 at 15:07

answered Jun 16 '12 at 14:49

Tim Pote
25k ● 6 ● 59 ● 64

3 +1, but a more related example to demonstrate usefulness might be using a `List<FlyingMachine>` rather than a `List<Jet>`. – cheeken Jun 16 '12 at 15:00

@cheeken I was just updating my answer as you posted that. See my updates. – Tim Pote Jun 16 '12 at 15:02

26

> What is the advantage of polymorphism when both `flm.fly()` and `j.fly()` give me the same answer?

The advantage is that

```
FlyingMachines flm = new Jet();
flm.fly();
```

returns

```
"Start, Taxi, Fly"
```

instead of

```
"No implementation"
```

That's polymorphism. You call `fly()` on an object of type `FlyingMachine` and it still knows that it is in fact a `Jet` and calls the appropriate `fly()` method instead of the wrong one which outputs `"No implementation"`.

That means you can write methods that work with objects of type `FlyingMachine` and feed it with all kinds of subtypes like `Jet` or `Helicopter` and those methods will always do the right thing, i.e. calling the `fly()` method of the appropriate type instead of always doing the same thing, i.e. outputting `"No implementation".`

Polymorphism

Polymorphism is not useful in your example.

- a) It gets useful when you have different types of objects and can write classes that can work with all those different types because they all adhere to the same API.
- b) It also gets useful when you can add new `FlyingMachine`s to your application without changing any of the existing logic.

a) and b) are two sides of the same coin.

Let me show how.

Code example

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

import static java.lang.System.out;

public class PolymorphismDemo {

    public static void main(String[] args) {
        List<FlyingMachine> machines = new ArrayList<FlyingMachine>();
        machines.add(new FlyingMachine());
        machines.add(new Jet());
        machines.add(new Helicopter());
        machines.add(new Jet());

        new MakeThingsFly().letTheMachinesFly(machines);
    }
```

```
}

class MakeThingsFly {
    public void letTheMachinesFly(List<FlyingMachine> flyingMachines) {
        for (FlyingMachine flyingMachine : flyingMachines) {
            flyingMachine.fly();
        }
    }
}

class FlyingMachine {
    public void fly() {
        out.println("No implementation");
    }
}

class Jet extends FlyingMachine {
    @Override
    public void fly() {
        out.println("Start, taxi, fly");
    }

    public void bombardment() {
        out.println("Fire missile");
    }
}

class Helicopter extends FlyingMachine {
    @Override
    public void fly() {
        out.println("Start vertically, hover, fly");
    }
}
```

Explanation

a) The `MakeThingsFly` class can work with everything that is of type `FlyingMachine`.

b) The method `letTheMachinesFly` also works without any change (!) when you add a new class, for example `PropellerPlane`:

```
public void letTheMachinesFly(List<FlyingMachine> flyingMachines) {
        for (FlyingMachine flyingMachine : flyingMachines) {
            flyingMachine.fly();
        }
    }
}
```

That's the power of polymorphism. You can implement the open-closed-principle with it.

Share Edit Follow

edited Oct 2 '15 at 14:29

answered Oct 2 '15 at 14:06

Lernkurve

18k ● 24 ● 79 ● 111

Very nicely explained. Thank you! – Neha Chaudhary Dec 28 '20 at 17:10

Happy to hear that, @NehaChaudhary! – Lernkurve Dec 29 '20 at 11:31

15

The reason why you use polymorphism is when you build generic frameworks that take a whole bunch of different objects **with the same interface**. When you create a new type of object, you don't need to change the framework to accommodate the new object type, as long as it follows the "rules" of the object.

So in your case, a more useful example is creating an object type "Airport" that accepts different types of FlyingMachines. The Airport will define a "AllowPlaneToLand" function, similar to:

```
//pseudocode
void AllowPlaneToLand(FlyingMachine fm)
{
    fm.LandPlane();
}
```

As long as each type of FlyingMachine defines a proper LandPlane method, it can land itself properly. The Airport doesn't need to know anything about the FlyingMachine, except that to land the plane, it needs to invoke LandPlane on the FlyingMachine. So the Airport no longer needs to change, and can continue to accept new types of FlyingMachines, be it a handglider, a UFO, a parachute, etc.

So polymorphism is useful for the frameworks that are built around these objects, that can generically access these methods without having to change.

Share Edit Follow

answered Jun 16 '12 at 15:04

steve8918

1,688 ● 4 ● 24 ● 38

10

let's look at OO design first, inheritance represents a IS-A relationship, generally we can say something like "let our `FlyingMachines` fly". every specific `FlyingMachines` (sub class) IS-A `FlyingMachines` (parent class), let say `Jet`, fits this "let our `FlyingMachines` fly", while we want this flying actually be the fly function of the specific one (sub class), that's polymorphism take over.

so we do things in abstract way, oriented interfaces and base class, do not actually depend on detail implementation, polymorphism will do the right thing!

Share Edit Follow

answered Jun 16 '12 at 15:30

Cruis

347 ● 2 ● 9

6

[Polymorphism] (both runtime and compile time) is necessary in Java for quite a few reasons.

Method overriding is a *run time polymorphism* and overloading is *compile time polymorphism.*

Few of them are(some of them are already discussed):

1. **Collections**: Suppose you have multiple type of flying machines and you want to have them all in a single collection. You can just define a list of type `FlyingMachines` and add them all.

   ```
   List<FlyingMachine> fmList = new ArrayList<>();
   fmList.add(new new JetPlaneExtendingFlyingMachine());
   fmList.add(new PassengerPlanePlaneExtendingFlyingMachine());
   ```

   The above can be done only by polymorphism. Otherwise you would have to maintain two separate lists.

2. **Caste one type to another** : Declare the objects like :

   ```
   FlyingMachine fm1 = new JetPlaneExtendingFlyingMachine();
   FlyingMachine fm2 = new PassengerPlanePlaneExtendingFlyingMachine();
   fm1 = fm2; //can be done
   ```

3. **Overloading**: Not related with the code you gave, But overloading is also another type of polymorphism called **compile time polymorphism**.

4. Can have a single method which accepts type `FlyingMachine` handle all types i.e. subclasses of FlyingMachine. Can only be achieved with `Polymorphism`.

Share Edit Follow edited Jun 19 '17 at 7:27 answered Jun 6 '17 at 18:37

Pritam Banerjee
15.7k ● 10 ● 73 ● 95

4

It doesn't add much if you are going to have just Jets, the advantage will come when you have different FlyingMachines, e.g. Aeroplane

Now that you've modified to include more classes, the advantage of polymorphism is that abstraction from what the specific type (and business concept) of the instance you receive, you just care that it can fly

Share Edit Follow edited Jun 16 '12 at 15:06 answered Jun 16 '12 at 14:52

Sebastian Piu
7,558 ● 1 ● 30 ● 49

Thanks @Sebastian Piu - Let me add two more child classes 1) Heli 2) Aeroplane - So i know my question is still same. – baig772 Jun 16 '12 at 14:59

3

Both `flm.fly()` and `j.fly()` give you the same answer because of the type of the instance is actually the same, which is `Jet`, so they are behave the same.

You can see the difference when you:

```
FlyingMachines flm = new FlyingMachines();
flm.fly();

Jet j = new Jet();
j.bombarment();
j.fly();
```

Polymorphism is define as same method signature with difference behaviour. As you can see, both `FlyingMachines` and `Jet` have method `fly()` defined, but the method is implemented differently, which consider as behave differently.

See aa

- Polymorphism

Share Edit Follow

edited Jun 16 '12 at 14:57

answered Jun 16 '12 at 14:51

Pau Kiat Wee
9,185 ● 40 ● 38

1 No, overriding is defined as the same method signature with different behavior. Polymorphism is a broader view that essentially encompassing constructing an object of one type, and treating it as another type (ie, treating Jet as a FlyingMachine). Inheritance, overriding and overloading are tools that allow you to achieve effective polymorphism (overriding and overloading are what is called ad-hoc polymorphism). – Matt Jun 16 '12 at 16:02

3

Let's add one more class in this, It help's you to understand use of polymorphism..

```
class FlyingMachines {
    public void fly() {
        System.out.println("No implementation");
    }
}

class Jet extends FlyingMachines {
    public void fly() {
        System.out.println("Start, Jet, Fly");
    }
}
```

```
class FighterPlan extends FlyingMachines {
    public void fly() {
        System.out.println("Start, Fighter, Fight");
    }
}

public class PolymorphicTest {
    public static void main(String[] args) {
        FlyingMachines flm = new Jet();
        flm.fly();

        FlyingMachines flm2 = new FighterPlan();
        flm2.fly();
    }
}
```

Output:

```
Start, Jet, Fly
Start, Fighter, Fight
```

Share Edit Follow

answered Mar 29 '17 at 9:00

Vivek Buddhadev

137 ● 16

3

Polymorphism can also help our code to remove the "if" conditionals which is intended to produce production level code because removing conditionals will increase the code readability and helps us to write better unit test cases, we know for "n" if cases there comes n!(n factorial) possibilities.

Let us see how

if you have class FlyingMachine and which takes a string in the constructor defining the type of FlyMachine as below

```
class FlyingMachine{

    private type;

    public FlyingMachine(String type){
            this.type = type;
    }

    public int getFlyingSpeedInMph {
         if(type.equals("Jet"))
              return 600;
         if(type.equals("AirPlane"))
              return 300;
    }

 }
```

We can create two instances of FlyingMachine as

```
FlyingMachine jet = new FlyingMachine("Jet");
FlyingMachine airPlane = new FlyingMachine("AirPlane");
```

and get the speeds using

```
jet.fylingSpeedInMph();
airPlane.flyingSpeedInMph();
```

But if you use polymorphism you are going to remove the if conditions by extending the generic FlyMachine class and overriding the getFlyingSpeedInMph as below

```
class interface FlyingMachine {

     public int abstract getFlyingSpeedInMph;
}


class Jet extends FlyingMachine {
    @Override
    public int getFlyingSpeedInMph(){
          return 600;
    }
}

class Airplane extends FlyingMachine {
    @Override
    public int getFlyingSpeedInMph(){
          return 600;
    }
}
```

Now you can get the flying speeds as below

```
FlyingMachine jet = new Jet();
jet.flyingSpeed();

FlyingMachine airPlane = new AirPlane();
airPlane.flyingSpeed();
```

Share Edit Follow

edited Oct 8 '18 at 15:14

answered Dec 3 '16 at 13:39

murali krish

1,255 ● 15 ● 22

1

Polymorphism gives you benefits only if you need Polymorphism. It's used when an entity of your conceptual project can be seen as the specialization of another entity. The main idea is "specialization". A great example stands in the so called

Taxonomy,for example applied to living beings. Dogs and Humans are both Mammals. This means that, the class Mammals group all the entities that have some properties and behaviors in common.

Also, an ElectricCar and a DieselCar are a specialization of a Car. So both have a isThereFuel() because when you drive a car you expect to know if there's fuel enough for driving it. Another great concept is "expectation".

It's always a great idea to draw an ER (entity relationship) diagram of the domain of your software before starting it. That's because your are forced to picture which kind of entities are gonna be created and, if you're able enough, you can save lot of code finding common behaviors between entities. But saving code isn't the only benefit of a good project.

You might be interested in finding out the so called "software engineering" that it's a collection of techniques and concepts that allows you to write "clean code" (there's also a great book called "Clean code" that's widely suggested by pro-grammes).

Share Edit Follow

answered Oct 2 '15 at 14:20

Luca Marzi
748 ● 2 ● 6 ● 21

Here, for this particular code, there is no need of polymorphism.

0

Let's understand why and when we need polymorphism.

Suppose there are different kinds of machines (like car, scooter, washing machine, electric motor, etc.) and we know that every machine starts and stops. But the logic to start and stop a machine is different for each machine. Here, every machine will have different implementations to start and stop. So, to provide different implementations we need polymorphism.

Here we can have a base class machine with `start()` and `stop()` as its methods and each machine type can extend this functionality and `@Override` these methods.

Share Edit Follow

edited May 17 '18 at 15:55

answered May 17 '18 at 14:21

Jayshree
21 ● 4

---

polymorphism as stated clear by itself, a one which mapped for many.

0

java is a oops language so it have implementation for it by abstract, overloading and overriding

remember java would not have specification for run time polymorphism.

it have some best of example for it too.

```
public abstract class Human {

    public abstract String getGender();

}

class Male extends Human
{
    @Override
    public String getGender() {
        return "male";
    }
}

class Female extends Human
{
    @Override
    public String getGender() {
        return "female";
    }
}
```

Overriding

redefine the behavior of base class. for example i want to add a speed count int the existing functionality of move in my base Car.

Overloading

can have behavior with same name with different signature. for example a particular president speaks clear an loud but another one speaks only loud.

Share Edit Follow

edited Sep 17 '14 at 17:47

Jonathan Drapeau

2,599 ● 2 ● 24 ● 32

answered Jan 9 '14 at 7:11

RTA

1,191 ● 2 ● 13 ● 31

0

The good reason for why Polymorphism is need in java is because the concept is extensively used in implementing inheritance.It plays an important role in allowing objects having different internal structures to share the same external interface.

Share Edit Follow

answered Apr 13 '14 at 5:49

Harkaran

1