HowToDoInJava

Guide to Inheritance

Last Updated: January 29, 2022 By: Lokesh Gupta Java Object Oriented Programming Inheritance, Java OOP

Inheritance in java (IS-A relationship) is referred to the ability where child objects inherit or acquire all the properties and behaviors from parent object. In object oriented programming, inheritance is used to promote the code re-usability.

In this Java tutorial, we will learn about **inheritance types** supported in Java and **how inheritance is achieved** in Java applications.

Table of Contents

1. What is inheritance in Java

As said before, inheritance is all about inheriting the **common state and behavior** of parent class (super class) by it's derived class (sub class or child class). A sub class can inherit all **non-private members** from super class, by default.

In java, **extends** keyword is used for inheritance between classes. let's see a quick inheritance example.

1.1. Java inheritance example

Let's say we have `Employee` class. Employee class has all common attributes and methods which all employees must have within organization. There can be other specialized employees as well e.g. `Manager`. Managers are regular employees of organization but, additionally, they have few more attributes over other employees e.g. they have reportees or subordinates.

Let's design above classes.

```
Employee.java

public class Employee
{
    private Long id;
    private String firstName;
    private String lastName;

    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    @Override
    public String toString() {
        return "Employee [id=" + id + ", firstName=" + firstName + ", lastName="
    }
}
```

```
Manager.java

import java.util.List;

public class Manager extends Employee
{
    private List<Employee> subordinates;

    public List<Employee> getSubordinates() {
        return subordinates;
    }

    public void setSubordinates(List<Employee> subordinates) {
        this.subordinates = subordinates;
    }

    @Override
    public String toString() {
        return "Manager [subordinates=" + subordinates + ", details=" + super.toS
    }
}
```

In above implementation, employees have common attributes like `id`, `firstName` and `lastName`; while manager has it's specialized `subordinates` attribute only. To inherit all non-private members from `Employee` class (in this case getter and setter methods), `Manager extends Employee` is used.

Let's see how it works?

```
Main.java

public class Main
{
    public static void main(String[] args)
    {
        Manager mgr = new Manager();
        mgr.setId(1L);
        mgr.setFirstName("Lokesh");
        mgr.setLastName("Gupta");

        System.out.println(mgr);
    }
}
```

Program Output.

```
Console

Manager [subordinates=null, details=Employee [id=1, firstName=Lokesh, lastName=Gu
```

Clearly, `Manager` class is able to use members of `Employee` class. This very behavior is called inheritance. Simple, isn't it?

Now consider if we do not use inheritance. Then we would have defined id, firstName and lastName in both classes. It would have caused code duplication which always create problems in code maintenance.

2. Types of inheritance in Java

In Java, inheritance can be one of **four types** – depending on classes hierarchy. Let's learn about all four types of inheritances.

2.1. Single inheritance

This one is simple. There is one Parent class and one Child class. **One child class extends one parent class**. It's single inheritance. The above example code (employee and manager) is example of single inheritance.

Java Single Inheritance

2.2. Multi-level inheritance

In multilevel inheritance, there will be inheritance between more than three classes in such a way that a **child class will act as parent class for another child class**. Let's understand with a diagram.

Multilevel Inheritance

In above example, Class `B` extends class `A`, so class `B` is child class of class `A`. But `C` extends `B`, so `B` is parent class of `C`. So `B` is parent class as well as child class also.

2.3. Hierarchical inheritance

In hierarchical inheritance, there is **one super class and more than one sub classes** extend the super class.

Hierarchical Inheritance

These subclasses `B`, `C`, `D` will share the common members inherited from `A`, but they will not be aware of members from each other.

2.4. Multiple inheritance

In multiple inheritance, a class can **inherit the behavior from more than one parent classes** as well. Let's understand with diagram.

Multiple inheritance

In diagram, `D` is extending class `A` and `B`, both. In this way, `D` can inherit the non-private members of both the classes.

BUT, in java, you cannot use `extends` keyword with two classes. So, how multiple inheritance will work?

> Till JDK 1.7, multiple inheritance was not possible in java. But **from JDK 1.8 onwards, multiple inheritance is possible via use of interfaces with default methods**.

3. Accessing inherited parent class members

Now we know that using four types of inheritance mechanisms, we can access non-private members of parent classes. Let's see how individual member can be accessed.

3.1. Parent class constructors

Constructors of super class can be called via `super` keyword. There are only two rules:

1. `super()` call must be made from child class constructor.
2. `super()` call must be first statement inside constructor.

```
public class Manager extends Employee
{
    public Manager()
    {
        //This must be first statement inside constructor
        super();

        //Other code after super class
    }
}
```

3.2. Parent class fields

In java, non-private member fields can be inherited in child class. You can access them using dot operator e.g. `manager.id`. Here `id` attribute is inherited from parent class `Employee`.

You need to be careful when dealing with fields with same name in parent and child class. Remember that **java fields cannot be overridden**. Having same name field will hide the field from parent class – while accessing via child class.

In this case, attribute accessed will be decided based on the **class of reference type**.

```
ReferenceClass variable = new ActualClass();
```

In above case, member field will be accessed from `ReferenceClass`. e.g.

```
//Parent class
public class Employee
{
    public Long id = 10L;
}

//Child class
public class Manager extends Employee
```

```
{
    public Long id = 20L;   //same name field
}

public class Main {
    public static void main(String[] args)
    {
        Employee manager = new Manager();
        System.out.println(manager.id);     //Reference of type Employee

        Manager mgr = new Manager();
        System.out.println(mgr.id);     //Reference of type Manager
    }
}

Output:

10
20
```

3.3. Parent class methods

Opposite to field access, method access uses the type of actual object created in runtime.

java]ReferenceClass variable = new ActualClass();[/java]

In above case, member method will be accessed from `ActualClass`. e.g.

```
public class Employee
{
    private Long id = 10L;

    public Long getId() {
        return id;
    }
}

public class Manager extends Employee
{
    private Long id = 20L;

    public Long getId() {
        return id;
    }
}

public class Main
{
    public static void main(String[] args)
    {
        Employee employee = new Employee();     //Actual object is Employee Type
        System.out.println(employee.getId());

        Employee manager = new Manager();       //Actual object is Manager Type
        System.out.println(manager.getId());

        Manager mgr = new Manager();       //Actual object is Manager Type
        System.out.println(mgr.getId());
    }
}

Output:

10
20
20
```

4. Summary

Let's summarize what we learned about **java inheritance**:

- Inheritance is also known **IS-A** relationship.
- It provides child class the ability to inherit non-private members of parent class.
- In java, inheritance is achieved via `extends` keyword.
- From Java 8 onward, you can use interfaces with default methods to achieve multiple inheritance.
- Member fields are accessed from reference type class.
- Member methods are accessed from actual instance types.

Drop me any question, you might have, in comments section.

Happy Learning !!

Was this post helpful?

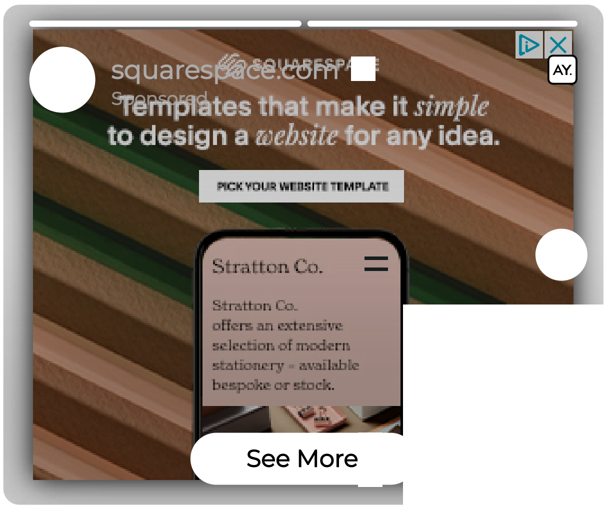
Let us know if you liked the post. That's the only way we can improve.

Yes

No

Recommended Reading:

1. Multiple Inheritance in Java
2. Guide to Abstraction
3. Guide to Polymorphism
4. Overloading vs Overriding in Java
5. Interface vs Abstract Class in Java
6. Encapsulation vs Abstraction in Java
7. Overriding final static method in Java
8. Java Access Modifiers
9. Constructors in Java
0. Java extends vs implements Keywords


squarespace.com
Sponsored
Templates that make it simple to design a website for any idea.
PICK YOUR WEBSITE TEMPLATE
Stratton Co.
Stratton Co. offers an extensive selection of modern stationery – available bespoke or stock.
See More

Join 7000+ Awesome Developers

Get the latest updates from industry, awesome resources, blog updates and much more.

Email Address

Subscribe

** We do not spam !!*

3 thoughts on “Guide to Inheritance”

yee

December 15, 2019 at 3:06 pm

is there a typo here ?

```
ReferenceClass variable = new ActualClass();
```

Reply

Jay Al Serna Gallenero

May 2, 2020 at 12:57 pm

ActualClass mus be replace with ReferenceClasss

Reply

Prashant Raghav

March 3, 2018 at 1:04 pm

Hi lokesh,
In Java can I do
Child_class_object.baseclassmethod(),
Directly

Reply

Leave a Comment

Name *

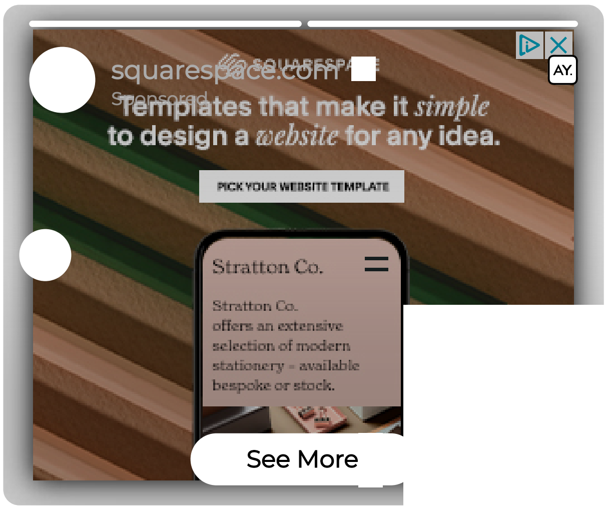
Email *

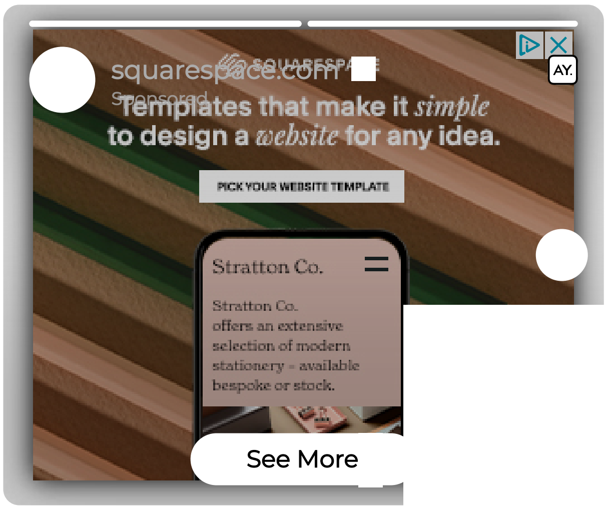
Website

☐ Add me to your newsletter and keep me updated whenever you publish new blog posts

Post Comment

Search …


squarespace.com
Sponsored
SQUARESPACE
Templates that make it simple to design a website for any idea.
PICK YOUR WEBSITE TEMPLATE
Stratton Co.
Stratton Co. offers an extensive selection of modern stationery - available bespoke or stock.
See More


squarespace.com
Sponsored
SQUARESPACE
Templates that make it simple to design a website for any idea.
PICK YOUR WEBSITE TEMPLATE
Stratton Co.
Stratton Co. offers an extensive selection of modern stationery - available bespoke or stock.
See More

HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us
- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial