

Professional Skills
Agile Fundamentals
Jira
Git
Databases Introduction
Java Beginner
What is Java?
Installation
Hello World Example
Data Types
Packages
Naming Conventions Cheat Sheet
Flow of Control
Class Members
Operators
Conditionals
Iteration
Arrays
ArrayList
Enhanced For Loops
String Manipulation
Class Constructors
Access Modifiers
Installing Java & Maven To PATH
Object-Oriented Programming Principles
Encapsulation
Inheritance
Polymorphism
Abstraction
Interfaces
Type Casting
Static
Final
Garbage Collection
Input With Scanner
Pass by Value/Reference
JUnit

Polymorphism

Contents

- [Overview](#)
- [Tutorial](#)
 - [Example](#)
 - [Animal Class](#)
 - [Cow Class](#)
 - [Main Class](#)
- [Exercises](#)

Overview

Polymorphism is one of the four principles of object-orientated programming (OOP). It is the ability of an object being able to take on many forms, the most common use of this is when a parent class reference is used to refer to a child class object. We can check if a Java object is polymorphic by creating **instanceof** tests, if an object can pass more than one of these "is a" tests then it is considered to be polymorphic. However in Java since every class inherits from the **Object** class they will always pass at least two "is a" tests since they will be an instance of their class and the **Object** class, therefore all Java objects are considered to be polymorphic.

Tutorial

Example

If we have the following classes of **Animal** and **Cow** which inherits from **Animal** then **Cow** will be polymorphic because it passes three "is a" tests, those being:

- Cow is a Cow
- Cow is a Animal
- Cow is a Object

Animal Class

```
public class Animal {  
  
    public void sleep() {  
        System.out.println("zzz");  
    }  
}
```

Cow Class

```
public class Cow extends Animal {  
  
    public void speak() {  
        System.out.println("moo");  
    }  
}
```

Main Class

We can now instantiate an object of **Cow** using the three classes that it belongs to.

<div><div></div><div>Test Driven Development</div></div> <div><div></div><div>UML Basics</div></div> <div><div></div><div>JavaDoc</div></div> <div><div></div><div>Peer Programming</div></div> <div><div></div><div>Code Reviews</div></div>
Maven
Testing (Foundation)
Java Intermediate
HTML
CSS
Javascript
Spring Boot
Selenium
Sonarqube
Advanced Testing (Theory)
Cucumber
MongoDB
Express
NodeJS
React
Express-Testing
Networking
Security
Cloud Fundamentals
AWS Foundations
AWS Intermediate
Linux
DevOps
Jenkins Introduction
Jenkins Pipeline
Markdown
IDE Cheatsheet

```
public class Main {

    public static void main(String[] args) {
        Cow cow = new Cow();
        Animal animal = new Cow();
        Object object = new Cow();
    }
}
```

We can also run the methods within the classes against these new objects. However, each object will only be able to run the methods inside the class that it is a type of and the methods within the class or classes that it inherits from. So the `cow` object will be able to run all methods within `Cow`, `Animal`, and `Object`; but `object` will only be able to run methods within the `Object` class.

If we want to be able to use the methods within the `Cow` class from the `object` reference variable then we would simply have to cast the method call. Below we are calling both the `speak()` and `sleep()` method on each of the objects that we created, casting where needed.

```
public class Main {

    public static void main(String[] args) {
        System.out.println("====Cow====");
        Cow cow = new Cow();
        cow.speak();
        cow.sleep();
        System.out.println("====");
        System.out.println("====Animal====");
        Animal animal = new Cow();
        ((Cow) animal).speak();
        animal.sleep();
        System.out.println("====");
        System.out.println("====Object====");
        Object object = new Cow();
        ((Cow) object).speak();
        ((Cow) object).sleep();
        System.out.println("====");
    }
}
```

As you can see we need to cast both method calls when running them against the `object` reference variable, this is because neither method belongs to the `Object` class, and instead belong to its subclasses. Whereas when we run the methods against the `animal` reference variable we need to cast the `speak()` method call because it belongs to the `Cow` class which is a subclass of `Animal`, however the `sleep()` method belongs to the `Animal` class so we do not need to cast that method call.

If we run the above code we get the following console output.

```
====Cow====
moo
ZZZ
=====
====Animal====
moo
ZZZ
=====
====Object====
moo
ZZZ
=====
```

Exercises

There are no exercises for this module.

